Реферат

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Реферат

Лит.

Листов

1

74417023, 2024

Пояснительная записка содержит 68 страниц, 33 рисунка, 21 таблицу, 16 листингов, 23 источника, 13 приложений.

ВЕБ-ПРИЛОЖЕНИЕ, EXPRESS, MONGODB, NODE.JS, REACT JS, JAVASCRIPT, TYPESCRIPT, FIREBASE

Целью дипломного проектирования является разработка веб-приложения «Финансовый помощник», которое обеспечивает удобное и эффективное администрирование данных о счетах, доходах, расходах и ограничениях на расходы.

При разработке дипломного проекта была использована платформа Node.js, языки программирования JavaScript и TypeScript, библиотека React.js, протоколы обмена данными HTTP и HTTPS, а также облачные платформы Atlas и Google.

Пояснительная записка состоит из введения, шести разделов и заключения.

В первом разделе описаны анализ предметной области, аналитический обзор литературы, включая обзор аналогов, обзор средств разработки, выбор технологий и библиотек, а также постановка задачи.

Во втором разделе представлены проектирование диаграммы вариантов использования, архитектура приложения, структура базы данных, блок-схема алгоритма приглашения пользователя для совместного администрирования счета, а также блок-схема алгоритма стратегии расходов.

Третий раздел посвящен разработке веб-приложения.

В четвертом разделе проведено тестирование разработанного приложения.

В пятом разделе приведено руководство пользователя.

В шестом разделе рассчитываются экономические параметры, включая себестоимость приложения и его экономическую эффективность.

В заключении представлены итоги дипломного проектирования и задачи, которые были решены в ходе разработки веб-приложения «Финансовый помощник».

Графическая часть представлена в объеме 1,5 листа формата А1.

**Abstract**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Abstract

Лит.

Листов

1

74417023, 2024

The explanatory note of the graduation project contains 68 pages, 33 figures, 21 tables, 16 listings, 23 sources, 13 appendices.

WEB APP, EXPRESS, EXPRESS, MONGODB, NODE JS, REACT JS, TYPESCRIPT, JAVASCRIPT, FIREBASE

The aim of the diploma project is to develop a web application «Financial Assistant», which provides convenient and efficient administration of data on accounts, income, expenses and spending limits.

Node.js platform, JavaScript and TypeScript programming languages, React.js library, HTTP and HTTPS data exchange protocols, as well as Atlas and Google cloud platforms were used in the development of the diploma project.

The explanatory note consists of an introduction, six sections and a conclusion.

The first section describes the analysis of the subject area, analytical literature review including a review of analogues, review of development tools, selection of technologies and libraries, and the problem statement.

The second section presents the design of the use-case diagram, the application architecture, the database structure, the flowchart of the algorithm for inviting a user to co-administer an account, and the flowchart of the spending strategy algorithm.

The third section deals with the development of the web application.

The fourth section tests the developed application.

The fifth section provides the user manual.

The sixth section calculates the economic parameters including the cost-value of the application and its cost-effectiveness.

The conclusion presents the results of the thesis design and the tasks that were solved during the development of the web application «Financial Assistant».

The graphic part is presented in the volume of 1.5 sheets of A1 format.

**Содержание**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Содержание

Лит.

Листов

2

74417023, 2024

[Введение 7](#_Toc168874179)

[1 Аналитический обзор литературы и постановка задачи 8](#_Toc168874180)

[1.1 Анализ предметной области 8](#_Toc168874181)

[1.2 Аналитический обзор литературы 8](#_Toc168874182)

[1.2.1 Сайт MoneyTracker 9](#_Toc168874183)

[1.2.2 Сайт Newretirement 10](#_Toc168874184)

[1.3 Обзор средств разработки 12](#_Toc168874185)

[1.3.1 Обоснование выбора языка программирования 12](#_Toc168874186)

[1.3.2 Обоснование выбора платформы разработки 12](#_Toc168874187)

[1.4 Выбор технологий и библиотек 13](#_Toc168874188)

[1.4.1 Фреймворк Express 13](#_Toc168874189)

[1.4.2 Библиотека React 13](#_Toc168874190)

[1.5 Постановка задачи 14](#_Toc168874191)

[1.6 Выводы по разделу 14](#_Toc168874192)

[2 Проектирование веб-приложения 15](#_Toc168874193)

[2.1 Архитектура приложения 15](#_Toc168874194)

[2.2 Проектирование диаграммы вариантов использования 17](#_Toc168874195)

[2.3 Проектирование структуры базы данных 18](#_Toc168874196)

[2.4’Блок-схема алгоритма приглашения пользователя для совместного администрирования счетов 22](#_Toc168874197)

[2.5 Блок-схема алгоритма стратегии расходов 23](#_Toc168874198)

[2.6 Выводы по разделу 24](#_Toc168874199)

[3 Разработка веб-приложения 25](#_Toc168874200)

[3.1 Разработка серверной части 25](#_Toc168874201)

[3.1.1 Файл package.json 26](#_Toc168874202)

[3.1.2 Взаимодействие с базой данных 27](#_Toc168874203)

[3.1.3 Модели данных 27](#_Toc168874204)

[3.1.4 Реализация маршрутизации 29](#_Toc168874205)

[3.1.5 Контроллеры 30](#_Toc168874206)

[3.1.6’Реализация приглашения пользователя для совместного администрирования счета 31](#_Toc168874207)

[3.1.7 Реализация телеграм-бота 32](#_Toc168874208)

[3.2 Разработка клиентской части 34](#_Toc168874209)

[3.2.1 Реализация аутентификации 35](#_Toc168874210)

[3.2.2 Реализация глобального контекста 36](#_Toc168874211)

[3.2.3 Реализация маршрутизации 38](#_Toc168874212)

[3.2.4 Отправка запросов к серверу 39](#_Toc168874213)

[3.2.5 Реализация стратегии расходов на следующий месяц 40](#_Toc168874214)

[3.3 Выводы по разделу 40](#_Toc168874215)

[4 Тестирование веб-приложения 41](#_Toc168874216)

[4.1 Тестирование страниц авторизации и регистрации 41](#_Toc168874217)

[4.2 Тестирование процесса создания нового счета 42](#_Toc168874218)

[4.3 Тестирование операции создания ограничений 42](#_Toc168874219)

[4.4 Тестирование процесса добавления новой операции 43](#_Toc168874220)

[4.5 Тестирование функций депозитного счета 44](#_Toc168874221)

[4.6 Тестирование функций телеграм-бота 45](#_Toc168874222)

[4.7 Выводы по разделу 46](#_Toc168874223)

[5 Руководство пользователя 47](#_Toc168874224)

[5.1 Методика использования приложения 47](#_Toc168874225)

[5.2 Выводы по разделу 56](#_Toc168874226)

[6 Технико-экономическое обоснование проекта 57](#_Toc168874227)

[6.1 Общая характеристика разрабатываемого программного средства 57](#_Toc168874228)

[6.2 Исходные данные для проведения расчетов и маркетинговый анализ 57](#_Toc168874229)

[6.3 Обоснование цены программного средства 59](#_Toc168874230)

[6.3.1 Расчет затрат рабочего времени на разработку программного средства 59](#_Toc168874231)

[6.3.2 Расчет основной заработной платы 59](#_Toc168874232)

[6.3.3 Расчет дополнительной заработной платы 60](#_Toc168874233)

[6.3.4 Расчет отчислений в ФСЗН и по обязательному страхованию 60](#_Toc168874234)

[6.3.5 Расчет суммы прочих прямых затрат 61](#_Toc168874235)

[6.3.6 Расчет суммы накладных расходов 61](#_Toc168874236)

[6.3.7 Сумма расходов на разработку программного средства 62](#_Toc168874237)

[6.3.8 Расходы на реализацию 62](#_Toc168874238)

[6.3.9 Полная себестоимость 62](#_Toc168874239)

[6.3.10 Определение цены, оценка эффективности 63](#_Toc168874240)

[6.4 Выводы по разделу 64](#_Toc168874241)

[Заключение 66](#_Toc168874242)

[Список использованных источников 67](#_Toc168874243)

[Приложение А. Диаграмма развертывания 69](#_Toc168874244)

[Приложение Б. Диаграмма вариантов использования 70](#_Toc168874245)

[Приложение В. Логическая схема базы данных 71](#_Toc168874246)

[Приложение Г. Блок-схема алгоритма приглашения пользователя для совместного администрирования счета 72](#_Toc168874247)

[Приложение Д. Блок-схема алгоритма стратегии расходов 73](#_Toc168874248)

[Приложение Е. Скриншот работы приложения 74](#_Toc168874249)

[Приложение Ж. Таблица экономических расчетов 75](#_Toc168874250)

[Приложение И. Листинг package.json 76](#_Toc168874251)

[Приложение К. Листинг transactions.ts 77](#_Toc168874252)

[Приложение Л. Листинг телеграм-бота 78](#_Toc168874253)

[Приложение М. Листинг UserAuthContext.js 88](#_Toc168874254)

[Приложение Н. Листинг globalContext.js 89](#_Toc168874255)

[Приложение П. Листинг navigation.js 96](#_Toc168874256)

# Введение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

74417023, 2024

В наше время эффективное администрирование финансов становится крайне важной задачей для многих людей. Необходимость контролировать расходы, следить за доходами и планировать бюджет становится все более актуальной в условиях увеличивающейся финансовой нестабильности и быстро меняющихся экономических условий. Хоть современные технологии и позволяют автоматизировать многие аспекты нашей повседневной жизни, однако многие люди все еще сталкиваются с трудностями при администрировании своих финансов.

Одной из основных причин таких затруднений является отсутствие финансовой грамотности. Многие люди не имеют достаточных знаний о том, как правильно распределять свои доходы, инвестировать средства или планировать крупные покупки. Без понимания базовых принципов финансового планирования и управления деньгами, сложно достигнуть финансовой стабильности и избежать финансовых проблем. Современные образовательные программы и онлайн-курсы могут помочь в приобретении необходимых знаний, но это требует времени и усилий, на которые не каждый готов пойти.

В дополнение к недостатку финансовой грамотности, значительную роль играют и психологические факторы. Страх перед сложными финансовыми решениями, склонность к импульсивным покупкам, неспособность контролировать свои траты и отсутствие четких финансовых целей могут существенно затруднять процесс эффективного администрирования финансов. Приложения для учета расходов и планирования бюджета могут значительно облегчить эту задачу, предоставляя удобные и доступные инструменты для мониторинга и анализа финансовых потоков, что помогает достичь финансового благополучия.

Целью данного дипломного проекта является разработка веб-приложения «Финансовый помощник», которое позволит пользователям эффективно администрировать свои финансовые данные. Проект нацелен на создание всех необходимых функций, учитывая недостатки аналогичных решений, а также внедрение усовершенствований для создания более удобного и комплексного пользовательского опыта. В результате, пользователи смогут лучше контролировать свои финансовые данные, что повысит их финансовую стабильность и уверенность в будущем.

Для достижения поставленной цели необходимо решить следующие задачи:

* + исследовать преимущества и недостатки аналогичных приложений;
  + разработать структуру базы данных;
  + спроектировать диаграмму вариантов использования;
  + спроектировать диаграмму развертывания;
  + реализовать веб-приложение;
  + провести тестирование веб-приложения.

# Аналитический обзор литературы и постановка задачи

ФИО

Подпись

Дата

Лист

1

БГТУ 01.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

1 Аналитический обзор литературы и постановка задачи

Лит.

Листов

7

74417023, 2024

## Анализ предметной области

В настоящее время рынок финансовых приложений находится в стадии активного развития и конкуренции. Пользователи все больше обращают внимание на удобство, разнообразие функций и безопасность таких приложений при выборе инструмента для администрирования своих финансов.

Основными тенденциями на рынке финансовых приложений являются рост спроса на цифровые решения. С каждым годом все больше людей предпочитают использовать цифровые инструменты для управления своими финансами вместо традиционных методов учета и планирования.

Расширение и улучшение доступности для всех пользователей также является важной тенденцией. С учетом разнообразия пользовательской базы, включая людей с ограниченными возможностями и тех, кто не имеет доступа к современным технологиям, разработчики финансовых приложений стараются сделать свои продукты более доступными и удобными для всех категорий пользователей и возможностью доступа через различные устройства и платформы.

Пользовательский опыт также играет значительную роль. Удобство использования и интуитивно понятный интерфейс становятся ключевыми факторами при выборе финансового приложения. Пользователи ожидают от приложения быстрого доступа к нужной информации и возможности быстро совершать необходимые действия.

Персонализация и адаптивность ценятся пользователями. Они хотят иметь возможность настройки приложения под свои потребности и предпочтения, а также его способность адаптироваться к изменяющимся условиям и потребностям.

Анализ существующих финансовых приложений позволяет выявить их преимущества и недостатки, что поможет определить требования к разрабатываемому веб-приложению «Финансовый помощник». Изучение конкурентов поможет выявить успешные стратегии и особенности приложений, а также обнаружить пробелы для улучшения разрабатываемого продукта.

## Аналитический обзор литературы

На сегодняшний день в области администрирования финансов ключевым направлением становится повышение финансовой грамотности населения и обеспечение доступности информации о финансовых инструментах и услугах [1]. Подходящие веб-приложения могут стать неотъемлемой частью этого процесса. Для выполнения основной задачи необходимо провести аналитический обзор литературы. Рассмотрим результаты анализа аналогов веб-приложений в области администрирования финансов.

### Сайт MoneyTracker

MoneyTracker [2] – это сайт для учета финансов, предоставляющий пользователям возможность эффективно отслеживать свои расходы и доходы. Он разработан с упором на простоту использования, обеспечивая удобство в управлении финансами. Главная страница MoneyTracker приведена на рисунке 1.1.

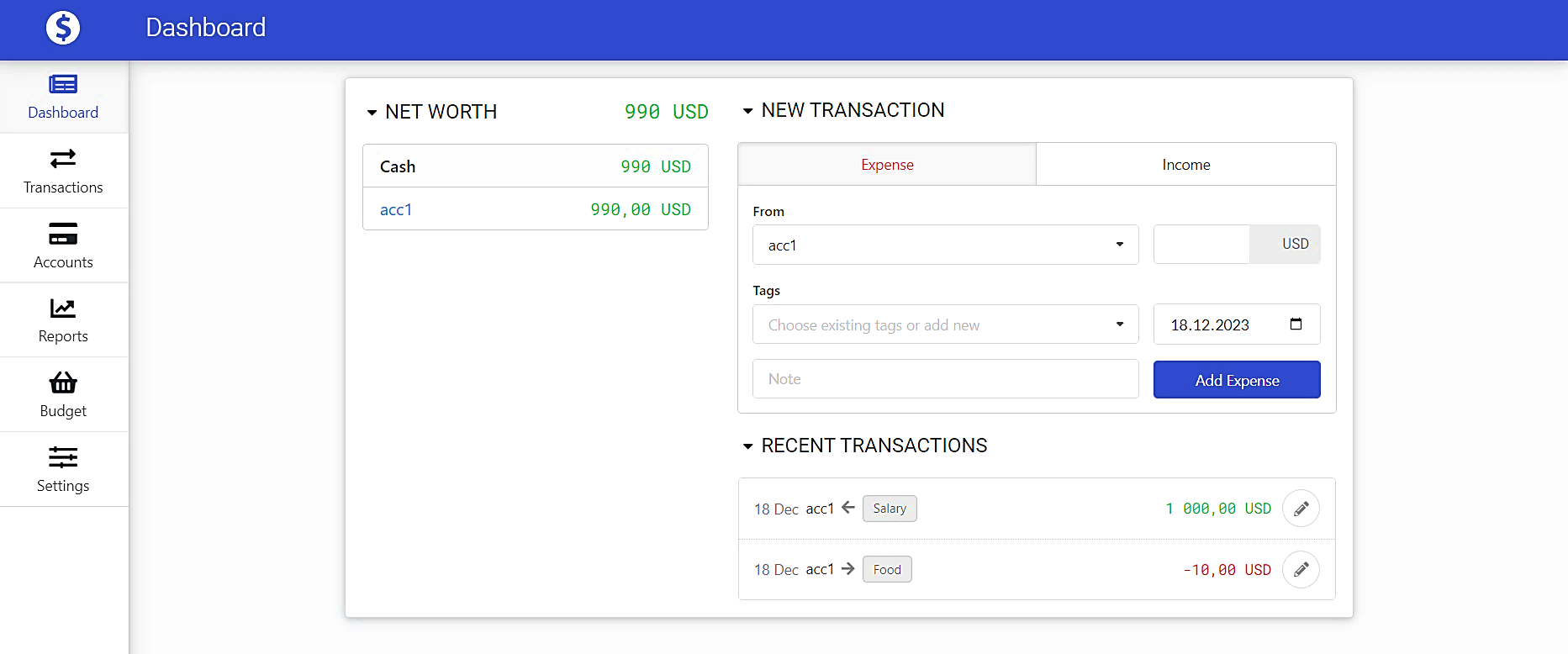


Рисунок 1.1 – Возможности, предлагаемые сайтом MoneyTracker

На данном сайте пользователи могут легко добавлять свои транзакции, управлять счетами, отслеживать расходы и доходы, а также управлять своим бюджетом. Кроме того, оно позволяет в любой момент добавить и/или изменить валюту счета и импортировать или экспортировать транзакции. Данный функционал представлен на рисунке 1.2.

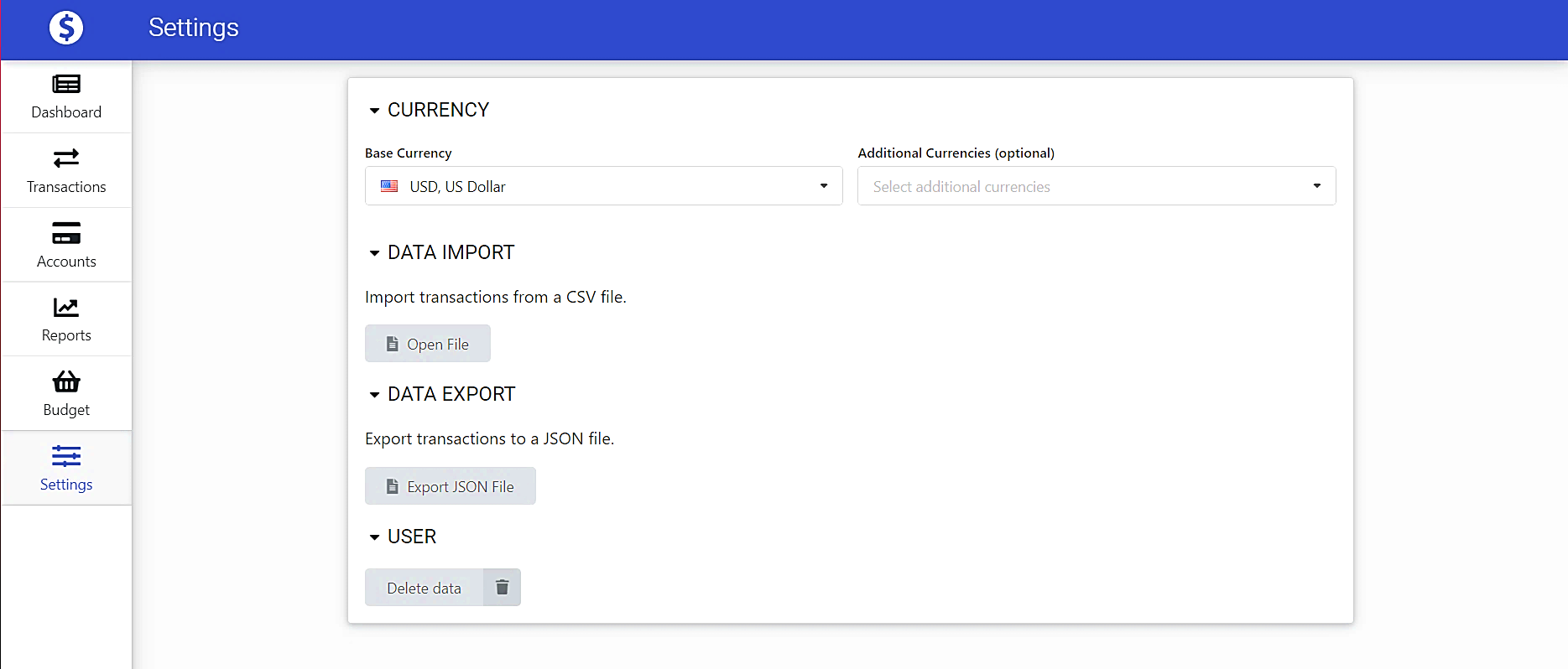


Рисунок 1.2 – Пользовательские настройки

Данный сервис имеет ряд плюсов, таких как: простота в использовании и минималистичный дизайн, графическое представление финансов, возможность отслеживать доходы и расходы нескольких счетов. Также, MoneyTracker обеспечивает возможность генерации отчетов и аналитики по финансовым операциям, что позволяет пользователям более детально изучить свои траты и доходы, выявить тренды и принимать более обоснованные финансовые решения.

Однако, у данного сервиса есть и недостатки: отсутствует функция для установки ограничений, хотя она отображена на навигационной панели, а также ограниченный функционал по сравнению с конкурентами. Кроме того, пользователи отмечают, как недостаток отсутствие мобильного приложения, что снижает удобство использования сервиса на ходу. Еще одним минусом является недостаток аналитических инструментов для детального анализа расходов, что может быть важно для более продвинутых пользователей. Также, в данном веб-приложении нет возможности совместно управлять данными счетов, что может быть полезно для управления семейным бюджетом.

### Сайт Newretirement

Newretirement [3] – это сайт, который предоставляет широкий спектр бесплатных инструментов для управления личными финансами и планирования пенсии. Пользователи могут воспользоваться различными функциями сайта, включая анализ своего текущего финансового положения, определение своих целей на пенсию, планирование бюджета на будущее, а также получение персонализированных рекомендаций по наиболее эффективному управлению своими средствами. Дополнительно, функционал данного сайта позволяет пользователям легко вводить и отслеживать свои финансовые данные. Секции для ввода информации о доходах, расходах, активах и обязательствах, а также детализированные графики и отчеты представлены на рисунке 1.3.



Рисунок 1.3 – Пользовательский интерфейс сайта Newretirement

На данном сайте можно администрировать данные нескольких счетов, просматривать график доходов и налогов, общую сумму сбережений, готовую оценку компетентности своего плана по администрированию финансов. Описанный функционал представлен на рисунке 1.4.

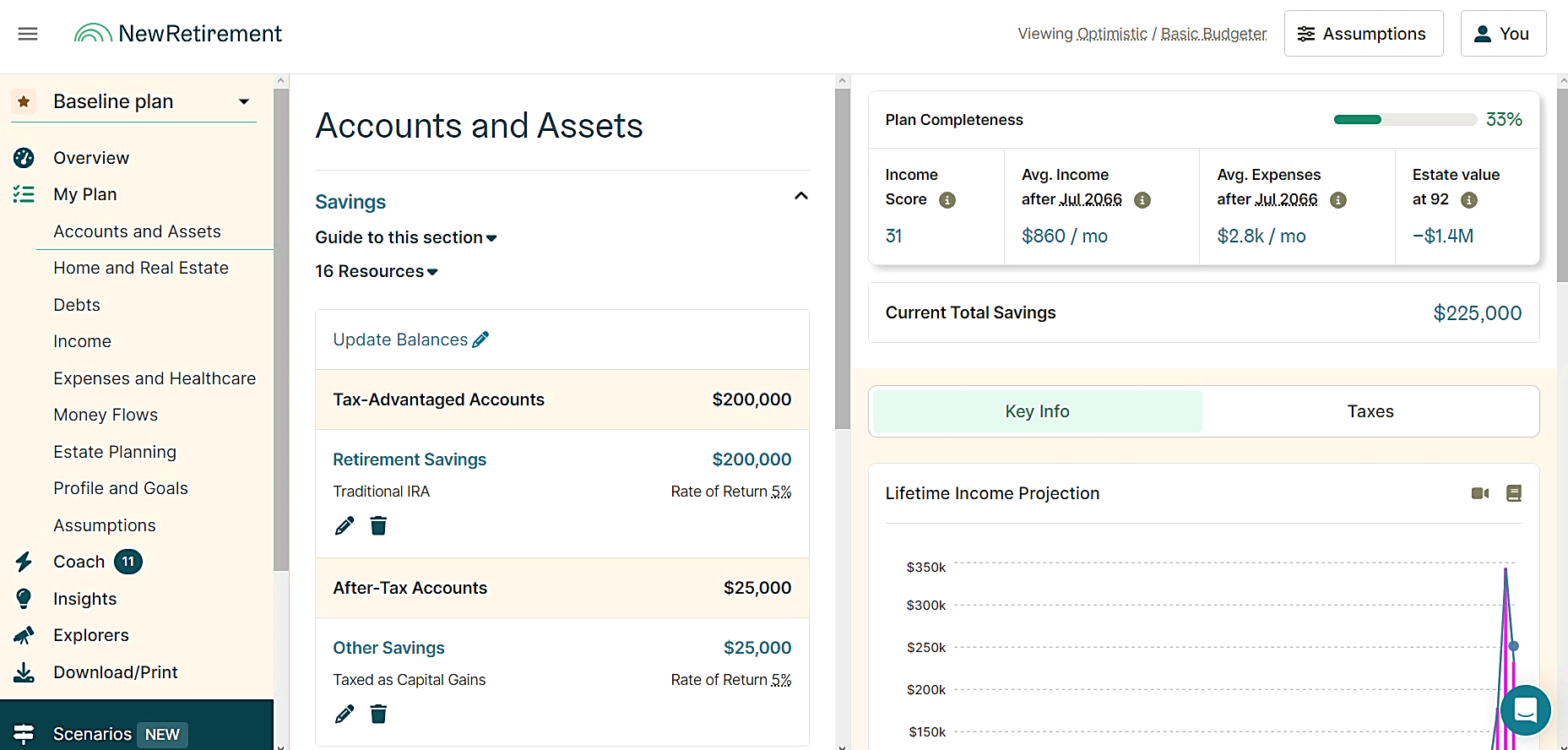


Рисунок 1.4 – Пример настройки счетов на сайте Newretirement

Также, данный ресурс позволяет составить план сбережений и инвестиций с учетом собственных стратегий, касающихся инфляции, роста собственных расходов, продолжительности жизни, возраста выхода на пенсию. Еще, данное веб-приложение предоставляет пользователям возможность моделировать различные финансовые сценарии и их потенциальные последствия для личного бюджета. Страница c выбором своих предположений указана на рисунке 1.5.

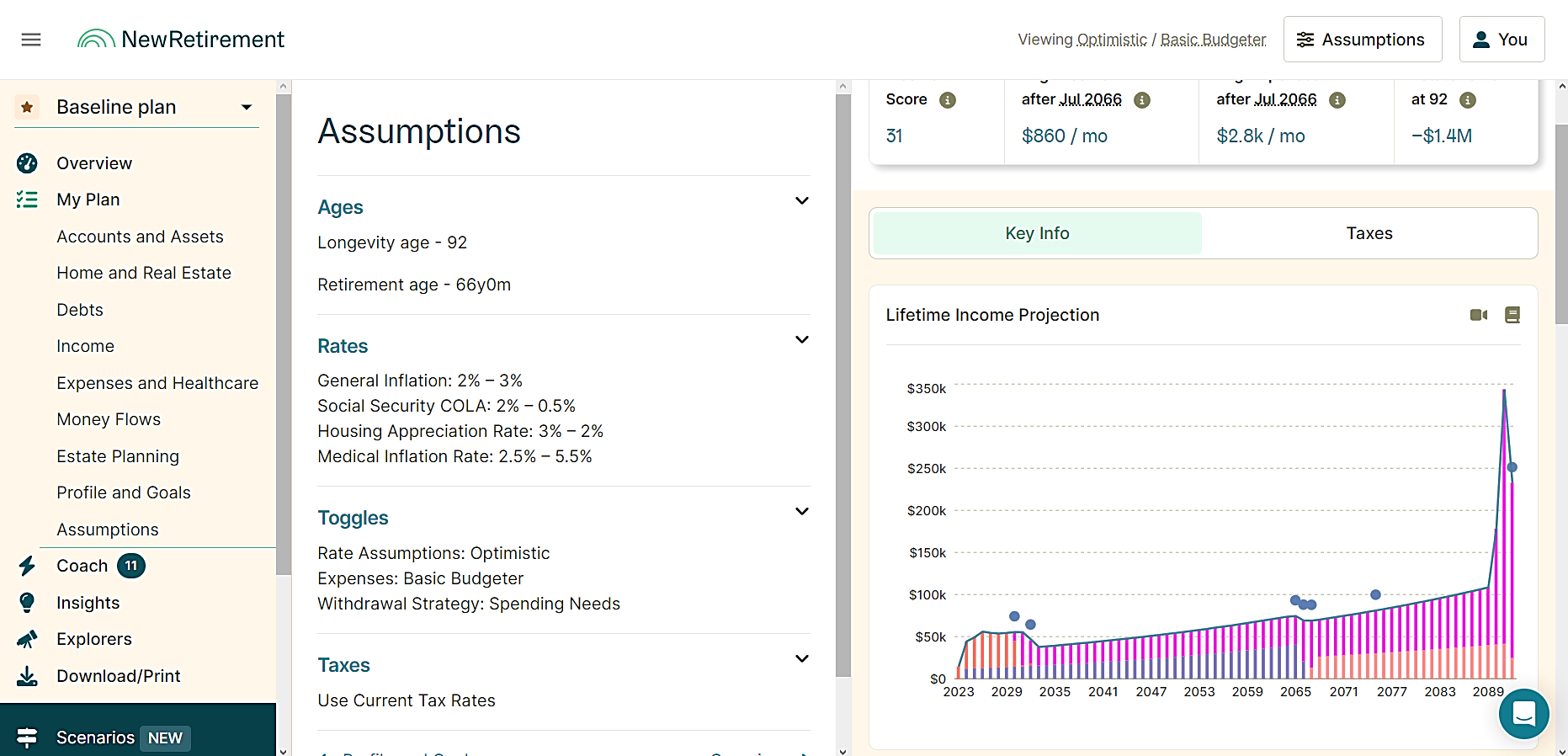


Рисунок 1.5 – Страница с выбором стратегий

Данный сервис предоставляет удобные бесплатные инструменты для администрирования личных финансов и планирования пенсии. Среди преимуществ можно выделить наличие бесплатных инструментов, возможность просматривать прогнозы по накоплениям и доходам в пенсии, а также гибкий подход к учету всех аспектов личных финансов.

Однако, следует учесть некоторые недостатки. В частности, отсутствие некоторых продвинутых функций, доступных только в платных версиях приложения, может быть ограничивающим фактором для пользователей, ищущих более расширенные возможности. Также стоит отметить, что интерфейс сервиса может восприниматься как перегруженный, что усложняет его использование для новых пользователей, несмотря на его многогранный функционал.

## Обзор средств разработки

В данном подразделе будет обоснован выбор языка программирования, среды разработки и платформы, которые будут использованы в разработке приложения.

### Обоснование выбора языка программирования

Для разработки веб-приложения был выбран JavaScript [4]. Этот язык программирования обладает обширным набором библиотек и фреймворков, которые упрощают процесс разработки, предоставляя разнообразные инструменты и решения для разнообразных задач. Благодаря поддержке практически всеми современными браузерами, JavaScript становится идеальным выбором для создания веб-приложений, которые должны корректно функционировать на различных устройствах и платформах [5]. Важно отметить, что сообщество разработчиков JavaScript активно развивается, постоянно внедряя новые возможности и улучшения в язык и его экосистему.

### Обоснование выбора платформы разработки

Выбор Node.js [6] для серверной разработки обоснован несколькими ключевыми преимуществами. Node.js позволяет создавать полноценные приложения на JavaScript, обеспечивает эффективное взаимодействие с внешними библиотеками и командами JavaScript, включая работу в качестве веб-сервера. Кроме того, его широкое сообщество разработчиков и большое количество доступных модулей способствуют быстрому и удобному развитию проектов.

Преимущества Node.js включают асинхронную модель выполнения и событийно-ориентированную архитектуру, которые позволяют обрабатывать множество запросов без блокировки основного потока исполнения, обеспечивая высокую отзывчивость и производительность веб-приложений.

Единый язык программирования – JavaScript – как на клиентской, так и на серверной стороне упрощает разработку, обучение и поддержку приложений, позволяя использовать одни и те же навыки для обеих сторон, что повышает производительность и эффективность работы над проектом.

Совместно эти преимущества делают Node.js привлекательным выбором для разработки современных и высоконагруженных веб-приложений.

## Выбор технологий и библиотек

В данном разделе представлено обоснование выбора используемых в проекте технологий и библиотек. Для создания веб-приложения «Финансовый помощник» были выбраны технологии, обладающие необходимой функциональностью, производительностью и удобством использования. Особое внимание уделено выбору языка программирования, фреймворка для серверной части приложения, а также библиотек для разработки клиентской части. Кроме того, в разделе приведены аргументы и преимущества выбранных технологий и библиотек.

### Фреймворк Express

Express [7] – это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений. Он обладает простым интерфейсом для обработки HTTP-запросов [8] и маршрутизации, что упрощает создание эффективных обработчиков для различных маршрутов.

Благодаря своей производительности и небольшому размеру, Express идеально подходит для создания быстрых и масштабируемых веб-приложений. Он предоставляет необходимый минимум функциональности для работы с HTTP-запросами без излишних ограничений.

Большое сообщество разработчиков активно поддерживает и развивает Express, что обеспечивает надежность и стабильность фреймворка, а также облегчает процесс изучения и использования его для разработки веб-приложений.

Таким образом, для разработки веб-приложения «Финансовый помощник» был выбран фреймворк Express, учитывая его простоту, производительность и широкую поддержку сообщества.

### Библиотека React

React [9] – это библиотека JavaScript с открытым кодом для создания внешних пользовательских интерфейсов. В отличие от других библиотек JavaScript, предоставляющих полноценную платформу приложений, React ориентируется исключительно на создание представлений приложений через инкапсулированные единицы (называются компонентами), которые сохраняют состояние и генерируют элементы пользовательского интерфейса.

Одним из ключевых преимуществ React является виртуальный DOM (Document Object Model)  [10], который позволяет эффективно обновлять только те части пользовательского интерфейса, которые изменились, без перерисовки всего дерева DOM. Это улучшает производительность приложений и обеспечивает более плавное пользовательское взаимодействие.

React также обладает хорошей документацией и множеством обучающих материалов, что делает его доступным для новичков и облегчает процесс изучения и использования библиотеки для разработки веб-приложений.

Таким образом, для разработки веб-приложения «Финансовый помощник» была выбрана библиотека React, благодаря ее производительности, удобству работы с виртуальным DOM и обширной документации.

## Постановка задачи

На основе проведенного анализа предметной области, существующих решений и средств разработки можно сформулировать требования к дипломному проекту. Главной целью дипломного проекта является разработка веб-приложения «Финансовый помощник», которое обеспечивает удобное и эффективное администрирование данных о счетах, доходах, расходах и ограничениях на расходы.

Разрабатываемое веб-приложение должно обеспечивать:

* возможность регистрации и авторизации;
* создание счетов типов «депозитный» и «расчетный»;
* возможность добавлять расходы, доходы, ограничения на расходы, категории для доходов и расходов;
* возможность создания и изменения финансовых целей;
* обеспечение функции приглашения других пользователей для совместного администрирования счетов;
* получение уведомлений при превышении ограничений по расходам;
* получение стратегии расходов на следующий месяц;
* просмотр и экспорт статистики и отчетов о финансовом состоянии;
* возможность расчета инвестиций с помощью финансового калькулятора.

## Выводы по разделу

В данном разделе был проведен анализ предметной области и аналитический обзор литературы, включающий обзор аналогов. Целью этого было выявление существующих решений в сфере разработки веб-приложений для администрирования финансов. Результаты анализа позволили определить основные характеристики подобных приложений, а также выделить ключевые особенности, которые должны быть реализованы в проекте «Финансовый помощник».

Кроме того, был проведен обзор средств разработки, включая выбор языка программирования и платформы разработки. Были рассмотрены и выбраны технологии и библиотеки, необходимые для реализации функционала приложения.

В данном разделе также были определены требования для достижения цели дипломного проекта – разработки веб-приложения «Финансовый помощник», предназначенного для удобного и эффективного администрирования данных о счетах, доходах, расходах и ограничениях на расходы. А именно, был сформулирован набор функций, необходимых для его реализации.

Таким образом, данный раздел является ключевым для последующего проектирования веб-приложения «Финансовый помощник».

# Проектирование веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 02.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

2 Проектирование

веб-приложения

Лит.

Листов

10

74417023, 2024

Проектирование веб-приложения включает создание структуры, функционала и внешнего вида для обеспечения эффективной работы и удобства использования. Оно включает анализ требований, выбор технологий, разработку архитектуры и интерфейса, а также тестирование и оптимизацию.

Архитектура веб-приложения должна быть масштабируемой, чтобы эффективно реагировать на рост числа пользователей и объема данных без потери производительности. Она должна обеспечивать высокую надежность и безопасность, включая защиту данных пользователей и предотвращение утечек и внешних атак.

Гибкость и расширяемость важны для легкого внедрения новых функций и изменений без нарушения работы приложения, позволяя ему эволюционировать с изменяющимися потребностями пользователей и рынка. Понятность и легкость поддержки архитектуры также важны, так как простота архитектуры облегчает ее поддержку и развитие, снижая затраты на разработку и обеспечивая долгосрочную жизнеспособность приложения.

В этой главе будут спроектированы архитектура приложения, диаграмма развертывания, диаграмма вариантов использования и структура базы данных.

## Архитектура приложения

Для разработки веб-приложения «Финансовый помощник» была выбрана архитектура клиент-сервер. Это решение обусловлено необходимостью разделения функций обработки данных и представления пользовательского интерфейса. В данной архитектуре сервер отвечает за обработку бизнес-логики, управление базами данных и предоставление API, а клиентская часть – за взаимодействие с пользователем и отображение данных. Архитектура клиент-сервер предполагает наличие двух основных компонентов:

1. Серверная часть – это центральный компонент системы, который будет отвечать за обработку запросов от клиентов, управление данными и выполнение бизнес-логики. Для реализации серверной части будет использоваться платформа Node.js с фреймворком Express, что обеспечит высокую производительность и масштабируемость приложения. Сервер будет взаимодействовать с базой данных, предоставляя RESTful API для клиентских запросов;
2. Клиентская часть – это фронтенд-приложение, которое будет работать в браузере пользователя. Для его разработки будет использована библиотека React, обеспечивающая создание динамических и интерактивных пользовательских интерфейсов. Клиентская часть будет отправлять запросы к серверу, получать данные и отображать их пользователю.

Для визуализации процесса развертывания веб-приложения, представления архитектуры системы и лучшего понимания, как компоненты системы будут размещены и взаимодействовать друг с другом в реальной среде эксплуатации, была разработана диаграмма развертывания. Она представлена на рисунке 2.1 и в приложении А.

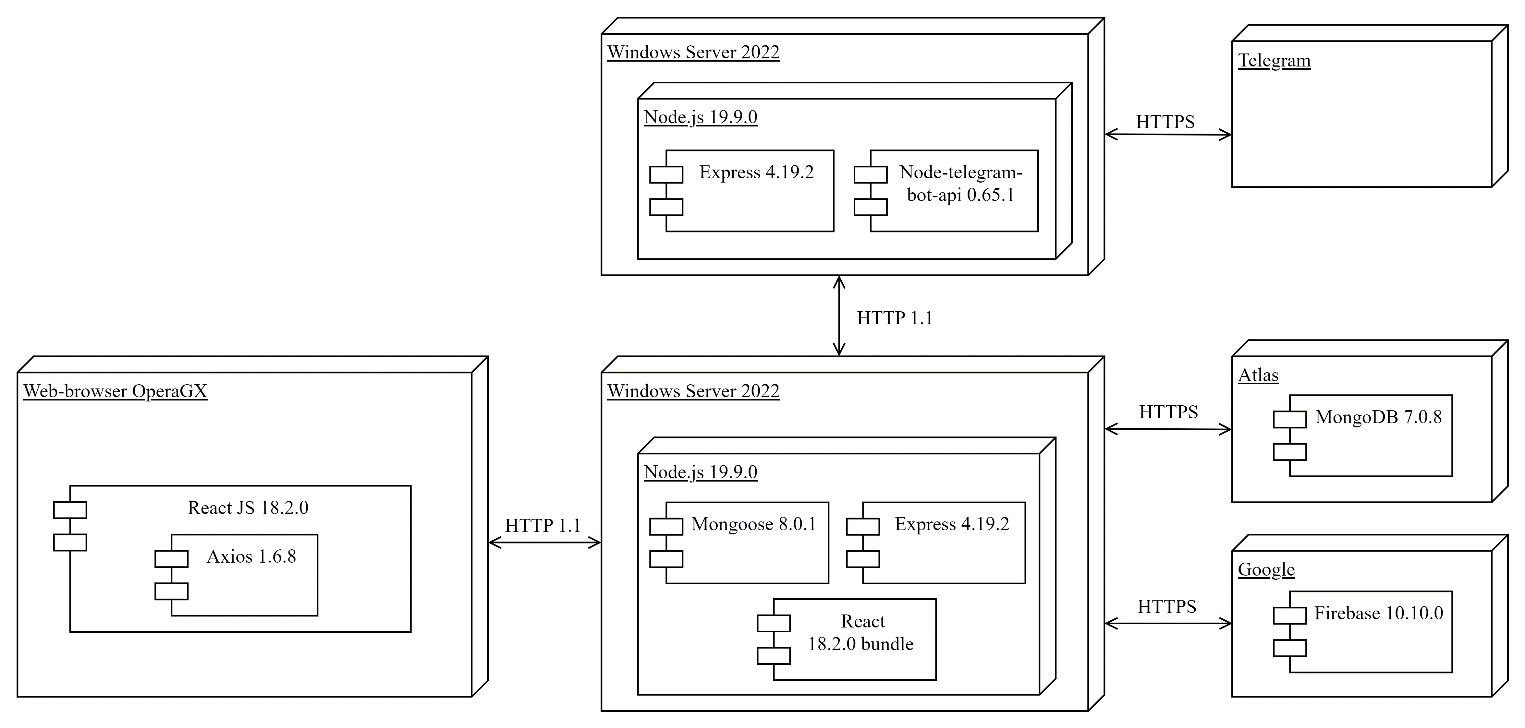


Рисунок 2.1 – Диаграмма развертывания

На диаграмме развертывания представлено шесть основных компонентов: веб-клиент, два сервера, облачные платформы Google [11] и Atlas, а также сервер Telegram.

Веб-клиент будет использовать OperaGX для работы с веб-приложением на React и Axios для обмена данными с сервером через HTTP 1.1.

Первый сервер, работающий на Windows Server 2022 с использованием Node.js и Express, будет обрабатывать запросы и взаимодействовать с MongoDB через Mongoose. Он также будет предоставлять бандл React для рендеринга интерфейса веб-клиента и обслуживать запросы, управлять данными и взаимодействовать с другими частями системы.

Этот сервер будет интегрирован с облачными платформами, такими как Atlas для хранения данных MongoDB [12] и Google Firebase [13] для регистрации и аутентификации пользователей, обеспечивая масштабируемое хранение данных и дополнительные функции.

Второй сервер, также на Windows Server 2022 с использованием Node.js и Express, будет обрабатывать HTTP-запросы и взаимодействовать с API Telegram через node-telegram-bot-api. Он будет обмениваться данными с первым сервером по HTTP 1.1, обеспечивая интеграцию с ботом Telegram и обмен данными между пользователями и системой. Кроме того, он будет взаимодействовать с сервером Telegram по HTTPS [14] для безопасного обмена данными.

Таким образом, все компоненты, представленные на диаграмме развертывания, будут функционировать независимо, что облегчит их обслуживание, обновление и масштабирование. Эта архитектура позволит эффективно разделить функции и обеспечить надежное взаимодействие между всеми компонентами системы, создавая гибкую и масштабируемую инфраструктуру для приложения.

## Проектирование диаграммы вариантов использования

Диаграмма вариантов использования [15] представляет собой графическое изображение функциональности системы, позволяя иллюстрировать взаимодействие между актерами и различными вариантами использования, или сценариями, которые могут возникать в процессе работы приложения.

В результате проектирования была разработана диаграмма вариантов использования, представляющая функционал разрабатываемого программного средства и наглядно иллюстрирующая взаимодействие между актерами и различными сценариями работы приложения.

Для данного приложения предусмотрены одна роль: пользователь.

Зарегистрировавшись и войдя в аккаунт, пользователь может создавать счета различных типов, добавлять и удалять информацию о доходах, расходах, ограничениях на расходы с получением уведомлений и их превышении, а также о финансовых целях. Также, он может пригласить другого аутентифицированного пользователя для совместного управления счетом. Кроме того, пользователь может просмотреть статистику и отчеты по конкретному счету, а также скачать их. Более того, пользователь может создать стратегию расходов на следующий месяц и рассчитать план инвестиций с помощью финансового калькулятора. Вдобавок, пользователь может воспользоваться телеграм-ботом для быстрого доступа к своим данным.

Спроектированная диаграмма вариантов использования в сокращенном виде приведена на рисунке 2.2, а полная – в приложении Б.

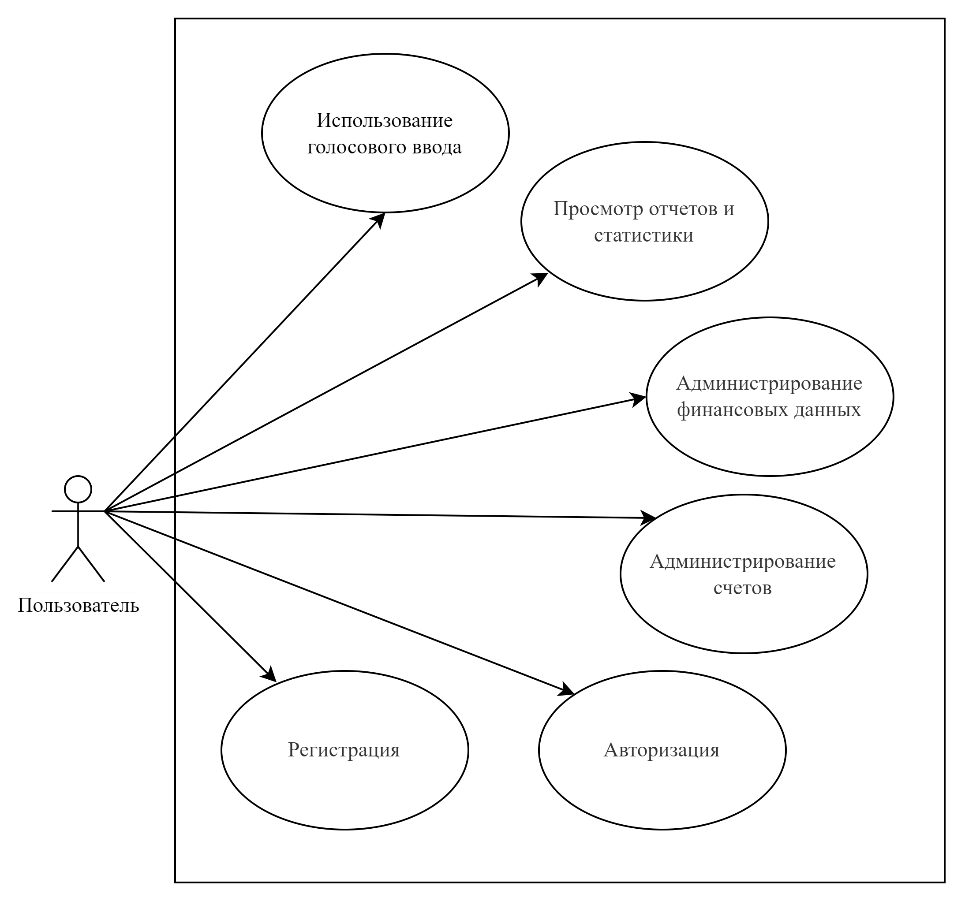


Рисунок 2.2 – Диаграмма вариантов использования в сокращенном виде

Разработанная диаграмма вариантов использования позволит более четко представить потребности и ожидания пользователей, а также определить основные сценарии использования приложения, что в свою очередь способствует более точному определению функциональных требований и эффективной реализации проекта. Разработка веб-приложения будет происходить на основе тщательно продуманных сценариев использования, выявленных и представленных в диаграмме вариантов использования.

## Проектирование структуры базы данных

Для хранения данных в данном приложении будет использоваться облачная NoSQL [16] база данных MongoDB Atlas, обеспечивающая надежность и масштабируемость хранения информации.

В ходе проектирования базы данных было разработано семь таблиц. Данные таблицы включают: Accounts (счета), AccountsOwners (Счета владельцев), AccountsInvited (Счета приглашенных), Operations (операции), Categories (Категории), Goals (Цели), Limitations (Ограничения). В данном разделе будет рассмотрена каждая из этих таблиц для лучшего понимания структуры базы данных и взаимосвязей между ними.

Таблица «Accounts» является хранилищем данных о счетах. Рассмотрим ее структуру для лучшего понимания информации, содержащейся в таблице 2.1.

Таблица 2.1 – Структура таблицы «Accounts»

|  |  |
| --- | --- |
| Название | Описание |
| \_id | Уникальный идентификатор счета |
| name | Поле типа String, содержащее информацию о названии счета |
| type | Поле типа String, содержащее информацию о типе счета |
| amount | Поле типа Number, содержащее информацию о сумме счета |
| currency | Поле типа String, содержащее информацию о валюте счета |

Таким образом, таблица «Accounts» обеспечивает централизованное хранение и управление данными о счетах. Она является основной и центральной в системе, поскольку от нее зависят остальные таблицы.

Таблица «AccountsOwners» играет роль связующего звена между счетами и их владельцами. Рассмотрим подробнее структуру этой таблицы в таблице 2.2.

Таблица 2.2 – Структура таблицы «AccountsOwners»

|  |  |
| --- | --- |
| Название | Описание |
| \_id | Уникальный идентификатор записи о владельце счета |
| uidOwner | Поле типа String, содержащее информацию о владельце счета |
| accNum | Поле типа ObjectId, ссылающееся на таблицу «Accounts» и содержащее информацию о номере счета |

Таким образом, таблица «AccountsOwners» служит для хранения о информации о всех созданных счетах и их владельцах. Как только создается новый счет, сразу же создается новая запись в таблице «AccountsOwners».

Таблица «AccountsInvited» играет роль связующего звена между счетами и пользователями, которых пригласили управлять счетом. Рассмотрим подробнее структуру этой таблицы в таблице 2.3.

Таблица 2.3 – Структура таблицы «AccountsInvited»

|  |  |
| --- | --- |
| Название | Описание |
| \_id | Уникальный идентификатор записи о приглашенном пользователе |
| uidInvited | Поле типа String, содержащее информацию о приглашенном пользователе |
| accNum | Поле типа ObjectId, ссылающееся на таблицу «Accounts» и содержащее информацию о номере счета |

Таким образом, таблица «AccountsInvited» служит для хранения информации о счете и о пользователе, которого пригласили для совместного управления счетом.

Таблица «Operations» играет важную роль в организации информации о расходах и доходах по конкретному счету. Рассмотрим подробнее структуру этой таблицы в таблице 2.4.

Таблица 2.4 – Структура таблицы «Operations»

|  |  |
| --- | --- |
| Название | Описание |
| \_id | Уникальный идентификатор операции |
| accNum | Поле типа ObjectId, ссылающееся на таблицу «Accounts» и содержащее информацию о номере счета |
| title | Поле типа String, содержащее информацию о названии операции |
| amount | Поле типа Number, содержащее информацию о сумме операции |
| date | Поле типа Date, содержащее информацию дате операции |
| category | Поле типа String, содержащее информацию о категории операции |
| type | Поле типа String, содержащее информацию об типе операции |
| email | Поле типа String, содержащее информацию о почте пользователя, который добавил операцию |

Таблица «Categories» служит для организации информации о категориях по доходам и расходам по конкретному счету. Рассмотрим подробнее структуру в таблице 2.5.

Таблица 2.5 – Структура таблицы «Categories»

|  |  |
| --- | --- |
| Название | Описание |
| \_id | Уникальный идентификатор категории |

Продолжение таблицы 2.5

|  |  |
| --- | --- |
| Название | Описание |
| accNum | Поле типа ObjectId, ссылающееся на таблицу «Accounts» и содержащее информацию о номере счета |
| title | Поле типа String, содержащее информацию о названии категории |
| type | Поле типа Number, содержащее информацию о типе категории |

Перейдем к следующей таблице. Таблица «Goals» содержит данные о финансовых целях для счетов с типом «депозитный». Поля перечислены в таблице 2.6.

Таблица 2.6 – Структура таблицы «Goals»

|  |  |
| --- | --- |
| Название | Описание |
| \_id | Уникальный идентификатор финансовой цели |
| accNum | Поле типа ObjectId, ссылающееся на таблицу «Accounts» и содержащее информацию о номере счета |
| goalAmount | Поле типа Number, содержащее информацию о числовом значении цели |

Таблица «Limitations» содержит данные об ограничениях по расходам, рассмотрим подробнее из каких полей она состоит. Список полей представлен в таблице 2.7.

Таблица 2.7 – Структура таблицы «Limitations»

|  |  |
| --- | --- |
| Название | Описание |
| \_id | Уникальный идентификатор ограничения |
| accNum | Поле типа ObjectId, ссылающееся на таблицу «Accounts» и содержащее информацию о номере счета |
| amount | Поле типа Number, содержащее информацию сумме ограничения |
| category | Поле типа String, содержащее информацию о категории расхода |

Таблица «Limitations» позволяет добавить ограничение на расходы по конкретной категории для конкретного счета.

Таким образом, были спроектированы семь таблиц: Accounts, AccountsOwners, AccountsInvited, Operations, Categories, Goals и Limitations, каждая из которых играет важную роль в организации данных для веб-приложения. Выбранная в качестве базы данных облачная MongoDB Atlas при разработке приложения поможет обеспечить высокую доступность и масштабируемость системы. Благодаря облачному хранилищу, будет получена отказоустойчивость и возможность мгновенного масштабирования ресурсов в зависимости от текущей нагрузки. Это обеспечит стабильную и эффективную работу приложения даже в условиях интенсивного использования и роста числа пользователей.

Результатом проведенного анализа и проектирования, является созданная логическая схема базы данных. Она представлена на рисунке 2.3 и в приложении В.

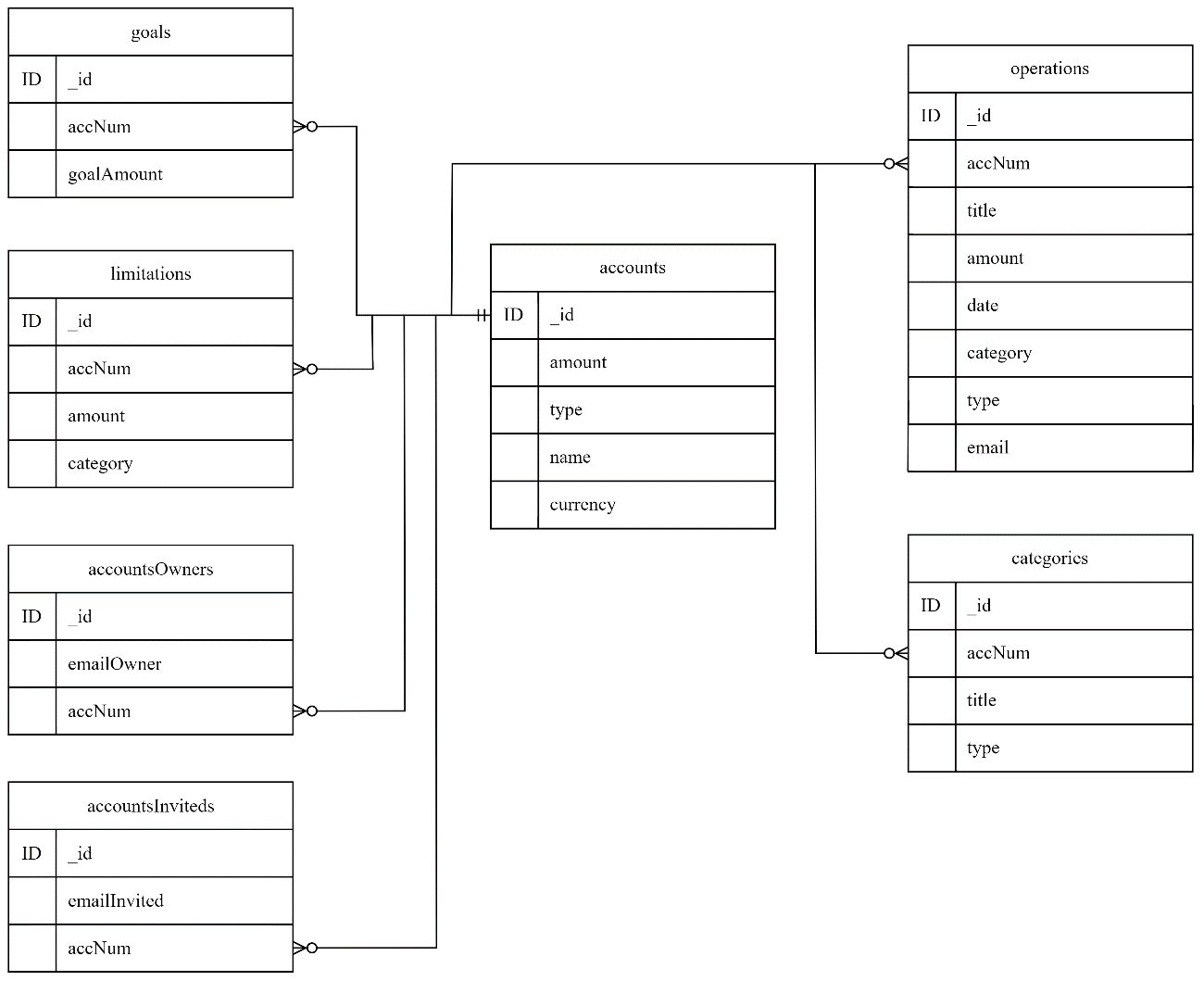


Рисунок 2.3 – Логическая схема базы данных

На приведенной выше логической схеме базы данных видно, что все таблицы ссылаются на поле \_id в таблице accounts. Это ключевая таблица, которая описывает все созданные в веб-приложении счета пользователей. От нее по полю \_id зависят доходы, расходы, категории, ограничения, финансовые цели, а также приглашенные пользователи и владельцы счетов. В таблице operations хранятся данные о финансовых операциях, таких как доходы и расходы, привязанных к определенным счетам. Таблица categories позволяет классифицировать эти операции, предоставляя дополнительную информацию о типе операций.

Таблица limitations хранит данные об ограничениях, связанных с расходами на определенные категории, что позволяет пользователям контролировать свои траты. Таблица goals хранит финансовые цели, которые пользователи устанавливают для своих счетов, помогая им отслеживать прогресс в достижении этих целей.

Связь между таблицами accountsOwners и accounts позволяет определить владельцев счетов, что важно для управления доступом и обеспечения безопасности данных. Таблица accountsInviteds хранит информацию о приглашенных пользователях, которые могут иметь ограниченные права доступа к счетам, что обеспечивает гибкость в управлении совместным использованием финансовой информации.

Разработанная логическая схема базы данных будет служить основой для создания физической структуры базы данных. Она описывает структуру данных, их взаимосвязи и ограничения целостности, что обеспечит эффективное хранение и управление информацией в приложении. Данная схема предоставляет понятное представление о структуре данных, что упрощает работу разработчиков и администраторов баз данных, а также способствует более гибкому и масштабируемому развитию приложения в будущем.

Эта схема демонстрирует, как различные аспекты финансового управления связаны между собой через централизованную систему учета, что позволяет обеспечить согласованность и целостность данных. Использование такой схемы позволяет строить сложные запросы для анализа данных и генерации отчетов, что значительно повышает функциональность и ценность приложения для конечных пользователей.

## Блок-схема алгоритма приглашения пользователя для совместного администрирования счетов

В данном подразделе представлена структура блок-схемы алгоритма приглашения пользователя для совместного управления счетом. На блок-схеме будут описаны важные элементы и последовательность действий, необходимых для эффективного функционирования алгоритма приглашения пользователей.

Разработанная блок-схема представлена на рисунке 2.4 и в приложении Г.

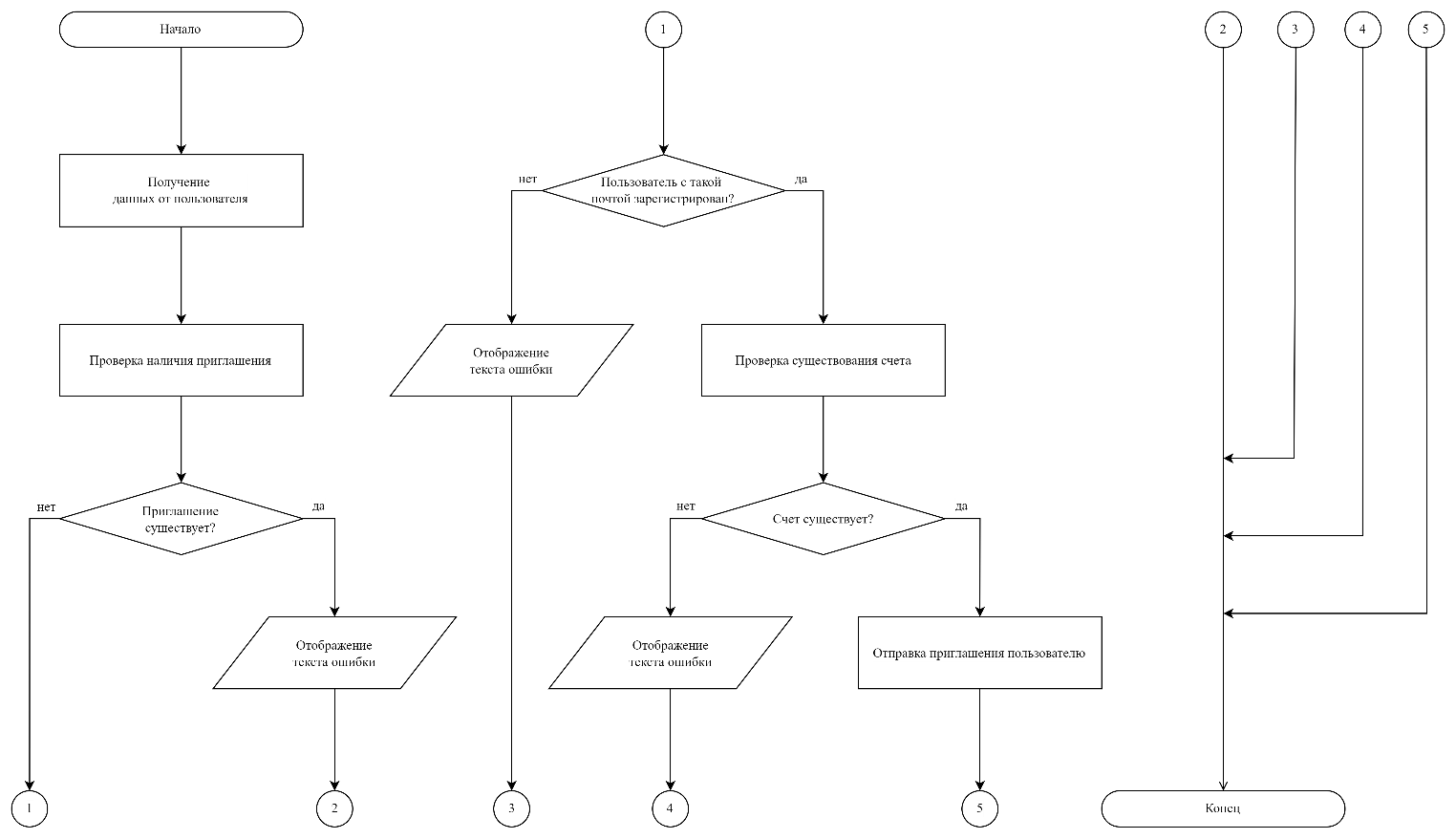


Рисунок 2.4 – Блок-схема алгоритма приглашения пользователя для совместного управления счетом

В начале происходит получение почты от пользователя, который хочет отправить пригласить другого пользователя управлять счетом. Далее, происходит попытка найти данную почту в базе данных. Если пользователь не найден, а значит не зарегистрирован, то выводится соответствующая ошибка. Если же пользователь найден, далее происходит поиск уже существующего приглашения. Если оно найдено, то выводится соответствующая ошибка о том, что данный пользователь уже был приглашен управлять счетом. Если же нет, то приглашение управлять счетом отправляется, и пользователь видит соответствующее сообщение.

Таким образом, спроектированная блок-схема поможет разработать функцию для приглашения пользователя для совместного управления счетом.

## Блок-схема алгоритма стратегии расходов

В данном подразделе представлена структура блок-схемы алгоритма стратегии расходов. Здесь будут рассмотрены основные компоненты и последовательность операций, необходимых для точного прогнозирования расходов.

Разработанная блок-схема представлена на рисунке 2.5 и в приложении Д.

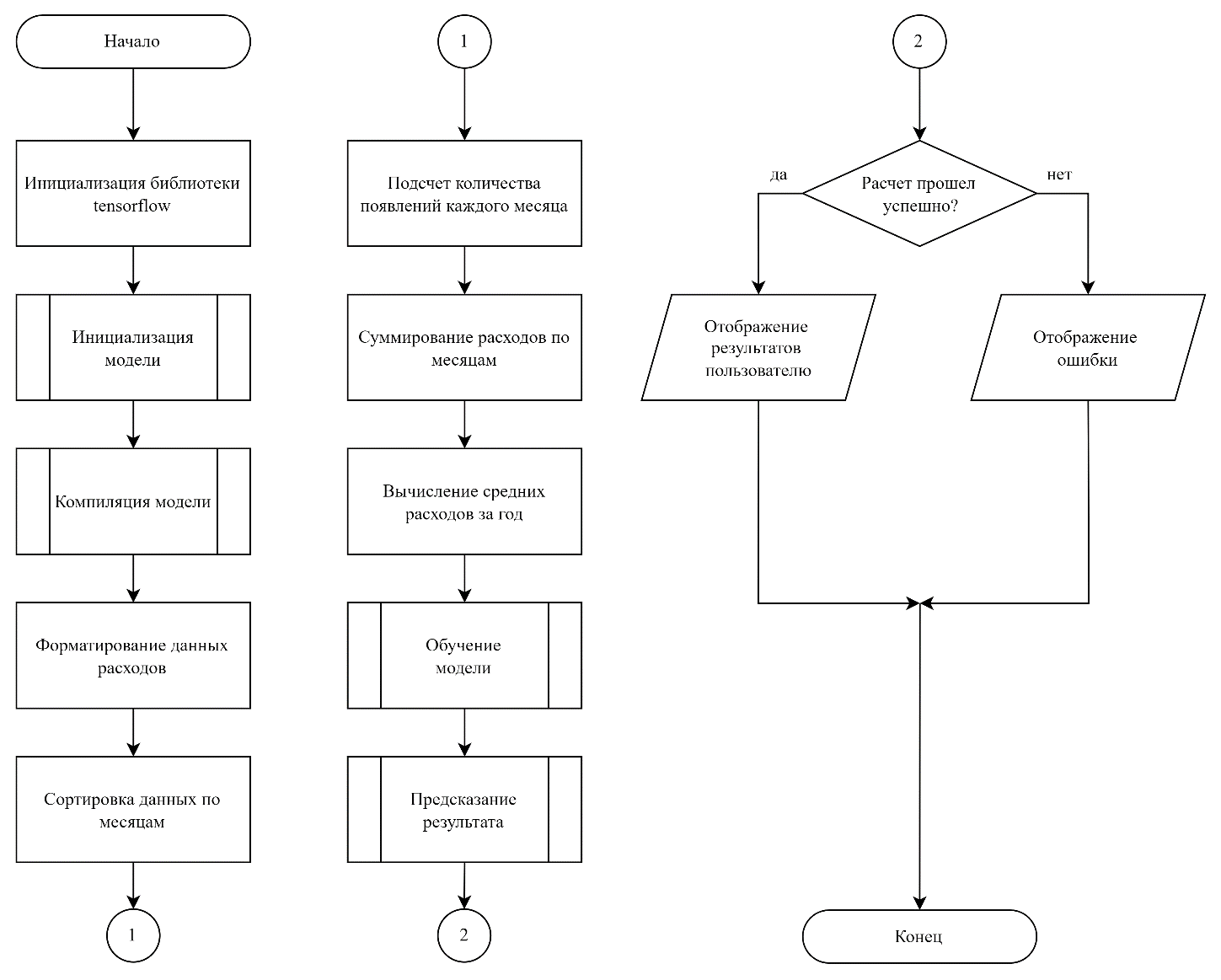


Рисунок 2.5 – Блок-схема алгоритма стратегии расходов

В начале инициализируется библиотека для машинного обучения tensorflow [17]. С помощью функции данной библиотеки инициализируется модель для последующего создания стратегии расходов. Затем она компилируется. После этого полученные от пользователя данные расходов форматируются. Они сортируются по месяцам. Следующим шагом происходит подсчет количества появлений каждого месяца. Затем расходы суммируются по каждому месяцу. Вычисляются средние расходы за год по каждому месяцу. Затем модель обучается на этих данных и вызывается предсказание результата. Если расчет прошел успешно, то пользователь получает стратегию расходов на следующий месяц, если нет – выводится соответствующая ошибка.

Таким образом, спроектированная блок-схема позволяет разработать более эффективные стратегии финансового планирования и управления ресурсами, а также разработать функцию, которая поможет этого добиться.

## Выводы по разделу

В данном разделе были представлены ключевые моменты проектирования веб-приложения «Финансовый помощник». В начале была определена архитектура приложения, которая базируется на клиент-серверной модели. Для визуализации этой архитектуры и определения взаимодействия компонентов была разработана диаграмма развертывания. На ней представлено шесть основных компонентов: веб-клиент, два сервера, облачные платформы Google и Atlas, а также сервер Telegram.

Следующим этапом была разработка диаграммы вариантов использования, которая детально описывает сценарии взаимодействия пользователей с веб-приложением. На ней была детально описана единственная существующая роль-пользователя, что упрощает понимание функционала системы и его использование.

Важным этапом проектирования была разработка логической схемы базы данных. В результате было создано семь таблиц для хранения данных о сущностях приложения. Каждая таблица была тщательно проработана с указанием всех полей и их назначения, обеспечивая эффективное хранение и доступ к информации.

Кроме того, были разработаны две блок-схемы для последующей реализации функций приложения. Первая блок-схема описывает алгоритм приглашения пользователей для совместного администрирования счетов, а вторая - алгоритм стратегии расходов. Эти блок-схемы представляют собой план действий для разработки и внедрения функций в приложение.

Спроектированные компоненты и структура базы данных создают основу для перехода к реализации приложения. Они позволяют создать приложение, которое соответствует требованиям пользователей, обладает эффективной архитектурой и готово к дальнейшему развитию и масштабированию.

# Разработка веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

3 Разработка

веб-приложения

Лит.

Листов

16

74417023, 2024

Для реализации цели дипломного проекта был применен архитектурный стиль REST API [18]. Он предполагает разделение приложения на две основные части: backend, отвечающий за серверную часть, и frontend, обеспечивающий клиентское взаимодействие. Для реализации каждой из этих частей были применены соответствующие технологии, включая разнообразные библиотеки и инструменты, необходимые для обеспечения надежного и полноценного функционирования проекта.

## Разработка серверной части

Серверная часть реализована с использованием платформы Node.js, фреймворка Express.js и языка Typescript [19] и представляет собой API, через который клиентское приложение может взаимодействовать с сервером, отправляя запросы и получая необходимые данные в ответ. Структура серверной части веб-приложения представлена на рисунке 3.1.

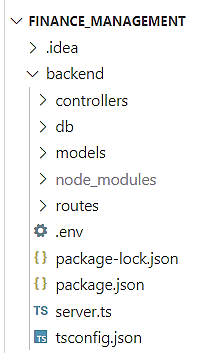


Рисунок 3.1 – Структура серверной части

Файл server.ts играет ключевую роль в запуске приложения, обрабатывая запросы, поступающие на определенный порт. В этом файле веб-сервер загружает необходимые модули для своей работы, проводит начальную настройку приложения и добавляет маршруты, которые будут обрабатывать входящие запросы.

Файл package.json – основной файл конфигурации для проектов на Node.js. Он содержит метаданные о проекте: название, версия, авторы, лицензия и прочее. Главная функция – управление зависимостями и скриптами с помощью npm.

Файл tsconfig.json – основной конфигурационный файл для TypeScript-проектов. Он определяет настройки компилятора TypeScript и трансляции кода в JavaScript.

Файл .env содержит конфигурационные переменные: порт сервера, строки подключения к базе данных и секретный ключ для токенов. Это упрощает настройку приложения для разных сред без изменения кода.

Папка node\_module хранит все зависимости проекта, установленные через npm: модули, библиотеки и фреймворки, необходимые для работы проекта.

Папка controllers содержит файлы с логикой управления запросами и взаимодействия с моделью данных и представлением. Контроллеры обрабатывают HTTP-запросы, извлекают данные из базы данных и передают их для отображения.

Папка db содержит конфигурационные файлы для управления соединением с MongoDB Atlas. Здесь находятся параметры подключения к базе данных, такие как URL и настройки соединения.

Папка modules содержит модели для базы данных MongoDB. Модели определяют структуру данных, выполняют запросы, добавляют, обновляют и удаляют записи. Каждая модель соответствует коллекции в базе данных.

Папка routes содержит файл маршрутов, который импортирует функции контроллеров для обработки запросов к базе данных. Каждый маршрут определяет тип запроса и путь для обработки клиентских запросов. Файл экспортирует объект router, используемый в server.ts для настройки HTTP-запросов.

### Файл package.json

Разработанный файл package.json содержит метаданные о проекте, такие как название, версия, описание, автор и лицензия. Основные скрипты для запуска и сборки проекта указаны в разделе «scripts». Зависимости проекта включены как «dependencies», включая библиотеки и фреймворки, необходимые для работы приложения, а также инструменты разработки, такие как тестовые фреймворки и сборщики, в разделе «devDependencies». Фрагмент файла package.json представлен в листинге 3.1, а весь код представлен в приложении И.

{

'name': 'backend',

'version': '1.0.0',

'description': '',

'main': 'server.ts',

'scripts': {

'start': 'nodemon server.ts',

'build': 'tsc -p',

'dev': 'nodemon server.ts'

},

"keywords": [],

"author": "Ksu",

Листинг 3.1 – Фрагмент файла package.json

Таким образом, package.json важен для разработки веб-приложений на Node.js. Он содержит основные данные о проекте, зависимости и скрипты для автоматизации задач, облегчает управление зависимостями, помогает в совместной работе над проектами, а также поддерживает единообразие и переносимость проекта.

### Взаимодействие с базой данных

Для удобного управления данными веб-приложения используется Mongoose, инструмент, который обеспечивает Object-Document Mapping для MongoDB.

Пример подключения к базе данных MongoDB Atlas приведен в листинге 3.2.

import mongoose from 'mongoose';

import { config } from './config';

export async function connectToDatabase()

{

try {

await mongoose.connect(config.mongo.url, {

retryWrites: true,

w: 'majority',

dbName: 'finance\_management',

});

console.log('Успешное подключение к бд!');

} catch (error) {

console.error('Ошибка при подключении к бд:', error);

}

}

Листинг 3.2 – Подключение базы данных

В данном листинге импортируется модуль для работы с MongoDB и настройки конфигурации, затем происходит попытка установить соединение с базой данных с указанными параметрами, включая имя базы данных и параметры записи. В случае успеха выводится сообщение об успешном подключении, а в случае ошибки – сообщение с описанием ошибки.

### Модели данных

Для работы с данными были разработаны модели, которые определяют структуру и типы данных для базы данных MongoDB. Эти модели играют ключевую роль в управлении информацией в приложении, обеспечивая ее структурирование и согласованность. В таблице 3.1 приведен перечень моделей данных разработанного приложения.

Таблица 3.1 – Описание моделей данных приложения

|  |  |
| --- | --- |
| Название модели | Описание модели |
| accounts | Включает информацию о счете, такую как название, баланс, тип счета и валюта счета. |
| accountsInviteds | Содержит данные о приглашенных к счетам пользователях. |
| accountsOwners | Содержит данные о владельцах счетов. |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Название модели | Описание модели |
| categories | Содержит данные о категориях для расходов и доходов. |
| goals | Содержит информацию о финансовых целях. |
| limitations | Содержит информацию об ограничениях на расходы. |
| operations | Содержит данные о расходах и доходах. |

Каждая модель данных взаимосвязана с конкретной коллекцией в базе данных MongoDB, отражая структуру и поля этой коллекции. Каждое свойство модели соответствует полю сущности в базе данных, определяя их характеристики и типы данных. В листинге 3.3 приведена модель данных для счетов.

const AccountSchema = new mongoose.Schema(

{

name:

{

type: String,

required: true,

trim: true

},

amount: {

type: Number,

required: false,

trim: true

},

type: {

type: String,

default: 'Checking'

},

currency: {

type: String,

enum: ['USD', 'EUR', 'RUB', 'BYN'],

default: 'BYN',

required: true

}

}

)

module.exports = mongoose.model('Account', AccountSchema)

Листинг 3.3 – Модель данных для счетов

В представленном коде определена схема для модели «Account» в MongoDB с использованием библиотеки Mongoose. Эта схема определяет структуру данных для коллекции «accounts» в базе данных. Поля схемы определяют свойства сущности «Account».

Поле «name» представляет собой строку, содержащую имя счета. Оно обязательно для заполнения и проходит обрезку лишних пробелов.

Поле «amount» представляет собой числовое значение, указывающее сумму на счете. Оно необязательно для заполнения и также проходит обрезку пробелов.

Поле «type» определяет тип счета и является строкой. По умолчанию устанавливается значение «Checking».

Поле «currency» определяет валюту счета. Оно также является строкой и может принимать только определенные значения из перечисления [«USD», «EUR», «RUB», «BYN»]. Значение по умолчанию установлено на «BYN», и его заполнение обязательно.

Эти параметры в схеме помогают контролировать структуру данных и их корректность, обеспечивая надежное хранение информации о счетах в базе данных.

### Реализация маршрутизации

В данном приложении используется фреймворк Express.js, который обеспечивает возможность создания серверов и обработки маршрутов в веб-приложениях на платформе Node.js. Этот фреймворк предоставляет удобные инструменты для определения маршрутов, обработки различных типов запросов и использования middleware. Код, представленный в листинге 3.4, разработан для маршрутизации и обработки запросов в данном контексте.

const app = express();

const corsOptions = { origin: process.env.CLIENT\_ORIGIN };

app.use(express.json());

const routes = readdirSync('./routes');

routes.forEach((route: string) => {

const routePath = `./routes/${route}`;

const routeModule = require(routePath);

app.use('/api', routeModule);});

connectToDatabase()

.then(() => {

app.listen(process.env.PORT, () => {

console.log('Server listening at http://localhost:' + process.env.PORT);

}); }).catch((error: Error) => {console.log(error);});

Листинг 3.4 – Добавление роутеров

Листинг 3.6 содержит код файла server.ts. Он создает экземпляр приложения Express и использует express.json() для парсинга JSON. Далее загружает маршруты из папки ./routes и применяет их к приложению. Функция connectToDatabase() устанавливает соединение с базой данных. Роутер transactions.ts, главный роутер сервера, размещает все маршруты и их обработчики. Он управляет потоком запросов и определяет, какой код выполняется в ответ на каждый запрос. Листинг 3.5 показывает часть кода роутера transactions.ts, полный код приведен в приложении К.

router.get('/get-incomes/:accNum', getIncomesByAccount)

.post('/get-all-user-incomes', getAllIncomesByUser)

.get('/get-expenses/:accNum', getExpensesByAccount)

.post('/get-all-user-expenses', getAllExpensesByUser)

.delete('/delete-operation/:id', deleteOperation)

.post('/add-operation', addOperation)

Листинг 3.5 – Роутер transactions.ts

Маршруты в роутере определяются методами Express.js: router.get(), router.post(), router.put(), router.delete(). Каждый маршрут связан с функцией-обработчиком из контроллера, содержащей логику для операций, таких как добавление доходов и расходов, просмотр баланса и приглашение пользователей.

### Контроллеры

Контроллеры играют важную роль в веб-приложении, отвечая за обработку запросов и формирование соответствующих ответов. В данном контексте, контроллеры обрабатывают запросы с определенными URL и вызывают соответствующие обработчики функций. Таким образом, контроллеры являются основой маршрутизации запросов в приложении. Подробное описание контроллеров представлено в таблице 3.2.

Таблица 3.2 – Описание контроллеров приложения

|  |  |
| --- | --- |
| Название контроллера | Описание контроллера |
| accountController | Предназначен для работы со счетами |
| goalController | Предназначен для работы с целями |
| limitationAndCategoryController | Предназначен для работы с ограничениями и категориями |
| operationController | Предназначен для работы с финансовыми операциями |

В качестве примера в листинге 3.6 приведен код функции удаления счета в контроллере accountController.

exports.deleteAccount = async (req: Request, res: Response) => {

    const { id } = req.params;

    try {const deletedAccount = await AccountModel.findByIdAndDelete(id);

        if (!deletedAccount) {return res.status(404).json({ message: 'Счет не найден' });}

        await mongoose.model('Operation').deleteMany({ accNum: deletedAccount.\_id });

        await mongoose.model('Limitation').deleteMany({ accNum: deletedAccount.\_id });

        await mongoose.model('Goal').deleteMany({ accNum: deletedAccount.\_id });

        await mongoose.model('AccountsOwners').deleteMany({ accNum: deletedAccount.\_id });

        await mongoose.model('AccountsInvited').deleteMany({ accNum: deletedAccount.\_id });

        await mongoose.model('Category').deleteMany({ accNum: deletedAccount.\_id });

        res.status(200).json({ message: 'Счет удален', account: deletedAccount });

    } catch (error) {console.error('Ошибка при удалении счета:', error); res.status(500).json({ message: 'Ошибка сервера' });

}}

Листинг 3.6 – Контроллер accountController

Эта функция представляет собой обработчик для запроса на удаление аккаунта. Она принимает объект запроса и объект ответа в качестве параметров. Внутри функции сначала извлекается идентификатор аккаунта из параметров запроса. Затем выполняется попытка удаления аккаунта из базы данных с использованием метода findByIdAndDelete модели AccountModel. После удаления аккаунта из базы данных, функция обработчика выполняет дополнительные операции по удалению связанных с ним данных. Это включает в себя удаление операций, ограничений и других сущностей, связанных с данным аккаунтом. Для этого используются методы deleteMany соответствующих моделей mongoose. Такой подход обеспечивает целостность данных и их связей в базе данных, предотвращая возможные проблемы при удалении аккаунтов.

### Реализация приглашения пользователя для совместного администрирования счета

В данном веб-приложении есть возможность пригласить пользователя для совместного администрирования счета посредством отправки письма на почту. Для этого была разработана функция invite, представленная в листинге 3.7.

exports.invite = async (req: Request, res: Response) => {

const { email, accNum } = req.body;

try {

if (!email) {

return res.status(400).json({ message: 'Необходимо указать почту' });}

const ownerInvited = await AccountsOwnersModel.find({ owner: email, accNum })

if (ownerInvited.length > 0) {

return res.status(400).json({ message: 'Данный пользователь не может быть приглашен, так как является его владельцем' });

}

const userInvited = await AccountsInvitedModel.find({ invited: email, accNum })

if (userInvited.length > 0) {

return res.status(400).json({ message: 'Данный пользователь уже был приглашен' });}

const isAccountExist = await AccountsOwnersModel.find({ accNum });

if (!isAccountExist || isAccountExist.length === 0) {

return res.status(400).json({ message: 'Счет для привязки не найден' });

const inviteLink = `http://localhost:8080/api/accept-invitation?email=${email}&accNum=${accNum}`;

await sendInviteEmail(email, inviteLink);

res.status(200).json({ message: 'Пригласительная ссылка была отправлена' });

} catch (error) {

res.status(500).json({ message: error });

}}

Листинг 3.7 – Реализация функции invite()

Как видно из листинга, для отправки пригласительной ссылки на почту была использована функция sendInviteEmail(). Для ее реализации была использована библиотека Nodemailer. Код данной функции приведен в листинге 3.8.

async function sendInviteEmail(email: String, inviteLink: String) {

    const transporter = nodemailer.createTransport(

{

        service: 'yandex',

        host: 'smtp.yandex.ru',

        port: 465,

        auth: {

            user: 'k.budan0va@yandex.by',

            pass: 'mdyukcccmwqcxnso'

        },

        secure: true

    }

);

    const mailOptions = {

        from: 'k.budan0va@yandex.by',

        to: email,

        subject: 'Приглашение к совместному управлению счетом',

        html: `<p>

Вы были приглашены совместно управлять счетом. Перейдите по <a href='${inviteLink}'>ссылке</a> для принятия приглашения.

</p>`

    };

    await transporter.sendMail(mailOptions);

}

Листинг 3.8 – Реализация функции sendInviteEmail()

В данном коде реализована функция sendInviteEmail, которая отправляет приглашения пользователям по электронной почте для совместного управления счетами. Функция создает транспортное соединение с помощью Nodemailer, настроенного для работы с сервисом Yandex. Далее формируются параметры письма, включая получателя, тему и содержание, которое содержит ссылку для принятия приглашения.

Таким образом, использование Nodemailer представляет собой эффективный и гибкий подход для отправки электронных писем из приложений на Node.js. Использование этого модуля способствует повышению надежности и безопасности почтовой системы приложения, а также упрощает процесс управления и отслеживания отправленных сообщений.

### Реализация телеграм-бота

Данное веб-приложение предоставляет быстрый и удобный доступ к финансовым данным с помощью телеграм-бота, что позволяет использовать приложение через мобильное устройство. Часть кода разработанного телеграм-бота представлена в листинге 3.9, а полный код представлен в приложении Л.

    bot.on('message', async (msg) => {

const chatId = msg.chat.id;

const text = msg.text;

try {

if (userUid === '') {

if (text === '/start') {

bot.sendMessage(chatId, 'Добро пожаловать в финансовый помощник! Чтобы продолжить, введите команду /login для входа через почту и пароль.');

} else if (text === '/login') {

currentStep = 'email';

bot.sendMessage(chatId, 'Введите email');

Листинг 3.9 – Часть кода разработанного телеграм-бота

Кроме того, телеграм-бот позволяет обрабатывать голосовые сообщения для добавления новых операций. Часть кода представлена в листинге 3.10.

const client = new AssemblyAI(

{

apiKey: apiKey

})

axios.get(`https://api.telegram.org/bot${BOT\_KEY}/getFile?file\_id=${file\_id}`).then(async response =>

{

const filePath = response.data.result.file\_path;

fileName = filePath.split("/").slice(-1)[0].split(".")[0];

fileName = fileName + file\_id.slice(-5);

const fullUrl = `https://api.telegram.org/file/bot${BOT\_KEY}/${filePath}`;

const writer = fs.createWriteStream(`${fileName}.ogg`);

axios.get(fullUrl, { responseType: 'stream' })

.then(response => {

response.data.pipe(writer);

writer.on('finish', async () => {

const audioUrl = `${fileName}.ogg`

const config = {

audio\_url: audioUrl,

language\_code: 'ru'

}

const transcript = await client.transcripts.create(config)

// Удаляем все точки и запятые, кроме случаев, когда запятая используется в числе

const cleanedText = transcription.replace(/[\.,]/g, (match, offset, string) => {

if (match === ',') {

Листинг 3.10 – Реализация распознавания голосовых сообщений

Приведенный выше код создает экземпляр библиотеки AssemblyAI для распознавания голосовых сообщений, настраивает конфигурацию, а затем создает транскрипцию, в которой находится распознанный текст.

## Разработка клиентской части

Клиентская часть приложения представляет собой отдельный проект, построенный с использованием React. Благодаря React возможно создание сложных пользовательских интерфейсов из небольших и автономных компонентов. Эти компоненты позволяют разбить интерфейс на независимые и многократно используемые части, что упрощает процесс разработки и поддержки. Такой подход способствует гибкости и эффективности разработки клиентской части приложения.

На рисунке 3.2 представлена архитектура клиентской части.



Рисунок 3.2 – Архитектура клиентской части

Файл index.js – это точка входа в React-приложение. Он обычно находится в корне директории проекта и используется для рендеринга основного компонента приложения в DOM браузера.

Файл index.css – это файл стилей, который обычно используется для глобальных стилей всего приложения.

Файл app.js содержит основной компонент приложения. Он определяет структуру и логику приложения, включая маршрутизацию, обработку состояний и взаимодействие с сервером.

Файл app.css – это файл стилей, который связан непосредственно с компонентом App, основным компонентом приложения.

Файл firebase.js – это файл конфигурации, который используется для настройки подключения к Firebase, которая используется для регистрации и авторизации.

Папка assets содержит статические ресурсы, такие как изображения.

Папка components хранит компоненты React, которые могут быть многократно использованы на страницах. В этой папке компоненты разделены по функциональности, облегчая повторное использование и обслуживание кода.

Папка context содержит файлы, отвечающие за контексты React. Контексты используются для передачи данных через дерево компонентов без явной передачи пропсов через каждый компонент.

Папка navigation содержит файл, который отвечает за управление маршрутами в веб-приложении. Этот файл определяет основные маршруты и защищенные маршруты, а также содержит логику для отображения соответствующих компонентов в зависимости от выбранного маршрута.

Папка utils содержит вспомогательные функции и утилиты, которые многократно используются в различных частях веб-приложения, что улучшает повторное использование кода и обеспечивает его модульность.

Папка node\_modules представляет собой каталог, в котором хранятся все зависимости проекта, установленные через пакетный менеджер npm.

### Реализация аутентификации

Для реализации аутентификации в клиентской части веб-приложения была использована интеграция с Firebase. Сервер взаимодействует с Firebase по протоколу HTTPS, обеспечивая безопасное соединение для передачи данных. Каждый запрос на аутентификацию, отправленный с клиента, передается на сервер Firebase через защищенные каналы. Сервер Firebase обрабатывает запросы, такие как регистрация нового пользователя, вход в систему, выход из системы и аутентификация с помощью сторонних провайдеров, возвращая соответствующие ответы клиенту. Эти запросы включают данные, такие как адрес электронной почты и пароль пользователя, которые передаются в зашифрованном виде для обеспечения конфиденциальности.

В файле firebaseConfig.js содержится конфигурация Firebase, включая ключ API, домен аутентификации, идентификатор проекта и другие параметры, необходимые для инициализации приложения. Код данного файла представлен в листинге 3.11.

const firebaseConfig =

{

apiKey: "AIzaSyDvbtB4EnJuXLkweBELx4Ab1k5DFXK00cQ",

authDomain: "finance-management-b6823.firebaseapp.com",

projectId: "finance-management-b6823",

storageBucket: "finance-management-b6823.appspot.com",

messagingSenderId: "221792020390",

appId: "1:221792020390:web:4fce8cbe92f5c438b15167",

measurementId: "G-B426266BLH"

};

const app = initializeApp(firebaseConfig);

getAnalytics(app);

export const auth = getAuth(app);

Листинг 3.11 – Код файла firebaseConfig.js

Далее, в AuthContext.js определен контекст аутентификации пользователя с помощью createContext. Этот контекст создает провайдер, который предоставляет доступ к функциям аутентификации, таким как вход, регистрация, выход и еще одному способу аутентификации через Google. Также в этом контексте хранится информация о текущем пользователе и его состоянии.

В файле AuthContext.js также устанавливается прослушивание состояния аутентификации с помощью useEffect. Это позволяет автоматически обновлять информацию о пользователе при изменении его статуса, например, при входе или выходе из системы.

Часть кода файла UserAuthContext.js содержится в листинге 3.12, а полный код представлен в приложении М.

export function UserAuthContextProvider({children})

{

const [user, setUser] = useState({});

const [loading, setLoading] = useState(true);

function logIn(email, password) {

return signInWithEmailAndPassword(auth, email, password);

}

function signUp(email, password) {

return createUserWithEmailAndPassword(auth, email, password);

}

function logOut() {

setUser(null);

return signOut(auth);

}

function googleSignIn() {

const googleAuthProvider = new GoogleAuthProvider();

return signInWithPopup(auth, googleAuthProvider);

}

return (

<userAuthContext.Provider value={{user, logIn, signUp, logOut, googleSignIn}}

>

{children}

</userAuthContext.Provider>

);

Листинг 3.12 – Часть кода файла AuthContext.js

Эти файлы вместе обеспечивают возможность аутентификации пользователей на клиентской стороне приложения с использованием Firebase, что делает процесс аутентификации простым и удобным для разработчика и пользователей.

### Реализация глобального контекста

Для реализации глобального контекста в клиентской части веб-приложения был использован React Context API. Глобальный контекст предоставляет общий доступ к состоянию и функциям для всего приложения, что облегчает управление состоянием и обмен данными между компонентами. Контекст для клиентской части приложения реализован в файле globalContext.js. Часть кода данного файла представлена в листинге 3.13, а весь код данного файла представлен в приложении Н.

export function UserAuthContextProvider({children})

{

const deleteInvitation = async (data) => {

const res = await axios.delete(`${BASE\_URL}delete-invitation`, { data: data }).catch((err) => {

setError(err.response.data.message);

return err; });

if (res && res.status === 200) setSuccessMessage(res.data.message);

await getEmailsInvited(data);

}

const getOwnerAccounts = async (uid) => {

const res = await axios.get(`${BASE\_URL}get-owner-accounts/${uid}`).catch((err) => {

setError(err.response.data.message)

}

);

setOwnerAccounts(res.data);

}

return (

<GlobalContext.Provider value={{

addIncome, getIncomes, incomes, deleteIncome, expenses, totalIncomes, addExpense, getExpenses, deleteExpense, totalExpenses, totalBalance, transactionHistory, categories, addCategory, getCategories, deleteCategory, successMessage, setSuccessMessage, error, setError }}>

{children}

</GlobalContext.Provider>

)

Листинг 3.13 – Часть кода файла globalContext.js

В файле GlobalContext.js создается контекст с помощью React.createContext(), и этот контекст предоставляет доступ к различным состояниям и функциям, необходимым для работы приложения. Константа BASE\_URL определяет базовый URL для всех запросов к серверу.

Компонент GlobalProvider оборачивает все дочерние компоненты, предоставляя им доступ к глобальному контексту. Внутри этого компонента определены различные состояния, такие как incomes, expenses, categories, limitations, accountInfo, goal, error, emails, userAccounts, ownerAccounts, successMessage и другие. Эти состояния используются для хранения данных, полученных с сервера, и для управления состоянием приложения.

Контекст GlobalContext предоставляет все эти состояния и функции через провайдер, что позволяет компонентам приложения получать доступ к данным и функциям контекста через хук useContext.

Такой подход обеспечивает централизованное управление состоянием и логикой работы приложения, улучшает структуру кода и облегчает его сопровождение и масштабирование.

### Реализация маршрутизации

Маршрутизация на клиентской части была реализована с помощью библиотеки React Router, которая предоставляет возможность навигации в приложении на основе URL-адресов. Для этого в коде используется компонент BrowserRouter, который оборачивает все приложение, позволяя использовать маршрутизацию на основе HTML5 History API. Этот подход делает навигацию по приложению более интуитивной и естественной для пользователя.

Кроме того, использование React Router позволяет создавать защищенные маршруты, ограничивая доступ к определенным страницам в зависимости от статуса аутентификации пользователя. Для более детального понимания реализации данного механизма представлен фрагмент кода в листинге 3.14, а весь код представлен в приложении П.

return (<BrowserRouter>

<UserAuthContextProvider><Routes>

{/\* ALL USERS\*/}

<Route path='/home' element={

<ProtectedRoute>

<MainLayout>

<GlobalProvider>

<LeftBlock active={active} setActive={setActive}

activeAccount={activeAccount} setActiveAccount={setActiveAccount} />

</GlobalProvider>

<GlobalProvider activeAccount={activeAccount !== null ? activeAccount.\_id : undefined}>

<main>{displayData()}</main>

</GlobalProvider></MainLayout></ProtectedRoute>}/>

{/\* CHECK\*/}

<Route path='/login' element={

<ProtectedRouteAuthorized>

<Login />

</ProtectedRouteAuthorized>} />

<Route path='/register' element={

<ProtectedRouteAuthorized>

<Register />

</ProtectedRouteAuthorized>} />

<Route path='\*' element={<Navigate to='/home' replace />} />

</Routes>

</UserAuthContextProvider>

</BrowserRouter>

)

Листинг 3.14 – Часть кода файла navigation.js

Компонент UserAuthContextProvider предоставляет контекст аутентификации пользователей, который используется в различных частях приложения для определения доступа к защищенным маршрутам.

Внутри Routes определены различные маршруты. Маршруты позволяют указать, какой компонент отображать в зависимости от текущего URL-адреса.

Маршрут /home обрабатывает запросы к главной странице приложения. Этот маршрут защищен и доступен только аутентифицированным пользователям. При успешной аутентификации отображается компонент MainLayout, который включает в себя глобальный провайдер для управления данными, левую панель и основной контент приложения.

Маршруты /login и /register предназначены для страниц входа и регистрации соответственно. Они доступны только не аутентифицированным пользователям, что обеспечивается компонентами ProtectedRouteAuthorized, которые перенаправляют аутентифицированных пользователей на главную страницу.

Последний маршрут «\*» является маршрутом по умолчанию и обрабатывает все остальные пути, перенаправляя пользователя на главную страницу приложения.

Таким образом, маршрутизация на клиентской части обеспечивает удобное управление навигацией и контролем доступа к различным частям приложения в зависимости от статуса аутентификации пользователя.

### Отправка запросов к серверу

В разрабатываемом приложении для обеспечения корректного получения данных с сервера используется Axios API [20], который представляет собой JavaScript библиотеку, предназначенную для выполнения HTTP-запросов и обработки полученных ответов. Применение вызова Axios представлено в листинге 3.15.

const getIncomes = async (activeAccount) =>

{

        const response = await axios.get(`${BASE\_URL}get-incomes/${activeAccount}`)

        setIncomes(response.data)

    }

    const addIncome = async (income, uid) => {

        await axios.post(`${BASE\_URL}add-operation`, income).catch((err) => {

                setError(err.response.data.message)});

        await getIncomes(activeAccount)

        await getUserBalance(uid);

    }

    const deleteIncome = async (id) =>

{

        await axios.delete(`${BASE\_URL}delete-operation/${id}`);

        await getIncomes(activeAccount)

}

Листинг 3.15 – Пример использования Axios

Этот код представляет собой набор функций для работы с доходами в приложении. Он обрабатывает запрос по определенному маршруту и обновляет значение переменной, отвечающей за доходы. Функция getIncomes получает данные о доходах с сервера и устанавливает их в состояние. Функция addIncome добавляет новый доход на сервер, обновляет список доходов и баланс пользователя. Функция deleteIncome удаляет выбранный доход с сервера и обновляет список доходов.

### Реализация стратегии расходов на следующий месяц

Для реализации стратегии расходов на следующий месяц была выбрана библиотека tensorflow/tfjs. Она позволяет разрабатывать и обучать модели машинного обучения и нейронные сети прямо в браузере или на сервере с использованием JavaScript. Она предоставляет широкий набор инструментов для работы с данными, создания и обучения моделей, а также составления стратегий на основе этих моделей. Часть кода данного функционала представлена в листинге 3.16.

const sums = result.map(item => item[1]);

const averageSumsByYear = sums.map((num, index) => num / occuresNumber[index]);

//обучаем модель данных для месяцев и соответсвующих сумм

const xs = tf.tensor2d(months, [months.length, 1]);

const ys = tf.tensor2d(averageSumsByYear, [sums.length, 1]);

await model.fit(xs, ys, { epochs: 150 }); //количество обучений

const prediction = model.predict(tf.tensor2d([Number(month)], [1, 1])).dataSync();

setPredictedExpense(prediction[0].toFixed(2));

Листинг 3.16 – Часть кода функции стратегии расходов на следующий месяц

Этот код обучает модель машинного обучения с использованием tensorflow.js, чтобы составлять стратегии расходов по месяцам. Сначала он преобразует массивы месяцев и средних сумм расходов в двумерные тензоры. Затем модель обучается с использованием этих данных за 150 эпох. После обучения модель используется для создания стратегии расходов для заданного месяца, и результат округляется до двух знаков после запятой и сохраняется.

## Выводы по разделу

В данной главе описан процесс разработки серверной и клиентской частей приложения по архитектуре REST. Серверная часть включает проектирование структуры сервера с роутерами и контроллерами, взаимодействие с базой данных и настройку package.json для управления зависимостями. Реализован алгоритм приглашения пользователей для совместного администрирования счетов и телеграм-бот. Клиентская часть включает проектирование архитектуры, описание директорий и файлов, маршрутизацию для запросов к серверу, маршрутизацию на клиентской части и получение данных. Разработана функция для реализации алгоритма стратегии расходов на следующий месяц, а также компоненты интерфейса, стили и другие ресурсы для корректной работы клиента, уделено внимание удобству использования. Данный раздел дает полное представление о разработке веб-приложения с применением REST, что обеспечивает его надежность и масштабируемость.

# Тестирование веб-приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

4 Тестирование

веб-приложения

Лит.

Листов

6

74417023, 2024

Тестирование веб-приложений необходимо для обеспечения их качества и надежности. В этой главе будет описан процесс тестирования дипломного проекта при помощи ручного тестирования [21], которое позволяет проверить функциональность, интерфейс и общую работоспособность приложения, также будут приведены таблицы с описанием проведенных тестов.

## Тестирование страниц авторизации и регистрации

После того, как пользователь вошел в систему, он имеет доступ только к страницам авторизации и регистрации. Как на странице регистрации, так и на странице авторизации пользователь видит два поля: почту и пароль. Результаты тестирования этих страниц представлены в таблице 4.1.

Таблица 4.1 – Результаты тестирования страниц авторизации и регистрации

|  |  |  |
| --- | --- | --- |
| Шаги | Ожидаемый результат | Результат теста |
| Тестирование страницы авторизации | | |
| Попытка входа с корректными данными (почта и пароль) | Успешный вход в систему с переходом на главную страницу | Пройден |
| Попытка авторизации через Google Auth | Успешная авторизация пользователя через Google | Пройден |
| Попытка регистрации без заполненного пароля | Выводится сообщение об ошибке заполнения поля пароля | Пройден |
| Попытка регистрации без заполненной почты | Выводится сообщение об ошибке заполнения поля почты | Пройден |
| Попытка входа под несуществующим пользователем или с некорректным паролем | Выводится ошибка с сообщением об ошибке аутентификации | Пройден |
| Тестирование страницы регистрации | | |
| Попытка регистрации с корректно заполненными полями (почта и пароль) | Успешная регистрация пользователя с переходом на страницу авторизации | Пройден |
| Попытка регистрации без заполненного пароля | Выводится сообщение об ошибке заполнения поля пароля | Пройден |
| Попытка регистрации без заполненной почты | Выводится сообщение об ошибке заполнения поля почты | Пройден |

Исходя из результатов тестирования, обе страницы - и страница регистрации, и страница авторизации – успешно прошли все тесты. Это подтверждает корректную работу функционала входа и регистрации в систему. Также успешное прохождение тестов на странице авторизации через Google Auth свидетельствует о правильной интеграции этого механизма аутентификации. Все это гарантирует удобство и безопасность процесса взаимодействия пользователя с приложением.

## Тестирование процесса создания нового счета

После входа в приложение у пользователя есть возможность создания счетов. Результаты тестирования данного функционала представлены в таблице 4.2.

Таблица 4.2 – Результаты тестирования процесса создания нового счета

|  |  |  |
| --- | --- | --- |
| Шаги | Ожидаемый результат | Результат теста |
| Тестирование функционала создания нового счета | | |
| Ввод корректных данных (название, тип, сумма, валюта) | Новый счет успешно создается с указанными данными | Пройден |
| Тестирование функционала создания нового счета с ошибками | | |
| Попытка создания счета с слишком длинным названием | Выводится сообщение об ошибке о превышении допустимой длины названия | Пройден |
| Попытка ввода некорректной суммы | Выводится сообщение об ошибке о некорректной сумме | Пройден |
| Попытка создания расчетного счета с введением баланса | Выводится сообщение об ошибке, так как баланс не вводится для расчетного счета | Пройден |

Проведенное тестирование функционала создания нового счета показало, что при правильном вводе данных счет успешно создается, а также корректно обрабатываются различные ошибочные сценарии, такие как превышение допустимой длины названия, ввод некорректной суммы и попытка введения баланса для расчетного счета.

## Тестирование операции создания ограничений

После создания счета у пользователя есть возможность создавать ограничения на категории расходов. Однако повторяться ограничения по категориям не могут. Результаты тестирования данного функционала представлены в таблице 4.3.

Таблица 4.3 – Результаты тестирования процесса создания ограничений на категории расходов

|  |  |  |
| --- | --- | --- |
| Шаги | Ожидаемый результат | Результат теста |
| Тестирование функционала создания ограничения на категории расходов | | |
| Ввод корректных данных (сумма и выбор категории) | Новое ограничение успешно создается с указанными данными | Пройден |

Продолжение таблицы 4.3

|  |  |  |
| --- | --- | --- |
| Шаги | Ожидаемый результат | Результат теста |
| Тестирование функционала создания ограничения на категории расходов с ошибками | | |
| Попытка создания ограничения с некорректной суммой | Выводится сообщение об ошибке о некорректной сумме | Пройден |
| Попытка создания ограничения без выбора категории | Выводится сообщение об ошибке о необходимости выбрать категорию | Пройден |
| Попытка создания ограничения с уже существующей категорией | Выводится сообщение об ошибке о невозможности повторного ограничения по категории | Пройден |

Проведенное тестирование функционала создания ограничений на категории расходов показало, что при правильном вводе данных ограничение успешно создается, а также корректно обрабатываются различные ошибочные сценарии, такие как ввод некорректной суммы, необходимость выбора категории из списка и невозможность повторного ограничения по одной категории. Это гарантирует корректное функционирование данного функционала в приложении.

## Тестирование процесса добавления новой операции

После создания счета пользователь может добавлять финансовые операции. Результаты тестирования данного функционала представлены в таблице 4.4.

Таблица 4.4 – Результаты тестирования процесса добавления новой операции

|  |  |  |
| --- | --- | --- |
| Шаги | Ожидаемый результат | Результат теста |
| Тестирование функционала добавления новой операции | | |
| Попытка ввода корректных данных (название, сумма, тип операции, категория, дата) | Новая операция успешно добавляется с указанными данными | Пройден |
| Тестирование функционала добавления новой операции с ошибками | | |
| Попытка добавления операции с некорректной суммой | Выводится сообщение об ошибке о некорректной сумме | Пройден |
| Попытка добавления операции с некорректным названием | Выводится сообщение об ошибке о некорректном названии | Пройден |

Проведенное тестирование функционала добавления новой финансовой операции показало, что при правильном вводе данных операция успешно добавляется, а также корректно обрабатываются различные ошибочные сценарии, такие как ввод некорректной суммы и названия операции. Это гарантирует корректное функционирование данного функционала в приложении.

## Тестирование функций депозитного счета

После создания депозитного счета вначале есть только одно поле – новый баланс. То есть пользователь может изменить баланс депозитного счета. Далее, пользователь может добавить финансовую цель. После создания финансовой цели пользователь может рассчитать план инвестиций. Результаты тестирования данного функционала представлены в таблице 4.5.

Таблица 4.5 – Результаты тестирования функций депозитного счета

|  |  |  |
| --- | --- | --- |
| Шаги | Ожидаемый результат | Результат теста |
| Тестирование функционала обновления баланса депозитного счета | | |
| Ввод корректного нового баланса | Баланс депозитного счета успешно обновляется с указанным новым балансом | Пройден |
| Тестирование функционала обновления баланса депозитного счета с ошибками | | |
| Попытка обновления баланса с некорректным значением | Выводится сообщение об ошибке о некорректном значении баланса | Пройден |
| Тестирование функционала добавления финансовой цели | | |
| Ввод корректной финансовой цели (желаемого баланса) | Выводится сообщение об ошибке о некорректном значении финансовой цели | Пройден |
| Тестирование функционала добавления финансовой цели с ошибками | | |
| Попытка добавления финансовой цели с некорректным значением | Выводится сообщение об ошибке о невозможности повторного ограничения по категории | Пройден |
| Тестирование функционала расчета плана инвестиций | | |
| Ввод корректного процента инвестирования, срока вклада и частоты переинвестирования | План инвестиций успешно рассчитывается с указанным процентом инвестирования, сроком вклада и частотой переинвестирования | Пройден |
| Тестирование функционала расчета плана инвестиций с ошибками | | |
| Попытка расчета плана инвестиций с некорректным значением процента инвестирования | Выводится сообщение об ошибке о некорректном значении процента инвестирования | Пройден |
| Попытка расчета плана инвестиций с некорректным значением срока вклада | Выводится сообщение об ошибке о некорректном значении срока вклада | Пройден |
| Попытка расчета плана инвестиций с некорректной частотой переинвестирования | Выводится сообщение об ошибке о некорректной частоте переинвестирования | Пройден |

Тестирование функционала депозитного счета подтвердило его работоспособность при правильном вводе данных, а также корректную обработку ошибочных сценариев.

## Тестирование функций телеграм-бота

Телеграм-бот предоставляет пользователю множество функций, однако для их корректной работы необходимо соблюдать определенные правила. Среди функций телеграм-бота есть аутентификация, добавление новых финансовых операций, добавление счета, добавление новой финансовой операции с помощью голосового ввода. Результаты тестирования данного функционала представлены в таблице 4.6.

Таблица 4.6 – Результаты тестирования функций телеграм-бота

|  |  |  |
| --- | --- | --- |
| Тестирование функционала аутентификации пользователя | | |
| Ввод корректных данных при аутентификации | Успешная аутентификация пользователя | Пройден |
| Тестирование функционала аутентификации пользователя с ошибками | | |
| Попытка ввода некорректных данных при аутентификации | Выводится сообщение об ошибке о некорректных данных | Пройден |
| Тестирование функционала добавления новой операции | | |
| Ввод корректной суммы и названия при добавлении новой операции | Новая операция успешно добавляется с указанным названием и суммой | Пройден |
| Тестирование функционала добавления новой операции с ошибками | | |
| Попытка ввода некорректной суммы при добавлении новой операции | Выводится сообщение об ошибке о некорректной сумме | Пройден |
| Попытка ввода некорректного названия при добавлении новой операции | Выводится сообщение об ошибке о некорректном названии | Пройден |
| Отправка команды добавления операции без выбора счета | Выводится сообщение об ошибке о необходимости выбора счета | Пройден |
| Тестирование функционала добавления нового счета | | |
| Ввод корректного названия и валюты при добавлении нового счета | Новый счет успешно добавляется с указанным названием и валютой | Пройден |
| Тестирование функционала добавления нового счета с ошибками | | |
| Попытка ввода некорректного названия при добавлении нового счета | Выводится сообщение об ошибке о некорректном названии счета | Пройден |
| Тестирование функционала голосового ввода | | |
| Отправка голосового сообщения с корректными данными | Голосовое сообщение успешно обрабатывается и создается новая финансовая операция | Пройден |

Продолжение таблицы 4.6

|  |  |  |
| --- | --- | --- |
| Шаги | Ожидаемый результат | Результат теста |

|  |  |  |
| --- | --- | --- |
| Тестирование функционала голосового ввода с ошибками | | |
| Отправка голосового сообщения с некорректными данными | Выводится сообщение об ошибке о некорректных данных | Пройден |
| Отправка голосового сообщения без предварительного выбора счета | Выводится сообщение об ошибке о необходимости выбора счета | Пройден |

Исходя из результатов тестирования, функционал телеграм-бота успешно прошел все проверки как при вводе корректных, так и некорректных данных.

## Выводы по разделу

Исходя из результатов тестирования, проведенного в рамках данной главы, можно сделать вывод, что все ключевые функциональные компоненты веб-приложения и телеграм-бота прошли проверки и показали свою надежность и устойчивость.

Страницы регистрации и авторизации успешно прошли тестирование, как при корректном, так и некорректном вводе данных. Приложение правильно обрабатывает случаи некорректного ввода, отображая соответствующие сообщения об ошибках.

Функционал создания нового счета был проверен на ввод корректных данных, а также на случаи ввода некорректной суммы и превышения допустимой длины названия счета. Все тесты прошли успешно, подтверждая, что система надежно обрабатывает различные сценарии ввода данных.

Создание ограничений по категориям расходов также было протестировано на корректность ввода суммы и выбора категории. Результаты тестирования показали, что система правильно обрабатывает как корректные данные, так и ошибки ввода.

Добавление финансовых операций было протестировано на ввод названия, суммы, категории и даты операции. Приложение успешно справилось с обработкой как корректных данных, так и различных ошибок ввода, подтверждая свою надежность.

Функции депозитного счета включали обновление баланса, добавление финансовой цели и расчет плана инвестиций. Тестирование показало, что система корректно обрабатывает ввод данных для всех этих функций, а также надежно распознает и обрабатывает ошибки.

Телеграм-бот был протестирован на корректность выполнения функций аутентификации, добавления новых финансовых операций, создания нового счета и добавления операций с помощью голосового ввода. Результаты тестирования показали, что бот корректно обрабатывает как правильные команды и данные, так и ошибки ввода, выводя соответствующие сообщения.

Таким образом, тестирование всех вышеуказанных компонентов веб-приложения и телеграм-бота подтвердило их готовность к использованию, обеспечивая надежность и стабильность работы в различных сценариях.

# Руководство пользователя

ФИО

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

5 Руководство пользователя

Лит.

Листов

10

74417023, 2024

В данной главе представлено подробное руководство по использованию веб-приложения «Финансовый помощник». Оно подразумевает использование только одной роли – пользователя. Поэтому описанное далее руководство будет фокусироваться на потребностях и задачах, которые возникают у авторизованного пользователя в процессе работы с приложением. Каждый этап процесса будет рассмотрен подробно с примерами и рекомендациями, чтобы обеспечить максимальную удобство и эффективность использования приложения.

Данное руководство призвано облегчить процесс освоения и использования приложения, помогая пользователям получить максимальную выгоду от его работы.

## Методика использования приложения

Открыв веб-приложение или попытавшись перейти на другую страницу, неавторизованный пользователь попадает на страницу регистрации. Данная страница приведена на рисунке 5.1.

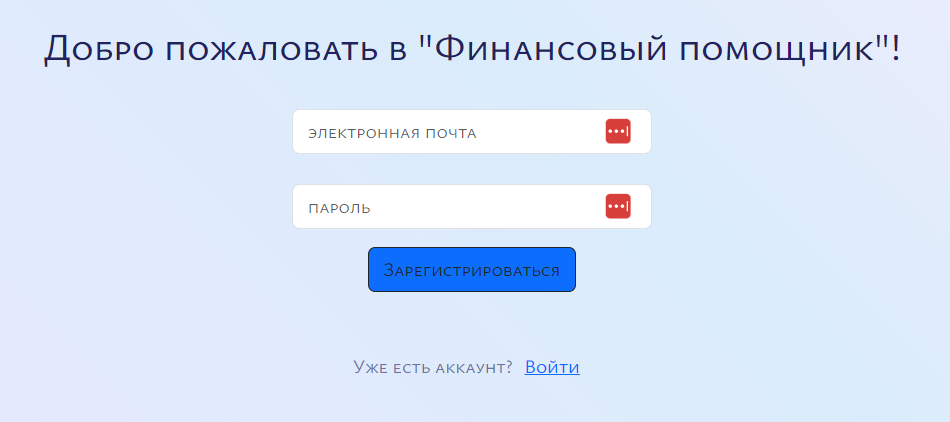


Рисунок 5.1 – Страница регистрации

Чтобы пользователь попал в систему, ему нужно зарегистрироваться, введя при этом свою почту и надежный пароль.

Следующим шагом необходимо нажать на кнопку «Войти». Чтобы пользователь попал в систему, ему необходимо ввести свою почту и пароль, которые были введены при регистрации, или же использовать аутентификацию через Google OAuth 2.0 [22]. Страница входа в приложение представлена на рисунке 5.2.

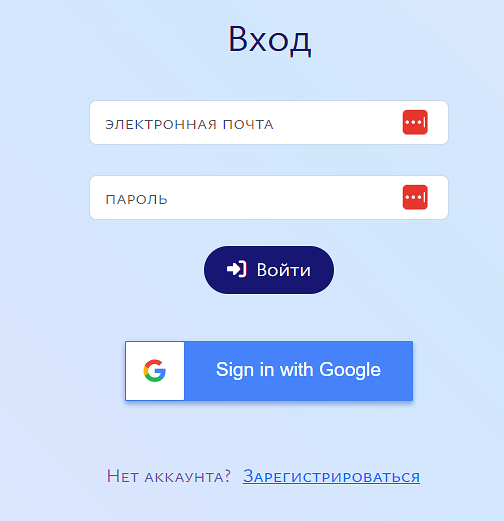


Рисунок 5.2 – Страница авторизации

После входа в приложение, пользователь попадает на главную страницу, на которой он может администрировать свои финансовые данные. Исходное состояние страницы представлено на рисунке 5.3.

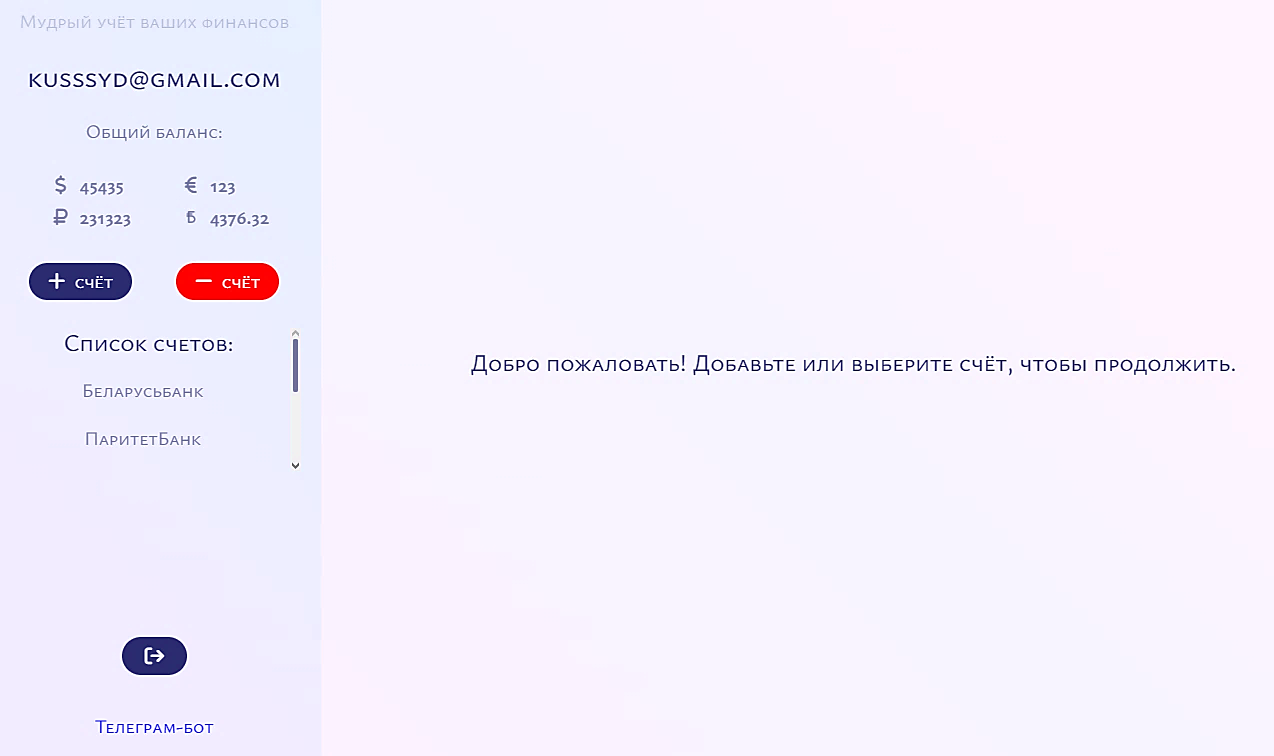


Рисунок 5.3 – Главная страница

Далее, пользователь имеет возможность создавать счеты двух типов: расчетный и депозитный. Для этого нужно нажать на кнопку «+ счет». Пример правильного создания расчетного счета представлен на рисунке 5.4.



Рисунок 5.4 – Пример корректного создания счета с типом «расчетный»

Как говорилось ранее, пользователь может создать счет с типом депозитный. Тогда в появившемся модальном окне необходимо ввести сумму, которая есть на счете. Пример правильного создания депозитного счета представлен на рисунке 5.5.



Рисунок 5.5 – Пример корректного создания счета с типом «депозитный»

После создания счета с типом «расчетный», пользователю доступны 5 пунктов меню: обзор, доходы, расходы, категории и ограничения.

Помимо этих функций, у пользователя есть возможность взаимодействия с уже созданными счетами. Для этого, ему нужно нажать на значок шестеренки возле названия выбранного счета. После этого появится модальное окно, на котором отображены все функции, которые можно сделать с счетом. Однако этот функционал доступен только если пользователь является владельцем счета. Приглашенный пользователь не увидит значка шестеренки. Модальное окно для взаимодействия с счетом представлено на рисунке 5.6.

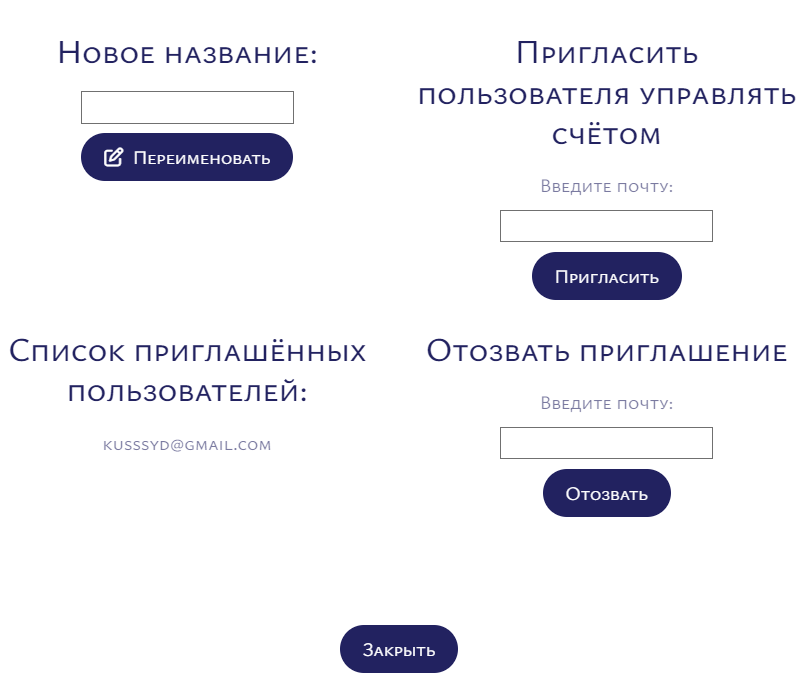


Рисунок 5.6 – Модальное окно для взаимодействия с счетом

Здесь пользователь может переименовать счет, пригласить или отозвать пользователя от управления счетом, а также увидеть список приглашенных пользователей.

Далее рассмотрим пункт «категории». Здесь добавляются категории для расходов и доходов. По умолчанию создается категория «другое», которая относится к типу «общая». Пример этой страницы представлен на рисунке 5.7.

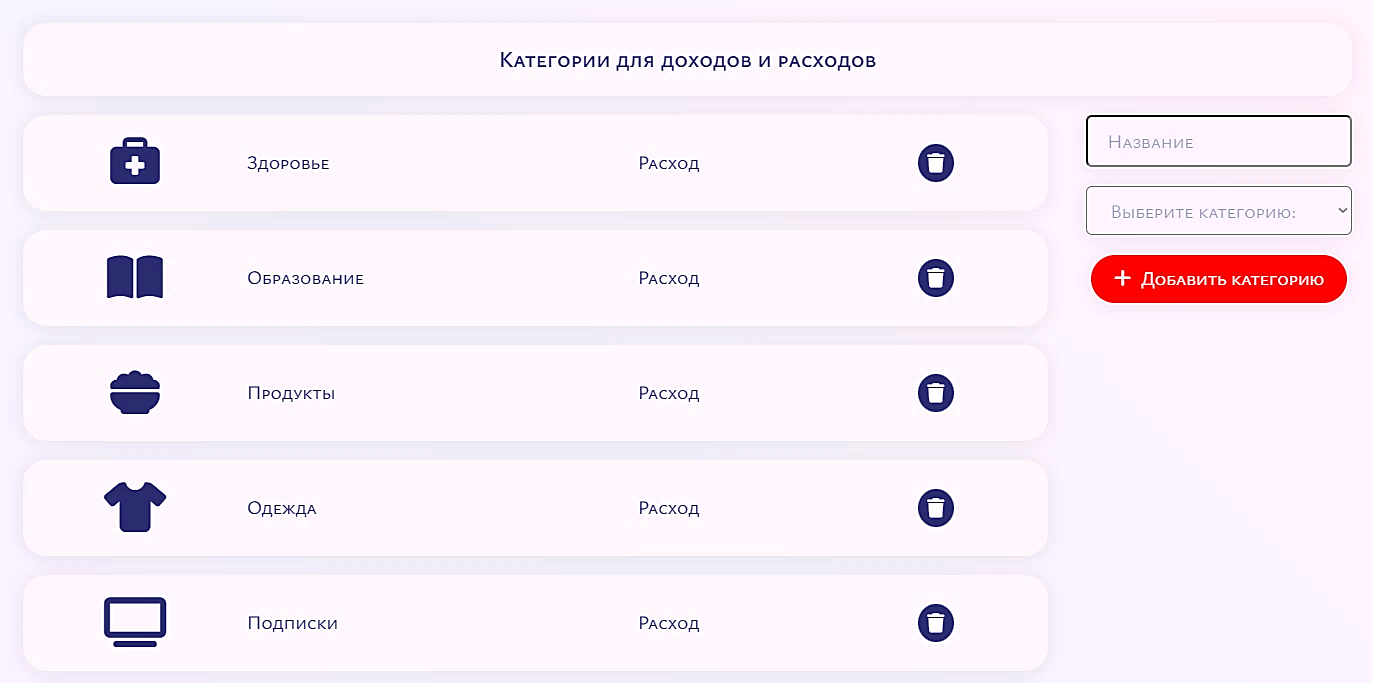


Рисунок 5.7 – Пример созданных категорий

Во вкладке ограничений пользователь может добавить ограничения по расходом на желаемую категорию. Пример представлен на рисунке 5.8.

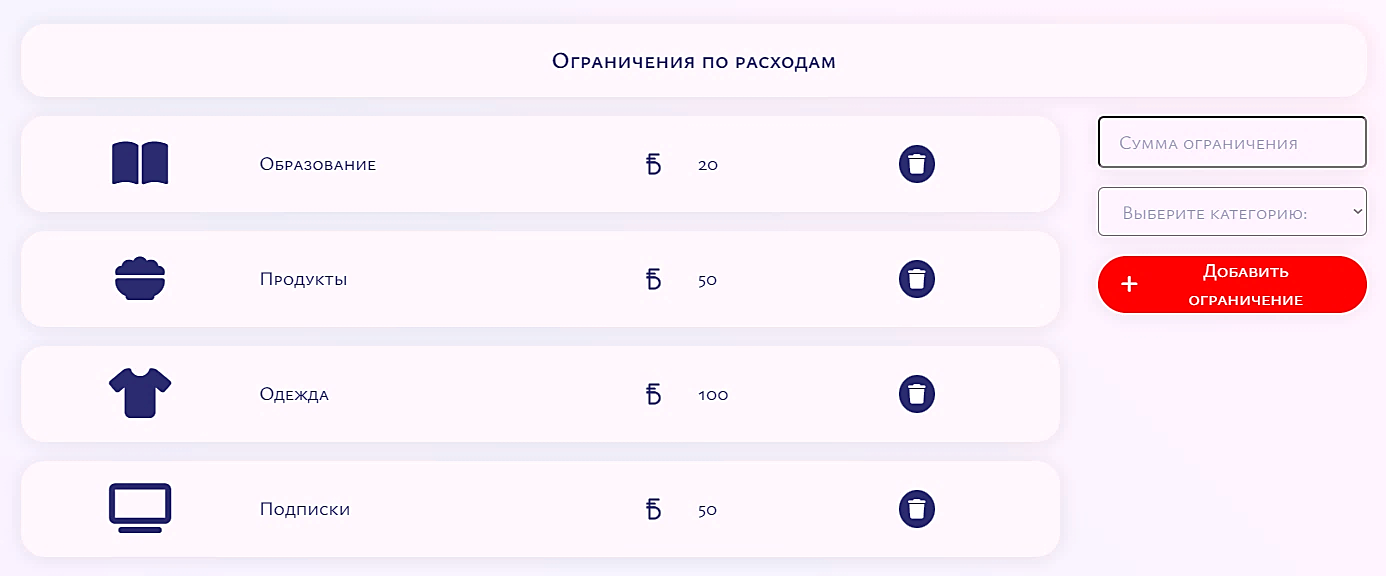


Рисунок 5.8 – Пример ограничений по категориям расходов

В разделах «расходы» и «доходы» пользователь имеет возможность вносить информацию о соответствующих финансовых операциях. Они включают в себя доходы и расходы. Пример добавления корректно заполненных новых доходов представлен на рисунке 5.9.

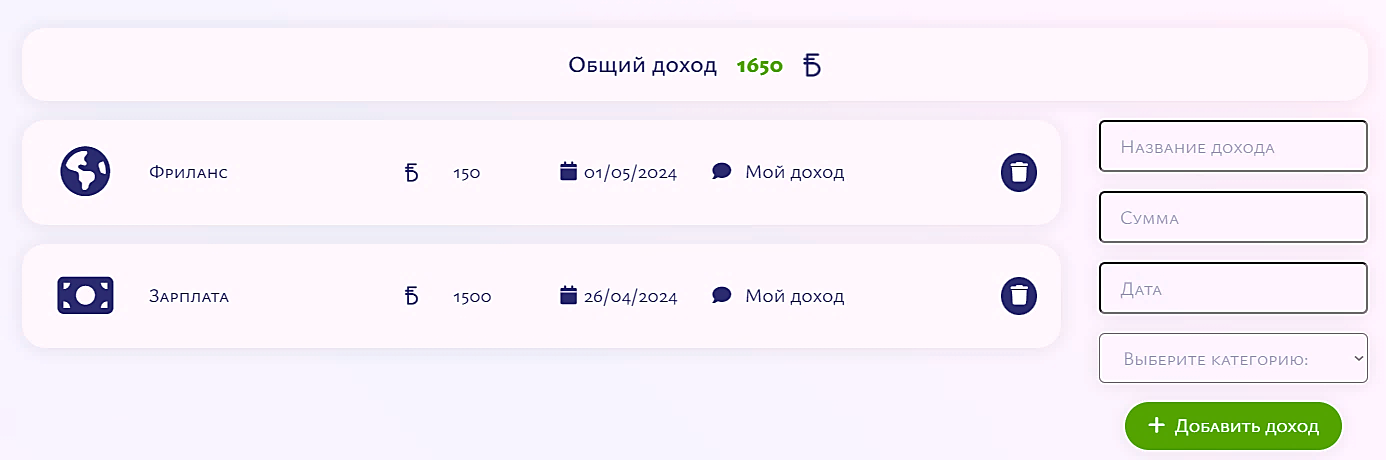


Рисунок 5.9 – Информация о доходах

Пример добавления новых расходов представлен на рисунке 5.10.

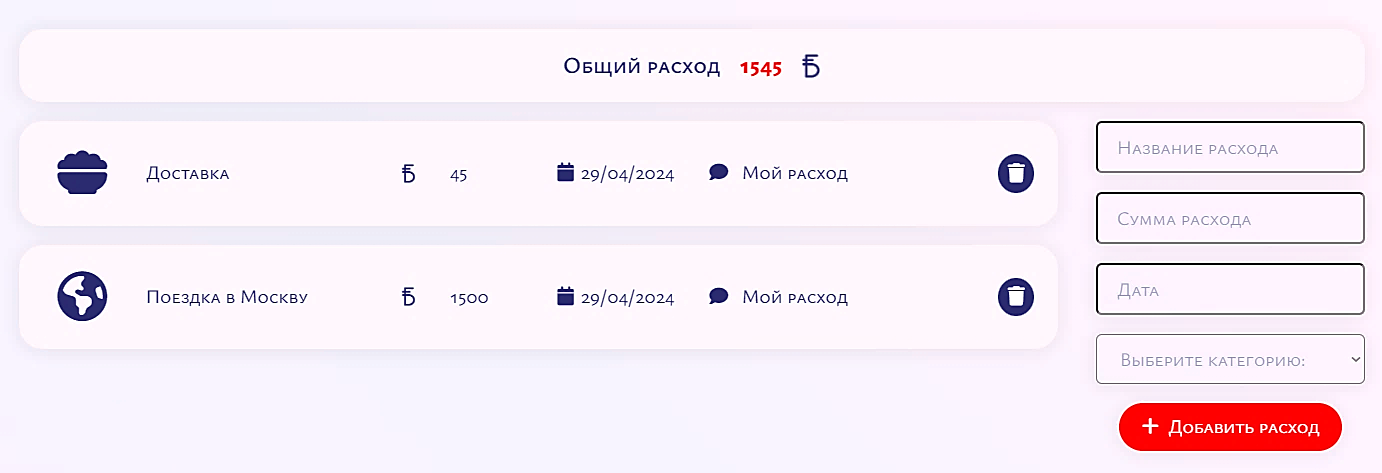


Рисунок 5.10 – Информация о расходах

Кроме того, если ограничение по расходам было превышено, то пользователь получит уведомление, как это показано на рисунке 5.11.

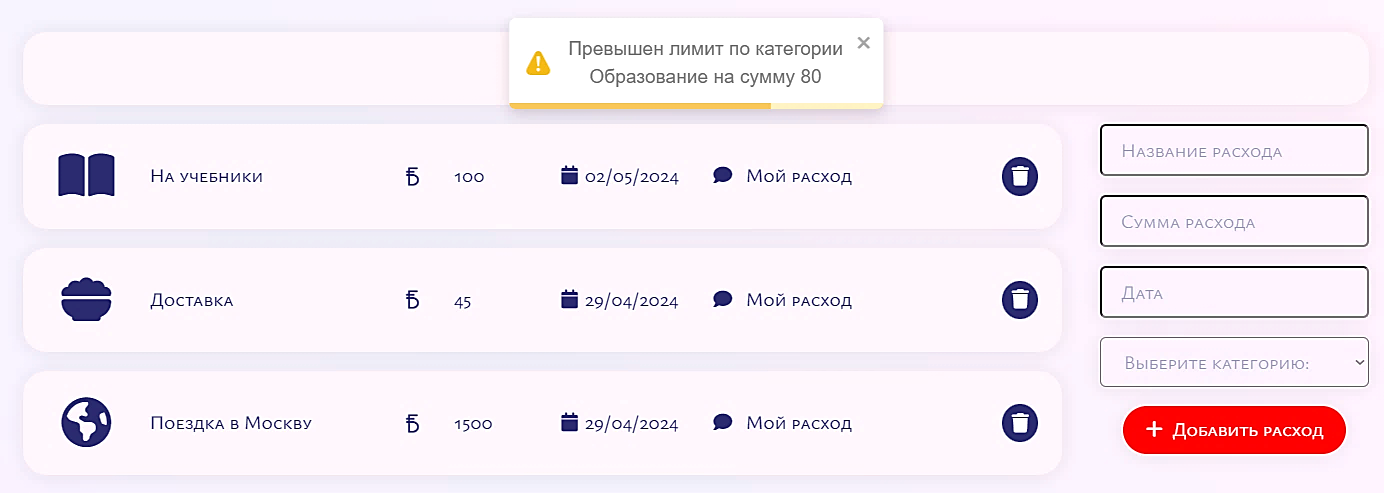


Рисунок 5.11 – Уведомление о превышении ограничения по категории расходов

В обзорной панели пользователь получает доступ к такой информации, как история последних доходов и расходов, максимальный и минимальный доход или расход, общая сумма доходов и расходов, текущий баланс счета, статистика по доходам и расходам, а также отчет по доходам и расходам. У пользователя есть возможность отфильтровать диаграмму доходов и расходов по датам. Как статистику, так и отчет, пользователь может экспортировать в формат PDF. Общая статистика по счету вместе с диаграммой представлена на рисунке 5.12.

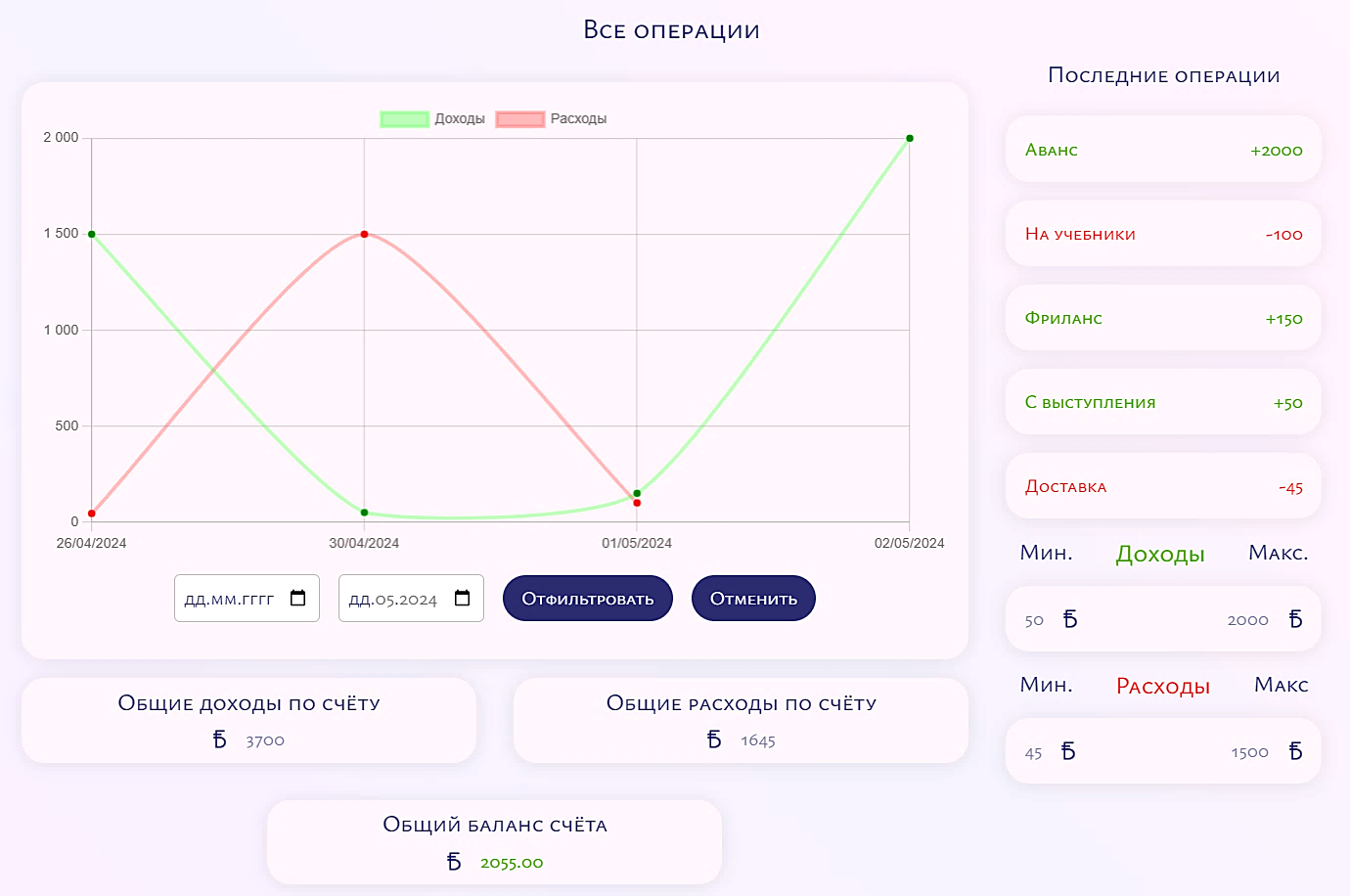


Рисунок 5.12 – Общая информация о финансах пользователя

Круговые диаграммы, отражающие статистику по расходам и доходам по категориям отображены на рисунке 5.13.



Рисунок 5.13 – Статистика по доходам и расходам

Отчет по доходам и расходам в виде таблицы представлен на рисунке 5.14.



Рисунок 5.14 – Отчет по доходам и расходам

Кроме того, пользователь на этой странице может создать стратегию расходов на следующий месяц. Пример представлен на рисунке 5.15.



Рисунок 5.15 – Создание стратегии расходов на следующий месяц

Для счета с типом «депозитный» пользователю доступен всего один пункт меню – финансы. Здесь пользователь может добавлять свои финансовые цели и рассчитать план сбережений. Пример представлен на рисунке 5.16.

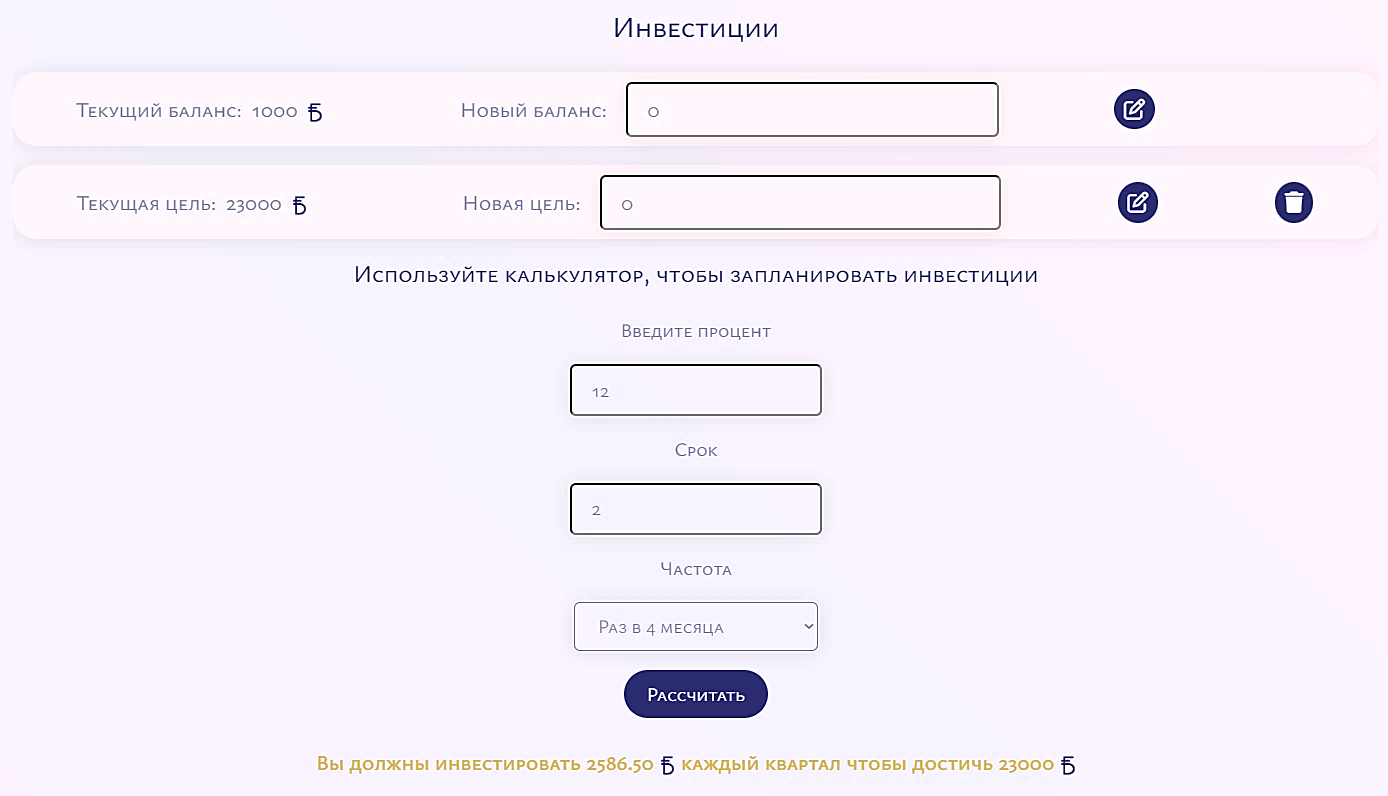


Рисунок 5.16 – Управление депозитным счетом

Далее перейдем к взаимодействию с телеграм-ботом. После запуска бота мы видим сообщения с предложением войти в систему. Это приведено на рисунке 5.17.

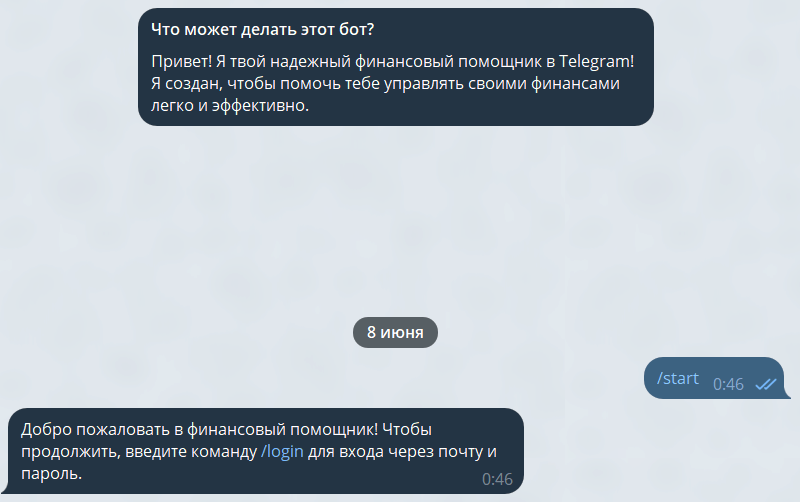


Рисунок 5.17 – Приветствие в телеграм-боте

Следующим шагом, мы входим в систему. Для этого нужно отправить команду /login, после чего ввести логин и пароль. После успешной авторизации бот предлагает выбрать один из счетов. И только после выбора счета мы можем отправлять команды. Пример представлен на рисунке 5.18.

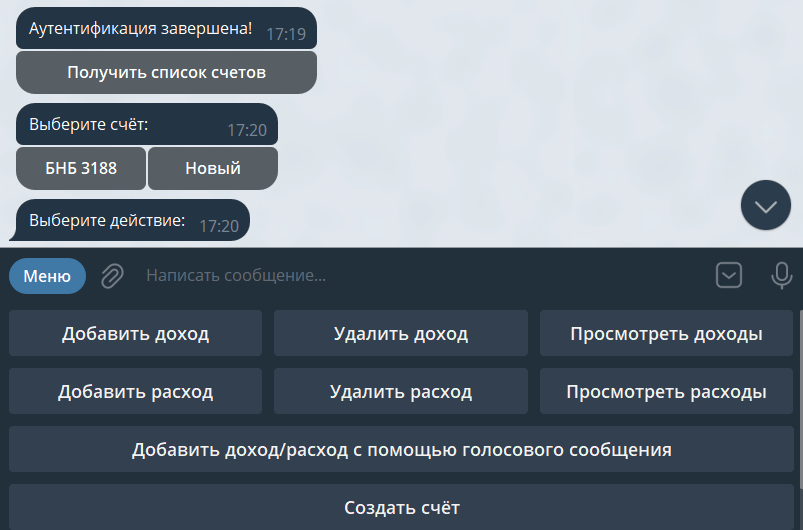


Рисунок 5.18 – Действия, доступные после авторизации и выбора счета.

Если же счет еще не был создан, будет выведено соответствующее сообщение. На этом же этапе мы можем его создать. Данный пример представлен на рисунке 5.19.

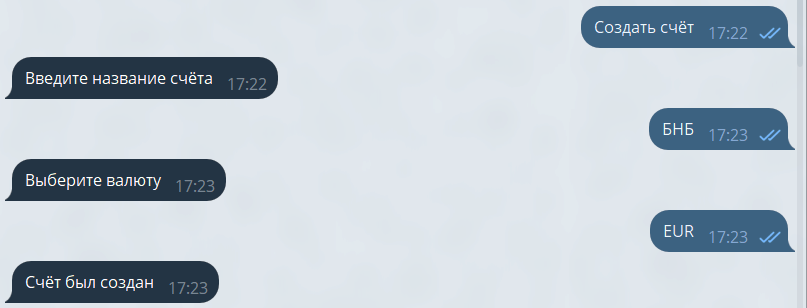


Рисунок 5.19 – Пример создания нового счета

Следующим шагом после создания счета являются стандартные операции, доступные и в основной клиентской части. Одной из такой операций является добавление нового дохода. Пример добавления нового дохода представлен на рисунке 5.20.



Рисунок 5.20 – Пример добавления нового дохода

Одной из уникальных возможностей бота, которой нет на основной клиентской части является добавление финансовой операции с помощью голосового сообщения. Однако, прежде чем отправить на обработку голосовое сообщение, необходимо войти в систему, выбрать существующий счет или создать новый и выбрать его, после чего можно будет отправить голосовое сообщение. Пример представлен на рисунке 5.21.

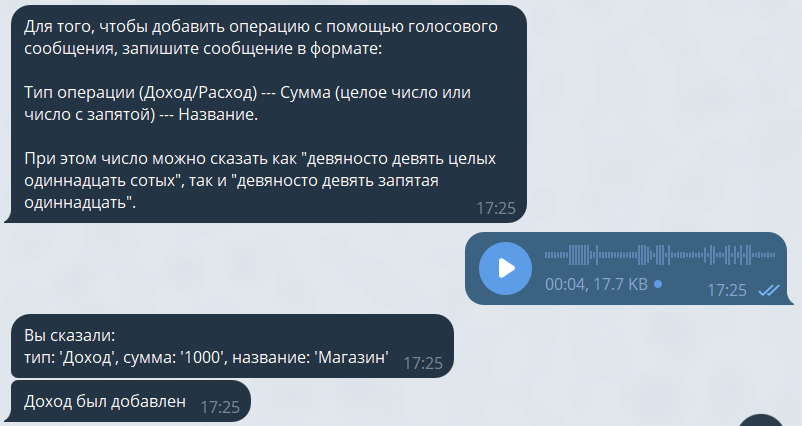


Рисунок 5.21 – Пример добавления операции с помощью голосового сообщения

Стоит отметить, что при добавлении новой операции с помощью голосового сообщения не нужно указывать категорию дохода или расхода, как это сделано в основной клиентской части. Это сделано для того, чтобы ускорить процесс добавления новой финансовой операции с мобильного устройства. Поэтому, чтобы добавить новую операцию, необходимо назвать три слова: тип операции (доход или расход), сумму и название операции.

Также стоит отметить, что для добавления новой финансовой операции с помощью голосового сообщения не нужно выбирать команду из списка доступных операций со счетом. Команда «Добавить доход/расход с помощью голосового сообщения» нужна только для информирования пользователя о правильном формате ввода сообщения. После выбора счета нужно просто записать голосовое сообщение, которое бот обработает и добавит соответствующую операцию в базу данных.

## Выводы по разделу

В данном разделе представлены ключевые аспекты использования веб-приложения и подробные инструкции по доступным функциям приложения, что поможет пользователю быстро освоиться с интерфейсом и достичь желаемых результатов. Приведены скриншоты интерфейса приложения с подробными пошаговыми инструкциями по взаимодействию пользователя с каждой страницей. Описан доступный функционал на различных страницах, представлены алгоритмы использования, а также описаны основные элементы интерфейса.

# Технико-экономическое обоснование проекта

ФИО

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

Буданова К.А.

И.И.

Провер.

Сухорукова И.Г.

Консульт.

Евлаш А.И.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

6 Технико-экономическое

обоснование проекта

Лит.

Листов

9

74417023, 2024

## Общая характеристика разрабатываемого программного средства

При выполнении дипломного проекта было разработано веб-приложение «Финансовый помощник» для эффективного администрирования финансовых данных.

Целью веб-приложения было улучшение и объединение всех необходимых функций, учитывая проблемы и недостатки существующих аналогов, а также внедрение усовершенствований для создания более удобного и полного пользовательского взаимодействия.

При помощи данного веб-приложения пользователи могут администрировать свои доходы и расходы, добавлять ограничения на расходы и получать уведомления при их превышении, предсказывать расходы на следующий месяц, создавать финансовые цели, планировать инвестиции, совместно с другими пользователями управлять данными созданных счетов, а также взаимодействовать с телеграм-ботом для быстрого добавления данных о совершенных финансовых операциях.

Разработанное программное решение имеет следующие преимущества перед рассмотренными в главе 1 аналогичными решениями:

* интуитивно понятный и неперегруженный интерфейс;
* возможность быстрого доступа к данным через телеграм-бот и добавление новой финансовой операции с помощью голосового ввода;
* наличие функций для предоставления отчетности финансовых операций;
* возможность приглашения другого пользователя для совместного администрирования финансовых данных счета;
* возможность создания как депозитного счета, так и расчетного;
* наличие финансового калькулятора для планирования инвестиций.

Во время разработки дипломного проекта для создания серверной части веб-приложения использовалась платформа Node.js, язык программирования TypeScript и фреймворк Express.js. Для разработки клиентской части приложения была использована библиотека React.js. Взаимодействие клиента с сервером происходит посредством использования протокола HTTP. Данное веб-приложение разработано для последующего использования в коммерческих целях.

Стратегия монетизации основана на рекламе, что позволяет пользователям пользоваться приложением бесплатно.

## Исходные данные для проведения расчетов и маркетинговый анализ

Источниками исходных данных для расчетов выступают действующие законы и нормативно-правовые акты. Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Численность разработчиков | чел. | Чр | 3 |
| Норматив дополнительной заработной платы | % | Ндз | 12 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нфсзн | 34 |
| Ставка отчислений в БРУСП «Белгосстрах» | % | Нбгс | 0,6 |
| Норматив прочих прямых затрат | % | Нпз | 20 |
| Норматив накладных расходов | % | . | 43 |
| Норматив расходов на реализацию | % | Нрса | 8 |

Для анализа стоимости разработки программного обеспечения необходимо прежде всего изучить стоимость создания аналогичных приложений. В этой связи было выбрано три приложения-аналога, каждое из которых отличается уникальным функционалом и набором предлагаемых услуг.

Результаты проведения анализа стоимости представлены в таблице 6.2.

Таблица 6.2 – Анализ стоимости разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Продукт-аналог | Источник | Стоимость, руб | Примечание |
| Quicken Home & Business | quicken.com | 45 790 | Учет основных финансовых операций. Относительно высокая стоимость в сравнении с аналогичными продуктами, не такой широкий функционал как у конкурентов. |
| PocketGuard Ultimate | pocketguard.com | 35 660 | Автоматическое отслеживание расходов, бюджетирование, анализ финансов. Ограниченные возможности категоризации расходов. |
| Banktivity | banktivity.com | 24 567 | Управление бюджетом, отслеживание расходов, управление инвестициями, отчетность, поддержка многопользовательского режима. Сложность настройки для новых пользователей. |

В процессе проведения маркетингового анализа была определена стоимость разработки аналогичного программного продукта для организации взаимоотношений с клиентами. Средняя стоимость разработки такого продукта оценивается в диапазоне от 24 567 до 45 790 рублей. Следовательно, общая стоимость разработки данного программного средства, выбранного в качестве базы для сравнения, составляет 19 000 рублей. Разрабатываемый продукт не только дешевле и выгоднее самого доступного аналога, но и превосходит его по функциональности. Веб-приложения аналоги имеют меньшие возможности: пользователи не могут категоризировать доходы и расходы, создавать отчеты, планировать инвестиции, приглашать других пользователей для совместного администрирования данных, а также не могут получить быстрый доступ к данным через телеграм-бот и добавлять новые финансовые операции с помощью голосового ввода.

## Обоснование цены программного средства

### Расчет затрат рабочего времени на разработку программного средства

В таблице 6.3 указаны в укрупненном виде все работы, реально выполненные для создания, указанного в дипломной работе программного средства, и количество рабочих дней, реально потраченных для выполнения этих работ. Были учтены затраты на разработку программного средства, поиск нужных компонентов платформы, с сервисами которой будет интегрировано программное средство, а также тестирование и написание руководства пользователя.

Таблица 6.3 – Затраты рабочего времени на разработку ПС

|  |  |  |
| --- | --- | --- |
| Содержание работ | Исполнитель | Затраты рабочего времени, часов |
| Исследование и анализ требований | Бизнес-аналитик | 10 |
| Анализ аналогичных приложений | Бизнес-аналитик | 6 |
| Подготовка технического задания | Бизнес-аналитик | 15 |
| Разработка UI/UX дизайна | Дизайнер | 50 |
| Создание архитектуры приложения | Программист | 60 |
| Проектирование базы данных | Программист | 15 |
| Разработка серверной части приложения | Программист | 75 |
| Разработка клиентской части приложения | Программист | 125 |
| Тестирование серверной части приложения | Программист | 20 |
| Тестирование клиентской части приложения | Дизайнер | 15 |
| Написание руководства пользователя | Дизайнер | 5 |
| Всего |  | 396 |

В данной таблице представлены затраты рабочего времени специалистами на различные этапы разработки программного обеспечения. Общие затраты времени всех специалистов составляют 396 часов.

### Расчет основной заработной платы

После определения часовых ставок и трудозатрат исполнителей определяются основные заработные платы всех исполнителей [23]. Основная заработная плата отдельного специалиста будет рассчитываться по формуле 6.1. Результаты подсчетов представлены в таблице 6.4.

|  |  |  |
| --- | --- | --- |
|  |  | (6.1) |

где Соз– основная заработная плата, руб.;

Траз – трудоемкость (чел./час.);

Сзп – средняя часовая ставка руб./час.

Таблица 6.4 – Расчет основной заработной платы специалистов

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель | Затраты рабочего времени, часов | Средняя часовая ставка, руб./час | Основная заработная плата, руб. |
| Программист | 295 | 9,37 | 2 764,15 |
| Бизнес-аналитик | 31 | 17,59 | 545,29 |
| Дизайнер | 70 | 15,63 | 1 094,1 |
| Всего | 396 |  | 4 403,54 |

Таким образом, при разработке программного средства основная заработная плата программиста составит 2 764,15 руб., бизнес-аналитика – 545,29 руб., а дизайнера – 1 094,1 руб. Суммарная основная заработная плата всех специалистов проекта по разработке веб-приложения для администрирования финансов «Финансовый помощник» составит 4 403,54 руб.

### Расчет дополнительной заработной платы

Дополнительная заработная плата на конкретное программное средство включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах к основной заработной плате по формуле 6.2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | (6.2) | , | (6.2) |

где Соз– основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.



Таким образом, дополнительная заработная плата составила 528,42 рубля.

### Расчет отчислений в Фонд социальной защиты населения и по обязательному страхованию

Отчисления в Фонд социальной защиты населения (ФСЗН) и по обязательному страхованию от несчастных случаем на производстве и профессиональных заболеваний определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей.

Отчисления в Фонд социальной защиты вычисляются по формуле 6.3.

|  |  |  |
| --- | --- | --- |
|  |  | (6.3) |

где Соз – основная заработная плата, руб.;

Сдз – дополнительная заработная плата на конкретное ПС, руб.;

Нфсзн – норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.4.

|  |  |  |
| --- | --- | --- |
|  |  | ((6.4) |





Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 29,59 руб., а в фонд социальной защиты населения – 1 676,87 руб.

### Расчет суммы прочих прямых затрат

Сумма прочих затрат Спз определяется как произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив прочих затрат в целом по организации Нпз, и находится по формуле 6.5.

|  |  |  |
| --- | --- | --- |
|  |  | (6.5) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму прочих затрат.



В итоге получена сумма прочих прямых затрат равная 880,71 руб.

### Расчет суммы накладных расходов

Сумма накладных расходов Сн.р. – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нн.р., по формуле 6.6.

|  |  |  |
| --- | --- | --- |
|  |  | (6.6) |

Сумма накладных расходов составит:



В итоге была получена сумму накладных расходов на сумму 1 893,52 руб.

### Сумма расходов на разработку программного средства

Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, суммы прочих затрат и суммы накладных расходов, по формуле 6.7.

|  |  |
| --- | --- |
|  | (6.7) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму расходов на разработку программного средства.



Таким образом, итоговая сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в текущем разделе, и составила 9 412,65 руб.

### Расходы на реализацию

Сумма расходов на реализацию программного средства Срса определяется как произведение суммы расходов на разработку на норматив расходов на реализацию Нрса, и находится по формуле 6.8.

|  |  |  |
| --- | --- | --- |
|  |  | (6.8) |



Сумма расходов на реализацию была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 753,01 руб.

6.3.9 Полная себестоимость

Полная себестоимость Сп определяется как сумма двух элементов: суммы расходов на разработку Ср и суммы расходов реализацию программного средства Срса. Полная себестоимость Сп вычисляется по формуле 6.9.

|  |  |  |
| --- | --- | --- |
|  |  | (6.9) |



Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 10 165,66 руб.

6.3.10 Определение цены, оценка эффективности

Прежде всего, необходимо определить сумму денежных поступлений и окупаемость затрат на разработку продукта, т.е. целесообразность и эффективность. Для этого необходимо приблизительно рассчитать количество подписок/установок на основании данных о продуктах-аналогах. Рассмотрим аналоги из пункта 6.2 и дадим им краткую характеристику.

Продукты-аналоги:

* Quicken Home & Business – программное обеспечение для управления финансами, которое помогает отслеживать доходы и расходы, составлять бюджеты, управлять инвестициями и генерировать налоговые отчеты;
* PocketGuard Ultimate – приложение для контроля расходов и оптимизации бюджетов, которое автоматически синхронизируется с банковскими счетами, классифицирует транзакции, отслеживает доходы и расходы, а также предлагает функции для отслеживания долгов, сбережений и финансовых целей;
* Banktivity – финансовое программное обеспечение, позволяющее отслеживать доходы и расходы, управлять бюджетами и генерировать подробные отчеты.

Были выбраны следующие характеристики для показателей качества рассматриваемого программного продукта и программного продукта конкурента:

1. Дизайн – то, что пользователь видит и с чем непосредственно взаимодействует – очень важно для любого современного веб-приложения;
2. Юзабилити – насколько приложение удобно в использовании;
3. Функциональность – количество инноваций, внедряемых в приложение;
4. Отсутствие багов – наличие несущественных ошибок в работе.

Расчет показателей качества базовых и нового продуктов, согласно балловому методу, приводится в таблице 6.5.

Таблица 6.5 – Оценка качества программного средства

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатель качества | Весовой коэффициент | Разрабатываемый продукт | Продукт-аналог 1 | Продукт-аналог 2 | Продукт-аналог 3 |
| Дизайн | 0,3 | 7 | 5 | 8 | 7 |
| Юзабилити | 0,3 | 8 | 7 | 6 | 6 |
| Функциональность | 0,3 | 8 | 6 | 7 | 8 |
| Отсутствие багов | 0,1 | 7 | 5 | 7 | 6 |
| Всего | 1 | 7,5 | 5,75 | 7 | 6,75 |

Исходя из данных в таблице, можно сделать вывод, что разрабатываемый продукт имеет самую высокую общую оценку качества (7,5), что делает его наилучшим вариантом среди всех анализируемых аналогов.

Расчет прогнозного количества установок программного средства рассчитывается исходя из среднего значения установок по каждому аналогу. Количество установок К аналогов при монетизации методом подписок, скорректированная на оценку показателей качества, рассчитывается по формуле 6.10.

|  |  |
| --- | --- |
|  | (6.10) |

где К0 – количество установок ПС конкурента;

T0 – количество лет существования приложения;

ИР – показатель рассматриваемого программного продукта;

ИК – показатель программного продукта конкурента.









По данным исследований CTR за тысячу показов составляет в среднем 1%. Ожидается, что в среднем каждый пользователь будет добавлять около 30 ссылок. Реклама будет отображаться как нативная реклама, каждый переход будет оцениваться в 0,1 белорусских рублей.

Теперь рассчитаем денежные поступления от рекламы. Для этого сначала рассчитаем общее количество показов рекламы в год, затем умножим это число на CTR и затем на сумму за каждый переход.



Таким образом, получаем Пост.в год = 4 707,89 рублей за год.

Срок окупаемости приложения Tок вычисляется по формуле 6.11.

|  |  |  |
| --- | --- | --- |
|  |  | (6.11) |

где Сп – полная себестоимость, руб.;

Пост.в год – денежные поступления от показа рекламы в приложении за год, руб.



Таким образом, мы получили, что данное веб-приложение окупится за 2,16 года.

6.4 Выводы по разделу

После тщательного изучения были определены основные факторы, влияющие на доход веб-приложения для администрирования финансов за счет рекламы. В таблице 6.6 представлены результаты расчетов основных экономических показателей, включая стоимость разработки и ожидаемую прибыль, а также годовые денежные поступления от рекламы и срок окупаемости.

Таблица 6.6 – Результаты расчетов основных экономических показателей разработки веб-приложения для организации взаимоотношений с клиентами

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, часов | 396 |
| Количество разработчиков, чел. | 3 |
| Основная заработная плата, руб. | 4 403,54 |
| Дополнительная заработная плата, руб. | 528,42 |
| Отчисления в Фонд социальной защиты населения и БРУСП «Белгосстрах», руб. | 1 706,46 |
|  |  |
| Прочие прямые затраты, руб. | 880,71 |
| Накладные расходы, руб. | 1 893,52 |
| Себестоимость разработки программного средства, руб. | 9 412,65 |
| Расходы на реализацию, руб. | 753,01 |
| Полная себестоимость, руб. | 10 165,66 |
| Годовые денежные поступления от рекламы, руб. | 4 707,89 |
| Срок окупаемости, лет | 2,16 |

В результате технико-экономического обоснования разработки веб-приложения для администрирования финансов были получены следующие ключевые показатели. Полная себестоимость разработки составила 10 165,66 руб., включая все затраты на заработную плату, отчисления в социальные фонды, прочие прямые затраты и накладные расходы. Ожидаемые годовые денежные поступления от рекламы составляют 4 707,89 руб., что демонстрирует способность приложения приносить стабильный доход уже в первый год эксплуатации, базируясь на консервативных оценках рыночного уровня цен на рекламу.

Срок окупаемости веб-приложения составляет 2,16 года, указывая на экономическую эффективность и быструю возвратность инвестиций. Учитывая тенденцию роста цен на рекламные показы, срок окупаемости может быть сокращен. Повышение стоимости рекламных переходов увеличит доходы, что позволит достичь точки безубыточности быстрее. Рост цен на рекламные услуги увеличивает доходы без значительных дополнительных затрат, усиливая привлекательность проекта.

В итоге, технико-экономическое обоснование показывает, что разработка веб-приложения для администрирования финансов является выгодным и перспективным проектом. Полная себестоимость разработки и ожидаемые годовые поступления от рекламы подтверждают его экономическую целесообразность. Быстрый срок окупаемости и потенциал для роста доходов от рекламы подчеркивают его инвестиционную привлекательность. Проект не только обеспечит стабильный доход, но и может стать значительным источником прибыли в будущем.

# Заключение

В результате успешного выполнения поставленных задач, цель дипломного проекта – разработка веб-приложения «Финансовый помощник» – была достигнута.

Перед началом работы был проведен аналитический обзор существующих прототипов приложений в области администрирования финансовых данных. Этот этап позволил выявить их преимущества и недостатки, что помогло определить требования к разрабатываемому веб-приложению.

Далее были выбраны средства разработки, технологии и библиотеки. Описана архитектура приложения, а также спроектированы диаграмма развертывания, диаграмма вариантов использования и логическая схема базы данных.

Было разработано веб-приложение по клиент-серверной архитектуре. Серверная часть реализована с использованием платформы Node.js, фреймворка Express.js и языка Typescript, а для связи с базой данных была использована облачная база данных MongoDB Atlas. Клиентская часть приложения представляет собой отдельный проект, построенный с использованием React. Также был разработан телеграм-бот, размещенный на отдельном сервере, который предоставляет дополнительный функционал и дублирует основные возможности клиентского приложения для взаимодействия с пользователями через мессенджер Telegram.

Было проведено ручное тестирование основных функций веб-приложения с целью проверки их работоспособности, а также выявления возможных ошибок и недочетов. Тестирование показало, что разработанное приложение «Финансовый помощник» функционирует стабильно и надежно в соответствии с поставленными к нему требованиями.

Для облегчения использования приложения было составлено подробное руководство пользователя, в котором описаны основные функции и возможности приложения, а также шаги по их использованию. Руководство содержит пошаговые инструкции с иллюстрациями и примерами, которые помогут пользователям быстро и легко освоить интерфейс и функционал приложения.

Также, была выбрана стратегия монетизации разработанного веб-приложения, основанная на рекламе, и рассчитаны основные экономические показатели, включая стоимость разработки и ожидаемую прибыль, а также годовые денежные поступления от рекламы и срок окупаемости.

Полученные результаты свидетельствуют о правильном функционировании разработанного веб-приложения и полном соответствии требованиям технического задания. Таким образом, цель дипломного проекта считается достигнутой.

По результатам дипломного проектирования сделан доклад на 75-й научно-технической конференции студентов и магистрантов.

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Буданова К.А.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Заключение

Лит.

Листов

1

74417023, 2024

# Список использованных источников

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

Буданова К.А.

Провер.

Сухорукова И.Г.

Н. контр.

Нистюк О.А.

Утв.

Смелов В.В.

Список использованных

источников

Лит.

Листов

2

74417023, 2024

1. Ciberleninka [Электронный ресурс] – Режим доступа: <https://cyberleninka.ru/article/n/neobhodimost-finansovogo-planirovaniya-v-usloviyah-sovremennoy-ekonomiki> – Дата доступа: 26.03.2024.
2. Moneytracker [Электронный ресурс] – Режим доступа: <https://app.moneytracker.cc> – Дата доступа: 27.03.2024.
3. Newretirement [Электронный ресурс] – Режим доступа: <https://www.newretirement.com> – Дата доступа: 27.03.2024.
4. JavaScript vs другие технологии [Электронный ресурс] – Режим доступа: [https://ain.ua/special/zachem-perehodit-na-javascript](https://ain.ua/special/zachem-perehodit-na-javascript/) – Дата доступа: 28.03.2024.
5. Язык программирования JavaScript [Электронный ресурс] – Режим доступа: <https://flexberry.github.io/ru/gbt_javascript.html> – Дата доступа: 30.03.2024.
6. О Node.js [Электронный ресурс] – Режим доступа: https://nodejs.org/ru/about – Дата доступа: 01.04.2024.
7. Express.js [Электронный ресурс] – Режим доступа: [https://expressjs.com/ru](https://expressjs.com/ru/) – Дата доступа: 03.04.2024.
8. HTTP [Электронный ресурс] – Режим доступа: [https://habr.com/ru/articles/215117](https://habr.com/ru/articles/215117/) – Дата доступа: 04.04.2024.
9. React [Электронный ресурс] – Режим доступа: <https://react.dev/learn> – Дата доступа: 09.04.2024.
10. DOM-дерево [Электронный ресурс] – Режим доступа: https://learn.javascript.ru/dom-nodes – Дата доступа: 12.04.2024.
11. Google cloud [Электронный ресурс] – Режим доступа: <https://cloud.google.com/why-google-cloud> – Дата доступа: 13.04.2024.
12. MongoDB Atlas [Электронный ресурс] – Режим доступа: [https://www.mongodb.com/docs/atlas](https://www.mongodb.com/docs/atlas/) – Дата доступа: 15.04.2024.
13. Firebase [Электронный ресурс] – Режим доступа: <https://firebase.google.com/docs> – Дата доступа: 17.04.2024.
14. HTTPS [Электронный ресурс] – Режим доступа: <https://hostiq.ua/wiki/http-https> – Дата доступа: 18.04.2024.
15. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования / Фаулер М. – 3-е изд. // СПб: Символ-Плюс – 2004 – 192 с.
16. Что такое NoSQL? [Электронный ресурс] – Режим доступа: https://www.oracle.com/cis/database/nosql/what-is-nosql – Дата доступа: 25.04.2024.
17. Tensorflow [Электронный ресурс] – Режим доступа: https://www.tensorflow.org/learn – Дата доступа: 29.04.2024.
18. Изучаем REST: Руководство по созданию RESTful-сервиса [Электронный ресурс] – Режим доступа: <https://restapitutorial.ru> – Дата доступа: 01.05.2024.
19. Typescript [Электронный ресурс] – Режим доступа: [https://www.typescriptlang.org/docs](https://www.typescriptlang.org/docs/) – Дата доступа: 05.05.2024.
20. Axios API [Электронный ресурс] – Режим доступа: https://axios-http.com/docs/api\_intro – Дата доступа: 09.05.2024.
21. Ручное тестирование [Электронный ресурс] – Режим доступа: <https://www.zaptest.com/ru/ручное-тестирование-что-это-такое-вид> – Дата доступа: 15.05.2024.
22. Using OAuth 2.0 to Access Google APIs [Электронный ресурс] – Режим доступа: <https://developers.google.com/identity/protocols/oauth2> – Дата доступа: 16.05.2024.
23. Каштелян, Т.В. Экономическое обоснование дипломных проектов / Каштелян Т.В. // Минск: УО «Белорусский государственный технологический университет» – 2013 – 88 с.

# Приложение А. Диаграмма развертывания

# Приложение Б. Диаграмма вариантов использования

# Приложение В. Логическая схема базы данных

# Приложение Г. Блок-схема алгоритма приглашения пользователя для совместного администрирования счета

# Приложение Д. Блок-схема алгоритма стратегии расходов

# Приложение Е. Скриншот работы приложения

# Приложение Ж. Таблица экономических расчетов

# Приложение И. Листинг package.json

{

"name": "backend",

"version": "1.0.0",

"description": "",

"main": "server.ts",

"scripts": {

"start": "nodemon server.ts",

"build": "tsc -p",

"dev": "nodemon server.ts"

},

"keywords": [],

"author": "Ksu",

"license": "ISC",

"dependencies": {

"@types/dotenv": "^8.2.0",

"cors": "^2.8.5",

"dotenv": "^16.4.5",

"express": "^4.19.2",

"firebase": "^10.6.0",

"fs": "^0.0.1-security",

"mongoose": "^8.0.1",

"morgan": "^1.10.0",

"nodemailer": "^6.9.13",

"ts-node": "^10.9.1",

"typescript": "^5.3.3"

},

"devDependencies": {

"@types/express": "^4.17.21",

"@types/mongoose": "^5.11.97",

"@types/node": "^20.10.5",

"nodemon": "^3.1.0",

"ts-node": "^10.9.1"

}

}

# Приложение К. Листинг transactions.ts

const {addOperation, deleteOperation, getIncomesByAccount, getExpensesByAccount, getAllIncomesByUser, getAllExpensesByUser} = require('../controllers/operationController');

const {addAccount, getAccounts, deleteAccount, updateAccount, getAccountById, getOwnersAccounts, getInvitedAccounts,

addAccountOwner, addAccountInvited, getUserAccounts, getUserDepositsBalance, addCorrespondence, getCorrespondence,

deleteCorrespondence, invite, deleteAccountInvited, acceptInvitation, getEmailsOfInvited} = require('../controllers/accountController');

const {addLimitation, getLimitationsByAccount, deleteLimitation,

addCategory, getCategoriesByAccount, deleteCategory} = require('../controllers/limitationAndCategoryController');

const {addGoal, getGoalsByAccount, deleteGoal, updateGoal} = require('../controllers/goalController');

const router = require('express').Router();

router.get('/get-incomes/:accNum', getIncomesByAccount)

.post('/get-all-user-incomes', getAllIncomesByUser)

.get('/get-expenses/:accNum', getExpensesByAccount)

.post('/get-all-user-expenses', getAllExpensesByUser)

.delete('/delete-operation/:id', deleteOperation)

.post('/add-operation', addOperation)

.get('/get-accounts', getAccounts)

.delete('/delete-account/:id', deleteAccount)

.put('/update-account/:id', updateAccount)

.get('/get-accounts/:id', getAccountById)

.get('/get-user-accounts/:uid', getUserAccounts)

.get('/get-user-deposits/:uid', getUserDepositsBalance)

.get('/get-owner-accounts/:uid', getOwnersAccounts)

.get('/get-invite-accounts/:uid', getInvitedAccounts)

.post('/add-account-owner', addAccountOwner)

.post('/add-account-invited', addAccountInvited)

.post('/get-emails-invited', getEmailsOfInvited)

.get('/get-limitations/:accNum', getLimitationsByAccount)

.delete('/delete-limitation/:id', deleteLimitation)

.post('/add-limitation', addLimitation)

.get('/get-categories/:accNum', getCategoriesByAccount)

.post('/add-category', addCategory)

.delete('/delete-category/:id', deleteCategory)

.get('/get-goal/:accNum', getGoalsByAccount)

.delete('/delete-goal/:id', deleteGoal)

.put('/update-goal/:accNum', updateGoal)

.post('/add-goal', addGoal)

.post('/add-correspondence', addCorrespondence)

.get('/get-correspondence', getCorrespondence)

.delete('/delete-correspondence/:email',deleteCorresponence)

.post('/invite', invite)

.delete('/delete-invitation', deleteAccountInvited)

.get('/accept-invitation', acceptInvitation)

module.exports = router

# Приложение Л. Листинг телеграм-бота

import express, { text } from "express";

import axios from "axios";

import TelegramApi from "node-telegram-bot-api";

import dotenv from 'dotenv';

import moment from 'moment';

import fs from "fs";

import { AssemblyAI } from "assemblyai";

dotenv.config();

const BOT\_KEY = process.env.BOT\_KEY;

const BASE\_URL = process.env.BASE\_URL;

const PORT = process.env.PORT;

const apiKey = process.env.apiKey;

const app = express();

const bot = new TelegramApi(BOT\_KEY, { polling: true });

app.listen(PORT, () => {

console.log(`Express сервер запущен на порту ${PORT}`);

});

let userUid = "";

let email = "";

let activeAccount = "";

let currentStep = "";

let title = "";

let currency = "";

let commandsAllowedWithoutAcc = ["Создать счет", "/start", "Выйти", "/logout", "/choose\_account", "titleAccount", "chooseAccount", "currency"];

let commandsWithAccount = ["Добавить доход", "Удалить доход", "Просмотреть доходы", "Добавить расход", "Удалить расход",

"Просмотреть расходы", "Добавить доход/расход с помощью голосового сообщения"];

bot.setMyCommands([

{

command: "/start",

description: "Добро пожаловать в финансовый помощник! Чтобы продолжить, введите команду /login для входа через почту и пароль."

},

{

command: "/login",

description: "Авторизация"

},

{

command: "/logout",

description: "Выход",

},

{

command: "/choose\_account",

description: "Выбрать счет"

}

])

const optionsWithAccount = {

reply\_markup: JSON.stringify({

keyboard: [

[{ text: "Добавить доход" }, { text: "Удалить доход" }, { text: "Просмотреть доходы" }],

[{ text: "Добавить расход" }, { text: "Удалить расход" }, { text: "Просмотреть расходы" }],

[{ text: "Добавить доход/расход с помощью голосового сообщения" }],

[{ text: "Создать счет" }],

[{ text: "Выйти" }],

],

resize\_keyboard: true,

one\_time\_keyboard: false

})

}

async function getUserAccounts() {

return await axios.get(`${BASE\_URL}get-user-accounts/${userUid}`);

}

async function authorize(password) {

const res = await axios.post(

"https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyDvbtB4EnJuXLkweBELx4Ab1k5DFXK00cQ",

{

clientType: "CLIENT\_TYPE\_WEB",

email: email,

password: password,

returnSecureToken: true,

})

return res;

}

//OPERATIONS

const getOperations = async (type, action, chatId) => {

try {

let res;

bot.sendMessage(chatId, `Последние 15 ${type}ов:`);

if (type === "доход") {

res = await axios.get(`${BASE\_URL}get-incomes/${activeAccount}`);

currentStep = "deleteIncome";

} else if (type === "расход") {

res = await axios.get(`${BASE\_URL}get-expenses/${activeAccount}`);

currentStep = "deleteExpense";

}

if (res.status === 200) {

if (action === "удалить") {

const operations = res.data.slice(0, 15);

const buttons = operations.map(operation => ({

text: `${operation.title} ${operation.amount}${currency} ${moment(operation.date).format("DD-MM-YYYY")}`,

callback\_data: operation.\_id

}));

const replyMarkup = {

reply\_markup: {

inline\_keyboard: buttons.map(button => [{ text: button.text, callback\_data: button.callback\_data }]),

resize\_keyboard: true,

one\_time\_keyboard: true

}

};

bot.sendMessage(chatId, `Выберите ${type}:`, replyMarkup);

} else {

const operations = res.data.slice(0, 15);

operations.forEach(operation => (

bot.sendMessage(chatId, `${operation.title} ${operation.amount} ${currency} ${moment(operation.date).format("DD-MM-YYYY")}`)

))

}

}

} catch (err) {

throw err;

}

}

const addOperation = async (type, title, amount, chatId) => {

const data = {

accNum: activeAccount,

title: title,

amount: amount,

type: type,

email: email,

category: "Другое",

date: new Date()

}

try {

const res = await axios.post(`${BASE\_URL}add-operation`, data);

if (res.status === 200) {

bot.sendMessage(chatId, `${type} был добавлен`);

currentStep = "";

title = "";

amount = "";

}

}

catch (err) {

bot.sendMessage(chatId, `${err.response.data.message}. Попробуйте снова. Введите название расхода.`);

if (type === "Доход") {

currentStep = "titleIncome";

} else if (type === "Расход") currentStep === "titleExpense";

}

}

const deleteOperation = async (type, id, chatId) => {

const res = await axios.delete(`${BASE\_URL}delete-operation/${id}`);

if (res.status === 200) {

bot.sendMessage(chatId, "Доход был удален")

}

}

//ACCOUNT

const addAccountOwner = async (chatId) => {

const data = {

uidOwner: userUid,

name: title,

type: 'Checking',

currency: currency

}

try {

const res = await axios.post(`${BASE\_URL}add-account-owner`, data);

if (res.status === 201) {

bot.sendMessage(chatId, "Счет был создан");

currentStep = "";

title = "";

currency = "";

}

}

catch (err) {

bot.sendMessage(chatId, "Возникла ошибка при создании счета, попробуйте снова. Введите название счета.")

currentStep = "titleAccount";

}

}

const chooseAccount = async (chatId) => {

const accounts = await getUserAccounts(userUid);

const checkingAccounts = accounts.data.filter(account => account.type === 'Checking');

if (checkingAccounts.length === 0) {

bot.sendMessage(chatId, `У вас нет счетов с типом "Расчетный"`);

bot.sendMessage(chatId, "", {

reply\_markup: JSON.stringify({

keyboard: [

[{ text: "Добавить счет" }]

],

resize\_keyboard: true,

one\_time\_keyboard: true

})

})

} else {

const buttons = checkingAccounts.map(account => ({

text: account.name,

callback\_data: `${account.\_id}|${account.currency}`

}));

const replyMarkup = {

reply\_markup: {

inline\_keyboard: [buttons],

resize\_keyboard: true,

one\_time\_keyboard: true

}

};

bot.sendMessage(chatId, "Выберите счет:", replyMarkup);

currentStep = "chooseAccount";

}

}

bot.on("message", async (msg) => {

const chatId = msg.chat.id;

const text = msg.text;

try {

if (userUid === "") {

if (text === "/start") {

bot.sendMessage(chatId, "Добро пожаловать в финансовый помощник! Чтобы продолжить, введите команду /login для входа через почту и пароль.");

} else if (text === "/login") {

currentStep = "email";

bot.sendMessage(chatId, "Введите email");

} else if (currentStep === "email" && !commandsWithAccount.includes(text)) {

email = text;

currentStep = "password";

bot.sendMessage(chatId, "Введите пароль");

} else if (currentStep === "password") {

let password = text;

try {

const res = await authorize(password);

if (res.status === 200) {

userUid = res.data.localId;

bot.sendMessage(chatId, "Аутентификация завершена!", {

reply\_markup: {

inline\_keyboard: [

[{ text: "Получить список счетов", callback\_data: "getAccounts" }]

]

}

});

currentStep = "getAccounts";

}

} catch (error) {

console.log(error);

bot.sendMessage(chatId, "Данные введены некорректно! Заново введите почту:");

currentStep = "email";

}

} else {

bot.sendMessage(chatId, "Добро пожаловать! Чтобы продолжить, введите команду /login для входа через почту и пароль.");

}

} else {

if (text === "/start") {

bot.sendMessage(chatId, "Вы уже авторизованы!");

bot.sendMessage(chatId, "", {

reply\_markup: {

inline\_keyboard: [

[{ text: "Получить список счетов", callback\_data: "getAccounts" }]

]

}

});

}

if (text === "/logout" || text === "Выйти") {

userUid = "";

activeAccount = "";

currentStep = "";

email = "";

bot.sendMessage(chatId, "Вы вышли из аккаунта!");

}

if (text === "/choose\_account") {

chooseAccount(chatId);

}

if (text === "Создать счет") {

if (currentStep !== "titleExpense" && currentStep !== "amountExpense" && currentStep !== "titleAccount") {

currentStep = "titleAccount";

bot.sendMessage(chatId, "Введите название счета");

}

} else if (currentStep === "titleAccount" && !commandsWithAccount.includes(text)) {

title = text;

currentStep = "currency";

bot.sendMessage(chatId, "Выберите валюту", {

reply\_markup: JSON.stringify({

keyboard: [

[{ text: "USD" }, { text: "EUR" }, { text: "BYN" }, { text: "RUB" }]

],

resize\_keyboard: true,

one\_time\_keyboard: true

})

})

} else if (currentStep === "currency" && !commandsWithAccount.includes(text)) {

currency = text;

addAccountOwner(chatId);

}

if (activeAccount !== "") {

let amount = "";

if (text === "Добавить доход") {

if (currentStep !== "titleIncome" && currentStep !== "amountIncome") {

currentStep = "titleIncome";

bot.sendMessage(chatId, "Введите название дохода");

}

} else if (currentStep === "titleIncome" && !commandsWithAccount.includes(text)) {

title = text;

currentStep = "amountIncome";

bot.sendMessage(chatId, "Введите сумму дохода");

} else if (currentStep === "amountIncome" && !commandsWithAccount.includes(text)) {

amount = text;

addOperation("Доход", title, amount, chatId);

}

if (text === "Добавить расход") {

if (currentStep !== "titleExpense" && currentStep !== "amountExpense") {

currentStep = "titleExpense";

bot.sendMessage(chatId, "Введите название расхода");

}

} else if (currentStep === "titleExpense" && !commandsWithAccount.includes(text)) {

title = text;

currentStep = "amount";

bot.sendMessage(chatId, "Введите сумму расхода");

} else if (currentStep === "amount" && !commandsWithAccount.includes(text)) {

amount = text;

addOperation("Расход", title, amount, chatId);

}

if (text === "Просмотреть доходы") {

getOperations("доход", "", chatId);

}

if (text === "Просмотреть расходы") {

getOperations("расход", "", chatId);

}

if (text === "Удалить доход") {

getOperations("доход", "удалить", chatId);

}

if (text === "Удалить расход") {

getOperations("расход", "удалить", chatId);

}

if (text === "Добавить доход/расход с помощью голосового сообщения") {

bot.sendMessage(chatId, 'Для того, чтобы добавить операцию с помощью голосового сообщения, запишите сообщение в формате:\n\nТип операции (Доход/Расход) --- Cумма (целое число или число с запятой) --- Название.\n\nПри этом число можно сказать как "девяносто девять целых одиннадцать сотых", так и "девяносто девять запятая одиннадцать".');

}

} else if (activeAccount === "" && commandsWithAccount.includes(text)) {

bot.sendMessage(chatId, "Прежде чем отправлять команды, выберите счет!", {

reply\_markup: {

inline\_keyboard: [

[{ text: "Получить список счетов", callback\_data: "getAccounts" }]

]

}

});

}

}

} catch (error) {

console.error("Ошибка в обработчике сообщений:", error);

}

});

bot.on("callback\_query", async function (query) {

const chatId = query.message.chat.id;

const [data, accountCurrency] = query.data.split('|');

try {

if (userUid !== "") {

if (data === "getAccounts") {

chooseAccount(chatId);

}

if (currentStep === "chooseAccount" && data !== "getAccounts") {

currency = accountCurrency;

activeAccount = data;

bot.sendMessage(chatId, "Выберите действие: ", optionsWithAccount);

currentStep = "";

}

if (currentStep === "deleteIncome") {

deleteOperation("Доход", data, chatId);

}

if (currentStep === "deleteExpense") {

deleteOperation("Расход", data, chatId);

}

} else {

bot.sendMessage(chatId, "Вы не авторизованы! Чтобы продолжить, введите команду /login для входа через почту и пароль.");

}

} catch (err) {

bot.sendMessage(chatId, "Возникла непредвиденная ошибка...");

}

})

const client = new AssemblyAI({

apiKey: apiKey

})

bot.on('voice', async voice => {

if (activeAccount === "" && currentStep === "getAccounts") {

bot.sendMessage(voice.chat.id, "Прежде чем добавить доход или расход, выберите счет!", {

reply\_markup: {

inline\_keyboard: [

[{ text: "Получить список счетов", callback\_data: "getAccounts" }]

]

}

});

} else if (userUid !== "" && activeAccount !== "") {

try {

let file\_id = voice.voice.file\_id;

let fileName = '';

axios.get(`https://api.telegram.org/bot${BOT\_KEY}/getFile?file\_id=${file\_id}`)

.then(async response => {

const filePath = response.data.result.file\_path;

fileName = filePath.split("/").slice(-1)[0].split(".")[0];

fileName = fileName + file\_id.slice(-5);

const fullUrl = `https://api.telegram.org/file/bot${BOT\_KEY}/${filePath}`;

const writer = fs.createWriteStream(`${fileName}.ogg`);

axios.get(fullUrl, { responseType: 'stream' })

.then(response => {

response.data.pipe(writer);

writer.on('finish', async () => {

const audioUrl = `${fileName}.ogg`

const config = {

audio\_url: audioUrl,

language\_code: 'ru'

}

const transcript = await client.transcripts.create(config)

console.log(transcript.text)

const transcription = transcript.text;

console.log(transcription);

// Удаляем все точки и запятые, кроме случаев, когда запятая используется в числе

const cleanedText = transcription.replace(/[\.,]/g, (match, offset, string) => {

if (match === ',') {

const before = string[offset - 1];

const after = string[offset + 1];

// Если перед и после запятой цифры, заменяем ее на точку

if (/\d/.test(before) && /\d/.test(after)) {

return '.';

}

}

// Удаляем точку или запятую

return '';

}).replace(/\s{2,}/g, ' ');

console.log("Очищенный текст:", cleanedText);

const regex = /^(расход|доход|Расход|Доход)\s+(\d+(?:,\d+)?(?:\.\d+)?)\s+(.+)/;

const matches = cleanedText.match(regex);

if (matches) {

let typeName = matches[1]; // тип (расход или доход)

let amountNum = parseFloat(matches[2].replace(',', '.')); // сумма (преобразовываем запятую в точку)

let name = matches[3]; // название (все слова начиная с третьего)

typeName = typeName.charAt(0).toUpperCase() + typeName.slice(1);

name = name.charAt(0).toUpperCase() + name.slice(1);

bot.sendMessage(voice.chat.id, `Вы сказали:\nтип: '${typeName}', сумма: '${amountNum}', название: '${name}'`);

console.log("Тип:", typeName);

console.log("Сумма:", amountNum);

console.log("Название:", name);

addOperation(typeName, name, amountNum, voice.chat.id);

} else {

bot.sendMessage(voice.chat.id, `Строка не соответствует ожидаемому формату.\nВы сказали "${cleanedText}". Повторите ввод.`);

}

fs.unlinkSync(`${fileName}.ogg`);

});

})

.catch(error => {

console.error(error);

});

})

.catch(error => {

console.error(error);

});

}

catch (error) {

console.log(error);

}

}

}

)

# Приложение М. Листинг UserAuthContext.js

import React, {createContext, useContext, useEffect, useState, ReactNode} from 'react';

import {

UserCredential,

createUserWithEmailAndPassword,

signInWithEmailAndPassword,

onAuthStateChanged,

signOut,

GoogleAuthProvider,

signInWithPopup,

} from 'firebase/auth';

import {auth} from '../firebase';

const userAuthContext = createContext();

export function UserAuthContextProvider({children}) {

const [user, setUser] = useState({});

const [loading, setLoading] = useState(true);

function logIn(email, password) {

return signInWithEmailAndPassword(auth, email, password);

}

function signUp(email, password) {

return createUserWithEmailAndPassword(auth, email, password);

}

function logOut() {

setUser(null);

return signOut(auth);

}

function googleSignIn() {

const googleAuthProvider = new GoogleAuthProvider();

return signInWithPopup(auth, googleAuthProvider);

}

useEffect(() => {

const unsubscribe = onAuthStateChanged(auth, (currentUser) => {

console.log('Auth', currentUser);

setUser(currentUser || null);

setLoading(false);

});

return () => {

unsubscribe();

};

}, []);

return (

<userAuthContext.Provider value={{user, logIn, signUp, logOut, googleSignIn}}>

{children}

</userAuthContext.Provider>

);

}

export function useUserAuth() {

return useContext(userAuthContext);

}

# Приложение Н. Листинг globalContext.js

import React, { useContext, useState } from "react"

import axios from 'axios'

const BASE\_URL = 'http://localhost:8080/api/';

const GlobalContext = React.createContext();

export const GlobalProvider = ({ children, activeAccount }) => {

let [incomes, setIncomes] = useState([])

let [expenses, setExpenses] = useState([])

const [categories, setCategories] = useState([])

const [limitations, setLimitations] = useState([])

const [accountInfo, setAccountInfo] = useState(null)

const [goal, setGoal] = useState(null)

const [error, setError] = useState(null)

const [emails, setEmails] = useState([])

const [userAccounts, setUserAccounts] = useState([])

const [ownerAccounts, setOwnerAccounts] = useState([])

const [successMessage, setSuccessMessage] = useState('')

let [balanceUSD, setBalanceUSD] = useState(null)

let [balanceEUR, setBalanceEUR] = useState(null)

let [balanceRUB, setBalanceRUB] = useState(null)

let [balanceBYN, setBalanceBYN] = useState(null)

//INCOMES

const getIncomes = async (activeAccount) => {

const response = await axios.get(`${BASE\_URL}get-incomes/${activeAccount}`)

setIncomes(response.data)

}

const addIncome = async (income, uid) => {

await axios.post(`${BASE\_URL}add-operation`, income)

.catch((err) => {

setError(err.response.data.message)

});

await getIncomes(activeAccount)

await getUserBalance(uid);

}

const deleteIncome = async (id) => {

await axios.delete(`${BASE\_URL}delete-operation/${id}`);

await getIncomes(activeAccount)

}

const totalIncomes = () => {

let totalIncome = 0;

incomes.forEach((income) => {

totalIncome = totalIncome + income.amount

})

return parseFloat(totalIncome.toFixed(2).replace(/\.?0+$/, ''));

}

//EXPENSES

const getExpenses = async (activeAccount) => {

const response = await axios.get(`${BASE\_URL}get-expenses/${activeAccount}`)

setExpenses(response.data)

}

const addExpense = async (expense) => {

await axios.post(`${BASE\_URL}add-operation`, expense)

.catch((err) => {

setError(err.response.data.message)

});

await getExpenses(activeAccount)

}

const deleteExpense = async (id) => {

await axios.delete(`${BASE\_URL}delete-operation/${id}`);

await getExpenses(activeAccount)

}

const totalExpenses = () => {

let totalExpense = 0;

expenses.forEach((expense) => {

totalExpense = totalExpense + expense.amount

})

return parseFloat(totalExpense.toFixed(2).replace(/\.?0+$/, ''));

}

//ACCOUNT

const getAccountInfo = async (activeAccount) => {

const response = await axios.get(`${BASE\_URL}get-accounts/${activeAccount}`);

setAccountInfo(response.data);

}

const updateAccountInfo = async (activeAccount, newAmount) => {

try {

const response = await axios.put(

`${BASE\_URL}update-account/${activeAccount}`,

{ amount: newAmount }

);

setAccountInfo(response.data);

} catch (error) {

console.error('Ошибка при обновлении счета:', error);

}

}

const getUserAccounts = async (uid) => {

const response = await axios.get(`${BASE\_URL}get-user-accounts/${uid}`);

setUserAccounts(response.data);

}

const addAccountOwner = async (data, uid) => {

await axios.post(`${BASE\_URL}add-account-owner`, data);

await getUserAccounts(uid);

}

const deleteAccount = async (id, uid) => {

await axios.delete(`${BASE\_URL}delete-account/${id}`)

await getUserAccounts(uid);

}

const updateAccount = async (id, name, uid) => {

try {

await axios.put(`${BASE\_URL}update-account/${id}`, { name });

await getUserAccounts(uid);

} catch (error) {

console.error('Ошибка при обновлении счета:', error);

}

}

//CORRESPONDENCE

const getOrSetCorrespondence = async (email, uid) => {

try {

const response = await axios.get(`${BASE\_URL}get-correspondence?email=${email}`);

} catch (error) {

if (error.response && error.response.status === 202) {

await axios.post(`${BASE\_URL}add-correspondence`, { email, uid })

} else {

console.error('Произошла ошибка: ', error.message);

}

}

}

//FINANCIAL GOAL

const getGoal = async (activeAccount) => {

const response = await axios.get(`${BASE\_URL}get-goal/${activeAccount}`)

setGoal(response.data)

}

const addGoal = async (goal) => {

await axios.post(`${BASE\_URL}add-goal`, goal)

.catch((err) => {

setError(err.response.data.message)

});

await getGoal(activeAccount)

}

const updateGoal = async (activeAccount, newAmount) => {

await axios.put(`${BASE\_URL}update-goal/${activeAccount}`, { goalAmount: newAmount })

.catch((err) => {

setError(err.response.data.message)

});

await getGoal(activeAccount)

}

const deleteGoal = async (id) => {

await axios.delete(`${BASE\_URL}delete-goal/${id}`);

await getGoal(activeAccount)

}

//BALANCE & HISTORY

const totalBalance = () => {

return (totalIncomes() - totalExpenses()).toFixed(2).replace(/\.?0+$/, '');

}

const transactionHistory = () => {

const history = [...incomes, ...expenses]

history.sort((a, b) => {

return new Date(b.date) - new Date(a.date)

})

return history.slice(0, 5)

}

const getUserBalance = async (uid) => {

try {

const accountsResponse = await axios.get(`http://localhost:8080/api/get-user-accounts/${uid}`);

const accounts = accountsResponse.data;

const accountDetails = accounts.map(account => {

return {

account\_id: account.\_id,

currency: account.currency

};

});

const depositsResponse = await axios.get(`http://localhost:8080/api/get-user-deposits/${uid}`);

const deposits = depositsResponse.data;

const incomesResponse = await axios.post('http://localhost:8080/api/get-all-user-incomes', { accountDetails });

const incomes = incomesResponse.data;

const expensesResponse = await axios.post('http://localhost:8080/api/get-all-user-expenses', { accountDetails });

const expenses = expensesResponse.data;

let balanceByCurrency = {};

for (const currency in incomes) {

if (incomes.hasOwnProperty(currency)) {

if (!balanceByCurrency.hasOwnProperty(currency)) {

balanceByCurrency[currency] = 0;

}

balanceByCurrency[currency] += incomes[currency];

}

}

for (const currency in expenses) {

if (expenses.hasOwnProperty(currency)) {

if (!balanceByCurrency.hasOwnProperty(currency)) {

balanceByCurrency[currency] = 0;

}

balanceByCurrency[currency] -= expenses[currency];

}

}

for (const currency in deposits) {

if (deposits.hasOwnProperty(currency)) {

if (!balanceByCurrency.hasOwnProperty(currency)) {

balanceByCurrency[currency] = 0;

}

balanceByCurrency[currency] += deposits[currency];

}

}

console.log(balanceByCurrency);

if (balanceByCurrency.hasOwnProperty('BYN')) {

setBalanceBYN(balanceByCurrency.BYN);

}

if (balanceByCurrency.hasOwnProperty('USD')) {

setBalanceUSD(balanceByCurrency.USD);

}

if (balanceByCurrency.hasOwnProperty('RUB')) {

setBalanceRUB(balanceByCurrency.RUB);

}

if (balanceByCurrency.hasOwnProperty('EUR')) {

setBalanceEUR(balanceByCurrency.EUR);

}

console.log(balanceBYN);

console.log(balanceUSD);

console.log(balanceRUB);

console.log(balanceEUR);

} catch (error) {

console.error('Ошибка при получении данных:', error);

}

}

//CATEGORIES

const getCategories = async (activeAccount) => {

const response = await axios.get(`${BASE\_URL}get-categories/${activeAccount}`)

setCategories(response.data)

}

const addCategory = async (category) => {

await axios.post(`${BASE\_URL}add-category`, category)

.catch((err) => {

setError(err.response.data.message)

});

await getCategories(activeAccount)

}

const deleteCategory = async (id) => {

await axios.delete(`${BASE\_URL}delete-category/${id}`);

await getCategories(activeAccount)

}

//LIMITATIONS

const getLimitations = async (activeAccount) => {

const response = await axios.get(`${BASE\_URL}get-limitations/${activeAccount}`)

setLimitations(response.data)

}

const addLimitation = async (limitation) => {

await axios.post(`${BASE\_URL}add-limitation`, limitation)

.catch((err) => {

setError(err.response.data.message)

});

await getLimitations(activeAccount)

}

const deleteLimitation = async (id) => {

await axios.delete(`${BASE\_URL}delete-limitation/${id}`);

await getLimitations(activeAccount)

}

//INVITES

const invite = async (email, accNum) => {

const res = await axios.post(`${BASE\_URL}invite`, { email, accNum }).catch((err) => {

setError(err.response.data.message);

return err;

});

if (res && res.status === 200) setSuccessMessage(res.data.message);

}

const getEmailsInvited = async (data) => {

const response = await axios.post(`${BASE\_URL}get-emails-invited`, data);

if (response.status === 200) {

setEmails(response.data);

console.log(response.data);

}

}

const deleteInvitation = async (data) => {

const res = await axios.delete(`${BASE\_URL}delete-invitation`, {

data: data

}).catch((err) => {

setError(err.response.data.message);

return err;

});

if (res && res.status === 200) setSuccessMessage(res.data.message);

await getEmailsInvited(data);

}

const getOwnerAccounts = async (uid) => {

const res = await axios.get(`${BASE\_URL}get-owner-accounts/${uid}`).catch((err) => {

setError(err.response.data.message)

});

setOwnerAccounts(res.data);

}

return (

<GlobalContext.Provider value={{

addIncome,

getIncomes,

incomes,

deleteIncome,

expenses,

totalIncomes,

addExpense,

getExpenses,

deleteExpense,

totalExpenses,

totalBalance,

transactionHistory,

categories,

addCategory,

getCategories,

deleteCategory,

limitations,

addLimitation,

getLimitations,

deleteLimitation,

accountInfo,

getAccountInfo,

updateAccountInfo,

goal,

addGoal,

getGoal,

updateGoal,

deleteGoal,

getEmailsInvited,

emails,

deleteInvitation,

getUserAccounts,

userAccounts,

getOrSetCorrespondence,

getUserBalance,

balanceBYN,

balanceRUB,

balanceUSD,

balanceEUR,

addAccountOwner,

deleteAccount,

invite,

updateAccount,

getOwnerAccounts,

ownerAccounts,

successMessage,

setSuccessMessage,

error,

setError

}}>

{children}

</GlobalContext.Provider>

)

}

export const useGlobalContext = () => {

return useContext(GlobalContext)

}

# Приложение П. Листинг navigation.js

import React, { useEffect, useState } from 'react';

import { BrowserRouter, Navigate, Route, Routes } from 'react-router-dom';

import { Login } from '../components/Login/Login';

import { Register } from '../components/Register/Register';

import { UserAuthContextProvider } from '../context/UserAuthContext.jsx';

import { LeftBlock } from '../components/LeftBlock/LeftBlock';

import { ProtectedRouteAuthorized } from '../components/ProtectedRoutes/ProtectedRouteAuthorized';

import { ProtectedRoute } from '../components/ProtectedRoutes/ProtectedRoute';

import { Expenses } from '../components/Expenses/Expenses';

import { Dashboard } from '../components/Dashboard/Dashboard';

import { Incomes } from '../components/Incomes/Incomes';

import { MainLayout } from '../assets/styles/Layouts';

import { GlobalProvider, useGlobalContext } from '../context/globalContext';

import { Limitations } from '../components/Limitations/Limitations';

import { Savings } from '../components/Savings/Savings';

import { Categories } from '../components/Categories/Categories';

export const Navigation = () => {

const [active, setActive] = useState(null);

const [activeAccount, setActiveAccount] = useState(null);

const global = useGlobalContext()

useEffect(() => {

setActive(null);

}, [activeAccount]);

let displayData = () => {

if (activeAccount !== null) {

switch (active) {

case 1:

return <Dashboard activeAccount={activeAccount} />

case 2:

return <Incomes activeAccount={activeAccount} />

case 3:

return <Expenses activeAccount={activeAccount} />

case 4:

return <Savings activeAccount={activeAccount} />

case 5:

return <Categories activeAccount={activeAccount.\_id} />

case 6:

return <Limitations activeAccount={activeAccount} />

}

} else {

return (

<div style={{ top: '45%', left: '15%', position: 'absolute' }}>

<h5>Добро пожаловать! Добавьте или выберите счёт, чтобы продолжить.</h5>

</div>

)

}

}

return (

<BrowserRouter>

<UserAuthContextProvider>

{/\*<Header/>\*/}

<Routes>

{/\* ALL USERS\*/}

<Route path='/home' element={

<ProtectedRoute>

<MainLayout>

<GlobalProvider>

<LeftBlock active={active} setActive={setActive}

activeAccount={activeAccount} setActiveAccount={setActiveAccount} />

</GlobalProvider>

<GlobalProvider activeAccount={activeAccount !== null ? activeAccount.\_id : undefined}>

<main>

{displayData()}

</main>

</GlobalProvider>

</MainLayout>

</ProtectedRoute>

} />

{/\* CHECK\*/}

<Route path='/login' element={

<ProtectedRouteAuthorized>

<Login />

</ProtectedRouteAuthorized>

} />

<Route path='/register' element={

<ProtectedRouteAuthorized>

<Register />

</ProtectedRouteAuthorized>

} />

<Route path='\*' element={<Navigate to='/home' replace />} />

</Routes>

</UserAuthContextProvider>

</BrowserRouter>

)

}