МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

Допустить к защите
Заведующий кафедрой
канд. физ.-мат. наук, доц.
_____ В.В. Подколзин
(подпись)
_____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(БАКАЛАВРСКАЯ РАБОТА)

РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ ДЛЯ УЧЕТА ЛИЧНЫХ ФИНАНСОВ

Работу выполнил _____ К.А. Шабанов
(подпись)

Направление подготовки 09.03.03 Прикладная информатика

Направленность Прикладная информатика в экономике

Научный руководитель
канд. техн. наук, доц._____ С.Г. Синица
(подпись)

Нормоконтролер
ст. преп. _____ А.В. Харченко
(подпись)

Краснодар
2022

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(БАКАЛАВРСКАЯ РАБОТА)

РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ ДЛЯ УЧЕТА ЛИЧНЫХ ФИНАНСОВ

Работу выполнил _____ К.А. Шабанов
(подпись)

Направление подготовки 09.03.03 Прикладная информатика

Направленность Прикладная информатика в экономике

Научный руководитель
канд. техн. наук, доц._____ С.Г. Синица
(подпись)

Нормоконтролер
ст. преп. _____ А.В. Харченко
(подпись)

Краснодар
2022

РЕФЕРАТ

Выпускная квалификационная работа 45 страниц, 35 рисунков, 7 источников.

ФИНАНСЫ, УПРАВЛЕНИЕ, ИНСТРУМЕНТЫ, ЗАПИСИ, КАТЕГОРИИ, ПЛАНИРОВАНИЕ, UI, WEB ПРИЛОЖЕНИЕ

В данной работе изучен теоретический аспект управления персональными финансами, разобраны существующие инструменты для этой задачи, а также описано создание собственного инструмента на основе полученных знаний.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Финансы – это один из основных ресурсов для человека. Чтобы добиваться определенных результатов в жизни, надо уметь управлять ими, ведь мы все живем в рыночной системе, мы постоянно обмениваем свои деньги на товары или услуги, а затем работаем, обменивая свое время обратно на деньги и так по кругу. Люди, которые не умеют грамотно управлять своими финансами, через десятки лет работы не имеют никаких накоплений.

Чтобы понимать, как управлять – нужно быть финансово грамотным, а чтобы управлять, нужны соответствующие инструменты.

Современные проблемы требуют современных решений, поэтому цель данной курсовой работы – создать современный инструмент для управления персональными финансами.

Задачи:

– изучить теоретические аспекты управления персональными финансами;

– проанализировать имеющиеся инструменты для управления персональными финансами;

– создать свой инструмент управления персональными финансами.

Объект исследования – управление персональными финансами.

Предмет исследования – инструменты для управления персональными финансами.

Итог проделанной работы – web приложение для управления персональными финансами.

1 Теоретические аспекты управления персональными финансами

1.1 Финансовые цели

Управление персональными финансами должно начинаться с постановки финансовой цели. К финансовым целям нужно относить то, что реально измерить в деньгах. Но ставить цели недостаточно, потребуется разработать план, по которому надо будет двигаться. Здесь нужно сразу же считать:

- какое количество средств готовы ежемесячно откладывать;
- за какой период хочется достичь цели.

По срокам цели делятся на три типа [2]:

- краткосрочные (до 1 года);
- среднесрочные (до 5 лет);
- долгосрочные (от 5 лет).

Человеку свойственно распыляться, что выступает основой для уменьшения вероятности достижения всех целей. Поэтому нужно в процессе создания списка целей вспомнить, что именно является более важным для вас.

Среднесрочные и долгосрочные рекомендуется делить на этапы, это существенно упрощает процесс. Важно понимать, что накопить 5млн. рублей на дом за 5 лет – это абстрактная цель. Поскольку в один момент можно потерять мотивацию. Разбивайте цели на годы. В результате чего можно долгосрочные цели зафиксировать в виде краткосрочных.

Кажется, все просто, но на практике большинство допускают ошибки. Помним, что держать в голове цели не нужно, их всегда нужно записывать.

Также, нужно здраво оценивать свои возможности и бюджет. Планировать покупку самолета при зарплате даже 200 тыс. рублей ежемесячно – безрассудно. Не стоит вгонять себя в очень жесткие рамки, тем самым теряя заинтересованность.

Но какой бы точной ни была цель, ее не получиться добиться без финансовой дисциплины, надо работать над собой и не искать отговорки из месяца в месяц.

После того, как вы начнете фиксировать свои доходы и расходы, желательно, регулярно проводить их анализ, чтобы понимать, где можно уменьшить расходы и куда направить доходы.

1.2 Основные правила финансовой грамотности

Какой бы ни была цель или срок, общие принципы остаются неизменными:

- отслеживай свои доходы и расходы;
- планируй покупки, не совершай спонтанные решения;
- трать меньше, чем зарабатываешь;
- собери финансовую подушку в размере стандартных трат за месяц * ~6;
- погаси все задолженности;
- после выполнения двух предыдущих пунктов, покупай активы, чтобы деньги работали на тебя.

Исходя из них и стоит проектировать инструмент для управления личными финансами.

2 Анализ имеющихся инструментов для управления персональными финансами

Существует множество программ для управления личными финансами, у всех приблизительно один базовый функционал. Самые важные показатели, на которые ориентируются пользователи – это доп. функционал, UI/UX, кроссплатформенность и стоимость. Рассмотрим два приложения с отличиями в данных критериях.

2.1 Monefy

Monefy [6] – мобильное приложение для ios и android.

Функционал:

- категории;
- записи по категориям;
- счета в различных валютах;
- графическое представление истории записей;
- выбор временного промежутка для отображения истории;
- синхронизация через облака.

Однако, часть вышеперечисленных функций доступна только при покупке платной версии. Из недостатков можно выделить:

- отсутствие синхронизации между приложениями на разных платформах без подключения облаков;
- отсутствие синхронизации с банковскими картами.

Стоит отметить отличный UI/UX (приложение А) и активную поддержку от разработчиков.

Рейтинг в AppStore – 4.8 при 4807 отзывах.

2.2 DREBE DENGi

DREBE DENGi [7] работает на мобильных (ios, android, windows), десктопных (windows) платформах, есть web версия, а также бот в Telegram.

Функционал:

- категории;
- автоматический ввод трат, обработка СМС от банков;
- сканирование чеков по QR коду;
- планируемые операции с напоминаниями;
- список покупок для похода в магазин;
- отчёты для анализа расходов;
- мультивалютность, автоматическое обновление курс;
- еженедельная отправка бэкапов по Email;
- семейный бюджет.

Однако, часть функционала доступна только с платной версией.

Недостатки:

- устаревший UI;
- отсутствие графического представления истории;
- отсутствие тех поддержки. Последнее обновление более 3 лет назад.

Т. к. в приложении есть баги, обновления не выпускаются, а цена от 599р в год, то рейтинг в AppStore низкий – 3.3 при 74 отзывах. Пользователи жалуются на неработающие функции, которые как раз являются дополнительными, относительно базового функционала подобных приложений. Т. е. приложение хромает по все критериям, важным для многих современных пользователей (дизайн, стоимость, доп. функции (есть, но не всегда работают)).

2.3 Вывод

Оба рассмотренных приложения пригодны к использованию.

Monefy предоставляет относительно небольшой функционал, однако реализация и удобство использования на высоком уровне, что способствует быстрому освоению программы. Но мало поддерживаемых платформ, нет синхронизации с банковскими картами.

DREBE DENGi обладает сравнительно широким функционалом, но дизайн устаревший, нет графического представления истории записей, платная версия дорогая, а также отсутствует поддержка разработчиков, хотя баги, влияющие на функциональность, есть.

Мое приложение будет реализовано web технологиями, что сделает его более доступным, чем Monefy, т. к. к нему можно будет получить доступ независимо от платформы. Отображение расходов будет не только в виде списка, как в DREBE DENGi, но и в виде диаграммы, чтобы было проще воспринимать и анализировать информацию. А весь интерфейс будет в современном и минималистичном Material дизайне. Еще одним плюсом станет абсолютная бесплатность для пользователя, чем не могут похвастать вышеупомянутые конкуренты.

3 Программная реализация клиентской части web приложения для учета личных финансов

3.1 Описание бизнес-процесса

Опишем бизнес-процесс, руководствуясь [3].

Бизнес-процесс рассматриваемой задачи заключается в управлении личными финансами пользователя:

– добавление записей;

– постановка финансовых целей;

– просмотр истории записей;

– отслеживание курса валют.

Входные и выходные данные представим на схеме «черный ящик» (рисунок 1).

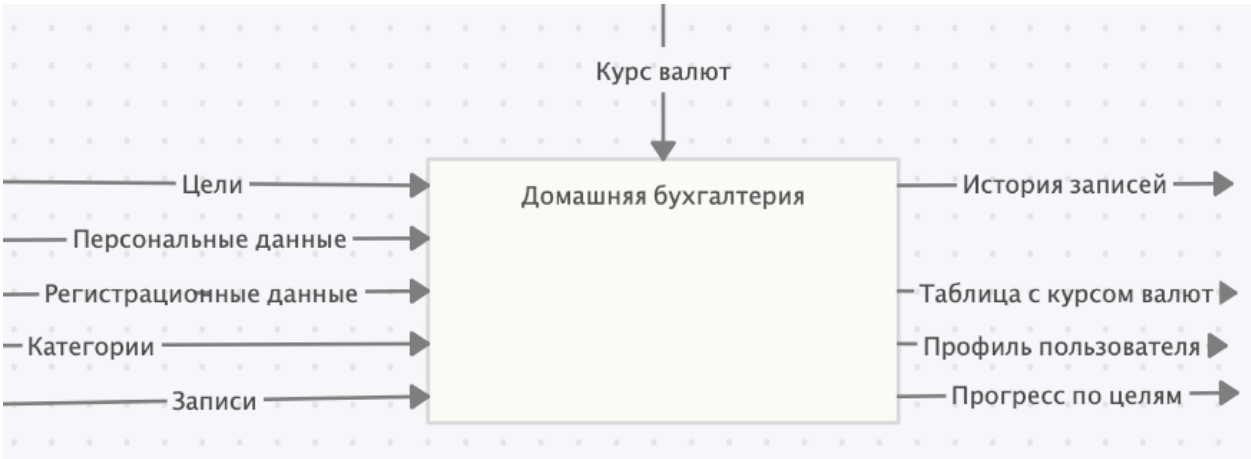

Рисунок 1 – Схема «черный ящик» бизнес-процесса

Чтобы понимать, как обрабатываются эти данные, надо разбить бизнес-процесс на подпроцессы и задачи (рисунок 2).

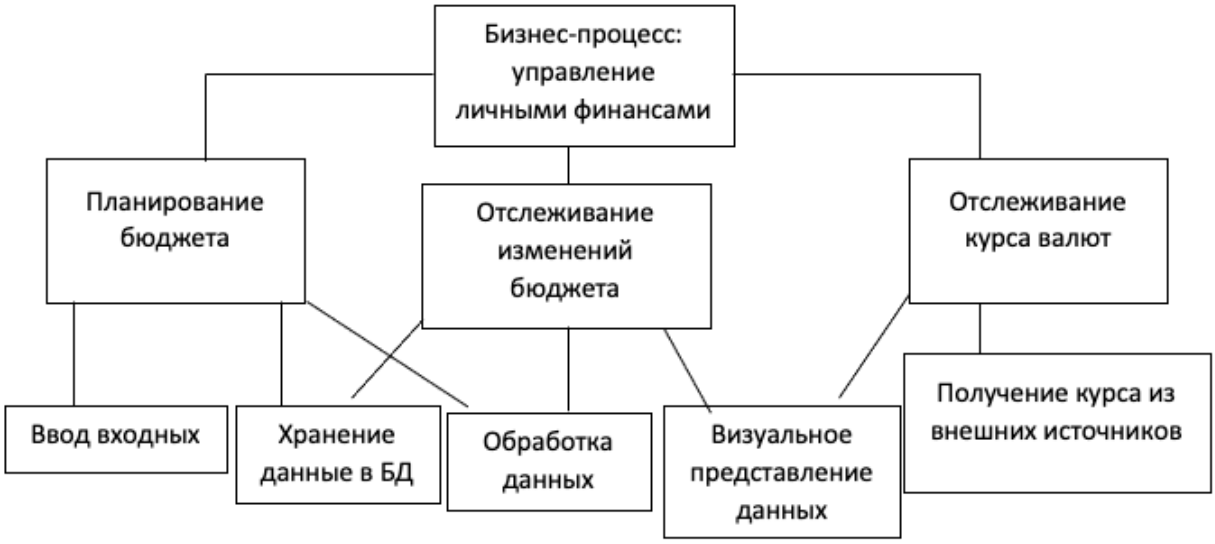

Рисунок 2 – Декомпозиция бизнес-процесса

Обозначим варианты использования будущего приложения с помощью UML диаграммы (рисунок 3).

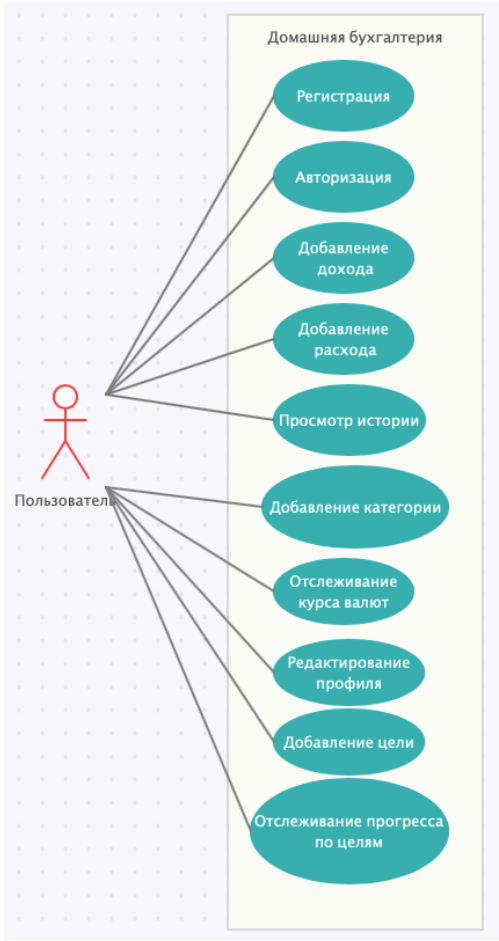

Рисунок 3 – UML диаграмма

3.2 Описание реализации

Исходный код можно найти на GitLab [4].

3.2.1 Используемые технологии

Приложение написано с использованием Vue фреймворка Nuxt.js [1] и является Single Page Application (SPA). Т. е. загружается один раз, а далее работает исключительно на JavaScript, делая только ajax запросы. Такой подход позволяет пользоваться приложением без перезагрузки страниц, хотя начальная загрузка длится немного дольше.

Используется библиотека компонентов Vuetify для минимизации времени верстки и тестирования. Компоненты данной библиотеки выполнены в стиле Material Design, а также имеют простой и стабильный API. В сравнении с библиотекой Materialize, для верстки на Vuetify понадобилось почти вдвое меньше кода. Для форматирования даты используется Nuxt модуль date-fns, вместо кастомных фильтров, как было ранее. Для ajax запросов – axios.

Весь код, который раньше был на JavaScript (JS), теперь переписан на TypeScript (TS). Строгая типизация позволяет допускать меньше ошибок и предотвращать некоторые потенциальные баги.

Во Vue <script> теперь для большего удобства используется class-api также на TS.

Добавлена mobile версия и Progressive Web Application (PWA) для более комфортного использования на mobile устройствах. PWA делается в Nuxt всего одним объектом в конфигурационном файле nuxt.config.js

```
pwa: {
    …
},
```

3.2.2 Структура проекта

Nuxt позволяет забыть про ручную настройку роутов, достаточно поместить `*.vue` файлы в каталог `pages` и Nuxt сам превратит их в страницы.

Похожая ситуация с store, Vuex по дефолту настроен и остается только создавать файлы в каталоге `store`, и они превратятся в модули Vuex.

И наконец layouts, по дефолту для всей страниц будет использовать `default.vue` из `layouts/`, а для страниц с ошибками `error.vue`. Можно добавлять новые Vue файлы в этот каталог и просто прописывать свойство в компонент страницы, чтобы применить новый layout.

Также, благодаря опции `components` в nuxt.config.js, можно забыть про ручной импорт компонентов, теперь они импортируются во `*.vue` файлы автоматические.

Все эти базовые настройки Nuxt позволяют экономить кучу времени на настройку и отладку базовых вещей.

В папке filters лежат фильтры, для преобразования входных значений.

Все Ajax запросы вынесены в каталог `services`, это позволяет переиспользовать запросы в любой части проекта и легко отлаживать их.

Во Vue в одном файле компонента находится его html шаблон, логика на TS и Cascading Style Sheets (CSS) стили, что делает компонентный подход очень удобным. При правильном написании компонентов их легко можно переиспользовать. А чистую JS логику можно выносить в mixins, делая ее максимально абстрактной, для возможности подключения к различным компонентам. При подключении к компоненту, миксин, как бы, встраивается в него, становясь одним целым, позволяя внутри компонента использовать логику, написанную в миксине.

Скриншот структуры проекта см. в приложении В.

3.2.4 Интерфейс программы

В данной версии реализованы как desktop, так и mobile версии приложения. Таблица с историей записей теперь более гибкая и функциональная. Пользователю доступна сортировка по любому из столбцов, а также редактирование количества отображаемых записей на одной страницы пагинации. Также теперь для просмотра описания записи не нужно переходить на новую страницу, достаточно просто раскрыть строку таблицы.

Также в разделе «Планирование» теперь отображается % выполнения цели.

Была переделана форма авторизации, теперь на одной страницы есть карточка с табуляцией, где в первой вкладке форма логина, во второй – форма регистрации. Это позволяет юзеру быстрее и удобнее переключаться между формами. Также добавлена возможность скрывать и показывать пароль, а для исключения введения некорректного пароля при регистрации добавлено поле подтверждения пароля.

В sidebar присутствуют следующие вкладки (приложение Г):

- главная – баланс пользователя и курс валют;
- история – история записей и графическим представлением;
- планирование – категории и их лимиты, а также прогресс;
- новая запись – создание дохода/расхода;
- категории – создание и редактирование категорий.

3.2.5 Общение с сервером

Для получения курса валют происходит обращение к API Fixer:

```
async get({ symbols = 'USD,EUR,RUB' } = {}) {
  return await axiosClient.get(
    'http://data.fixer.io/api/latest',
```

```
    {
      params: {
        access_key: process.env.fixerKey,
        // base: 'RUB', // не доступно в бесплатной версии
        symbols
      }
    }
    )
} // получаем данные по доллару, евро, рублю
```

```
const query = gql`
mutation myAdd {
  addCategory(      You, 1 second ago • Uncommitted changes
    profileId: ${profileId},
    name: "${title}",
    limit: "${limit}"
  ) {
    id
    name
    limit
  }
}
`;

const response = await this.$graphql.default.request(query);
```

Рисунок 4 – GraphQL запрос

4 Программная реализация серверной части web приложения для учета личных финансов

Исходный код можно найти на GitLab [5].

4.1 Используемые технологии

4.1.1 Python + Django

Из множества языков программирования, подходящих для серверной разработки, был выбран язык программирования общего назначения Python. Это объясняется тем, что он достаточно простой в изучении, но при этом обладает обширными возможностями. Он является одним из самых популярных языков для серверной разработки, что обуславливает большое количество доступных библиотек и фреймворков.

Один из доступных фреймворков для языка Python – это Django. Как его называют авторы, «фреймворк для перфекционистов с горящими сроками». Подразумевается то, что он является крайне обширным, и в нем реализовано «из коробки» огромное количество различных библиотек и инструментов для быстрого создания больших сайтов и веб-приложений.

Django содержит огромное количество функциональности для решения большинства задач веб-разработки. Вот некоторые из высокоуровневых возможностей Django, которые нужно искать отдельно при использовании других фреймворков:

– Object-relation mapper, ORM – это паттерн, который предназначен для написания запросов в объектном стиле, который переводится в SQL средствами фреймворка. Как правило, это сильно упрощает и ускоряет разработку.

– Миграции базы данных – то есть формальная схема, которая позволяет хранить историю изменения базы данных и безопасно применять

эти изменения с помощью программного кода без необходимости разбираться с деталями реализации

– Аутентификация пользователя – это одна из самых важных деталей современного веб-приложения, и она тоже реализована изначально, что позволяет избавиться от большого количества лишней работы.

– Панель администратора – считает одним из главных плюсов фреймворка, она позволяет в удобной форме иметь доступ к базе данных из панели с гибким редактированием прав доступа на чтение и запись различных полей и моделей.

– Формы – это функционал, который позволяет прототипировать веб-приложения без полноценного фронтенда за счет указания связей между моделями и формами. Это дает возможность быстро создавать гибкие формы с валидацией на стороне сервера и с возможностью автоматически применять изменения в форме на базу данных

Также в Django выделяют следующие существенные плюсы:

– Стандартизированная структура – Django как фреймворк задаёт структуру проекта. Она помогает разработчикам понимать, где и как добавлять новую функциональность. Это позволяет даже новичкам очень быстро входить в ход дела и реализовывать веб-приложения с широким функционалом. Благодаря одинаковой для всех проектов структуре гораздо проще найти уже готовые решения или получить помощь от сообщества. Огромное количество увлеченных разработчиков поможет справиться с любой задачей гораздо быстрее.

– Приложения Django позволяют разделить проект на несколько частей, чтобы они не разрастались и не превращались в огромный устаревший массив кода, который как правило называют «большой ком грязи». При необходимости эти приложения легко разделять на различные серверные приложения, если между ними правильно установлены границы. Приложения устанавливаются путём добавления в settings.INSTALLED_APPS. Этот подход

позволяет легко интегрировать готовые решения, что говорит об очень высокой расширяемости и масштабировании этого фреймворка.

– Безопасный по умолчанию – фреймворк включает механизмы предотвращения распространенных атак вроде SQL-инъекций (XSS) и подделки межсайтовых запросов (CSRF). Подробнее об этом можно почитать в официальном руководстве по безопасности.

– REST Framework для создания API – Django REST Framework, который часто сокращают до «DRF», является библиотекой для построения API. Он имеет модульную и настраиваемую архитектуру, которая хорошо работает для создания как простых, так и сложных API.

Панель администратора выглядит следующим образом:

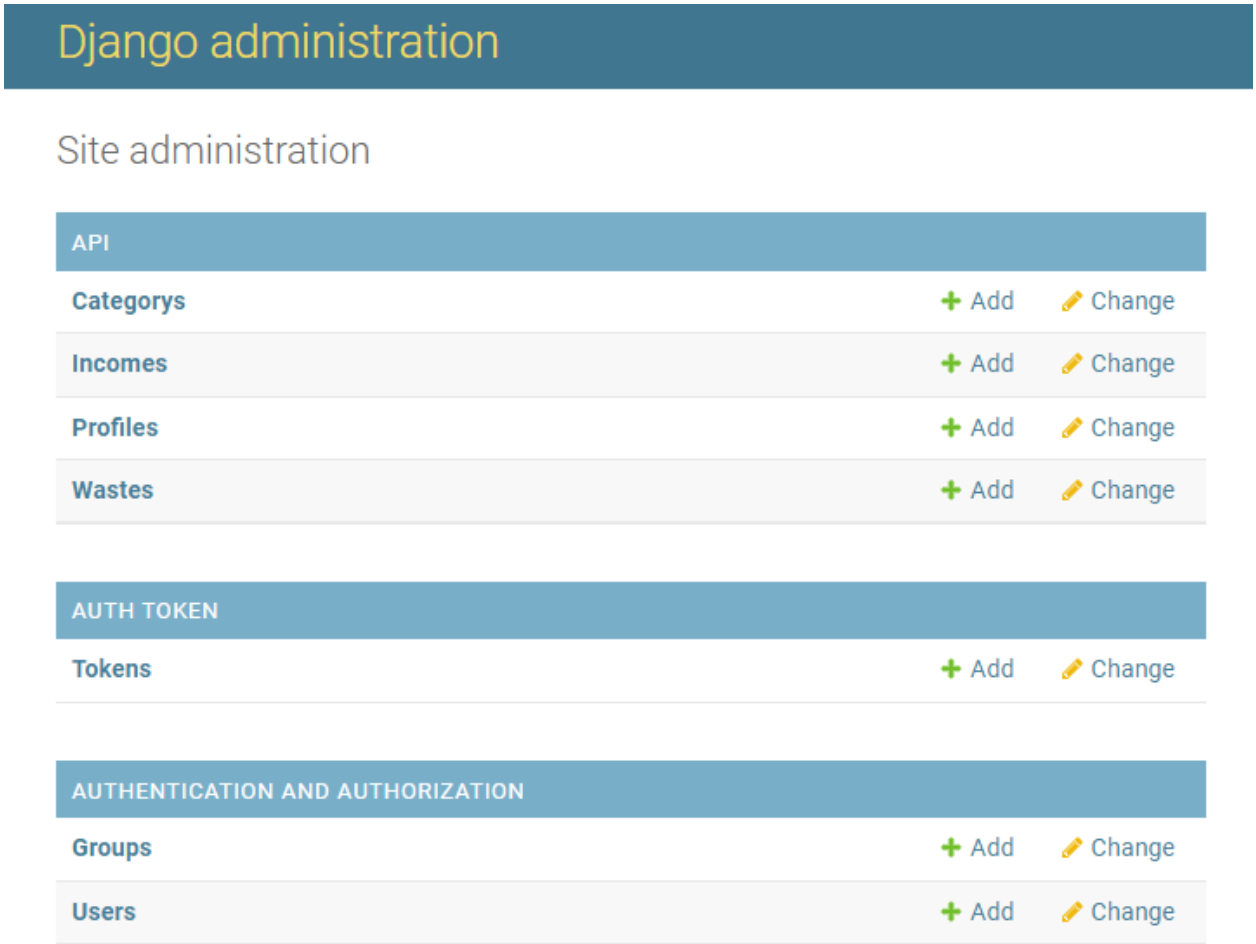

Рисунок 4 – Панель администратора Django

Как видно, здесь отображены все модели приложения, и есть возможность их добавлять и редактировать.

Но, конечно, все это невозможно без крупных недостатков:

– Django ORM – является не только плюсом, но и минусом. Будучи удобным, он имеет несколько существенных изъянов. Главный из них – отсутствие асинхронности даже в последней существующей версии Django. В этом аспекте основной конкурент, SQLAlchemy одерживает безоговорочную победу. Также у них есть разница в архитектуре: Django ORM основан на шаблоне Active Record, который менее гибкий и масштабируемый, чем шаблон Unit of Work, используемый в SQLAlchemy. На практике это выражается в том, что в Django модели могут «сохранять» себя по желанию, а транзакции отключены по умолчанию. Однако, этот недостаток заметен лишь при создании огромных информационных систем. В данной работе этот недостаток не создает никаких проблем.

– Django развивается медленно – это большой и монолитный фреймворк, что с одной стороны позволяет сообществу разрабатывать сотни универсальных модулей и приложений, но с другой стороны снижает скорость разработки самого Django. Кроме того, фреймворк должен поддерживать обратную совместимость, что тоже снижает темпы развития. Этот недостаток тоже проявляется только при разработке огромных систем.

Таким образом, сохраняется баланс преимуществ и недостатков, но в данной работе становится возможным использовать максимум от преимуществ и не испытывать никаких неудобств от недостатков. Поэтому в итоге был выбран именно Django.

Альтернативные легкие фреймворки типа Flask и FastAPI, хотя и позволяют быть свободнее Django в экосистеме и конфигурации, могут потребовать лишнего времени на поиск/создание дополнительных библиотек и функциональных возможностей в долгосрочной перспективе. Было решено

не переусложнять разработку и остановиться на варианте, который предоставляет весь необходимый функционал.

К тому же стабильность Django и сообщество вокруг него сильно выросли с первого релиза. Официальная документация и учебные пособия по фреймворку являются одними из лучших в своём роде.

4.1.2 GraphQL

GraphQL — это язык запросов, используемый клиентскими приложениями для работы с данными. С GraphQL связано такое понятие, как «схема» — это то, что позволяет организовывать создание, чтение, обновление и удаление данных в вашем приложении (то есть — четыре базовые функции, используемые при работе с хранилищами данных, которые обычно обозначают акронимом CRUD — create, read, update, delete).

Выше было сказано, что GraphQL используется для работы с данными в приложении, а не конкретно в базе данных. Дело в том, что GraphQL — это система, независимая от источников данных, то есть, для организации её работы неважно — где именно хранятся данные. Поэтому она совместима с любым способом их хранения.

Гибкость — это то, что отличает технологию GraphQL от широко известной технологии REST. При использовании REST, если всё сделано правильно, конечные точки обычно создают с учётом особенностей некоего ресурса или типа данных приложения.

Как было указано выше, GraphQL для передачи данных клиенту и получения их от него, полагается на простые GET или POST-запросы. Если подробнее рассмотреть эту мысль, то оказывается, что в GraphQL есть два вида запросов. К первому виду относятся запросы на чтение данных, которые в терминологии GraphQL называются просто запросами (query) и относятся к букве R (reading, чтение) акронима CRUD. Запросы второго вида — это

запросы на изменение данных, которые в GraphQL называют мутациями (mutation). Они относятся к буксам C, U и D акронима CRUD, то есть — с их помощью выполняют операции создания (create), обновления (update) и удаления (delete) записей. Все эти запросы и мутации отправляют на URL GraphQL-сервера, который, например, может выглядеть как https://myapp.com/graphql, в виде GET или POST-запросов.

4.2 Описание реализации

4.2.1 Django

В первую очередь необходимо указать модели данных, которые используется в приложении. Для этого нужно создать классы, которые наследуются от класса django.models.Model и указать поля, которые используются для доступа к данных и их редактирования.

```
class Profile(models.Model):
    """Профиль пользователя"""

    user = models.OneToOneField('auth.User', verbose_name='Пользователь-владелец профиля', on_delete=models.CASCADE)
    bill = models.DecimalField('Счет пользователя в рублях', decimal_places=2, default=0, max_digits=20)
    name = models.CharField('Имя пользователя', max_length=100)

    def __str__(self):
        return self.name
```

Рисунок 5 – Модель профиля пользователя

```python
class Category(models.Model):
    """Категория расходов, определяется пользователем"""

    name = models.CharField('Название категории', max_length=100)
    profile = models.ForeignKey(
        'api.Profile',
        verbose_name='Профиль, к которому привязана категория расходов',
        on_delete=models.CASCADE,
    )
    limit = models.DecimalField(
        'Ограничения на траты в категории',
        null=True,
        default=None,
        decimal_places=2,
        max_digits=20,
    )
    date_created = models.DateField('Дата создания категории', auto_now=True)

    def __str__(self):
        return self.name
```

Рисунок 6 – Модель категории расходов

```python
class Waste(models.Model):
    """Трата"""

    name = models.CharField('На что были потрачены деньги', max_length=200)
    amount = models.DecimalField('Количество потраченных денег', decimal_places=2, max_digits=20)
    category = models.ForeignKey('api.Category', verbose_name='Категория расходов', on_delete=models.PROTECT)
    datetime_created = models.DateTimeField('Время траты денег', auto_now=True)

    def __str__(self):
        return self.name


class Income(models.Model):
    """Доход"""

    name = models.CharField('Откуда были получены деньги', max_length=200)
    amount = models.DecimalField('Количество полученныь денег', decimal_places=2, max_digits=20)
    category = models.ForeignKey('api.Category', verbose_name='Категория доходов', on_delete=models.PROTECT)
    datetime_created = models.DateTimeField('Время получения денег', auto_now=True)

    def __str__(self):
        return self.name
```

Рисунок 7 – Модели расходов и доходов

Для создания и применения этих изменений к базе данных необходимо прописать `python manage.py makemigrations`, что создаст файлы миграций. Затем для их применения необходимо прописать в консоли `python manage.py

migrate`. Также эта строка должна быть указана в скрипте приготовления приложения к работе после внесения любых изменений.

Для авторизации и регистрации использовался упомянутый Django Rest Framework, который расширяет возможности Django и позволяет создавать запросы в стиле REST API и получать\возвращать запросы/ответы в формате JSON, который де-факто является общепринятым современным стандартом для передачи данных. Он используется гораздо чаще, чем, например, XML, и при этом является гораздо более удобным.

Для авторизации используется токен доступа. То есть, при регистрации создается временный токен, который позволяет серверу распознавать пользователя, как авторизованного. Этот токен добавляется в заголовок каждого запрос и выглядит как `Authorization: token xxx`. У него ограниченное время жизни, и при его окончании необходимо получить новый токен, указав логин и пароль.

Авторизация по токену является общепринятым способом авторизации, которые обеспечивает достаточную гибкость разработки и при этом является быстрым и безопасным.

4.2.2 Django + GraphQL

Для реализации запросов была использована библиотека graphene-django. Она позволяет интегрировать GraphQL в Django совместно использовать преимущество обоих технологий.

Сначала ее нужно установить с помощью менеджера зависимостей pip, затем ее можно использовать.

После этого нужно добавить graphene в список установленных «приложений» Django. Так выглядит список установленных приложений после добавления graphene:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'api',
    'graphene_django',
    'rest_framework',
    'rest_framework.authtoken',
    'corsheaders',
]
```

Рисунок 8 – Установленные приложения в Django

Как было упомянуто, GraphQL требует для работы указания схемы. В случае graphene-django в первую очередь необходимо привязать к GraphQL модели. Это позволит использовать все их возможности, не прописывая это в коде напрямую. Конечно, при более сложных и комплексных ситуациях все-таки придется указать, какие поля можно читать, указывать дополнительные поля. Однако, в данной работе в этом просто нет необходимости, потому что в базе данных нет служебных полей, все данные предназначены напрямую для пользователя. Поэтому можно позволить клиентскому приложению получить полный доступ ко всем таблицам и полям базы данных.

Делается это следующим образом:

```python
from graphene_django import DjangoObjectType

from api.models import Profile, Category, Waste, Income


class ProfileNode(DjangoObjectType):
    class Meta:
        model = Profile


class CategoryNode(DjangoObjectType):
    class Meta:
        model = Category


class WasteNode(DjangoObjectType):
    class Meta:
        model = Waste


class IncomeNode(DjangoObjectType):
    class Meta:
        model = Income
```

Рисунок 9 – Привязка моделей

Следующим шагом является указание, какие запросы на чтение может отправлять клиент серверному приложению с GraphQL. Эта часть схемы называется Query.

Одним из главных преимуществ библиотеки graphene-django является возможность получать напрямую доступ к пользовательскому запросу. Это объект класса Django.http.HttpRequest. Django при авторизации добавляет в запрос ссылку на пользователя. Это позволяет удобно фильтровать данные в Query и разграничивать доступы.

Например, в данном приложении добавлена фильтрация, которая не позволяет пользователям увидеть чужие данные. В этом смысле GraphQL обладает не меньшей гибкостью, чем стандартные средства Django.

```python
from django.http import HttpRequest
import graphene

from api.models import Profile, Category, Waste, Income
from api.schema.nodes import ProfileNode, CategoryNode, WasteNode, IncomeNode


class Query(graphene.ObjectType):
    """Описание запросов и типов данных"""
    profile = graphene.List(ProfileNode)
    categories = graphene.List(CategoryNode)
    wastes = graphene.List(WasteNode)
    incomes = graphene.List(IncomeNode)

    def resolve_categories(self, info: HttpRequest):
        user = info.context.user
        return Category.objects.filter(profile__user=user)

    def resolve_profile(self, info: HttpRequest):
        user = info.context.user
        return Profile.objects.filter(user=user)

    def resolve_wastes(self, info: HttpRequest):
        user = info.context.user
        return Waste.objects.all(category__profile__user=user)

    def resolve_incomes(self, info: HttpRequest):
        user = info.context.user
        return Income.objects.all(category__profile__user=user)
```

Рисунок 10 – Query-часть схемы GraphQL

Далее необходимо описать данные, которые доступны для изменения при помощи запросов в GraphQL. Эта часть называется Mutation. Также, как и Query, она позволяет ограничивать доступ пользователя к чужим данным.

Так как редактирование данных гораздо более чувствительная операция, требуется написать больше кода, чем для доступов на чтение данных. Для каждой операции надо прописать команды на изменение, что является не очень удобным способом разработки, однако позволяет гарантировать корректность доступов и предохраняет от несанкционированных запросов на изменение данных.

Например, так выглядит код для операций:

```
class CategoryMutation:
    add_category = graphene.Field(
        CategoryNode,
        name=graphene.String(required=True),
        limit=graphene.Decimal(required=False),
        profile_id=graphene.Int(required=True),
    )
    remove_category = graphene.Field(graphene.Boolean, category_id=graphene.ID())
    set_limit = graphene.Field(CategoryNode, category_id=graphene.ID(), limit=graphene.Decimal(required=True))
    set_category_name = graphene.Field(CategoryNode, category_id=graphene.ID(), name=graphene.String(required=True))

    def resolve_add_category(self, info: HttpRequest, name: str, profile_id: int, limit: Optional[Decimal] = None):
        """Описание мутации добавления новой категории"""
        profile = Profile.objects.get(pk=profile_id, user=info.context.user)
        return Category.objects.create(profile=profile, name=name, limit=limit)

    def resolve_remove_category(self, info: HttpRequest, category_id: int):
        """Мутация для удаления категории"""
        try:
            Category.objects.get(id=category_id, profile__user=info.context.user).delete()
        except Category.DoesNotExist:
            return False
        return True

    def resolve_set_limit(self, info: HttpRequest, category_id: int, limit: Decimal):
        """Мутация, которая позволяет установить лимит расходов на категорию"""
        category = Category.objects.get(pk=category_id, profile__user=info.context.user)
        category.limit = limit
        category.save()
        return category

    def resolve_set_category_name(self, info: HttpRequest, category_id: int, name: str):
        """Мутация для изменения имени категории"""
        category = Category.objects.get(pk=category_id, profile__user=info.context.user)
        category.name = name
        category.save()
        return category
```

Рисунок 11 – Операции изменения данных категории расходов

Далее необходимо создать объект, который ссылается на Query и Mutation части схемы:

```
import graphene

from api.schema.query import Query
from api.schema.mutation import Mutation

schema = graphene.Schema(query=Query, mutation=Mutation)
```

Рисунок 12 – Связывание Query и Mutation

После этого нужно указать в настройках Django путь к объекту Schema, который был создан ранее. Это делается довольно просто:

```
GRAPHENE = {
    'SCHEMA': 'api.schema.schema',
}
```

Рисунок 13 – Путь к схеме в настройках Django

4.2.3 Интерактивная консоль для проверки запросов

Graphene-django включает в себя приложения graphiql, который позволяет создавать интерактивную консоль с графическим интерфейсом для отлаживания и проверки запросов в GraphQL. В настройках указывается, в каких условиях она доступна. Как правило, ее оставляют на тестовых серверах, но скрывают на серверах, которые доступны пользователям. Или же, просто ограничивают доступ, чтобы ее видел только пользователь с административными правами.

Так выглядит консоль graphiql:

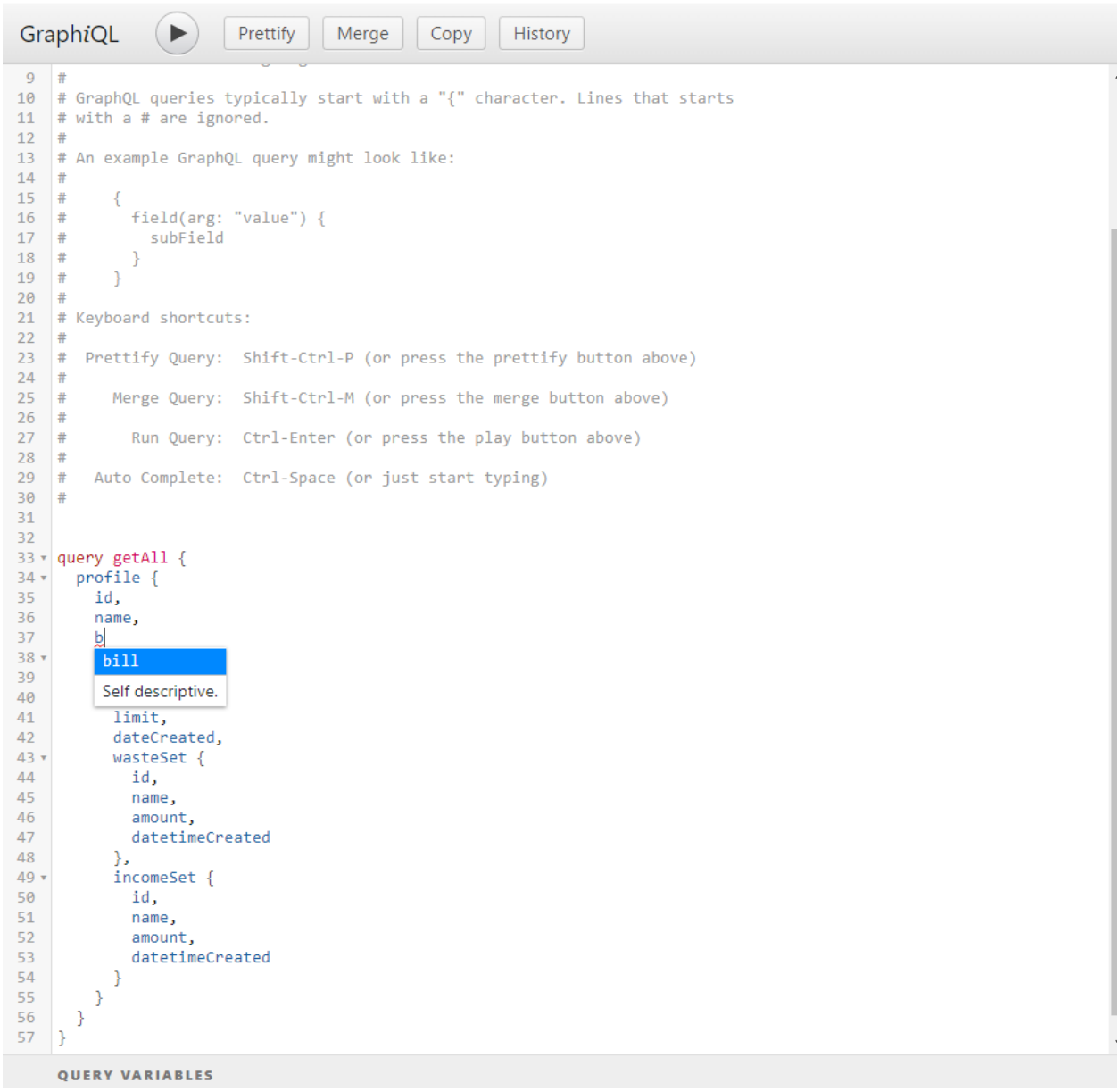

Рисунок 14 – Консоль GraphQL

Как видно на скриншоте, графический интерфейс обеспечивает подсказки и автоматические дополнение текста. Это сильно упрощает работу с graphql и позволяет разработчикам клиентской стороны приложения узнавать доступные им поля и операции в интерактивном режиме, не тратя время на взаимодействие с разработчиками серверной части приложения.

Так выглядит автодополнение с указанием доступных мутаций:

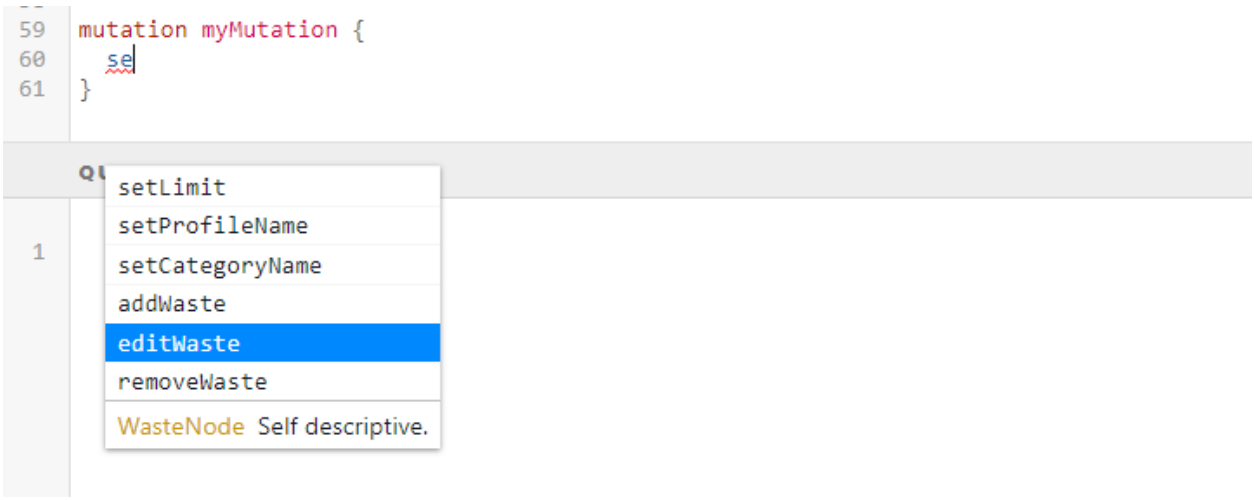

Рисунок 15 – Автодополнение мутация

Также консоль сразу указывает, если запрос является некорректным, подчеркивая красным фрагменты запроса, которые нужно поправить:

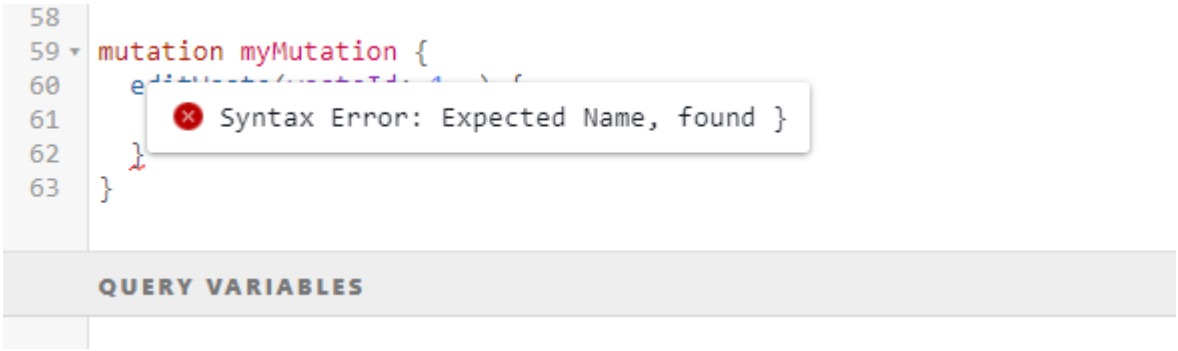

Рисунок 16 – Валидация запроса на ходу

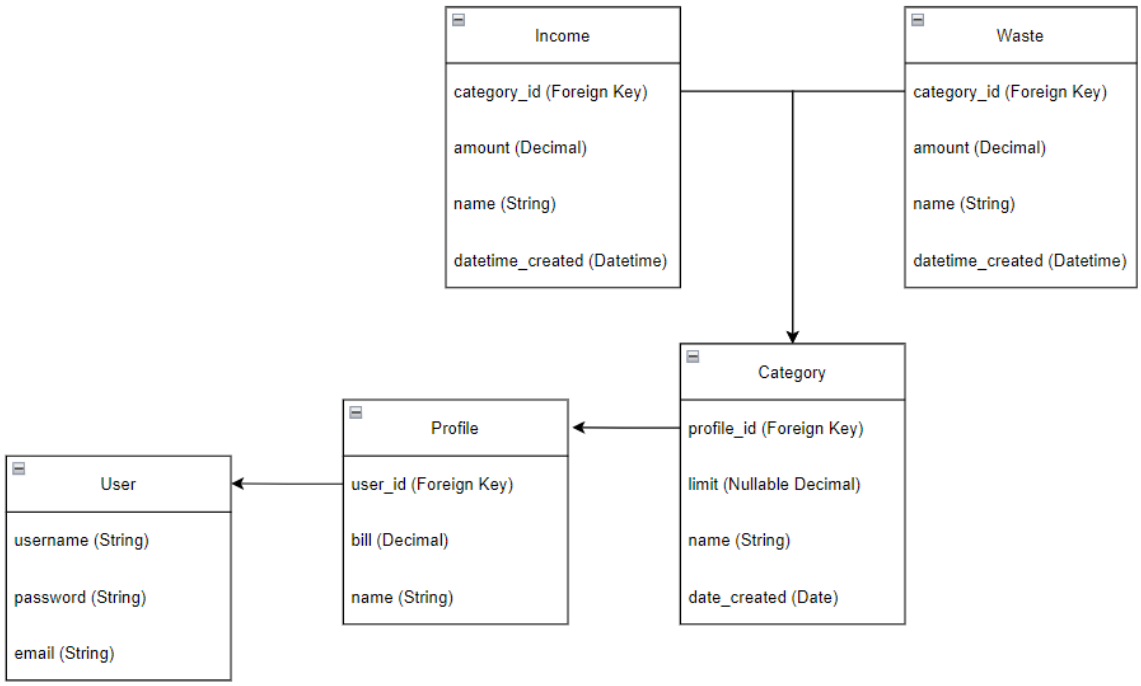

Рисунок 17 – Схема БД

Описание схемы БД:

1) Income – доход, внешний ключ ссылается на id категории
    а) Amount – сумма операции
    б) Name – название операции
    в) Datetime_created – дата операции
2) Waste – расход, поля аналогичны Income
3) Category – категория доходов или расходов, внешний ключ ссылается на id профиля
    а) Name – название
    б) Limit – ограничение расходов
    в) Datetime_created – дата создания
4) Profile – профиль пользователя, внешний ключ ссылается на id пользователя
    а) Bill – общее количество денег на счете

б) Name – имя

5) User – пользователь

а) Username – имя пользователя

б) Password – пароль

в) Email – электронная почта

ЗАКЛЮЧЕНИЕ

В 1-й главе мы выяснили, что необходимо для управления личными финансами, во 2-й проанализировали существующие инструменты, а в 3-й описали создание нового инструмента, на основе знаний, полученных в предыдущих главах.

Т. к. получилось web приложение, доступ к нему можно получить, независимо от платформы, что делает его более доступным, чем Monefy. А отображение расходов не только в виде списка, как в DREBE DENGi, но и в виде диаграммы (рисунок 5) позволяет проще воспринимать и анализировать информацию.

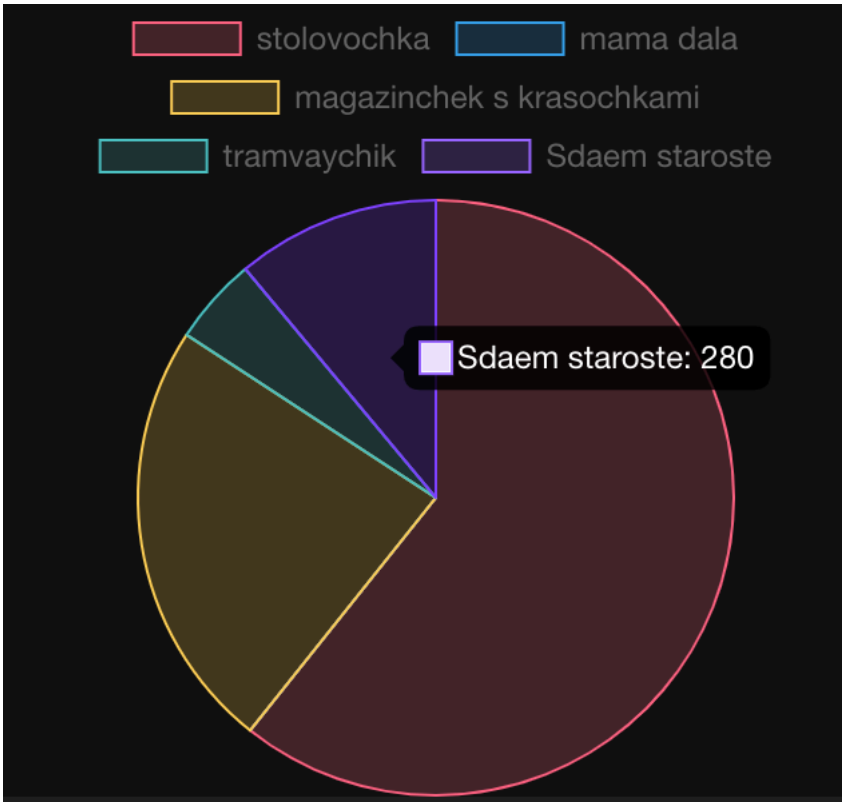

Рисунок 5 – История расходов в виде диаграммы

Также мое приложение является абсолютно бесплатным, чего не скажешь о вышеупомянутых конкурентах.

Приложение уже готово к использованию и способно удовлетворить базовые потребности пользователя.

Благодаря современному стеку, хорошей организации кода, качественным библиотекам и фреймворкам, проект легко расширяется и поддерживается, что упрощает его дальнейшее развитие.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Nuxt – The Intuitive Vue Framework // Nuxt.js : [сайт].– URL: https://nuxtjs.org/ (дата обращения 28.08.2021).

2 Финансовые цели: как правильно ставить и достигать? Личные финансы // InvestFuture: [сайт]. – URL: https://investfuture.ru/edu/articles/finansovye-tseli-kak-pravilno-stavit-i-dostigat-lichnye-finansy (дата обращения 24.04.2022).

3 Полетайкин А.Н. Социальные и экономические информационные системы: законы функционирования и принципы построения – Новосибирск.: СибГУТИ, 2016. – 241 с.

4 Репозиторий клиентской части // GitLab : [сайт]. – URL: https://gitlab.com/k.shabanov/moneyk

5 Репозиторий серверной части // GitLab : [сайт]. – URL: https://gitlab.com/k.shabanov/moneyk_back

6 Monefy : учет расходов / разработчик Aimbity AS. 2020г. : [сайт]. – URL: https://apps.apple.com/ru/app/monefy-учет-расходов/id1212024409 (дата обращения 01.05.2022).

7 Дребеденьги : приложение для учёта финансов / разработчик Alexander Evdokimov. 2017г. : [сайт]. – URL: https://apps.apple.com/ru/app/drebeden-gi/id571913431?ls=1 (дата обращения 05.05.2022).

ПРИЛОЖЕНИЕ А

UI «Monefy»

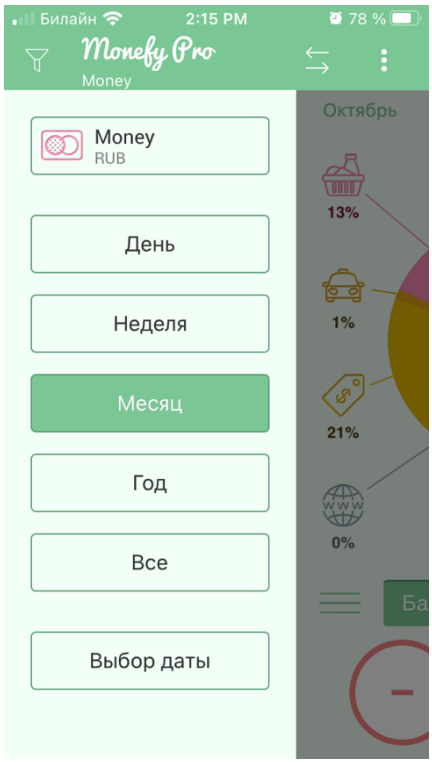

Рисунок А.1 – Временные промежутки

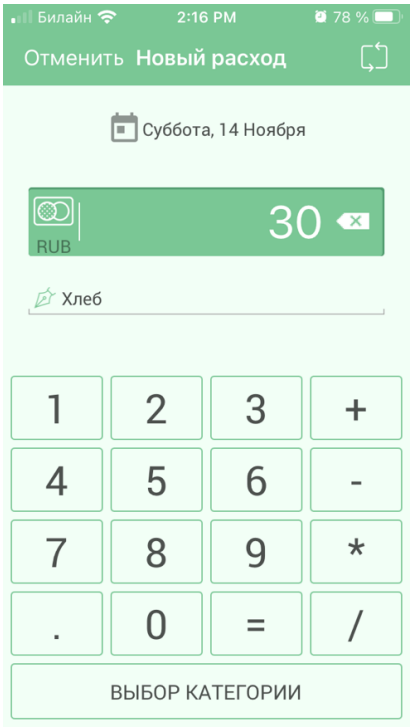

Рисунок А.2 – Добавление записи

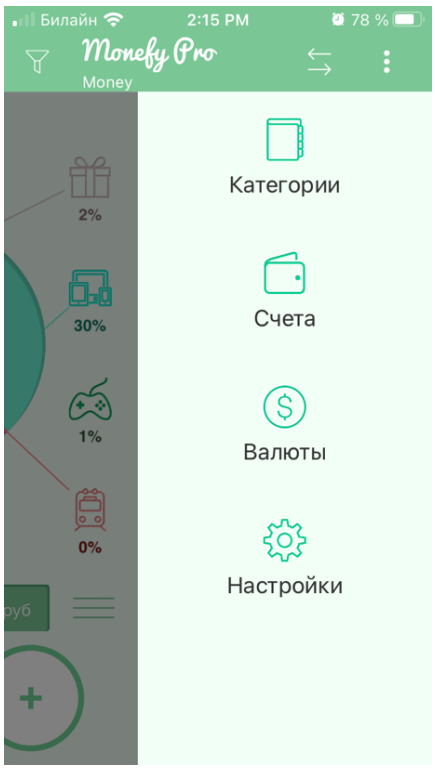

Рисунок А.3 – Настройки

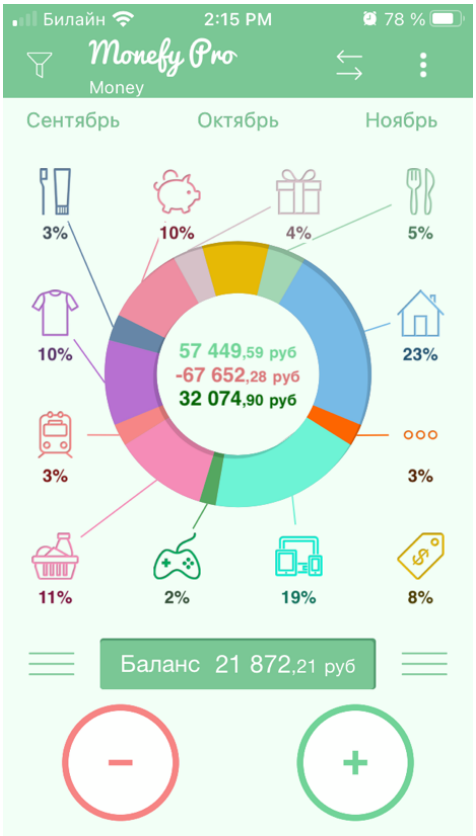

Рисунок А.4 – Отображение баланса

ПРИЛОЖЕНИЕ Б
UI «DREBE DENGi»

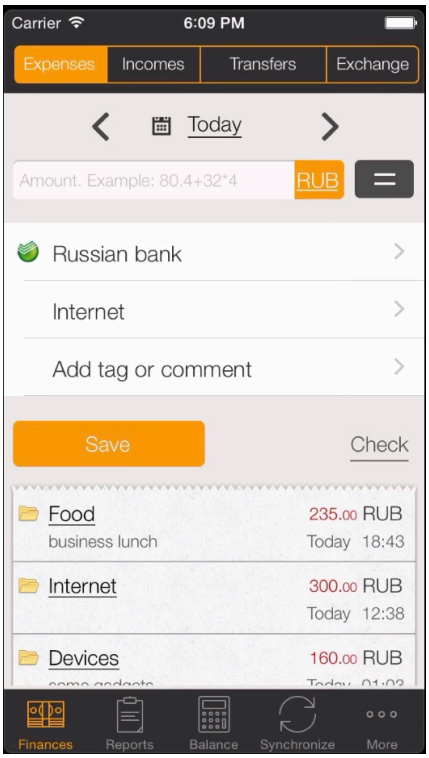

Рисунок Б.1 – Основная информация о счете

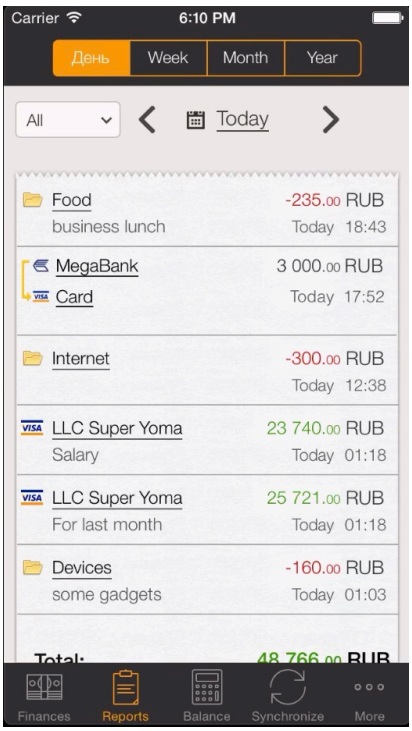

Рисунок Б.2 – Отчеты по категориям

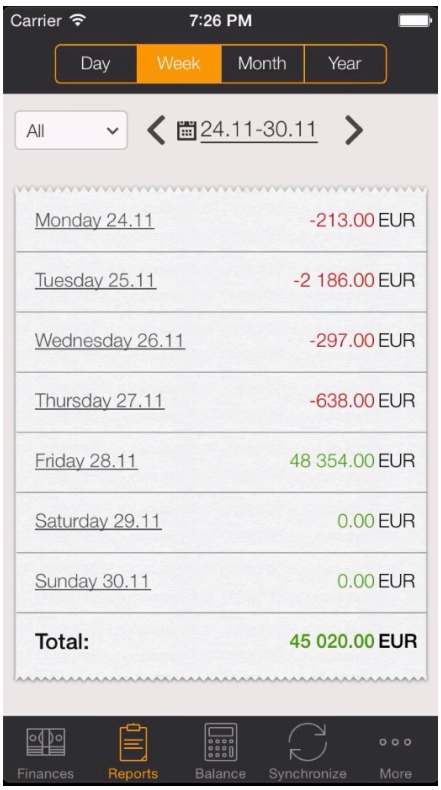

Рисунок Б.3 – Отчеты по дням

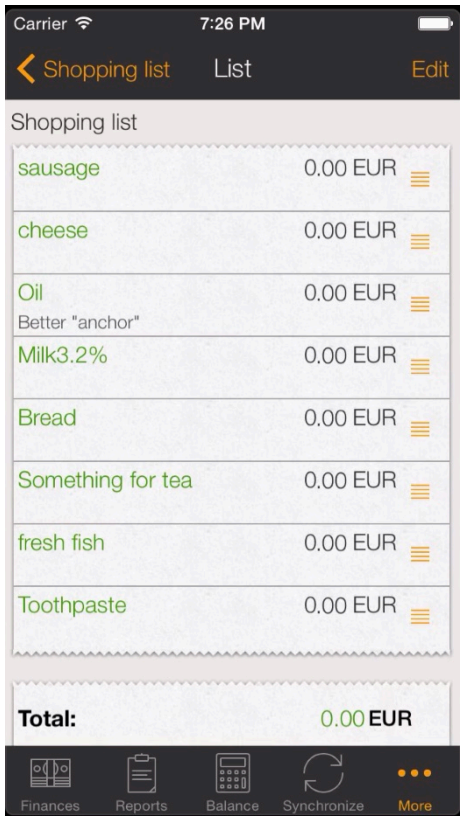

Рисунок Б.4 – Список покупок

ПРИЛОЖЕНИЕ В

Структура проекта

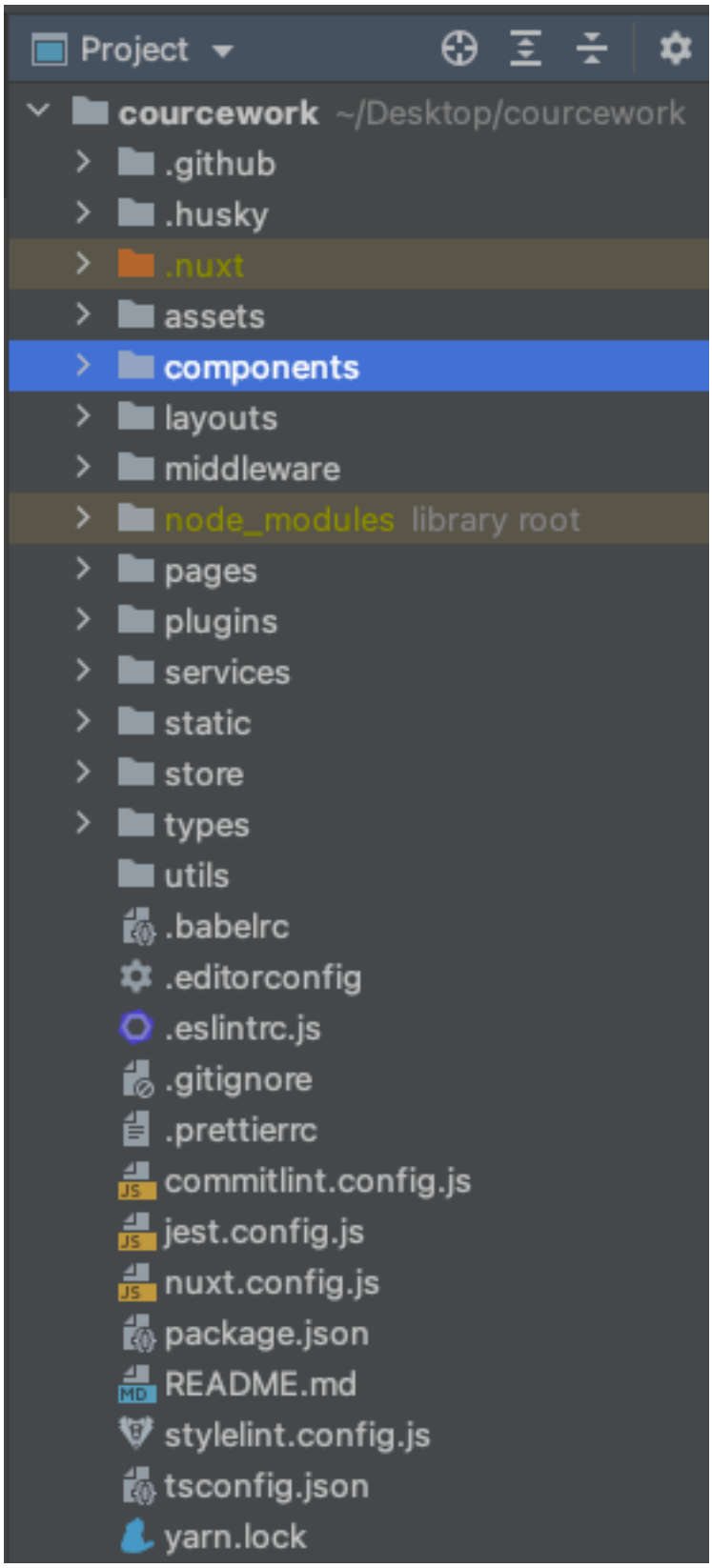

Рисунок В.1 – Структура проекта «MoneyK»

ПРИЛОЖЕНИЕ Г

UI «MoneyK»

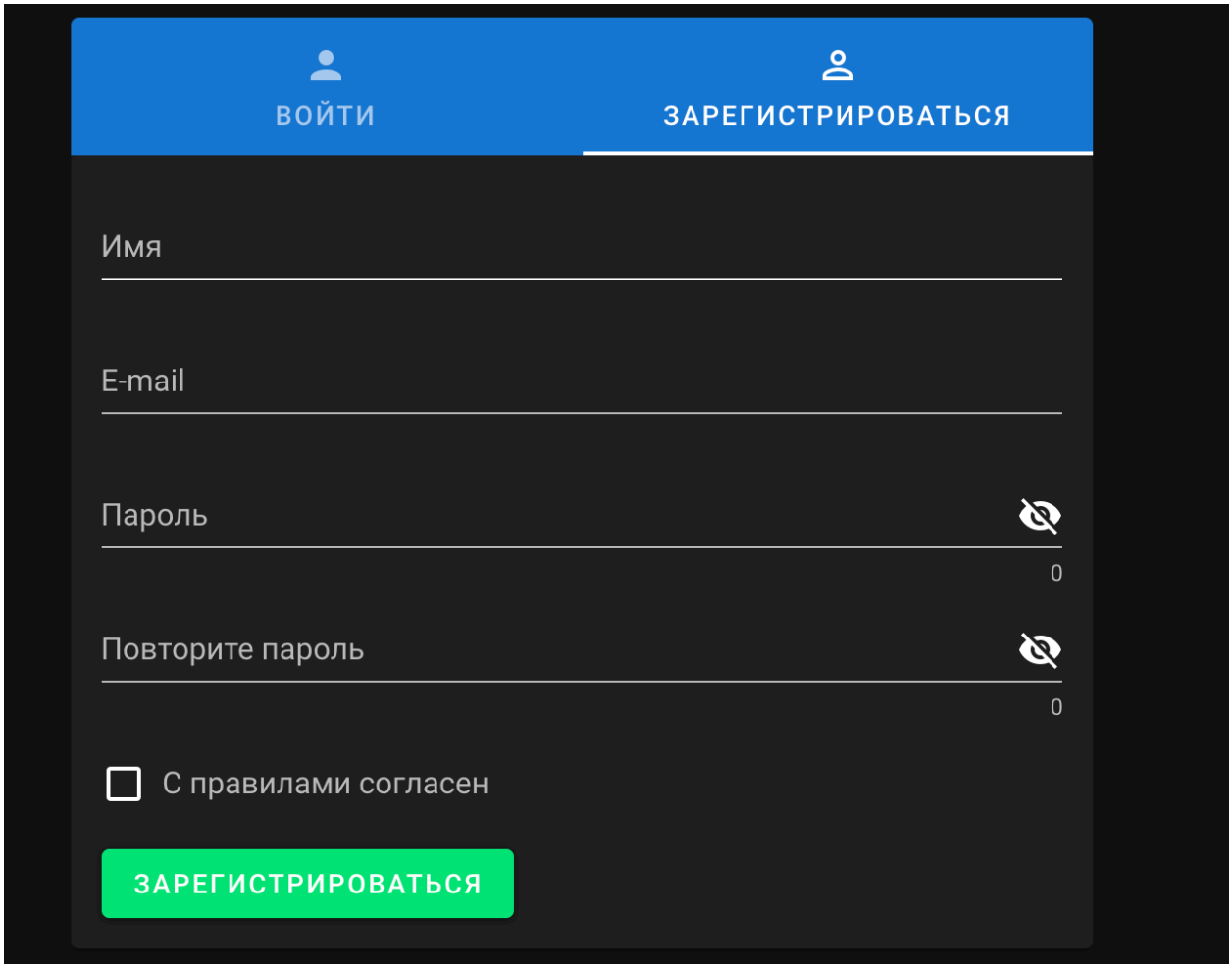

Рисунок Г.1 – Форма регистрации

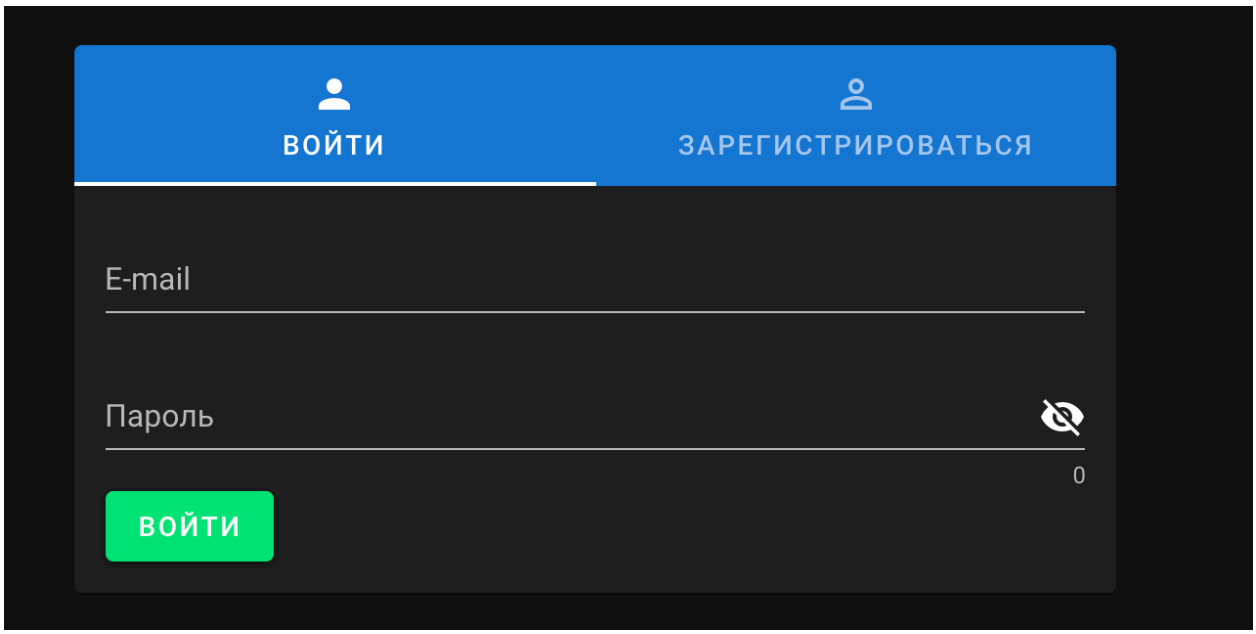

Рисунок Г.2 – Форма авторизации

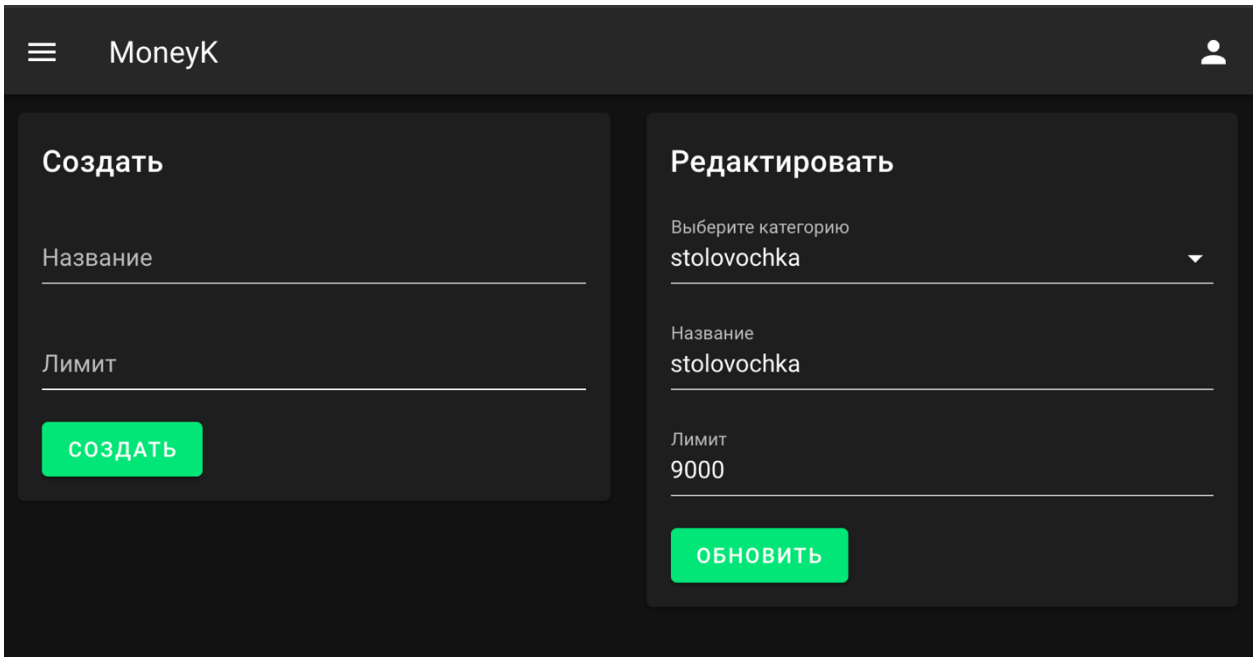

Рисунок Г.3 – Создание и редактирование категорий

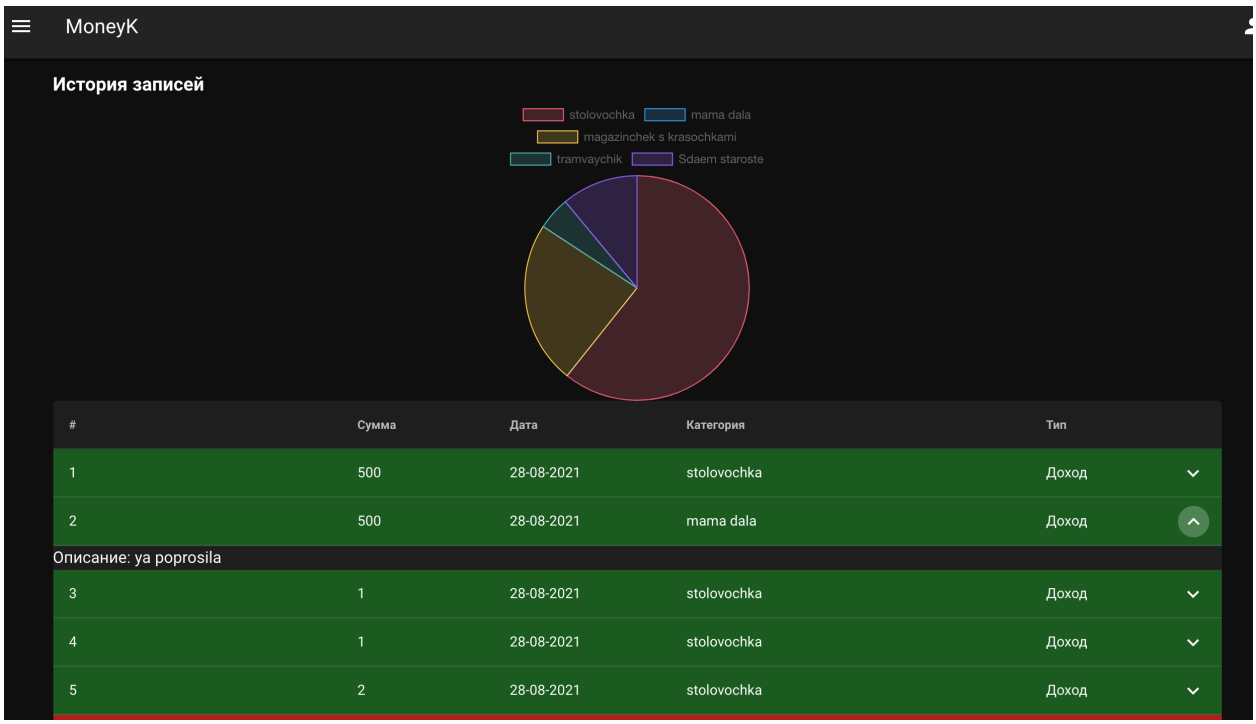

Рисунок Г.4 – История записей

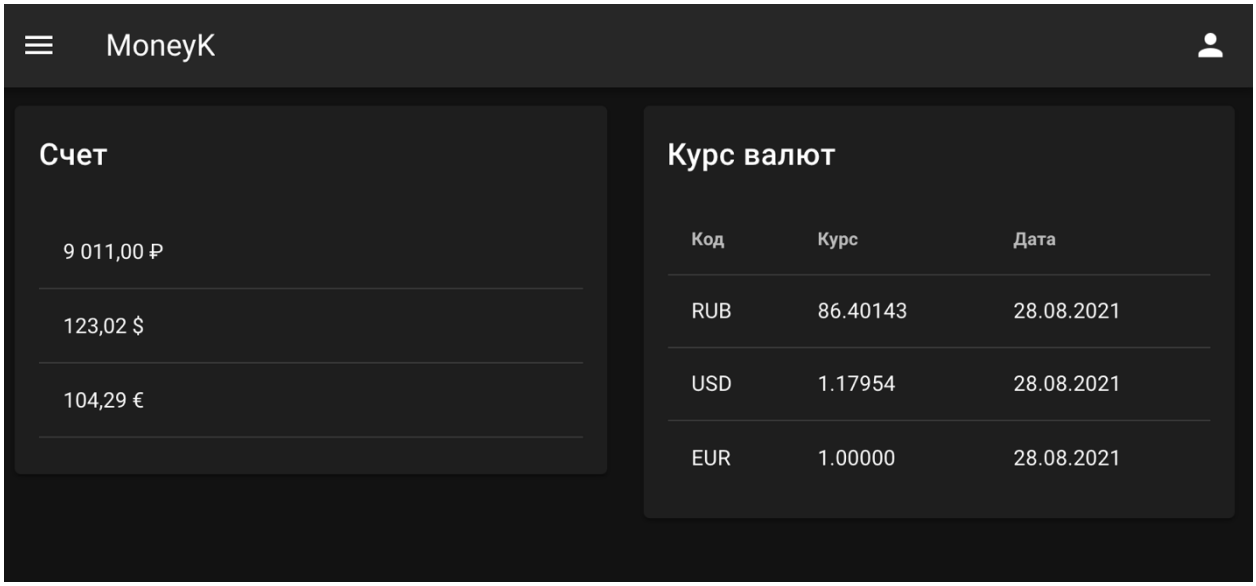

Рисунок Г.5 – Информация о счете, курс валют

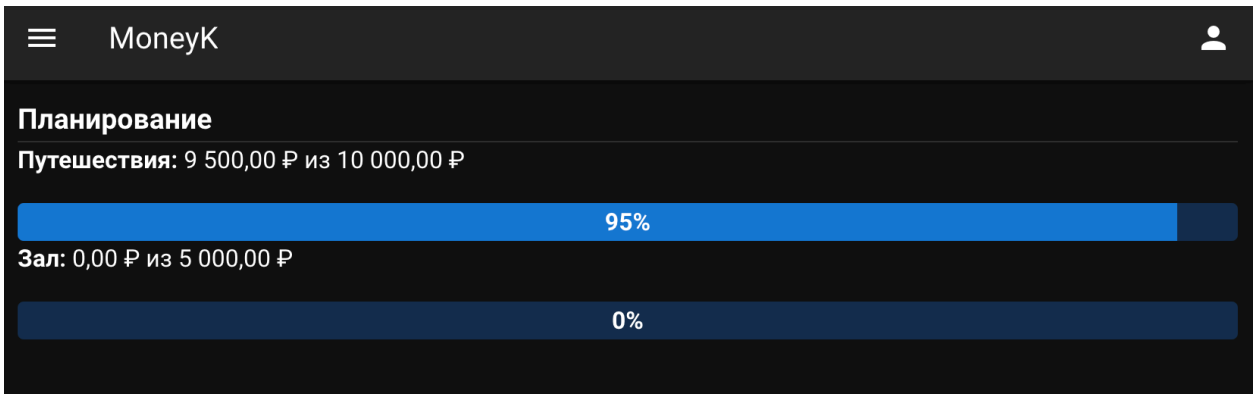

Рисунок Г.6 – Отображение прогресса по целям

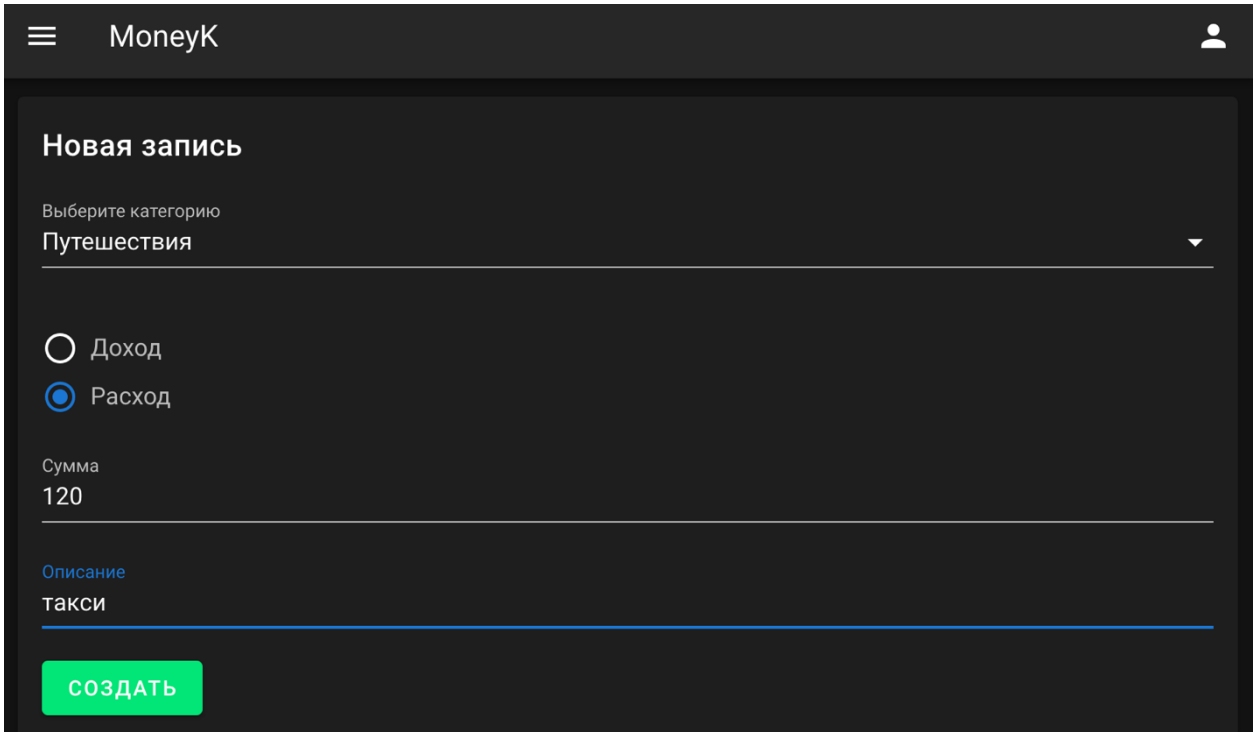

Рисунок Г.7 – Создание новой записи