

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

К защите допустить:

Заведующая кафедрой информатики

_____ Н. А. Волорова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту
на тему

ПРИЛОЖЕНИЕ ПЕРСОНАЛЬНЫЙ МЕНЕДЖЕР ФИНАНСОВ

БГУИР ДП 1-40 04 01 054 ПЗ

Студент	В.А. Попов
Руководитель	И.А. Удовин
Консультанты: <i>от кафедры информатики</i> <i>по экономической части</i>	И.А. Удовин Т.А. Рыковская
Нормоконтролер	Н.Н. Бабенко
Рецензент	И.П. Иванов

Минск 2021

РЕФЕРАТ

ПРИЛОЖЕНИЕ ПЕРСОНАЛЬНЫЙ МЕНЕДЖЕР ФИНАНСОВ: дипломный проект / В. А. Попов. – Минск : БГУИР, 2021, – п.з. – 58 с., чертежей (плакатов) – 6 л. Формата А4.

Объектом проектирования является программное средство, анализирующее финансы конкретного пользователя, позволяющее составлять заметки, посылающее уведомления.

Цель работы – разработка программного средства для обработки введённых данных, анализ и вывод статистики в разрезе дат и категорий транзакций.

Осуществлен полный цикл разработки программного обеспечения, включающий в себя анализ предметной области, изучение технологий и их отбор, проектирование, разработку и тестирование программного средства.

Разработанное программное средство должно помогать пользователю вести учёт финансов, уведомлять пользователя о заранее составленных планах.

Полученные в ходе технико-экономического обоснования результаты о прибыли для разработчика, пользователя, уровне рентабельности, а также экономическом эффекте доказывают целесообразность разработки приложения.

Министерство образования Республики Беларусь

+-----

Учреждение образования

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ
И РАДИОЭЛЕКТРОНИКИ**

Факультет КС и С Кафедра информатики
Специальность 1-40 04 01 Специализация 00

УТВЕРЖДАЮ

Н.А.Волорова

« » 20 г.

ЗАДАНИЕ

по дипломному проекту студента

Попова Вадима Андреевича

(фамилия, имя, отчество)

1. Тема проекта: **Приложение персональный менеджер финансов**

утверждена приказом по университету от « 19 » 04 2021 г. № 871-с

2. Срок сдачи студентом законченной работы 01 июня 2021 года

3. Исходные данные к проекту Тип операционной системы – ОС Windows 10;
Язык программирования – JavaScript; Перечень выполняемых функций:

Назначение разработки: подсчёт личных доходов/расходов, создание заметок и напоминаний

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)

Введение

1 Обзор предметной области. Постановка задачи

2 Используемые технологии

3 Проектирование программного средства

4 Создание программного средства

5 Тестирование программного средства

6 Техничко-экономическое обоснование

Заключение

Список использованных источников

Приложение А Текст программного модуля

Приложение Б Класс данных приложения

Приложение В Класс-маршрутизатор приложения

5. Перечень графического материала (с точным указанием наименования) и обозначения вида и типа материала)

Диаграмма декомпозиции веб-приложения. Схема программы – формат А4, лист 1.

Диаграмма способов использования приложения. Схема программы – формат А4, лист 1.

Схема-анализ структуры JSON. Схема программы – формат А4, лист 1.

Общая структура системы. Плакат – формат А4, лист 1.

Схема физической модели базы данных. Плакат – формат А4, лист 1.

Экранные формы программы. Плакат – формат А4, лист 1.

6. Содержание задания по технико-экономическому обоснованию

Расчет экономической эффективности от внедрения программного средства

Задание выдал _____ / Т.А. Рыковская /

Задание выдал: _____

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта (работы)	Объём этапа в %	Срок выполнения этапа	Примечан ие
Анализ предметной			
области, разработка технического задания	15	01.02–14.02	
Разработка функциональных требований,			
проектирование архитектуры программы	15	15.02–06.03	
Разработка схемы программы, алгоритмов,			
схемы данных	20	07.03–27.03	
Разработка программного средства	20	28.03–01.05	
Тестирование и отладка	10	02.05–08.05	
Оформление пояснительной записки			
и графического материала	20	09.05–31.05	

Дата выдачи задания 23 января 2021 г. Руководитель _____ /И.А. Удовин/

Задание принял к исполнению _____ / В.А. Попов /

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ. ПОСТАНОВКА ЗАДАЧИ	8
1.1 Менеджер финансов	8
1.2 Облачное хранение данных	8
1.2.1 Файловое хранилище	10
1.2.2 Блочное хранилище	11
1.2.3 Объектное хранилище	11
1.2.4 База данных	12
1.3 Обзор существующих аналогов	12
1.3.1 Monefy	12
1.3.2 Coinkeeper	13
1.3.3 Toshl	13
1.3.4 Money Manager	14
1.4 Постановка задачи	15
1.5 Перспективы развития программного средства	15
2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ	16
2.1 Язык разметки HTML	16
2.2 Каскадные таблицы стилей CSS	17
2.3 Язык программирования JavaScript	19
2.4 Облачная база данных Firebase	20
2.5 Git / GitHub	21
3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА	25
3.1 Создание и настройка репозитория	25
3.2 Модули приложения	25
3.2.1 Модуль Firebase	25
3.2.2 Модуль Router	26
3.2.3 Модуль рендеринга страниц	26
4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА	30
4.1 Хранение данных в Firebase Realtime Database	30
4.2 Добавление данных в Firebase Realtime Database	30
4.3 Визуализация данных в веб-приложении	32
4.3.1 Страницы регистрации и авторизации	32
4.3.2 Страница профиля	33
4.3.3 Страница с транзакциями	34
4.3.4 Страница с категориями	36
4.3.5 Страница со статистикой	38
4.3.6 Страница с заметками	40
5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА	42
6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ	43
6.1 Описание функций, назначения и потенциальных пользователей программного обеспечения	43

6.2	Расчёт затрат на разработку программного обеспечения.....	43
6.3	Оценка эффекта от продажи программного обеспечения	46
6.4	Расчёт показателей эффективности инвестиций в разработку программного обеспечения	47
	ЗАКЛЮЧЕНИЕ	50
	БИБЛИОГРАФИЧЕСКИЙ СПИСОК	51
	ПРИЛОЖЕНИЕ А (обязательное) Текст программного модуля	53
	ПРИЛОЖЕНИЕ Б (обязательное) Классы данных приложения.....	55
	ПРИЛОЖЕНИЕ В (обязательное) Класс-маршрутизатор приложения	56

ВВЕДЕНИЕ

Люди ведут персональный финансовый учет с различными целями: выплатить кредиты, организовать накопления, сократить расходы, спланировать даты платежей и поступлений, если с трудом хватает денег до зарплаты.

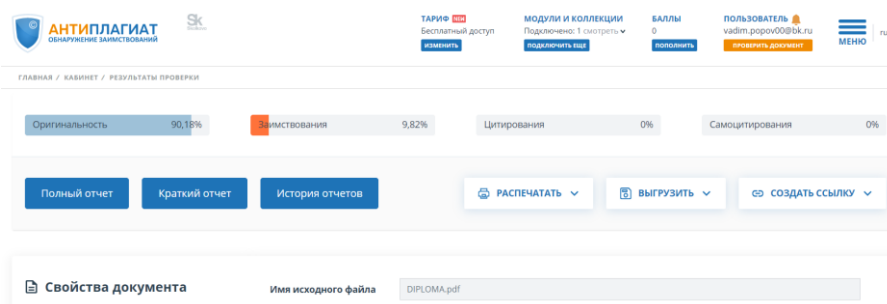
Мобильный банкинг уже давно умеет делить ваши расходы по категориям. Однако, ему ничего не известно о ваших наличных средствах, и он не знает, как планировать бюджет наперед. А учет финансов – первый шаг на пути к богатству. Магазиновые чеки собирать неудобно, а переносить из них цифры в блокнот или документ Excel долго и скучно. Поэтому проблему решают мобильные приложения и веб-сервисы.

В рамках данного проекта поставлена задача разработать веб-сервис для анализа статистики личных доходов/расходов, добавления заметок и напоминаний.

При разработке программного обеспечения были поставлены следующие задачи:

- разработать веб-приложение, содержащее в себе финансовый менеджер, заметки/напоминания;
- спроектировать хранилище данных, способное оперировать достаточно большими объёмами данных;
- создать интуитивно понятный и дружелюбный пользовательский интерфейс.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат.ру». Процент оригинальности соответствует норме, установленной кафедрой информатики. Цитирования и заимствования обозначены ссылками на публикации, указанные в «Списке использованных источников».



Скриншот с результатом проверки на плагиат

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ. ПОСТАНОВКА ЗАДАЧИ

1.1 Менеджер финансов

Учет личных финансов – первый шаг на пути к богатству. Так считают бизнесмены и консультанты по финансовой грамотности. Но собирать чеки неудобно, переносить в памяти цифры из магазина в компьютер долго и скучно. Записывать все в тетрадку у кассы – неловко. Поэтому проблему решают мобильные приложения и веб-сервисы.

Такое приложение должно предоставлять возможность:

- разделять бюджет на доходы и расходы;
- разделять доходы/расходы по категориям;
- разделять доходы/расходы по временному периоду;
- устанавливать лимит на расход личного бюджета;
- уведомлять пользователя, когда он приближается к лимиту;
- предоставлять итоговую статистику виде графиков;
- иметь интуитивно понятный интерфейс.

1.2 Облачное хранение данных

Отличительной чертой нашего времени является постоянный рост объема деловой информации. Дизайнеры, маркетологи, копирайтеры, представители IT-профессий, а также компании, работающие с огромными массивами данных, постоянно нуждаются в надежном месте, в котором можно было бы хранить ценные файлы. Если раньше их держали на дискетах, флешках и компакт-дисках, то сейчас лучше всего отправлять их в облачные хранилища.

В общем и целом, облачное хранилище - это специально выделенное место на серверах, куда любой пользователь может загрузить различные документы: текстовые файлы, любимые аудиозаписи и видеоролики, картинки, гиф-картинки, переписку из мессенджеров и многое другое. При этом серверы могут находиться где угодно: в Европе, Азии или Северной Америке.

Механизм облачного хранилища очень прост: нужно установить клиентское приложение и зарегистрироваться в нем. После чего можете спокойно сбрасывать в «облако» любую информацию, обмениваться ей с коллегами, обновлять ее, просматривать и так далее. Доступ к нему можно получить с любого устройства и из любого места, в котором есть Интернет [1].

У модели две стороны: клиент и поставщик услуги (провайдер). Клиент арендует место на серверах провайдера, сохраняет на них документы, приложения, статический контент сайтов, получая доступ к ним удаленно. Поставщик организует хранение, обслуживание, безопасность и доступ к данным. Эта модель имеет несколько преимуществ:

1) экономия – в случае с облачным хранилищем большинство операционных расходов (помещение, серверы, охлаждение, затраты на

резервное копирование с покупкой ПО и дополнительные накопители, обслуживание) сокращаются, а капитальные отсутствуют;

2) надежность – облачные хранилища обслуживаются инженерами со специализированным опытом в эксплуатации систем такого типа. Администраторы провайдеров регулярно обновляют железо, улучшают ПО, работают над безопасностью. При этом данные хранятся «с запасом»: для хранения 1 Гб данных клиента поставщик резервирует 2 Гб. Серверы часто распределены по нескольким городам или странам, что добавляет отказоустойчивости при форс-мажорах;

3) доступность – данными можно управлять через графические интерфейсы, консоль или API;

4) масштабирование – объем быстро увеличивается за счет подключения дополнительных серверов и СХД;

5) управление расходами – платить нужно ровно столько, сколько потреблять ресурсов. В объектных хранилищах есть классы — стандартное, «холодное», «ледяное». Классы помогают управлять стоимостью хранения. Например, когда к данным нужно часто обращаться, можно платить дороже за хранение, но дешевле за трафик (обращения). Для архивов наоборот — можно платить за работу с файлами дороже, но за хранение дешевле, потому что к ним редко обращаются;

6) бизнес-процессы упрощаются, когда облачное хранилище доступно для сотрудника, например, из дома на выходных. А еще не забываем о восстановлении данных, когда бизнес-процессы не прерываются форс-мажорами из-за потери документов или репозитория.

Облачное хранилище во многих случаях может стать хорошей альтернативой традиционным решениям по хранению в корпоративной системе (on-premise). Однако, во многих случаях резервирование файлов в облаке имеет некоторые минусы:

- 1) зависимость от Интернет-соединения;
- 2) зависимость от провайдера;
- 3) безопасность;
- 4) защита данных.

Хранение в облаке используется:

- для хранения массивных данных;
- в качестве репозитория контента, например, мультимедиа ресурсов;
- для Big Data, «Интернета вещей» и машинного обучения;
- для архивации или хранения для последующей аналитики;
- для хранения данных игровых платформ, вроде Google Stadia;
- видеохостингами для потоковой раздачи контента;
- в качестве хостинга интернет-магазинов, порталов, блогов.

Но чаще встречается пять сценариев:

1) резервное копирование и восстановление – большинство файловых систем облаков совместимы с базами данных, поэтому хранилища часто используют для резервирования, например, при обновлениях;

2) разработка ПО и тестирование – часто разработка требует дублирования сред, которые потом нужно удалять, и совместной работы. Использование облачных ресурсов для этого — стандартная практика среди разработчиков ПО. Также, облака интегрируются с разными приложениями без дополнительных «костылей»;

3) совместный доступ – например, для команд разработки и тестирования из разных офисов или городов. Если данные хранятся на сервере внутри сети предприятия, часто нужен VPN. Но можно обойтись без этого и перенести часть общих файлов, к которым обычно и нужен доступ, в облачное хранилище;

4) миграция данных в облако облегчает обслуживание своей инфраструктуры, но это серьезная задача, требующего многолетнего опыта у системного администратора;

5) Big Data и IoT – например, для Big Data массив данных в 100 Терабайт не так уж много, но держать на локальных серверах такой объем дорого, поэтому для этого часто используют облака. Хранить в «облаке» массивы удобно: в облачных сервисах обычно высокая пропускная способность, низкие задержки, и возможность настроить запросы не извлекая данные [2].

Поскольку данные бывают разные, то и хранить их лучше в подходящих для этого местах. По типу организации облачные хранилища делятся на:

- файловые;
- блочные;
- объектные;
- базы данных.

1.2.1 Файловое хранилище

В основе файловой системы лежит иерархическая структура: корневая запись, от которой отходят данные о файлах и их атрибутах. Все они, в свою очередь, организованы в удобную структуру каталогов – зная имя того или иного документа, доступ к нему можно получить, щелкнув мышью по его имени. С ними можно осуществлять любые операции – открывать, изменять, переименовывать, удалять, копировать, перемещать в другую папку.

Файловое хранилище может быть двух видов: физическим и виртуальным. В первом случае данные сохраняются на жестком диске, во втором – на виртуальном. Последний имеет намного больший объем чем жесткий, а еще туда можно настроить удаленный доступ. В качестве примера можно привести Dropbox, «Облако Mail.Ru», «Google Диск», «Яндекс. Диск» и другие аналогичные им сервисы.

Преимущества:

- простая и понятная структура;
- в таком хранилище легко ориентироваться, искать нужные документы.

Недостатки:

- ограниченность в объеме, по мере заполнения которого падает скорость доступа, а вместе с ней и производительность.

Для чего подходит: для работы с небольшими объемами разных данных.

1.2.2 Блочное хранилище

В блочном хранилище структура размещения та же, но все попадающие туда файлы делятся системой на блоки, каждому из которых присваивается свой идентификатор. С его помощью система собирает файлы в случае надобности.

Преимущества:

- каждая пользовательская среда находится отдельно, за счет чего можно рассортировывать данные и обеспечить отдельный доступ к ним;
- БХ обеспечивает повышенную производительность: благодаря хост-адаптеру шины, который разгружает процессор и освобождает его ресурсы для выполнения других задач.

Недостатки:

- оно дороже, и им трудно управлять, поскольку работа с блоками создает дополнительную нагрузку на базу данных;
- оно, как и файловое, ограничено в объеме.

Для чего подходит: для работы с корпоративными базами данных

1.2.3 Объектное хранилище

Это самый популярный тип хранилища. Вместо файловой системы в нем есть плоское пространство, состоящее из множества объектов, каждый из которых состоит из идентификатора и метаданных. Идентификатор – это присвоенный адрес, в роли которого выступает 128-битное число. Зная его можно без труда найти нужный файл. Метаданные (информация о файле) – его имя, размер, координаты и другая информация.

Достоинства:

- возможность работы с колоссальным объемом информации. Общий объем данных, хранящихся в Haystack Facebook, оценивается в 357 петабайт;
- возможность хранения резервных копий данных, особенно тех, от которых зависит жизнедеятельность системы (например, файлы для аварийного восстановления);
- возможность проверки корректности файлов и обеспечения быстрого доступа к ним.

Недостатки:

- в большинстве нет интерфейса для загрузки и управления файлами.

Для чего подходит: для хранения больших данных, текстовых документов, изображений, медиафайлов, переписок и многого другого.

1.2.4 База данных

База данных – это совокупность определенной информации, хранящаяся в строго установленном порядке на физических или виртуальных носителях. Она управляется специальной программой под названием СУБД (Система Управления Базами Данных). СУБД позволяет обрабатывать любые тексты, графику, медиа; с ними можно делать все что угодно: хранить, анализировать, тестировать продукты и обновления, запускать новые проекты.

Базы данных могут находиться либо на сервере, либо в облаке. Облачные СУБД сегодня являются самыми популярными в своей области. Согласно исследованиям Market Realist, их используют 35% респондентов, экспериментируют с ними 14%, планируют внедрение – 12%.

Преимущества:

- облачные базы данных имеют почти неограниченный объем хранения;
- есть функция резервного копирования;
- они обладают высоким внешним и внутренним уровнем безопасности;
- поддержка многозадачного и многопользовательского режимов.

Недостатки:

- сложность управления, что требует затрат на соответствующий персонал и ПО;
- в случае нахождения их на физическом носителе имеют ограниченный объем, так что может потребоваться увеличение дискового пространства;
- высокая стоимость разработки и эксплуатации.

Для чего подходят: для управления однородными массивами данных [3].

1.3 Обзор существующих аналогов

Рассмотрим существующие аналоги программных средств для мониторинга и анализа личных доходов и расходов, выявим их преимущества и недостатки, чтобы в последствие учесть их при составлении технических требований к разрабатываемому продукту. Источники информации об аналогах – магазины мобильных приложений Play Market и App Store, а также поисковые системы Google и Yandex.

1.3.1 Monefy

Monefy – крайне простое и удобное приложение для тех, кто хочет тщательно следить за расходами. Траты распределяются по категориям, а для каждой категории можно присвоить собственную иконку. Расходы отображаются в виде диаграммы. В приложении есть встроенный калькулятор и нет рекламы (рисунок 1.1).



Рисунок 1.1 - Интерфейс Monefy

1.3.2 Coinkeeper

У приложения необычный интерфейс, напоминающий монетницу. Контролировать расходы и доходы можно простым перетаскиванием монет из кошелька в расходные статьи. Облачная синхронизация поможет семейным парам вести общий бюджет на разных устройствах (рисунок 1.2).

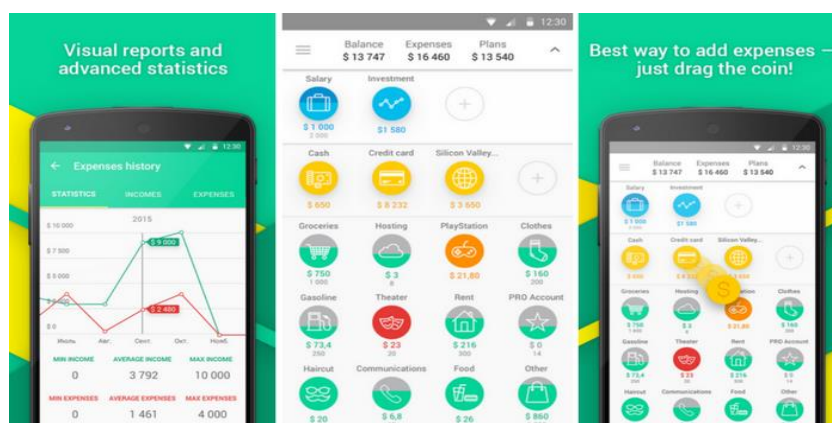


Рисунок 1.2 - Интерфейс Coinkeeper

1.3.3 Toshl

Самое важное в Toshl это легкость использования. Нет необходимости каждый раз пояснять полное наименование затраты, достаточно пользоваться тегами. К примеру, все напитки отметить тегом «drink», а затраты на мобильный — «mobile». Подобная концепция существенно упрощает ввод сведений, а также значительно уменьшает период, необходимое для этого (рисунок 1.3).

Во вкладке «Бюджет» вы добавляете все ваши доходы. Тут же отображается прогресс-бар со частями денег. Таким образом же можно выбрать на что выделен тот либо другой «бюджет».

При намерении можно экспортировать всю статистику в отдельный

документ формата .csv, .xls либо .pdf. В некоторых случаях программа присылает забавные уведомления наподобие «Я заметил, что вам много денег тратите в выходные, обратите на это внимание». При этом эффективность подобных рекомендаций достаточно высока.

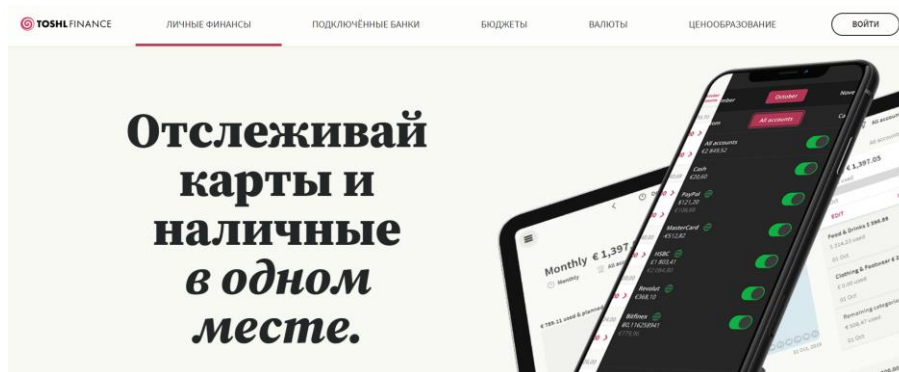


Рисунок 1.3 – Интерфейс Toshl

1.3.4 Money Manager

Money Manager - одно из наиболее функциональных приложений, с помощью которого можно не только держать под контролем свои траты, но и получать статистику за любой период времени (рисунок 1.4). Здесь можно управлять картами, получать статистику и отслеживать на графиках состояние активов. Кроме того, приложение имеет следующие особенности:

- система двойной записи;
- формирование бюджета по выбранным категориям;
- доступ с ПК;
- защита доступа паролем;
- перевод средств между активами;
- отслеживание расходов и доходов по активам;
- встроенный калькулятор;
- поиск по категориям [4].



Рисунок 1.4 - Интерфейс Money Manager

1.4 Постановка задачи

В рамках данного проекта поставлена задача разработать веб-сервис для анализа статистики личных доходов/расходов, добавления заметок и напоминаний.

Потенциальной целевой аудиторией программного продукта можно считать частных лиц, заинтересованных в удобном отслеживании своего личного бюджета, заметок и напоминаний, а также юридических лиц, заинтересованных в качественном ведении бухгалтерского учёта.

При разработке программного обеспечения были поставлены следующие задачи:

- разработать систему, реализующую деятельность персонального менеджера;
- спроектировать хранилище данных, способное оперировать достаточно большими объёмами данных;
- создать дружелюбный и интуитивно понятный пользовательский интерфейс.

1.5 Перспективы развития программного средства

Разрабатываемый продукт представляет собой веб-сервис, использующий реальные данные пользователя и имеющий реальное практическое применение. Однако, в перспективе, веб-сервис будет дополнен мобильным приложением под мобильные операционные Android и iOS. Таким образом продукт станет масштабируемым для разных платформ. В дополнение к этому, будет добавлена функция парсинга смс, приходящих от банков, для последующего автоматического разделения по категориям доходов и расходов.

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

2.1 Язык разметки HTML

HTML (Hypertext Markup Language) - это код, который используется для структурирования и отображения веб-страницы и её контента. Например, контент может быть структурирован внутри множества параграфов, маркированных списков или с использованием изображений и таблиц данных.

HTML был изобретён Тимом Бернерсом-Ли, физиком из исследовательского института ЦЕРН в Швейцарии. Он придумал идею интернет-гипертекстовой системы.

Hypertext означает текст, содержащий ссылки на другие тексты, которые зрители могут получить немедленно. Он опубликовал первую версию HTML в 1991 году, состоящую из 18 тегов HTML. С тех пор каждая новая версия языка HTML появилась с разметкой новых тегов и атрибутов (модификаторов тегов).

Согласно Справочнику HTML Element Reference от Mozilla Developer Network, в настоящее время существует 140 тегов HTML, хотя некоторые из них уже устарели (не поддерживаются современными браузерами).

Самым большим обновлением языка стало внедрение HTML5 в 2014 году. Было добавлено несколько новых семантических тегов к разметке, которые показывают смысл их собственного контента, например `<article>`, `<header>` и `<footer>`.

HTML-документы — это файлы, которые заканчиваются расширением `.html` или `.htm`. Вы можете просматривать его с помощью любого веб-браузера (например, Google Chrome, Safari или Mozilla Firefox). Браузер читает HTML-файл и отображает его содержимое, чтобы пользователи интернета могли его просматривать.

Каждая HTML-страница состоит из набора тегов (также называемых элементами), которые вы можете назвать строительными блоками веб-страниц. Они создают иерархию, которая структурирует контент по разделам, параграфам, заголовкам и другим блокам контента.

Большинство элементов HTML имеют открытие и закрытие, в которых используется синтаксис `<tag> </tag>` [5].

Рассмотрим элемент абзаца более подробно (рисунок 2.1).

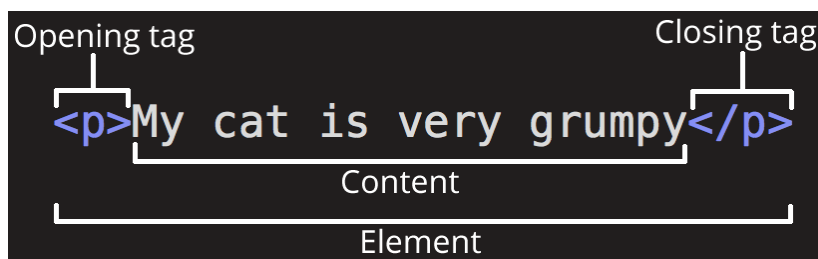


Рисунок 2.1 – Элемент абзаца

Главными частями нашего элемента являются:

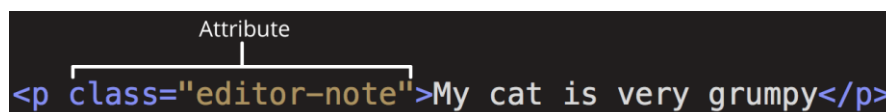
1. Открывающий тег (Opening tag): Состоит из имени элемента (в данном случае, "p"), заключённого в открывающие и закрывающие угловые скобки. Открывающий тег указывает, где элемент начинается или начинает действовать, в данном случае — где начинается абзац.

2. Закрывающий тег (Closing tag): Это то же самое, что и открывающий тег, за исключением того, что он включает в себя косую черту перед именем элемента. Закрывающий элемент указывает, где элемент заканчивается, в данном случае — где заканчивается абзац. Отсутствие закрывающего тега является одной из наиболее распространённых ошибок начинающих и может приводить к странным результатам.

3. Контент (Content): Это контент элемента, который в данном случае является просто текстом.

4. Элемент(Element): Открывающий тег, закрывающий тег и контент вместе составляют элемент.

Элементы также могут иметь атрибуты, которые выглядят так (рисунок 2.2):



```
<p class="editor-note">My cat is very grumpy</p>
```

Рисунок 2.2 – Атрибут элемента

Атрибуты содержат дополнительную информацию об элементе, которая не показывается в фактическом контенте. В данном случае, class это имя атрибута, а editor-note это значение атрибута. Класс позволяет дать элементу идентификационное имя, которое может позже использоваться, чтобы обращаться к элементу с информацией о стиле и прочих вещах.

Атрибут всегда должен иметь:

1. Пробел между ним и именем элемента (или предыдущим атрибутом, если элемент уже имеет один или несколько атрибутов).

2. Имя атрибута, за которым следует знак равенства.

3. Значение атрибута, заключённое с двух сторон в кавычки [6].

2.2 Каскадные таблицы стилей CSS

CSS – это язык, с помощью которого описывается внешний вид документа HTML, XML, XHTML. Название означает «каскадная таблица стилей», или Cascading Style Sheets. CSS-стили незаменимы при оформлении страниц сайтов: в одном файле содержатся сведения об отображении всех элементов документа.

По сути, таблица стилей – это файл, где описывается, как будет выглядеть каждый из элементов на странице. В HTML-документе, таким образом, остается только структура странички: сами блоки, их содержимое и

расположение. Создать страницу и оформить ее можно и без использования таблиц, прописывая визуальные свойства каждого элемента в его описании.

Но, если страниц сотни и тысячи, применять такой метод неудобно: при изменении оформления приходится менять множество документов, вдобавок это загромождает верстку. Поэтому использование CSS считается золотым стандартом оформления сайтов: так получилось благодаря гибкости и многообразию возможностей каскадных таблиц.

Преимущества CSS:

- это существенно упрощает верстку и снижает временные затраты – один созданный файл стилей можно распространить на множество страниц, так что внешний вид элементов достаточно описать один раз;
- если что-то нужно изменить, достаточно внести правки в один файл;
- применение CSS серьезно облегчает структуру документа, что хорошо и для пользователей, и для поисковых программ;
- вариативность оформления становится шире – CSS поддерживает намного больше возможностей, чем имеется при использовании чистого HTML, вдобавок к одной странице можно применить несколько стилей в зависимости от обстоятельств (размер монитора пользователя, устройство, с которого выполнен вход, – ПК или мобильное);
- страницы начинают загружаться быстрее: браузер кеширует таблицу стилей при первом посещении сайта, при последующих подгружаются только данные, что намного быстрее.

Начало развития было положено в 1990-х, когда консорциум W3C решил, что технология, позволяющая разделять содержание и представление документов, необходима. Стандарт CSS1 появился в 1996 году и позволял изменять с помощью таблиц параметры шрифтов, цвета элементов, свойства блоков и текстов, такие как отступы и выравнивание. Длина и ширина блоков задавались там же. С развитием интернета появились новые уровни:

- второй (CSS2) – стандарт расширил технические возможности, дал возможность работать с аудио и страничными носителями (например, при печати документов), включил в себя поддержку блочной структуры и генерируемого содержимого;
- третий (CSS3) – еще более масштабное расширение, находится в разработке до сих пор, поддерживает сглаживание, градиенты, тени и анимацию, для этого не приходится использовать JavaScript;
- четвёртый (CSS4) – находится в разработке, новые модули пока доступны как черновики.

Файл CSS сводится к набору правил, описанных по определенному синтаксису. Правило состоит из селекторной части и блока объявлений: ими описываются всевозможные элементы страницы. Формат примерно таков: селектор { параметр: значение }.

Селекторы указывают, к каким элементам будут применяться те или иные параметры стиля. Пишутся в начале строки, по сути, являются

названиями тегов, для которых справедливо правило. Среди особенности стоит выделить:

- можно использовать любой тег, написанный латиницей;
- если вариантов стиля для одного типа элементов несколько, используются так называемые классы. У одного тега их может быть несколько (применяются все стили, что описаны в таблице). Запись в этом случае выглядит так: `тег.Класс { параметр: значение };`
- есть возможность видоизменить только один конкретный элемент. Это делается с помощью идентификаторов – уникальных имен, которые можно присвоить элементам. Идентификатор будет использоваться как селектор [7].

2.3 Язык программирования JavaScript

JavaScript — это полноценный динамический язык программирования, который применяется к HTML документу, и может обеспечить динамическую интерактивность на веб-сайтах сайт (например: игры, отклик при нажатии кнопок или при вводе данных в формы, динамические стили, анимация). Его разработал Brendan Eich, сооснователь проекта Mozilla, Mozilla Foundation и Mozilla Corporation.

JavaScript сам по себе довольно компактный, но очень гибкий. Разработчиками написано большое количество инструментов поверх основного языка JavaScript, которые разблокируют огромное количество дополнительных функций с очень небольшим усилием. К ним относятся:

- программные интерфейсы приложения (API), встроенные в браузеры, обеспечивающие различные функциональные возможности, такие как динамическое создание HTML и установку CSS стилей, захват и манипуляция видеопотоком, работа с веб-камерой пользователя или генерация 3D графики и аудио сэмплов;
- сторонние API позволяют разработчикам внедрять функциональность в свои сайты от других разработчиков, таких как Twitter или Facebook;
- также к HTML применяются сторонние фреймворки и библиотеки, что позволит ускорить создание сайтов и приложений.

Базовой особенностью этого языка отмечается то, что на него повлияли другие (Python, Java и др.) языки программирования с целью придания максимального комфорта JavaScript и лёгкости в понимании его теми пользователями, которые не имеют соответствующего образования и глубинных знаний – не программистами. JavaScript – официально зарегистрированная торговая марка компании Oracle.

С помощью него доступны к исполнению следующие функции:

- возможность изменять страницы браузеров;
- добавление или удаление тегов;
- изменение стилей страницы;
- информация о действиях пользователя на странице;

- запрос доступа к случайной части исходного кода страницы;
- внесение изменений в этот код;
- выполнение действия с cookie-файлами.

Область применения этого языка удивительно обширна и ничем не ограничена: среди программ, которые используют JS, присутствуют и тестовые редакторы, и приложения (как для компьютеров, так и мобильные и даже серверные), и прикладное ПО.

Преимущества JavaScript:

- ни один современный браузер не обходится без поддержки JavaScript;
- с использованием написанных на JavaScript плагинов и скриптов справится даже не специалист;
- полезные функциональные настройки;
- взаимодействие с приложением может осуществляется даже через текстовые редакторы – Microsoft Office и Open Office;
- перспектива использования языка в процессе обучения программированию и информатике.

Недостатки JavaScript:

- пониженный уровень безопасности ввиду повсеместного и свободного доступа к исходным кодам популярных скриптов;
- множество мелких раздражающих ошибок на каждом этапе работы - их наличие позволяет считать этот язык менее профессиональным, сравнительно с другими;
- повсеместное распространение - своеобразным недостатком можно считать тот факт, что часть активно используемых программ (особенно приложений) перестанут существовать при отсутствии языка, поскольку целиком базируются на нем [8].

2.4 Облачная база данных Firebase

Firebase – это платформа разработки мобильных приложений с огромным функционалом. Начинаясь она как стартап, а сегодня ее используют при разработке лучших кроссплатформенных приложений. Главное достоинство платформы в том, что она позволяет разработчику не отвлекаться на создание бэкенда, то есть скрытой от пользователя программной части проекта, например, серверного кода. И это упрощает и ускоряет создание мобильных приложений, дает возможность полностью сосредоточиться именно на UX/UI, то есть, на пользовательском интерфейсе и опыте.

Firebase – это одно из BaaS-решений (Backend as a Service), которое дает разработчику массу возможностей.

Это и сервер, и база данных, и хостинг, и аутентификация в одной платформе. Так, Firebase Realtime Database предоставляет разработчикам API, который синхронизирует данные приложения между клиентами и хранит их в облачном хранилище.

Приложение подключается к базе данных через WebSocket, который отвечает за синхронизацию данных в течение всего сеанса.

Также Firebase выступает в качестве хранилища файлов. Firebase Storage обеспечивает надежную загрузку и выгрузку файлов для приложения. Облачное хранение файлов видео, аудио или любого другого типа поддерживается Google Cloud Storage. Содержимое облачного хранилища надежно защищено собственной системой безопасности.

Создавать систему аутентификации каждый раз с нуля довольно затратно, причем затраты эти чаще всего не оправданы. Справится с большинством вызовов позволяет система аутентификации Firebase Auth, в которой возможна аутентификация пользователя приложения по паролю и электронной почте. Поддерживает Firebase Auth также открытый протокол авторизации OAuth 2.0, используемый Google, Twitter, Facebook. Система аутентификации Firebase интегрируется непосредственно в базу данных.

Статические файлы приложения размещаются на хостинге Firebase. Поддерживается хостинг файлов JavaScript, HTML, CSS и других. Через Cloud Functions реализована динамическая поддержка Node.js. Передача файлов осуществляется через сеть доставки контента с использованием защищенных протоколов SSL и HTTPS [9].

2.5 Git / GitHub

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и дает возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранятся операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы.

Структура хранилища файлов не отражает реальную структуру

хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создает в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создается рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне. Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git свое приложение.

Для каждого объекта в репозитории вычисляется SHA-1-хеш, и именно он становится именем файла, содержащего данный объект в каталоге .git/objects. Для оптимизации работы с файловыми системами, не использующими деревья для каталогов, первый байт хеша становится именем подкаталога, а остальные — именем файла в нём, что снижает количество файлов в одном каталоге (ограничивающий фактор производительности на таких устаревших файловых системах).

В классическом обычном сценарии в репозитории git есть три типа объектов — файл, дерево и «коммит». Файл есть какая-то версия какого-то пользовательского файла, дерево — совокупность файлов из разных подкаталогов, «коммит» — дерево и некая дополнительная информация (например, родительские коммиты, а также комментарий).

Репозиторий Git бывает локальный и удаленный. Локальный репозиторий — это подкаталог .git, создаётся (в пустом виде) командой git init и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой git clone.

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным как

«вверх» (push), так и «вниз» (pull).

Наличие полностью всего репозитория проекта локально у каждого разработчика даёт Git ряд преимуществ перед SVN. Так, например, все операции, кроме push и pull, можно осуществлять без наличия интернет-соединения.

Очень мощной возможностью git являются ветви, реализованные куда более полно, чем в SVN: по сути, ветвь git есть не более чем именованная ссылка, указывающая на некий коммит в репозитории (используется подкаталог refs). Коммит без создания новой ветви всего лишь передвигает эту ссылку на себя, а коммит с созданием ветви — оставляет старую ссылку на месте, но создает новую на новый коммит, и объявляет её текущей. Заменить локальные девелоперские файлы на набор файлов из иной ветви, тем самым перейдя к работе с ней — так же тривиально.

Команда push передает все новые данные (те, которых еще нет в удалённом репозитории) из локального репозитория в репозиторий удаленный. Для исполнения этой команды необходимо, чтобы удалённый репозиторий не имел новых коммитов в себя от других клиентов, иначе push завершается ошибкой, и придётся делать pull и слияние.

Команда pull — обратна команде push. В случае, если одна и та же ветвь имеет независимую историю в локальной и в удаленной копии, pull немедленно переходит к слиянию.

Слияние в пределах разных файлов осуществляется автоматически (всё это поведение настраивается), а в пределах одного файла — стандартным двухпанельным сравнением файлов. После слияния нужно объявить конфликты как разрешенные.

Результатом всего этого является новое состояние в локальных файлах у того разработчика, что осуществил слияние. Ему нужно немедленно сделать коммит, при этом в данном объекте коммита в репозитории окажется информация о том, что коммит есть результат слияния двух ветвей и имеет два родительских коммита.

Также Git имеет временный локальный индекс файлов. Это — промежуточное хранилище между собственно файлами и очередным коммитом (коммит делается только из этого индекса). С помощью этого индекса осуществляется добавление новых файлов (git add добавляет их в индекс, они попадут в следующий коммит), а также коммит не всех измененных файлов (коммит делается только тем файлам, которым был сделан git add). После git add можно редактировать файл далее, получатся три копии одного и того же файла — последняя, в индексе (та, что была на момент git add), и в последнем коммите.

Имя ветви по умолчанию: master. Имя удалённого репозитория по умолчанию, создаваемое git clone во время типичной операции «взять имеющийся проект с сервера себе на машину»: origin.

Таким образом, в локальном репозитории всегда есть ветвь master, которая есть последний локальный коммит, и ветвь origin/master, которая есть

последнее состояние удаленного репозитория на момент завершения и исполнения последней команды pull или push.

Команда fetch (частичный pull) — берёт с удалённого сервера все изменения в origin/master, и переписывает их в локальный репозиторий, продвигая метку origin/master.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

- прямо на сайте можно просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования;
- можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям. Раньше возможность создавать приватные репозитории была платной;
- есть возможность прямого добавления новых файлов в свой репозиторий через веб-интерфейс сервиса;
- код проектов можно не только скопировать через Git, но и скачать в виде обычных архивов с сайта;
- кроме Git, сервис поддерживает получение и редактирование кода через SVN и Mercurial [10].

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Создание и настройка репозитория

Для хранения репозитория @vdmppv/expense-mgmt в интернете был выбран сервис GitHub. Выбрав соответствующие пункты меню создания репозитория были созданы следующие файлы:

- README.md — для предоставления полезной информации о репозитории другим разработчикам;
- .gitignore — для скрытия файлов и папок от системы контроля версий Git

3.2 Модули приложения

3.2.1 Модуль Firebase

Модуль Firebase отвечает за связь между базой данных и логикой приложения. Это подразумевает, что любая операция, направленная на манипуляцию данными в базе данных, будет проходить через данный модуль. К примеру, чтобы совершить добавление новой транзакции, необходимо обратиться сначала к методу writeTransaction в файле firebaseService.js, в который передаётся объект транзакции, и он преобразуется в поля, необходимые для создания записи в базы данных и заполнения её полей:

Листинг 3.1 – Создание записи с транзакцией в базе данных

```
async writeTransaction({uid}, transaction) {  
  const transactionNode = await this.transactionRef(uid).push();  
  
  await transactionNode.set(  
    {  
      amount: transaction.amount,  
      place: transaction.place,  
      description: transaction.description,  
      category_id: transaction.category_id,  
      date: transaction.date,  
      type: transaction.type,  
      uid: transactionNode.key,  
      image: null  
    }  
  );  
  if (transaction.image) {  
    this.uploadImage(uid, transaction.image, async (imageId) => {  
      await transactionNode.update(  
        {  
          image: imageId,  
        }  
      );  
    });  
  }
```

```

    });
  }
}

```

В дополнение к этому, модуль выступает как хостинг и отвечает за развёртывание разработанного программного средства как локально через команду `firebase serve`, так и на сайте через команду `firebase deploy`.

Настройка описана в файле `firebase.json`:

Листинг 3.2 – Настройки хостинга

```

{
  "hosting": {
    "public": "src",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}

```

3.2.2 Модуль Router

Модуль Router является маршрутизатором всего приложения. Он определяет какие компоненты должны быть запущены при заданных конечных точках. В любом реальном веб-приложении нужны маршруты. Пользователь должен видеть, где он находится в приложении в любой момент времени. А видит он свое текущее местоположение в адресной строке браузера. Следовательно, приложение должно уметь сопоставлять определённый URL с соответствующей ему страницей. То есть, если мы введём в адресную строку например <https://expenser-ab08b.web.app/categories>, то приложение должно направить нас на страницу категорий транзакций, но не на какую-либо другую.

3.2.3 Модуль рендеринга страниц

Данный модуль отвечает за отрисовку страниц, заполняя их тэги информацией, полученной из базы данных. Также, модуль отрисовывает верхний колонтитул страницы в зависимости от того, выполнен ли пользователем вход в систему или нет. Так, для неавторизованного пользователя доступны только кнопки регистрации и авторизации и курс

валют, а для авторизованного – весь функционал приложения, т.е. транзакции, категории, статистика, заметки и напоминания.

Вышеперечисленная логика хранится в файле header.js:

Листинг 3.3 – Рендер верхнего колонтитула

```
import {Router} from "../router.js";
import {linkNavigationHelper} from "../helpers/linkNavigationHelper.js";
import {getRate} from "../helpers/getRates.js";

let header = {
  render: async () => {

    let user = firebase.auth().currentUser;
    let username = localStorage.getItem("username");
    let headerView;
    if (user) {
      headerView = `
        <nav class="navigation-menu-block">
          <ul class="navigation-menu">
            <li class="navigation-item">
              <a class="navigation-
link" href="/transactions" id="header-transactions">Transactions</a>
            </li>
            <li class="navigation-item">
              <a class="navigation-
link" href="/categories" id="header-categories">Categories</a>
            </li>
            <li class="navigation-item">
              <a class="navigation-
link" href="/statistics/category" id="header-statistics">Statistics</a>
            </li>
            <li class="navigation-item">
              <a class="navigation-link" href="/notes" id="header-
notes">Daily Reminders</a>
            </li>
          </ul>
        </nav>

        <div class="user-block">
          <ul class="navigation-menu">
            <li class="navigation-item">
              <a class="navigation-
link" href="/profile" id="header-
username">${user ? user.displayName : username}</a>
            </li>
            <li class="navigation-item">
              <a class="navigation-link" id="header-
logout">Log Out</a>
            </li>
          </ul>
      `
    }
  }
}
```

```

        </div>
    ,
    } else {
    headerView = `
        <nav class="navigation-auth-block">
            <ul class="navigation-menu">
                <li class="navigation-item">
                    <a class="navigation-link" href="/login" id="header-
login">Log In</a>
                </li>
                <li class="navigation-item">
                    <a class="navigation-
link" href="/register" id="header-signup">Sign Up</a>
                </li>
            </ul>
        </nav>
    ,
    }
    let view = /*html*/`
    <div class="logo-block">
        
        <div class="rate-block">
            <p class="navigation-link">$:<span id='usd'></span>
                €:<span id='eur'></span></p>
        </div>
    </div>
    ${headerView}
    ,
    return view;
},
after_render: async () => {
    getRate();
    let user = firebase.auth().currentUser;
    if (user) {
        const transactionButton = document.getElementById("header-
transactions");
        transactionButton.onclick = linkNavigationHelper;

        const categoriesButton = document.getElementById("header-
categories");
        categoriesButton.onclick = linkNavigationHelper;

        const statisticsButton = document.getElementById("header-
statistics");
        statisticsButton.onclick = linkNavigationHelper;

        const usernameButton = document.getElementById("header-username");
        usernameButton.onclick = linkNavigationHelper;

        const notesButton = document.getElementById("header-notes");
    }
}

```

```
notesButton.onclick = linkNavigationHelper;

const logoutButton = document.getElementById("header-logout");
logoutButton.onclick = (event) => {
  firebase.auth().signOut();
  linkNavigationHelper(event);
  Router._instance.navigate("/");
};
} else {
  const loginButton = document.getElementById("header-login");
  loginButton.onclick = linkNavigationHelper;

  const registerButton = document.getElementById("header-signup");
  registerButton.onclick = linkNavigationHelper;
}
}
}
export default header;
```

4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

4.1 Хранение данных в Firebase Realtime Database

База данных Firebase Realtime позволяет создавать многофункциональные приложения для совместной работы, обеспечивая безопасный доступ к базе данных непосредственно из кода на стороне клиента. Данные сохраняются локально, и даже в автономном режиме события в реальном времени продолжают срабатывать, предоставляя пользователю полноценный отклик. Когда устройство восстанавливает соединение, база данных реального времени синхронизирует локальные изменения данных с удаленными обновлениями, которые произошли, когда клиент находился в автономном режиме, автоматически объединяя любые изменения.

База данных реального времени является базой данных NoSQL и, поэтому имеет иные виды оптимизации и функциональность по сравнению с реляционной базой данных. API-интерфейс базы данных реального времени позволяет выполнять только те операции, которые могут быть выполнены быстро. Это позволяет производить обработку данных в реальном времени, обслуживая миллионы пользователей без ущерба для скорости отклика. В связи с этим важно продумать то, как пользователи должны получать доступ к вашим данным, и затем соответствующим образом структурировать их.

Все данные базы данных Firebase Realtime хранятся в виде объектов JSON. Вы можете думать о базе данных как о дереве JSON, размещенном в облаке. В отличие от базы данных SQL, здесь нет таблиц или записей. Когда добавляются данные в дерево JSON, то оно становится узлом в существующей структуре JSON со связанным ключом (рисунок 4.1).



Рисунок 4.1 - Визуализация узла в Firebase Realtime Database

4.2 Добавление данных в Firebase Realtime Database

Перед сохранением данных в базу данных необходимо создать классы, объекты которых будут создаваться во время добавления новых карточек транзакций, категорий или заметок, через соответствующие модальные окна.

Рассмотрим на примере категорий. На странице Categories есть кнопка

Add Category, которая открывает модальное окно и требует заполнить в нём информацию о новой категории (рисунок 4.2), после чего создаётся объект класса Category и идёт запись в базу данных.

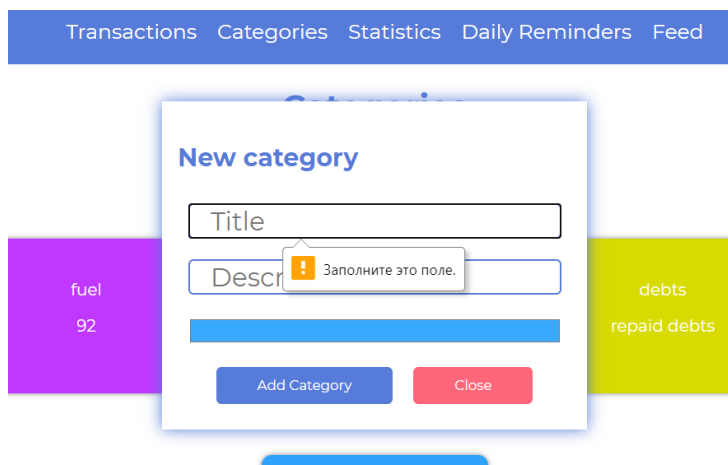


Рисунок 4.2 - Модальное окно Add Category

Вызов модального окна содержится в файле categoriesPage.js:

Листинг 4.2.1 – Открытие модального окна Add Category

```
const addCategoriesButton = document.getElementById("add-button");
addCategoriesButton.addEventListener("click", () => {
  showModal(AddCategoryModal);
});
```

Логика создания нового объекта класса Category содержится в файле addCategoryModal.js:

Листинг 4.2.2 – Модальное окно, реализующее создание объекта класса и вызов метода, записывающего данные в базу данных

```
after_render: async () => {
  const user = firebase.auth().currentUser;
  const categoryTitle = document.getElementById("category-title");
  const categoryDescription = document.getElementById("category-
description");
  const categoryColor = document.getElementById("category-color");
  const form = document.querySelector(".modal");
  form.addEventListener("submit", async (event) => {
    event.preventDefault();

    const category = new Category(
      {
        title: categoryTitle.value,
        description: categoryDescription.value,
        color: categoryColor.value
      }
    );
    await firebaseService.writeCategory(user, category);
    closeModal();
  });
});
```

```

        const categoryCloseButton = document.getElementById("category-close-
button");
        categoryCloseButton.addEventListener("click", (event) => {
            event.preventDefault();
            closeModal();
        });
    }
}

```

Логика добавления созданного объекта в базу данных содержится в файле `firebaseService.js`:

Листинг 4.2.3 – Метод записи новой категории в базу данных

```

categoryRef(uid) {
    return firebase.database().ref(`/category${uid}`);
}
async writeCategory({uid}, category) {
    const categoryNode = await this.categoryRef(uid).push();
    await categoryNode.set(
        {
            title: category.title,
            description: category.description,
            color: category.color,
            uid: categoryNode.key,
        }
    );
}
}

```

4.3 Визуализация данных в веб-приложении

Для каждой страницы веб-приложения в верхнем колонтитуле страницы отображается логотип сайта и курсы валют на текущий момент, полученные в результате запроса к API НБРБ.

Для неавторизованного пользователя доступны кнопки, ведущие на страницы:

- регистрации;
- входа в систему.

Для авторизованного:

- транзакций;
- категорий;
- статистики;
- заметок;
- профиля;
- выхода из системы.

4.3.1 Страницы регистрации и авторизации

Страница регистрации предоставляет форму, в которой содержатся следующие поля: электронная почта, имя, фамилия, логин, пароль, подтверждение пароля (рисунок 4.3).

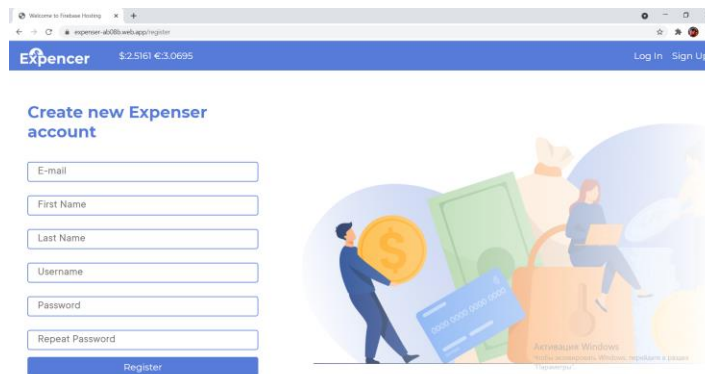


Рисунок 4.3 - Страница регистрации

Страница авторизации предлагает форму, в которой содержатся поля логина и пароля. Введённые пара данных логин-пароль сопоставляются с парой из Firebase Realtime Database (рисунок 4.4).

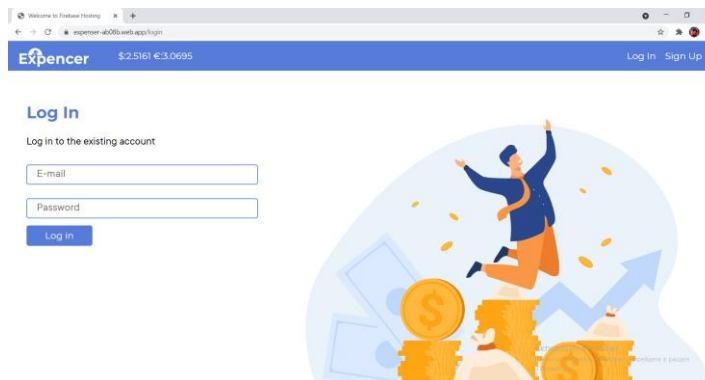


Рисунок 4.4 - Страница входа в систему

Для каждой строки выполняется базовая проверка на пустую строку – все строки должны быть заполнены.

4.3.2 Страница профиля

Страница профиля содержит информацию о текущем авторизованном пользователе: фотографию, логин, электронную почту, имя и фамилию указанные при регистрации (рисунок 4.5)

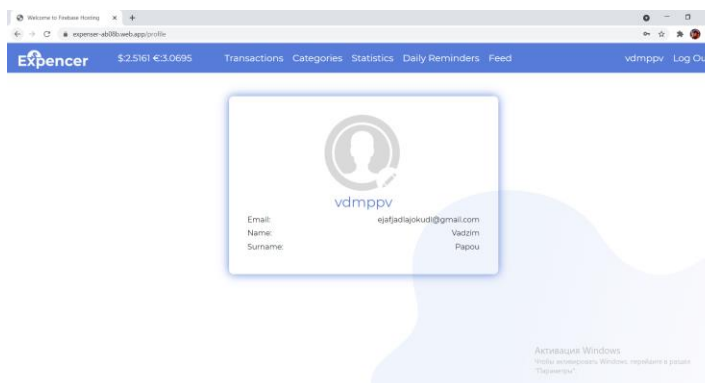


Рисунок 4.5 - Страница профиля

4.3.3 Страница с транзакциями

Страница с транзакциями содержит сетку с карточками транзакций авторизованного пользователя, на которых отображается сумма, место, описание и дата транзакции, а также кнопку добавления и удаления. Цвет транзакции соответствует цвету категории, к которой она относится (рисунок 4.6).

Логика отображения транзакций хранится в файлах `transactionsPage.js`, `transactionComponent.js`, `firebaseService.js`:

Листинг 4.3.3 – Отображение карточек транзакций

```
after_render: async () => {
  const user = firebase.auth().currentUser;
  const tableMain = document.querySelector(".table-main");

  firebaseService.getTransactions(user, async (data) => {
    let innerView = ``;
    if (!data.length) {
      tableMain.innerHTML = innerView;
      return;
    }
    for (const transaction of data) {
      const transactionComponent = TransactionComponent(transaction);

      innerView += await transactionComponent.render();
      await transactionComponent.after_render();
    }
    tableMain.innerHTML = innerView;
  });

  const addTransactionsButton = document.getElementById("add-transaction-button");
  addTransactionsButton.addEventListener("click", () => {
    showModal(AddTransactionModal);
  });

  tableMain.addEventListener("click", async (event) => {
    if (event.target.className.includes("category-block-button")) {
      const transactionUid = event.target.getAttribute("data-href");

      await firebaseService.removeTransaction(user, transactionUid);
    } else if (event.target.className.includes("fas fa-times")) {
      const transactionUid = event.target.parentNode.getAttribute("data-href");

      await firebaseService.removeTransaction(user, transactionUid);
    }
  });
}

export default TransactionsPage;

export const transactionComponent = (transaction) => {
  return {
```

```

        render: async () => {
            let user = firebase.auth().currentUser;
            let categories = await firebaseService.getCategoriesDict(user);
            let checkIncome = transaction["type"] == "income";
            let imageLink = transaction["image"] ? await firebaseService.retrieveImage(transaction["image"]) : "img/";

            let view = `
                <li>
                    <div class="transaction-card transaction-card-
margin" style="background: ${categories[transaction["category_id"]]["color"]};">
                        <div class="transaction-left">
                            <span class="transaction-
item">Place: ${transaction["place"]}</span>
                            <span class="transaction-
item">Desc: ${transaction["description"]}</span>
                            <span class="transaction-
item">Date: ${transaction["date"]}</span>
                        </div>
                        <div class="transaction-right">
                            <span class="transaction-
amount">${checkIncome ? "+" : "-"}${transaction["amount"]}</span>
                            
                        </div>
                    </div>
                </li>
            `;
            return view;
        },
        after_render: async () => {
        }
    }
}

export default transactionComponent;

getData(callback, ref) {
    ref.on("value", (snapshot) => {
        if (snapshot.exists()) {
            callback(Object.values(snapshot.val()));
        } else {
            callback([]);
        }
    });
}

transactionRef(uid) {
    return firebase.database().ref(`/transaction${uid}`);
}

getTransactions({uid}, callback) {
    this.getData(callback, this.transactionRef(uid));
}

async getTransactionsFromCategory({uid}, categoryId) {
    const snapshot = await this.transactionRef(uid).orderByChild("category_id").equalTo(categoryId).once("value");

```

```

        return Object.values(snapshot.val() ?? []);
    }

    async getTransactionsFromDate({uid}, fromDate) {
        const snapshot = await this.transactionRef(uid).orderByChild("date")
            .startAt(fromDate).once("value");
        return Object.values(snapshot.val() ?? []);
    }

    getCategories({uid}, callback) {
        this.getData(callback, this.categoryRef(uid));
    }

    async getCategoriesDict({uid}) {
        const snapshot = await this.categoryRef(uid).once("value");
        return snapshot.val();
    }
}

```

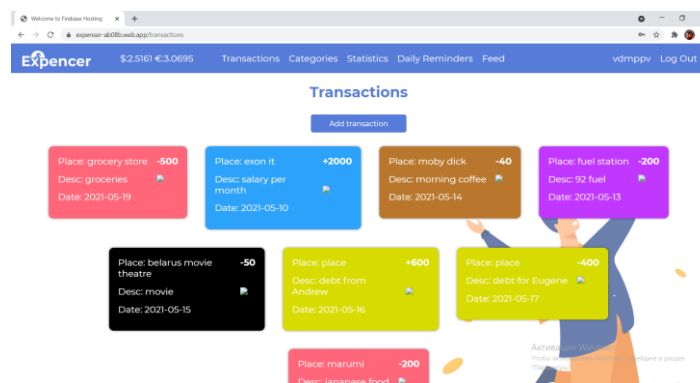


Рисунок 4.6 - Страница транзакций

4.3.4 Страница с категориями

Страница с категориями содержит сетку с карточками категорий, хранящими название, описание и цвет, а также кнопку добавления и удаления (рисунок 4.7).

Логика отображение категорий хранится в файлах categoriesPage.js, categoryComponent.js, firebaseService.js:

Листинг 4.3.4 – Отображение карточек категорий

```

after_render: async () => {
    const tableMain = document.querySelector(".table-main");
    const user = firebase.auth().currentUser;

    firebaseService.getCategories(user, async (data) => {
        let innerView = ``;
        if (!data.length) {
            tableMain.innerHTML = innerView;
            return;
        }
        for (const category of data) {
            const categoryComponent = CategoryComponent(category);
            innerView += await categoryComponent.render();
            await categoryComponent.after_render();
        }
    });
}

```

```

    }
    tableMain.innerHTML = innerView;
  });

  const addCategoriesButton = document.getElementById("add-button");
  addCategoriesButton.addEventListener("click", () => {
    showModal(AddCategoryModal);
  });
  tableMain.addEventListener("click", async (event) => {
    if (event.target.className.includes("category-block-button")) {
      const categoryUid = event.target.getAttribute("data-href");
      await firebaseService.removeCategory(user, categoryUid);
    } else if (event.target.className.includes("fas fa-times")) {
      const categoryUid = event.target.parentNode.getAttribute("data-href");
      await firebaseService.removeCategory(user, categoryUid);
    }
  });
}

export default CategoriesPage;

export const categoryComponent = (category) => {
  return {
    render: async () => {
      let view = `
        <li>
          <div class="category-block" style="background: ${category["color"]};">
            <div class="category-block-buttons">
              <button class="category-block-button"><i class="fas fa-cog"></i></button>
              <button class="category-block-button" id="category-component-delete" data-href="${category["uid"]}"><i class="fas fa-times"></i></button>
            </div>
            <div class="category-block-head">
              <span class="category-block-title">${category["title"]}</span>
            </div>
            <div class="category-block-main">
              <p class="category-block-text">${category["description"]}</p>
            </div>
          </div>
        </li>
      `;
      return view;
    },
    after_render: async () => {
    }
  }
}

export default categoryComponent;

categoryRef(uid) {
  return firebase.database().ref(`/category${uid}`);
}

```

```

    }

    getCategories({uid}, callback) {
      this.getData(callback, this.categoryRef(uid));
    }
  }
}

```

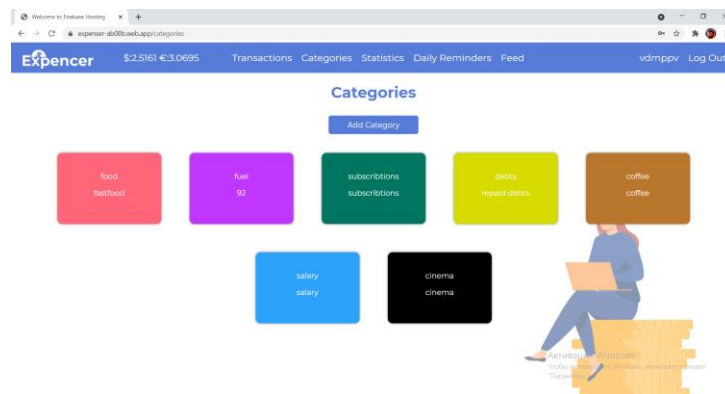


Рисунок 4.7 - Страница категорий

4.3.5 Страница со статистикой

Страница со статистикой позволяет отображать информацию о произведённых транзакциях в заданном периоде (рисунок 4.8) и по заданной категории в периоде (рисунок 4.9). Статистика в периоде содержит итог в виде круговых диаграмм по доходам и расходам. Таким образом можно отследить долю конкретной категории в доходах либо расходах. Для отображения круговых диаграмм используется библиотека с открытым кодом Chart.js.

Логика генерации круговых диаграмм на основе транзакций содержится в файле globalStatisticPage.js:

Листинг 4.3.5 – Генерация круговых диаграмм

```

async function getTransactionDateStatistic(user, fromDate, toDate) {

  const transactions = await firebaseService.getTransactionsFromDate(user, f
  romDate);

  var incomeTransactionsInfo = {};
  var incomeAmount = 0;
  var expenseTransactionsInfo = {};
  var expenseAmount = 0;
  for (const transaction of transactions) {
    if (transaction["date"] <= toDate) {
      if (transaction["type"] == "income") {
        incomeAmount += transaction["amount"];
        if (!incomeTransactionsInfo[transaction["category_id"]]) {
          incomeTransactionsInfo[transaction["category_id"]] = tra
          nsaction["amount"];
        } else {
          incomeTransactionsInfo[transaction["category_id"]] += tr
          ansaction["amount"];
        }
      } else {
        expenseAmount += transaction["amount"]
        if (!expenseTransactionsInfo[transaction["category_id"]]) {

```

```

        expenseTransactionsInfo[transaction["category_id"]] = transaction["amount"];
    } else {
        expenseTransactionsInfo[transaction["category_id"]] += transaction["amount"];
    }
}
}
return {
    "incomeTransactions": incomeTransactionsInfo,
    "incomeAmount": incomeAmount,
    "expenseTransactions": expenseTransactionsInfo,
    "expenseAmount": expenseAmount,
};
}

function createDataForChart(categoriesAmount, categories) {
    return {
        datasets: [{
            data: Object.values(categoriesAmount),
            backgroundColor: Object.keys(categoriesAmount).map(key => categories[key].color),
        }],
        labels: Object.keys(categoriesAmount).map(key => categories[key].title),
    }
}

function createPieChart(ctx, data) {
    return new Chart(ctx, {
        type: 'pie',
        data: data,
        options: {
            responsive: true,
            maintainAspectRatio: false
        }
    });
}

after_render: async () => {
    const user = firebase.auth().currentUser;
    const statisticCategoryButton = document.getElementById("statistic-category-button");
    statisticCategoryButton.onclick = () => {
        Router._instance.navigate("/statistics/category");
    }
    const chartContainer = document.getElementById("chart-pie-container");
    const form = document.querySelector(".statistics-main");
    const fromDate = document.getElementById("from-date");
    const toDate = document.getElementById("to-date");
    form.addEventListener("submit", async (event) => {
        event.preventDefault();
        const stat = await getTransactionDateStatistic(user, fromDate.value, toDate.value);
        const categories = await firebaseService.getCategoriesDict(user);

        chartContainer.innerHTML = chartComponent(stat);
    });
}

```

```

chart");
const expenseChartCanvas = document.getElementById("expense-
chart");
createPieChart(incomeChartCanvas, createDataForChart(stat["incom
eTransactions"], categories));
createPieChart(expenseChartCanvas, createDataForChart(stat["expe
nseTransactions"], categories));
});
}

```

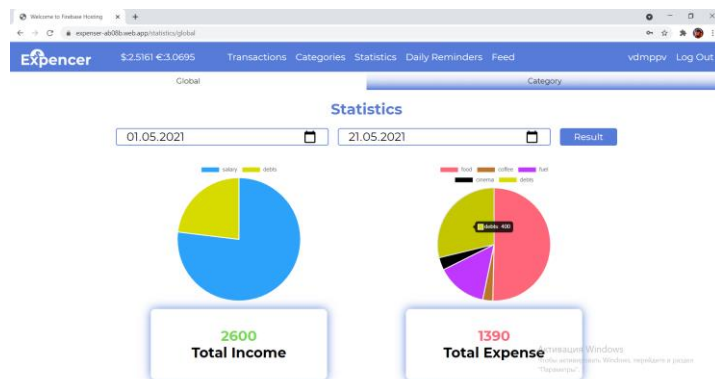


Рисунок 4.8 - Глобальная статистика

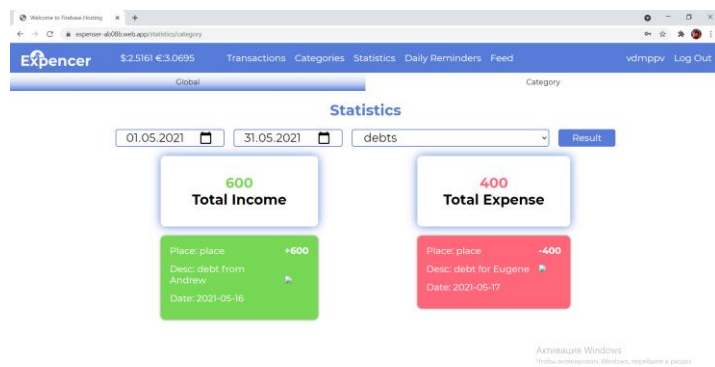


Рисунок 4.9 - Статистика по категориям

4.3.6 Страница с заметками

Страница с заметками отображает созданные напоминания — их название, описание, дату и время создания, время в которое должно сработать push уведомление, а также кнопки добавления и удаления (рисунок 4.10).

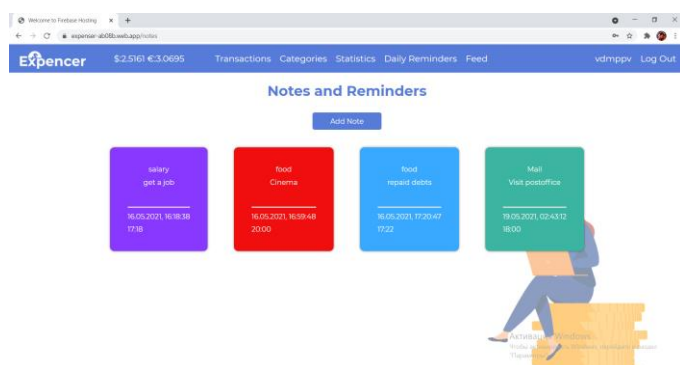


Рисунок 4.10 - Страница заметок и напоминаний

Push-уведомление срабатывает по таймеру заданному на карточке напоминания. В первой строке уведомления указано название заметки, во второй – её описание (рисунок 4.11).

Логика обработки push-уведомления описана в файле notesPage.js:

Листинг 4.3.6 – Обработка push-уведомления

```
function notifySet (note) {  
    var notification = new Notification (note["title"], {  
        tag : "Expencer",  
        body : note["description"]  
    });  
}  
function notifyMe(note) {  
    if (!("Notification" in window)) {  
  
        alert("This browser does not support desktop notification");  
    }  
    else if (Notification.permission === "granted" && currentTime === note["timeAlarm"]) {  
        notifySet(note);  
    }  
    else if (Notification.permission !== 'denied') {  
        Notification.requestPermission(function (permission) {  
            if (permission === "granted" && currentTime === note["timeAlarm"]) {  
                notifySet(note);  
            }  
        });  
    }  
}
```

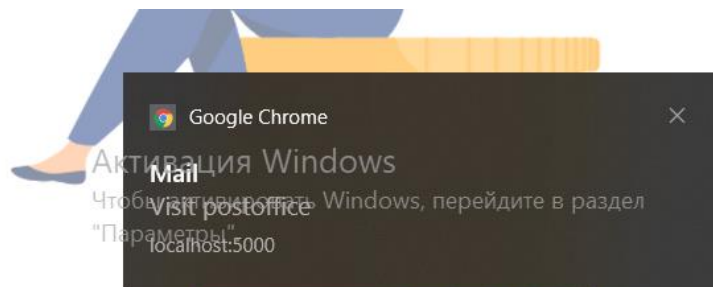


Рисунок 4.11 – push-уведомление

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование является неотъемлемым элементом разработки программного обеспечения. На него выделяется достаточно много времени в общем объеме трудозатрат, однако само по себе тестирование монотонный и однообразный процесс, поэтому хорошо поддается автоматизации.

Пока программное обеспечение использовало мало сторонних модулей и имело примитивный пользовательский интерфейс, для большинства задач хватало традиционных unit-тестов, проверяющих работоспособность отдельных модулей.

Модульное тестирование заключается в изолированной проверке каждого отдельного элемента путем запуска тестов в искусственной среде. Оценивая каждый элемент изолированно и подтверждая корректность его работы, установить проблему значительно проще чем, если бы элемент был частью системы.

Модульное тестирование также позволяет проводить рефакторинг, сохраняя корректную работоспособность модуля. Каждый тест вызывает определенный метод, передает ему тестовые параметры, проверяет результат работы метода. В ходе такого вызова из-за наличия операторов условного перехода в самом методе, некоторые операции могут выполняться, а некоторые – нет. Качество теста определяется покрытием всех операторов вызываемого метода, а также покрытием ветвей выполнения.

Исходя из частых проблемных мест, автотесты можно разделить на 3 основных вида:

1. Smoke-тесты для запуска собранного приложения в экстремальных: на самом старом и самом новом оборудовании. Это позволит убедиться в том, что, к примеру, обновление какой-либо зависимой библиотеки не повлияет на работу приложения.

2. Acceptance-тесты для проверки ключевых пользовательских сценариев. Это позволит убедиться, что последние правки не повлияли на сценарии, согласно которым будут действовать реальные пользователи в большинстве ситуаций.

3. Расширенные тесты для проверки верстки всех экранов на самых разных разрешениях, а также тесты для какой-либо специфической функциональности. Сюда же можно отнести и регрессионные автотесты. Для эффективного тестирования мобильных приложений можно также использовать различные инструменты, которые улучшают его качество и ускоряют весь процесс. Речь идет об эмуляторах, сервисах бета-тестирования, инструментах для сбора статистических данных.

В рамках тестирования данного приложения была поставлена задача покрыть тестами большинство пользовательских сценариев. Приложение успешно справилось со всеми из них.

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

6.1 Описание функций, назначения и потенциальных пользователей программного обеспечения

Разрабатываемый в дипломном проекте программный модуль предназначен для отслеживания доходов и расходов по категориям и периодам отдельного пользователя, для создания заметок/напоминаний, получения уведомлений

Разрабатываемый продукт представляет собой веб-сайт, позволяющий:

- отображать визуализацию общих доходов и расходов в периоде, а также доходов и расходов по категориям;
- отображать, создавать, редактировать и удалять заметки;
- хранить данные в облачной базе данных;
- работать адаптивно в веб-браузере как десктопных, так и мобильных устройств.

Среди потенциальной аудитории программного продукта можно выделить частных лиц, заинтересованных в удобном отслеживании своего бюджета, личных заметок и получения уведомлений. Модуль «Финансовый менеджер» позволит пользователю анализировать свои траты по заданным категориям, а также научит ограничивать и оптимизировать свой бюджет. Модуль «Заметки» позволит лёгким и доступным образом создавать себе напоминания о каких-либо грядущих событиях и получать уведомления по установленному таймеру.

Программное средство разрабатывается для свободной реализации на рынке информационных технологий и использования широким кругом потребителей.

Экономическое обоснование разработки и реализации ПО будет осуществляться в соответствии с п. 3.3. методического пособия [11].

6.2 Расчёт затрат на разработку программного обеспечения

Длительность разработки составит три месяца или 504 нормо-часов. Для разработки проекта «Персональный менеджер» необходимо привлечь трёх специалистов:

- менеджера проекта на 252 часа рабочего времени, должностной оклад которого равен 5208 руб./мес. (31 руб./ч.);
- программиста – на весь срок (504 часа), должностной оклад которого равен 4368 руб./мес. (26 руб./ч.);
- тестировщика на 120 часов, должностной оклад которого равен 2790 руб./мес. (15 руб./ч.);
- дизайнера на 120 часов, должностной оклад которого равен 3531 руб./мес. (19 руб./ч.).

Расчет затрат на основную заработную плату команды разработчиков

осуществляется по формуле:

$$З_0 = \sum_{i=1}^n З_{\text{чи}} \cdot t_i, \quad (6.2.1)$$

где n – количество исполнителей, занятых разработкой программного продукта;

$З_{\text{чи}}$ – часовая тарифная ставка i -го исполнителей, руб.;

t_i – трудоёмкость работ, выполняемых i -ым исполнителем, ч.

Расчёт затрат на основную заработную плату осуществляется в табличной форме (таблица 6.1).

Таблица 6.1 – Расчет затрат на основную заработную плату разработчиков

№	Участник команды	Должностной оклад, руб./ч.	Трудоемкость работ, ч.	Зарплата по тарифу, руб.
1	Менеджер проекта	31	252	7 812
2	Программист (разработчик веб-приложения и базы данных)	26	504	13 104
3	Тестирующий	15	120	1 800
4	Дизайнер	19	120	2 280
Итого затраты на основную заработную плату				24996

Затраты на дополнительную заработную плату команды разработчиков и определяется по формуле:

$$З_д = \frac{З_0 \cdot Н_д}{100}, \quad (6.2.2)$$

где $Н_д$ – норматив дополнительной заработной платы (25 %),

$З_0$ – затраты на основную заработную плату, руб.;

Затраты на дополнительную заработную плату составят:

$$З_д = \frac{24\,996 \cdot 25\%}{100\%} = 6\,249 \text{ руб.}$$

Отчисления на социальные нужды определяются в соответствии с действующими законодательными актами по формуле:

$$P_{\text{соц}} = \frac{(Z_o + Z_d) \cdot N_{\text{соц}}}{100}, \quad (6.2.3)$$

где $N_{\text{соц}}$ – норматив отчислений в ФСЗН и Белгосстрах (35 %).
Затраты на социальные нужды составят:

$$P_{\text{соц}} = (24996 + 6\,249) \cdot 0,35 = 10\,935,75 \text{ руб.}$$

Прочие затраты включаются в себестоимость разработки ПО в процентах от затрат на основную заработную плату команды разработчиков по формуле:

$$P_{\text{пр}} = \frac{Z_o \cdot N_{\text{пр}}}{100}, \quad (6.2.4)$$

где $N_{\text{пр}}$ – норматив прочих расходов, (35 %).
Прочие затраты составят:

$$P_{\text{пр}} = \frac{24\,996 \cdot 35\%}{100\%} = 8\,748,6 \text{ руб.}$$

Полная информация о формировании затрат на разработку программного средства приведена в таблице 6.2.

Таблица 6.2 – Затраты на разработку программного обеспечения

Статья затрат	Сумма, руб.
Основная заработная плата команды разработчиков, Z_o	24 996
Дополнительная заработная плата команды разработчиков, Z_d	6 249
Отчисления на социальные нужды, $P_{\text{соц}}$	10 935,75
Прочие затраты, $P_{\text{пр}}$	8 748,6
Всего	50 929,35

Таким образом, общая сумма затрат на разработку программного средства «Персональный менеджер» составит 50 929,35 руб.

6.3 Оценка эффекта от продажи программного обеспечения

Экономический эффект организации-разработчика программного средства представляет собой чистую прибыль от его продажи на рынке потребителям, величина которой зависит от объема продаж, цены реализации и затрат на разработку программного средства.

Организация-разработчик является налогоплательщиком (налог на прибыль), следовательно, экономический эффект можно рассчитать по формуле:

$$\Pi_{\text{ч}} = \Pi - \left(\Pi * \frac{H_{\text{п}}}{100\%} \right), \quad (6.3.1)$$

где Π – прибыль за использование программного продукта;

$H_{\text{п}}$ – ставка налога на прибыль, согласно законодательству, равная 18%.

Для оценки стоимости разработанного ПО уровень рентабельности определяется по формуле:

$$P_{\text{пр}} = \frac{\Pi(\Pi_{\text{ч}})}{Z_{\text{р}}} * 100\%, \quad (6.3.2)$$

Цена рассчитывается по формуле:

$$Ц = Z_{\text{р}} + \Pi + \text{НДС}, \quad (6.3.3)$$

Из формулы 6.3.2. следует, что Прибыль, включаемая в цену, рассчитывается по формуле:

$$\Pi = \frac{Z_{\text{р}} \cdot Y_{\text{р}}}{100\%}, \quad (6.3.4)$$

где $Z_{\text{р}}$ – затраты на разработку и реализацию ПО

$Y_{\text{р}}$ – запланированный норматив рентабельности, (по согласованию сторон равен 40%).

Налог на добавленную стоимость определяется по формуле:

$$\text{НДС} = \frac{(Z_{\text{р}} + \Pi) \cdot H_{\text{дс}}}{100\%}, \quad (6.3.5)$$

где $H_{\text{дс}}$ – ставка налога на добавленную стоимость в соответствии с действующим законодательством (20 %).

Прибыль равна:

$$\Pi = \frac{50\,929,35 \cdot 40\%}{100\%} = 20\,371,74 \text{ руб.}$$

Налог на добавленную стоимость равен:

$$\text{НДС} = \frac{(50\,929,35 + 20\,371,74) \cdot 20\%}{100\%} = 14\,260,22 \text{ руб.}$$

В таком случае, цена для заказчика равна:

$$\text{Ц} = 50\,929,35 + 20\,371,74 + 14\,260,22 = 85\,561,31 \text{ руб.}$$

Экономический эффект для разработчика равен:

$$\Pi_{\text{ч}} = 85\,561,31 - \left(85\,561,31 \cdot \frac{18\%}{100\%} \right) = 15\,401,04 \text{ руб.}$$

6.4 Расчёт показателей эффективности инвестиций в разработку программного обеспечения

Оценка экономической эффективности разработки и реализации программного средства на рынке зависит от результата сравнения инвестиций в его разработку (модернизацию, совершенствование) и полученного годового прироста чистой прибыли.

Был проведен социологический опрос среди потенциальных пользователей приложения (совершеннолетних работающих людей), в результате которого было выявлено, какую сумму они готовы ежемесячно тратить на данный продукт (рисунок 6.1-6.2).

Какую сумму Вы готовы потратить на приложение "Персональный Менеджер", которое позволит отслеживать и анализировать личные доходы/расходы, устанавливать напоминания, пролистывать ленту новостей и удобно просматривать статьи? *

☐ Бесплатное пользование

☐ Разовая оплата без подписочной системы

☐ До 2\$ в месяц

☐ До 3\$ в месяц

☐ До 5\$ в месяц

Рисунок 6.1 - Условие социологического опроса

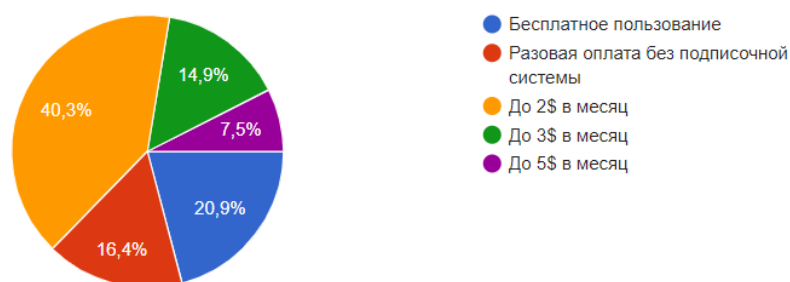


Рисунок 6.2 - Результат социологического опроса

Из социологического опроса следует, что оптимальная стоимость месячной подписки за пользование приложением - 2\$ (5 руб.).

Из пункта 2 следует, что разработка длилась 3 месяца – с января 2021 года до апреля 2021 года.

Для привлечения пользователей и дальнейшего увеличения количества подписчиков был произведён анализ рынка на предмет стоимости контекстной рекламы, администрирования баз данных, а также на продвижение веб-приложения в поисковых системах Google и Yandex. Затраты в течении девяти месяцев на привлечение пользователей и администрирование приложения:

- затраты на контекстную рекламу в социальных сетях и прочих веб-сайтах [12]: 700 руб. * 9 месяцев = 6 300 руб;
- затраты на продвижение в поисковых системах Google, Yandex [13]: (600 руб. + 350 руб.) * 9 месяцев = 8 550 руб;
- затраты на администрирование [14]: 400 руб. * 9 месяцев = 3 600 руб.

Общая сумма затрат на поддержку и продвижение программного средства в течение 9 месяцев будет равна:

$$З_{\text{тек}} = 6\,300 + 8\,550 + 3\,600 = 18\,450 \text{ руб.}$$

Можно спрогнозировать рост числа пользователей разрабатываемого веб-приложения в в первый год реализации проекта с помощью условий, гарантий и возможностей, которые предоставляют услуги рекламы, продвижения и администрирования программного продукта:

- 1-ый месяц: около 500 пользователей;
- 2-ой месяц: около 1 000 пользователей;
- 3-ий месяц: около 1 700 пользователей;
- 4-ый месяц: около 2 400 пользователей;
- 5-ый месяц: около 3 400 пользователей;
- 6-ой месяц: около 4 400 пользователей;
- 7-ой месяц: около 5 700 пользователей;
- 8-ой месяц: около 7 000 пользователей;
- 9-ый месяц: около 8 500 пользователей.

Из этого следует, что годовая прибыль равна:

$$\begin{aligned} \Pi &= (500 * 5 + 1\,000 * 5 + 1\,700 * 5 + 2\,400 * 5 + 3\,400 * 5 + \\ &+ 4\,400 * 5 + 5\,700 * 5 + 7\,000 * 5 + 8\,500 * 5) - Z_{\text{тек}} - \text{НДС} = \\ &= 173\,000 - 18\,450 - 34\,600 = 119\,950 \text{ руб.} \end{aligned}$$

Тогда, чистая прибыль равна:

$$\Pi_{\text{ч}} = \Pi - \left(\Pi * \frac{H_{\text{п}}}{100\%} \right) = 119\,950 - (119\,950 * 0,18) = 98\,359 \text{ руб.}$$

Экономический эффект равен:

$$\mathcal{E}_э = \Pi_{\text{ч}} - Z_{\text{р}} = 98\,359 - 50\,929,35 = 47\,429,65 \text{ руб.}$$

Оценка экономической эффективности инвестиций в разработку программного средства осуществляется с помощью расчета рентабельности инвестиций по формуле:

$$P_{\text{и}} = \frac{\Pi_{\text{ч}}}{Z_{\text{р}}} \cdot 100 \%, \quad (4.1)$$

где $\Pi_{\text{ч}}$ – прирост чистой прибыли, руб.;

$Z_{\text{р}}$ – затраты на разработку программного средства, руб.

Таким образом, рентабельность инвестиций будет равна:

$$P_{\text{и}} = \frac{98359}{50\,929,35} \cdot 100 \% = 193,13\%$$

Инвестиции на разработку программного средства и его реализация на рынке информационных технологий будут экономически эффективными, если рентабельность инвестиций превысит 100 % (100 % плюс ставка по банковским долгосрочным депозитам). А поскольку ставка по долгосрочным депозитам не превышает 15%, следовательно, программное средство целесообразно разрабатывать и реализовывать по установленной цене, т.к. рентабельность инвестиций превышает 115%.

Спустя год после внедрения данного программного средства заказчик не только покрывает собственные затраты, но и имеет прибыль. В свою очередь исполнитель также получает прибыль в короткие сроки.

На основании данных результатов можно сделать вывод, что проект представляется выгодным как для разработчика, так и для инвестора: реализация программного средства на рынке экономически эффективна.

ЗАКЛЮЧЕНИЕ

В настоящем документе описан полный цикл разработки программного средства для анализа личных доходов и расходов.

Основная часть работы посвящена сервисам, которые предоставляет платформа Firebase – один из лидеров рынка облачных технологий. Были использованы сервисы аутентификации, базы данных, хранилища и отправки сообщений.

Результатом дипломного проектирования стало веб-приложение, благодаря использованию которого пользователь сможет постоянно следить за изменением своих доходов и расходов, наблюдать за статистикой по категориям, оставлять себе заметки, получать уведомления. Наглядность и простота отображения данных, а также расположение отчетов непосредственно в мобильном телефоне привлекут большое количество заинтересованных пользователей.

Последующее развитие приложения представляет из себя интеграцию с банковским приложением, включающую в себя мобильное приложение, чтение и анализ смс от банков и реальных банковских транзакций. Таким образом веб-приложение составит конкуренцию существующим менеджерам финансов, опережая их по доступности, удобству и простоте использования.

В ходе разработки программного средства были не только получены новые навыки, но также закреплены значения, полученные за время обучения в университете.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Список используемых источников

- 1 Облачное хранилище что это: 10 облачных хранилищ, на которые стоит обратить внимание [Электронный ресурс]. Режим доступа: <https://iapple-59.ru/raznoe/oblachnoe-xranilishhe-cto-eto-10-oblachnyx-xranilishh-na-kotorye-stoit-obratit-vnimanie.html> Дата доступа: 13.05.21
- 2 Как работает облачное хранилище данных, его достоинства и недостатки [Электронный ресурс]. Режим доступа: <https://itelon.ru/blog/oblachnye-sistemy-khraneniya/> Дата доступа: 13.05.21
- 3 Облачные хранилища данных: для чего они нужны и какие типы существуют – База знаний Timeweb Community [Электронный ресурс]. Режим доступа: <https://timeweb.com/ru/community/articles/typy-oblachnih-hranilishch-dannyh> Дата доступа: 13.05.21
- 4 Топ-7 приложений для контроля расходов | Компьютерра [Электронный ресурс]. Режим доступа: <https://www.computerra.ru/263304/top-7-prilozhenij-dlya-kontrolya-rashodov/> Дата доступа: 13.05.21
- 5 Что такое HTML? Основы языка разметки гипертекста [Электронный ресурс]. Режим доступа: <https://www.hostinger.ru/rukovodstva/shto-takoe-html/> Дата доступа: 13.05.21
- 6 Основы HTML - Изучение веб-разработки | MDN [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/HTML_basics Дата доступа: 13.05.21
- 7 Что такое CSS, для чего нужны каскадные таблицы стилей CSS [Электронный ресурс]. Режим доступа: <https://blog.ingate.ru/seo-wikipedia/css/> Дата доступа: 13.05.21
- 8 Основы JavaScript - Изучение веб-разработки | MDN [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/JavaScript_basics Дата доступа: 13.05.21
- 9 Что такое Firebase? [Электронный ресурс]. Режим доступа: <https://avada-media.ua/services/firebase/> Дата доступа: 13.05.21
- 10 GitHub — Википедия [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/GitHub> Дата доступа: 13.05.21
- 11 Техничко-экономическое обоснование дипломных проектов (работ). Методические указания для студентов 1 ступени высшего образования специальностей, закрепленных за УМО/ А.А. Горюшкин, А.В. Грицай, В. Г. Горовой. –Минск.: БГУИР, 2020–100с.
- 12 Продвижение в социальных сетях: раскрутка бизнеса ВК, ФБ, Инстаграм [Электронный ресурс]. Режим доступа: <https://1ps.ru/cost/smm/> Дата доступа: 13.05.2021
- 13 Продвижение сайтов в Яндексe и Google [Электронный ресурс].

Режим доступа: <https://1ps.ru/blog/dirs/stoimost-prodvizheniya-sajta-v-yandekse-i-google/> Дата доступа: 13.05.2021

14 Администрирование веб-сервера | Настройка и техническое обслуживание веб-сервера [Электронный ресурс]. Режим доступа: <https://hoster.by/service/solutions/administration-web/> Дата доступа: 13.05.2021.

ПРИЛОЖЕНИЕ А

(обязательное)

Текст программного модуля

```
import {firebaseService} from "../services/index.js";
import {categoryStatisticComponent} from "../components/categoryStatisticComponent.js";
import {selectCategoryComponent} from "../components/selectCategoryComponent.js";
import {Router} from "../router.js";

let CategoryStatisticPage = {
  render: async () => {
    let user = firebase.auth().currentUser;
    let categories = await firebaseService.getCategoriesList(user);
    let view = `
      <div>
        <nav>
          <ul class="main-nav">
            <li class="main-nav-item main-nav-item-clicked" id="statistic-category-global">
              <a>Global</a>
            </li>
            <li class="main-nav-item">
              <a>Category</a>
            </li>
          </ul>
        </nav>
      </div>
      <div>
        <h1 class="page-title">Statistics</h1>
      </div>

      <div class="statistics-table">
        <form class="statistics-main">
          <div class="statistics-inputs">
            <input type="date" id="from-date" class="statistics-input">

            <input type="date" id="to-date" class="statistics-input">
            ${await selectCategoryComponent("statistics-input statistics-select", categories).render()}
            <button class="statistics-button" type="submit">Result</button>
          </div>

          <div class="statistics-transactions" id="statistics-transactions">

          </div>
        </form>
      </div>
    `;
    return view;
  },
  after_render: async () => {
    const user = firebase.auth().currentUser;
    const form = document.querySelector(".statistics-main");
```

```

        const statisticContainer = document.getElementById("statistics-
transactions");
        const fromDate = document.getElementById("from-date");
        const toDate = document.getElementById("to-date");
        const categorySelector = document.getElementById("category-selector");

        form.addEventListener("submit", async (event) => {
            event.preventDefault();
            const selectedCategoryId = categorySelector.options[categorySelector.
selectedIndex].id;
            const stat = await getCategoryStatistic(user, fromDate.value, toDate.
valueOf, selectedCategoryId);
            statisticContainer.innerHTML = await categoryStatisticComponent(stat)
;
        });

        const statisticCategoryGlobal = document.getElementById("statistic-
category-global");
        statisticCategoryGlobal.onclick = () => {
            Router._instance.navigate("/statistics/global");
        }
    }
}

async function getCategoryStatistic(user, fromDate, toDate, categoryId) {
    const transactions = await firebaseService.getTransactionsFromCategory(user,
categoryId);
    var incomeTransactionsInfo = [];
    var incomeAmount = 0;
    var expenseTransactionsInfo = [];
    var expenseAmount = 0;
    for (const transaction of transactions) {
        if (fromDate <= transaction["date"] && transaction["date"] <= toDate) {
            if (transaction["type"] == "income") {
                incomeTransactionsInfo.push(transaction);
                incomeAmount += transaction["amount"];
            } else {
                expenseTransactionsInfo.push(transaction);
                expenseAmount += transaction["amount"];
            }
        }
    }
    return {
        "incomeTransactions": incomeTransactionsInfo,
        "incomeAmount": incomeAmount,
        "expenseTransactions": expenseTransactionsInfo,
        "expenseAmount": expenseAmount,
    };
}

export default CategoryStatisticPage;

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Классы данных приложения

```
export class Category {
  constructor({title, description, color}) {
    this.title = title;
    this.description = description;
    this.color = color;
  }
}

export class Note {
  constructor({title, description, color, timeCreated, timeAlarm}){
    this.title = title;
    this.description = description;
    this.color = color;
    this.timeCreated = timeCreated;
    this.timeAlarm = timeAlarm;
  }
}

export class Transaction {
  constructor({amount, place, description, category_id, date, type, image}) {
    this.amount = amount;
    this.place = place;
    this.description = description;
    this.category_id = category_id;
    this.date = date;
    this.type = type;
    this.image = image;
  }
}
```

ПРИЛОЖЕНИЕ В

(обязательное)

Класс-маршрутизатор приложения

```
import Utils from "../helpers/utils.js"
import Header from "../components/header.js"
import NotFoundPage from "../pages/notFoundPage.js"

export class Router {
  static _instance = null;
  routes;

  constructor(routes) {
    this.routes = routes;
    window.addEventListener('popstate', event => this._onPopState(event));
  }

  _onPopState() {
    this.loadPage(this.parseCurrentURL());
  }

  static init(routes) {
    if (Router._instance != null) {
      return Router._instance;
    }

    const path = window.location.pathname;
    window.history.replaceState({path}, path, path);
    const router = new Router(routes);
    Router._instance = router;
    firebase.auth().onAuthStateChanged(() => {
      router._loadInitial();
    });
    return router;
  }

  navigate(url, render = true) {
    history.pushState({}, "", url);
    this.loadPage(url);
  }

  async loadPage(url) {
    const content = document.getElementById('page_id');

    const header = document.getElementById('header_id');
    header.innerHTML = await Header.render();
    await Header.after_render();

    let currentPage = NotFoundPage;
    for (const {path, page} of Router._instance.routes) {
      if (path === url) {
        currentPage = page;
      }
    }
    content.innerHTML = await currentPage.render();
    await currentPage.after_render();
  }
}
```



```
parseCurrentURL() {  
    let request = Utils.parseRequestURL()  
    return (request.resource ? '/' + request.resource : '/')  
        + (request.id ? '/' + request.id : '')  
        + (request.verb ? '/' + request.verb : '');  
}  
  
async _loadInitial() {  
    this.navigate(this.parseCurrentURL());  
}  
}
```

Обозначение					Наименование		Дополнительные сведения		
					Текстовые документы				
БГУИР ДП 1–40 04 01 054 ПЗ					Пояснительная записка		58 с.		
					Отзыв руководителя				
					Рецензия				
					Акт о внедрении				
					Графические документы				
ГУИР.753503-01 СП					Диаграмма декомпозиции веб-приложения.		Формат А4		
					Схема программы				
ГУИР.753503-02 СП					Диаграмма способов использования приложения.		Формат А4		
					Схема программы				
ГУИР.753503-03 СП					Схема-анализ структуры JSON.		Формат А4		
					Схема программы				
ГУИР.753503.001 ПЛ					Общая структура системы.		Формат А4		
					Плакат				
ГУИР.753503.002 ПЛ					Схема физической модели базы данных.		Формат А4		
					Плакат				
ГУИР.753503.003 ПЛ					Экранные формы программы.		Формат А4		
					Плакат				