МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой № 43

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| д-р техн. наук, профессор |  |  |  | М.Ю. Охтилев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

БАКАЛАВРСКАЯ РАБОТА

|  |  |
| --- | --- |
| на тему | Разработка мобильного приложения для управления личным бюджетом |
|  | |
|  | |

|  |  |
| --- | --- |
| выполнена | Кашиной Верой Андреевной |
| фамилия, имя, отчество студента в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.04 |  | Программная инженерия |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 02 |  | Проектирование программных систем |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы № | Z0431 |  |  |  | В.А. Кашина |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, канд. техн. наук, |  |  |  | А.А. Фоменкова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой №43

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

|  |  |  |  |
| --- | --- | --- | --- |
| студенту группы | Z0431 |  | Кашиной Вере Андреевне |
|  | номер |  | фамилия, имя, отчество |

|  |  |
| --- | --- |
| на тему | Разработка мобильного приложения для управления личным бюджетом |
|  | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| утвержденную приказом ГУАП от | 28.03.2025 | № | 11-390/25 |

|  |  |
| --- | --- |
| Цель работы: | Разработка кроссплатформенного приложения для учета личного бюджета |
| с аналитикой и геймификацией. | |
|  | |

|  |  |
| --- | --- |
| Задачи, подлежащие решению: | Создание кроссплатформенного приложения для |
| управления бюджетом, визуализация доходов и расходов, мотивирование пользователей с | |
| помощью внедрения системы достижений | |

|  |  |
| --- | --- |
| Содержание работы (основные разделы): | Введение, анализ предметной области, |
| разработка и анализ требований к программе, разработка программы, тестирование, | |
| заключение | |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Срок сдачи работы « | 06 | » | июня | 20 | 25 |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, к.т.н. |  |  |  | А.А. Фоменкова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Задание принял(а) к исполнению

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| студент группы № | Z0431 |  |  |  | В.А. Кашина |
|  |  |  | подпись, дата |  | инициалы, фамилия |

**РЕФЕРАТ**

Выпускная квалификационная работа содержит 148 страниц, 55 рисунков, 2 таблицы, 17 источников, 2 приложения.

Ключевые слова: личный бюджет, мобильное приложение, учет финансов, аналитика, кроссплатформенная разработка.

Актуальность: в условиях нестабильной экономической ситуации все больше людей стремятся к осонознанному управлению личными финансами. Существующие решения не всегда учитывают индивидуальные предпочтения пользователей и могут быть перегружены функциональностью. Поэтому разработка удобного и интуитивного приложения для ведения личного бюджета остается актуальной задачей.

Цель работы: разработка кроссплатформенного мобильного приложения для учета доходов и расходов, с поддержкой аналитики, визуализации финансовой информации и системой достижений, которая будет позволять мотивировать пользователя к регулярному использованию сервиса.

В процессе использовались программные средства: .NET MAUI (для создания кроссплатформенного мобильного клиента), ASP.NET Core (для реализации серверного API), PostgreSQL (СУБД), EF Core (для работы с базой данных), Swagger (для тестирования API).

Примененные языки программирования: C#, XAML.

Полученные результаты: разработано кроссплатформенное мобильное приложение, позволяющее пользователю вести учет доходов и расходов, просматривать аналитику в виде диаграмм и отчетов, сканировать QR-коды чеков для автоматического заполнения транзакций, получать достижения и уведомления, экспортировать отчеты в PDF.

Область применения: приложение может использоваться пользователями для ведения личного бюджета и формирования привычек финансового планирования.

**ABSTRACT**

The final qualifying work contains 148 pages, 55 figures, 2 tables, 17 sources, 2 appendices.

Keywords: personal budget, mobile application, financial accounting, analytics, cross-platform development.

Relevance: in the context of an unstable economic situation, more and more people strive for conscious management of personal finances. Existing solutions do not always take into account the individual preferences of users and can be overloaded with functionality. Therefore, the development of a convenient and intuitive application for maintaining a personal budget remains an urgent task.

Objective of the work: development of a cross-platform mobile application for accounting of income and expenses, with support for analytics, visualization of financial information and a system of achievements that will motivate the user to regularly use the service.

The following software tools were used in the process: .NET MAUI (to create a cross-platform mobile client), ASP.NET Core (to implement the server API), PostgreSQL (DBMS), EF Core (to work with the database), Swagger (for testing the API).

Programming languages used: C#, XAML.

Results obtained: a cross-platform mobile application has been developed that allows the user to keep track of income and expenses, view analytics in the form of charts and reports, scan QR codes of receipts to automatically fill in transactions, receive achievements and notifications, and export reports to PDF.

Scope: the application can be used by users to manage their personal budget and develop financial planning habits.

**СОДЕРЖАНИЕ**

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 8](#_Toc200226352)

[ВВЕДЕНИЕ 9](#_Toc200226353)

[1 Анализ предметной области 11](#_Toc200226354)

[1.1 Анализ предметной области 11](#_Toc200226355)

[1.1.1 Актуальность проблемы 11](#_Toc200226356)

[1.1.2 Сравнение существующих аналогов 12](#_Toc200226357)

[1.2 Цели и задачи разработки программы 19](#_Toc200226358)

[2 Разработка и анализ требований 21](#_Toc200226359)

[2.1 Функциональные требования 21](#_Toc200226360)

[2.2 Нефункциональные требования 25](#_Toc200226361)

[2.3 Требования к пользовательскому интерфейсу 25](#_Toc200226362)

[2.4 Выбор средств реализации 27](#_Toc200226363)

[2.4.1 Язык программирования 27](#_Toc200226364)

[2.4.2 Клиентская часть 28](#_Toc200226365)

[2.4.3 Серверная часть 28](#_Toc200226366)

[2.4.4 База данных 29](#_Toc200226367)

[2.4.5 Дополнительные библиотеки 29](#_Toc200226368)

[3 Разработка программы 31](#_Toc200226369)

[3.1 Проектирование архитектуры программы 31](#_Toc200226370)

[3.2 Описание данных, структур данных, файлов 34](#_Toc200226371)

[3.3 Описание сервисов и окружения приложения 38](#_Toc200226372)

[4 Тестирование 41](#_Toc200226373)

[4.1 Тестирование регистрации и авторизации 41](#_Toc200226374)

[4.2 Тестирование взаимодействия с транзакциями, получения достижений  
 и страницы профиля 44](#_Toc200226375)

[4.3 Тестирование аналитики и отчетов 60](#_Toc200226376)

[4.4 Тестирование ежедневного уведомления 65](#_Toc200226377)

[ЗАКЛЮЧЕНИЕ 66](#_Toc200226378)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 67](#_Toc200226379)

[ПРИЛОЖЕНИЕ А (обязательное) Код миграции 69](#_Toc200226380)

[ПРИЛОЖЕНИЕ Б (обязательное) Основной код программы 73](#_Toc200226381)

# **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

В настоящем отчете о ВКР применяют следующие сокращения и обозначения:

.NET – платформа с открытым исходным кодом для создания настольных, мобильных и веб-приложений, которые могут работать в любой операционной системе

UI – User Interface (пользовательский интерфейс)

API – Application Programming Interface (программный интерфейс приложения)

БД – база данных

HTTP – HyperText Transfer Protocol (протокол для получения ресурсов)

JWT-токен – JSON Web Token (открытый стандарт для создания токенов доступа, основанный на формате JSON)

QR-код – Quick Response code (двумерный штрихкод, который считывается устройствами обработки изображений)

EF Core – Entity Framework Core

DFD – data flow diagrams

# **ВВЕДЕНИЕ**

В условиях стремительного развития цифровых технологий, мобильные приложения уже стали неотъемлемой частью жизни практически любого человека, которые затрагивают все сферы жизни, включая и экономику. Цифровизация экономики приводит к росту безналичных платежей, увеличению количества кредитных предложений, увеличению популярности электронных торговых площадок и распространению бизнес-моделей, при которых потребитель циклично платит определенную сумму за определенный период. При этом, все сложнее становится управлять движением денежных средств, что приводит к тому, что многие люди теряют контроль над количеством потраченных денег. Кроме того, многие люди не обладают навыками равномерного планирования и ведения бюджета, что в свою очередь влияет на появление долгов, невозможность накопить на реализацию поставленных целей и, в целом, усиливает финансовые трудности.

В этой ситуации умение грамотно распоряжаться деньгами, в том числе анализировать расходы и доходы, минимизировать ненужные траты и следовать финансовому плану, является ключевым навыком. Для повышения финансовой грамотности и систематического анализа своих трат и доходов стоит начать с малого – например, отслеживать движения собственных финансов, используя ручное ведение записей доходов и затрат, после чего можно проводить простые анализы полученных результатов. Однако данный метод не является наилучшим, так как не всегда есть возможность вести учет затрат и аналитика по полученным данным не всегда является объективной.

Таким образом, необходимо иметь возможность автоматизировать учет и анализ, поэтому важно иметь доступное и полезное мобильное приложение, которое позволит компенсировать имеющиеся недостатки ручного учета и получить преимущества цифровизации.

Данное приложение должно помочь решить проблемы автоматизации контроля за финансами, проведения анализа затрат и получение графического отображения движения финансов. Наглядное представление движения финансов будет помогать пользователю более грамотно вести собственный бюджет, ставить перед собой личные цели по улучшение финансового благополучия и как следствие, выработать здоровые финансовые привычки. В результате пользователи получают возможность не только значительно упростить повседневные задачи, но и принимать более взвешенные решения, ориентированные на долгосрочные цели [1].

Целью данной выпускной квалификационной работы является разработка мобильного приложения для ведения личного бюджета, которое поможет пользователям удобно и эффективно проводить анализ своих финансов. Для повышения мотивации в приложении будет реализована система достижений, стимулирующая пользователей к оптимальному расходу средств и достижению поставленных целей.

Задачи бакалаврской работы:

* изучить аналоги данного приложения и провести их исследование,
* изучить теоретические основы реализации мобильного приложения,
* реализовать полнофункциональное приложение с использованием актуальных фреймворков и инструментов разработки,
* проведение комплексного тестирования разработанного приложения, включая проверку корректности работы алгоритмов финансовой аналитики.

1. **Анализ предметной области**
   1. **Анализ предметной области**
2. **Актуальность проблемы**

Бюджет – это сбалансированный план доходов и расходов физического лица, организации или государства за определенный период времени, служащий инструментом контроля над финансовыми потоками и достижения экономических целей [2].

Личным бюджетом является персональный план доходов (заработная плата, доходы от бизнеса, стипендии) и расходов (оплата транспорта, погашение кредитов, покупка продуктов питания) на определенный период.

В условиях стремительной цифровизации экономики контроль личного бюджета становится не просто полезной привычкой, а необходимым инструментом для финансовой стабильности и безопасности.

Даже учитывая то, что использование средств автоматизации финансового планирования приносит большую пользу, например, позволяет избежать ненужных трат и откладывать средства на поставленную цель, все равно можно столкнуться с неудобствами и проблемами при использовании.

Основными неудобствами и проблемами являются:

* трудности с регулярным внесением данных,
* отсутствие понятной аналитики, позволяющей оценить соотношение доходов на расходы,
* отсутствие мотивации:

1. многие избегают учета, потому что боятся увидеть свои долги, ненужные траты или несоответствия доходов и расходов;
2. безналичные платежи и кредитные карты создают ощущение, что деньги «не настоящие», а значит, их проще тратить;
3. недостаток знаний: люди не понимают, как бюджет влияет на их благосостояние [3].

* большое количество финансовых инструментов:

1. цифровые кошельки, криптовалюта, инвестиционные платформы и кредитные карты требуют системного отслеживания;
2. риск потери средств из-за мошенничества или технических сбоев повышает важность учета.

Основные компоненты учета личных финансов:

* фиксация операций, включающая в себя учет расходов, запись поступлений денежных средств,
* категоризация операций для детализации,
* анализ данных, например, построение графиков,
* планирование, в которое входит прогнозирование будущих расходов, создание финансовых целей.

Актуальность данной работы заключается в том, что учет личного бюджета является важной частью жизни многих современных людей. При этом цифровизация экономики делает учет финансов сложнее из-за большого количества финансовых инструментов и рисков. Поэтому необходимо использовать безопасные и проверенные автоматизированные сервисы для учета доходов и расходов, регулярно анализировать свои финансовые потоки и соблюдать цифровую гигиену для защиты данных.

1. **Сравнение существующих аналогов**

Современный рынок насыщен большим количеством приложений для учета финансов и поэтому выбрать действительно безопасное и хорошее приложение может вызывать проблемы у пользователей, которые не могут точно определить, какое приложение лучше всего соответствует их потребностям, сочетает в себе удобство, функциональность, безопасность и надежность.

Анализ будет проводится на основе оценки функциональных и эксплуатационных характеристик, для этого необходимо:

* определить ключевые потребности (простой учет, инвестиции, семейный бюджет),
* проверить репутацию разработчика (сколько лет на рынке, отзывы пользователей, прозрачность политики данных),
* протестировать бесплатную версию (если имеется),
* убедиться в безопасности (шифрование, аутентификация).

Для анализа были выбраны следующие популярные мобильные приложения:

* «Дзен-мани»,
* Monefy,
* MoneyLover,
* Wallet [4].

Для сравнения выбранных приложений можно выделить следующие ключевые критерии:

* функциональность:

а) учет доходов и расходов:

1. поддержка ручного и автоматического ввода операции;
2. категории и подкатегории (настраиваемые или готовые);
3. шаблоны и повторяющиеся платежи;

б) бюджетирование:

1. установка лимитов по категориям;
2. отчеты о перерасходе;
3. планирование долгосрочных целей (накопления);

в) отчеты и аналитика:

1. графики (круговые, линейные, столбчатые);
2. фильтры по времени, категориям, счетам;
3. экспорт данных (PDF, Excel, CSV);

г) мультивалютность и счета:

1. поддержка нескольких валют с конвертацией;
2. разные типы счетов (карты, наличные, депозиты);

д) синхронизация и автоматизация:

1. импорт банковских выписок (ручной или автоматический);
2. подключение банков (Open Banking, API);
3. резервное копирование (облачно, локально);

* удобство интерфейса:

а) простота добавления операций (быстрый ввод, виджеты);

б) наглядность статистики (дашборды, цветовые схемы);

в) кастомизация (темы, группировка категорий);

г) мультиплатформенность (iOS, Android, Web, Desktop);

* безопасность и конфиденциальность:

а) защита данных (пароль/PIN/биометрия, шифрование данных);

б) оффлайн-режим (работа без интернета);

в) политика конфиденциальности (хранение данных, передача третьим лицам);

* дополнительные возможности:

а) кредиты и долги (учет займов, напоминания);

б) инвестиции (отслеживание портфеля);

в) совместный учет (семейный/групповой доступ);

г) интеграции (банки, бухгалтерские сервисы).

Проведем оценку каждого приложения по выбранным критериям. Результаты проведенной оценки представлены в таблице 1.

Таблица 1 – Оценка выбранных приложения по критериям

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерии | | Приложения | | | |
| Дзен-мани | Monefy | Money Lover | Wallet |
| 1 | | 2 | 3 | 4 | 5 |
| Функциональность | | | | | |
| Учет расходов и расходов | поддержка ручного и автоматического ввода операции; | + | + | + | + |
| категории и подкатегории (настраиваемые или готовые); | + | + | + | + |
| шаблоны и повторяющиеся платежи; | + | + | + | + |
| Бюджетирование | установка лимитов по категориям; | + | + | + | + |
| отчеты о перерасходе; | + | - | + | + |
| планирование долгосрочных целей (накопления); | + | - | + | + |
| Отчеты и аналитика | графики (круговые, линейные, столбчатые); | + | + | + | + |
| фильтры по времени, категориям, счетам; | + | + | + | + |
| экспорт данных (PDF, Excel, CSV); | - | - | + | + |
| Мультивалютность и счета | поддержка нескольких валют с конвертацией; | - | + | + | + |
| резные типы счетов (карты, наличные, депозиты); | - | + | + | + |

Продолжение таблицы 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | | 2 | 3 | 4 | 5 |
| Синхронизация и автоматизация | импорт банковских выписок (ручной или автоматический); | - | + | + | + |
| подключение банков (Open Banking, API); | - | - | + | + |
| резервное копирование (облачно, локально). | + | + | + | + |
| Удобство интерфейса | | | | | |
| Простота добавления операций (быстрый ввод, виджеты); | - | + | + | + | + |
| Наглядность статистики (дашборды, цветовые схемы); | - | + | + | + | + |
| Кастомизация (темы, группировка категорий); | - | - | + | + | + |
| Мультиплатформенность (iOS, Android, Web, Desktop) | - | - | - | + | + |
| Безопасность и конфиденциальность | | | | | |
| Защита данных (пароль/PIN/биометрия, шифрование данных); | - | + | + | + | + |
| Оффлайн-режим (работа без интернета); | - | + | + | - | + |
| Политика конфиденциальности (хранение данных, передача третьим лицам). | - | + | + | + | + |

Продолжение таблицы 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Дополнительные возможности | | | | | |
| 1 | | 2 | 3 | 4 | 5 |
| Кредиты и долги (учет займов, напоминания) | - | - | - | + | + |
| Инвестиции (отслеживание портфеля) | - | - | - | - | + |
| Совместный учет (семейный/групповой доступ) | - | - | - | + | + |
| Интеграции (банки, бухгалтерские сервисы) | - | - | - | - | + |

Обобщим результаты оценки и проведем анализ приложений.

«Дзен-мани» имеет простой и понятный учет, отсутствует мультивалютность и нет синхронизации с банками. Не обладает возможностью автоматического обновления транзакций, не имеет глубокой аналитики и кастомизации отчетов, не имеет возможности учета инвестиций и семейного бюджета. Имеет бесплатную версию с основными функциями и платные опции (расширенные отчеты, дополнительные категории).

Monefy обладает быстрым и простым учетом расходов разных валют, минималистичным дизайном и малым набором функций. Не имеет автоматической синхронизации с банками, глубокой аналитики. Не позволяет вести совместный учет и учет инвестиций. Имеет бесплатную версию с базовыми функциями, но с рекламой. Предлагает платную версию, которая убирает рекламу, добавляет мультивалюность и экспорт данных.

MoneyLover предлагает мощные инструменты для учета финансов с лучшей аналитикой. Имеет возможность работать с мультивалютными счетами, ведением совместного бюджета. Отсутствует оффлайн-режим, что ограничивает возможности использования данного приложения при этом обладает облачной синхронизацией. Бесплатная версия предлагает ограниченное количество кошельков и бюджетов, есть реклама. В целях безопасности имеется защита PIN-кодом/биометрией и шифрованием данных. Предоставляет подписочный сервис (премиум), который открывает все функции (банковские подключения), является одним из самых дорогих.

Wallet предлагает расширенный выбор инструментов для автоматизации учета финансов. Обладает возможностью подключения к банку, удобные и подробные отчеты и ведение совместного бюджета. Есть возможность отслеживания акций, криптовалюты, учет займов с напоминаниями. В целях безопасности присутствует банковское шифрование (256-bit SSL), двухфакторная аутентификация. Доступ по Face ID/Touch ID, данные хранятся в облаке. Кроссплатформенный доступ к приложению (iOS, Android), имеет веб-версию приложения. Предлагает бесплатную версию с ограниченным функционалом (только ручной ввод операций, без банковской синхронизации). Расширенный функционал представляется по подписке (автоматическая синхронизация, веб-доступ, расширенная аналитика).

Таким образом, проведя анализ существующих решений и запросов пользователей, для разрабатываемого приложения будем использовать следующие функциональные характеристики, направленные на автоматизацию и упрощение учета личного бюджета, а также на создание мотивационной среды для формирования привычек грамотного финансового управления, при выборе функциональных характеристик следует учитывать избыточность и потенциальное использование функций приложения:

1. Упрощение этапа фиксаций операций доходов и расходов.

Введение функции сканирования с чеков QR-кодов с помощью встроенной камеры мобильного устройства и автоматическое внесение суммы и даты покупки сокращает время пользователя на внесение трат.

Сохранение истории доходов и расходов в цифровом виде обеспечивает удобный доступ к данным в любом месте.

1. Упрощение этапа анализа финансового состояния.

В приложении будет реализована интуитивно понятная аналитика, выводящая с помощью графиков и диаграмм, наглядную структуру расходов и доходов за выбранный период.

1. Упрощение формирования финансовых привычек и повышение мотивации у пользователей.

В приложение будет встроена система достижений, которая будет стимулировать пользователя к регулярному учету бюджета и соблюдению финансовых планов. Например, пользователь будет получать виртуальные значки за экономию средств и своевременное внесение данных. Также реализация системы уведомлений будет напоминать пользователям вносить свои траты за день и отслеживать выполнение целей.

Таким образом, предлагаемое мобильное приложение существенно упрощает и автоматизирует все ключевые этапы личного финансового учета: от фиксации затрат (вручную или с помощью сканирования чеков) до формирования отчетов и анализа расходов. Кроме того, добавление мотивационных элементов, таких как достижения и напоминания, помогает пользователям лучше следить за своими финансами и развивать привычку к финансовой дисциплине.

* 1. **Цели и задачи разработки программы**

Целью разработки данного мобильного приложения является создание удобного и современного инструмента для учета личного бюджета. Приложение должно помогать пользователям тратить меньше времени на ведение своих расходов и доходов, проводить их анализ и мотивировать к более осознанному управлению личными средствами.

Задача работы заключается в разработке мобильного приложения, которое должно обеспечивать выполнение следующих функций:

* разработка простого и понятного интерфейса. Большое количество приложений перегружены информацией, что отталкивает пользователей от их использования. Реализация интерфейса в минималистичном стиле позволяет быстро влиться в использование приложения, что также и экономит время,
* внедрение функции сканирования QR-кода с чеков и извлечением суммы покупки с датой ее совершения. Автоматическое сканирование позволяет быстро извлечь нужные данные и сохранять их в приложении,
* внесение вручную своих расходов и доходов с возможностью выбора, что данную операцию нужно повторять автоматически каждый раз в заданное количество времени (например, оплата аренды квартиры). Функция автоповтора позволяет заранее настроить регулярные операции, что также экономит время пользователя,
* разработка финансовой аналитики. при большом количестве трат трудно оценить, куда уходят расходы, поэтому внедрение аналитики поможет пользователям выявить проблемные категории, на которые уходит большая часть денег,
* обеспечение хранения данных и их редактирование,
* создание системы достижений и напоминаний. Напоминания помогут пользователям не забывать об учете движения своих финансов, в совокупности с системой достижений это поможет пользователям более ответственно относится к данному процессу,
* авторизация и регистрация пользователя в системе. Так пользователь сможет подключаться с разных устройств и не потерять данные. Также это обеспечивает защиту данных.

Таким образом, с помощью реализованных функций в приложении можно устранить самые распространенные причины, по которым пользователя не ведут свой учет финансов.

1. **Разработка и анализ требований**

В процессе создания программного продукта одним из ключевых этапов является разработка требований к программному обеспечению. Чаще всего требования делят на две категории: функциональные и нефункциональные.

Перед составления требований необходимо собрать информацию из ключевых источников.

Источниками требований могут являться (не ограничиваются):

* бизнес-требования (цели заказчика);
* анализ пользователей (целевая аудитория, сценарии использования);
* анализ конкурентов (плюсы/минусы аналогичных решений);
* ограничения (бюджет, сроки, технологии);
* изучение стандартов и нормативов [5].

1. **Функциональные требования**

Функциональные требования определяют поведение системы в различных сценариях. Они показывают, что разработчики должны сделать, чтобы пользователи могли выполнять свои задачи в соответствии с бизнес-требованиями [6].

Основные функциональные требования к продукту:

* управление пользователями:

1. регистрация (для предоставления доступа к функционалу приложения, пользователю необходимо пройти регистрацию, которая проходит с помощью заполнения формы. Форма регистрации представляет собой поля, в которые необходимо внести личный email, никнейм и пароль);
2. аутентификация (для входа в приложение необходимо пройти процедуру аутентификации, для этого необходимо ввести данные, указанные при регистрации. Сценарий входа будет считаться успешным если данные, записанные в БД, будут совпадать со значениями, которые ввел пользователь. На сервере осуществляется сравнение хеша пароля, сохранённого в БД и корректность введенного адреса почты. После успешного входа система генерирует JWT-токен, который содержит ID пользователя и время жизни (стандартно 3 часа). Параллельно с JWT-токеном выдается refresh-токен (время жизни 14 дней), который обновляет JWT-токен, если он истек [7]);
3. настройка профиля (после успешного входа пользователю должно быть доступно на вкладке «Профиль» смена пользовательских данных, таких как никнейм, email и пароль. Также приложение должно предоставлять возможность добавления личных категорий, которые пользователь сможет использовать при заполнении информации о транзакции);

* учет финансовых операций:

1. добавление транзакций (пользователю должно быть доступно добавление транзакций, указывая сумму, дату, категорию, описание и выбор типа операции (доход/расход);
2. отображение добавленных пользователем транзакций должно происходить на вкладке «Записи» (также при отображении должен быть доступен фильтр, позволяющий ограничить данные в пределах выбранного месяца заданного года);
3. редактирование и удаление транзакций;
4. категоризация транзакций с возможностью настройки пользовательских категорий;
5. возможность внесения периодических транзакций (при добавлении транзакций должен быть доступен чекбокс с возможностью выбора является ли транзакция повторяющейся (например, ежемесячная оплата аренды квартиры) с возможностью определения частоты повторения);

* сканирование чеков:

1. приложение должно обеспечивать возможность внесения транзакции через сканирование QR-кода на чеке (доступны варианты через выбор изображения с галереи и с помощью встроенной камеры в телефоне);
2. распознавание QR-кодов должно происходить через библиотеку ZXing.Net.Maui;
3. автоматическое заполнение полей (сумма и дата), с возможностью редактирования до сохранения;

* визуализация данных:

1. отображение аналитики в виде графиков и диаграмм (у пользователя должна быть возможность просмотреть на вкладке «Аналитика» распределение расходов по категориям за выбранный период с помощью круговой диаграммы. Также должна быть доступна столбчатая диаграмма, показывающая сравнение доходов и расходов между месяцами или годами;
2. просмотр и экспорт отчетов (должны быть доступны отчеты, выводящие общую статистику (вывод суммарных расходов и доходов, топ-3 категории расходов, круговая диаграмма расходов и доходов по категориям);
3. возможность фильтрации данных, которые можно получить на вкладках «Аналитика» и «Отчеты» по периодам;

* система достижений и мотивации:

1. награждение пользователей за регулярность использования. Пользователь должен иметь возможность видеть на вкладке «Профиль» достижения. Каждое достижение содержит значок, условие получения, название и дату получения, присваивающиеся за выполнение определенных действий: «Первая запись» – за создание первой транзакции, «Категорийный дегустатор» – за использование 3 разных категорий расходов, «Недельный дисциплинант» – за внесение транзакций 7 дней подряд, «Балансировщик» – за превышение доходов над расходами 2 месяца подряд;
2. геймификация процесса учета финансов (полученные достижения начисляют очки опыта, что повышает уровень, который можно отобразить в профиле, уровень должен расчитываться по формуле: 1+ количество полученных ачивок);
3. система уведомлений – ежедневное напоминание пользователям о необходимости внести расходы и доходы (пользователям должны приходить push-уведомления в 19:00);

* синхронизация:

1. хранение данных на сервере;
2. кроссплатформенная работа.

На рисунке 1 представлена диаграмма вариантов использования для пользователей приложения. Данная диаграмма отражает основные функции, которые может использовать пользователь в мобильном приложении.

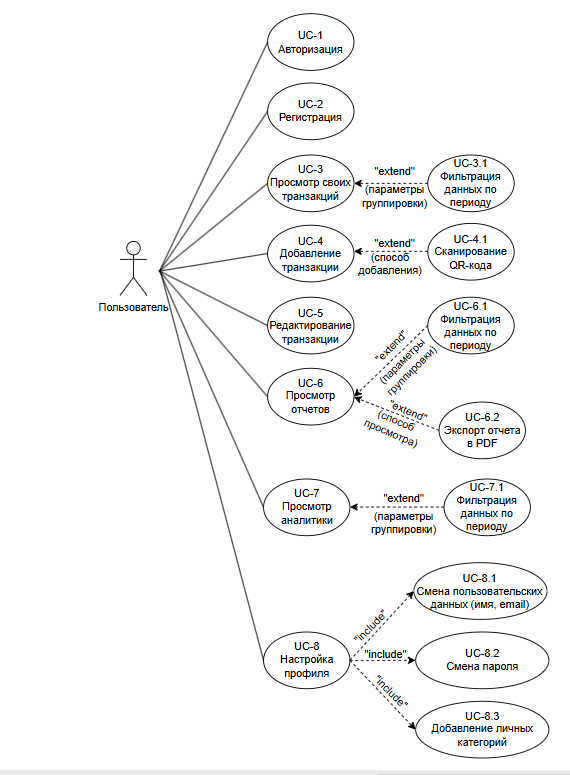


Рисунок 1 – Диаграмма вариантов использования

1. **Нефункциональные требования**

Нефункциональные требования фокусируются не на том, что система выполняет, а на том, насколько эффективно она это делает. Они могут описывать ключевые характеристики или качества системы, такие как доступность, удобство использования, производительность и другие важные аспекты [8].

Основные нефункциональные требования к продукту:

* производительность – время отклика приложения не должно превышать 1 – 2 секунд для основных операций.
* безопасность – хранение паролей в хешированном виде,
* кроссплатформенность:

1. поддержка iOS и Android;
2. адаптивный интерфейс под разные размеры экранов;

* удобство использования – интуитивно понятный дизайн.

1. **Требования к пользовательскому интерфейсу**

Для разработки пользовательского интерфейса были определены основные требования к нему:

* минималистичный дизайн,
* цветовая схема состоит из спокойных и холодных цветов,
* навигационное меню содержит краткое описание раздела,
* отклик системы на действия пользователя (графические изменения элементов интерфейса),
* должна быть реализована «сглаженная геометрия» интерфейса.

При разработке пользовательского интерфейса важно обеспечить интуитивную навигацию между страницами. Для визуализации структуры экранов и логики переходов между ними была разработана карта экранов и переходов между ними (рисунок 2).

Карта экранов и переходов включает в себя:

* экраны авторизации и регистрации,
* основные разделы приложения (финансы, отчеты, аналитика и профиль), перемещение между которыми происходит с помощью единого нижнего меню,
* экраны с добавлением транзакций и сканированием чека, на которые можно перейти из основных разделов приложения,
* логика переходов (указана стрелками).

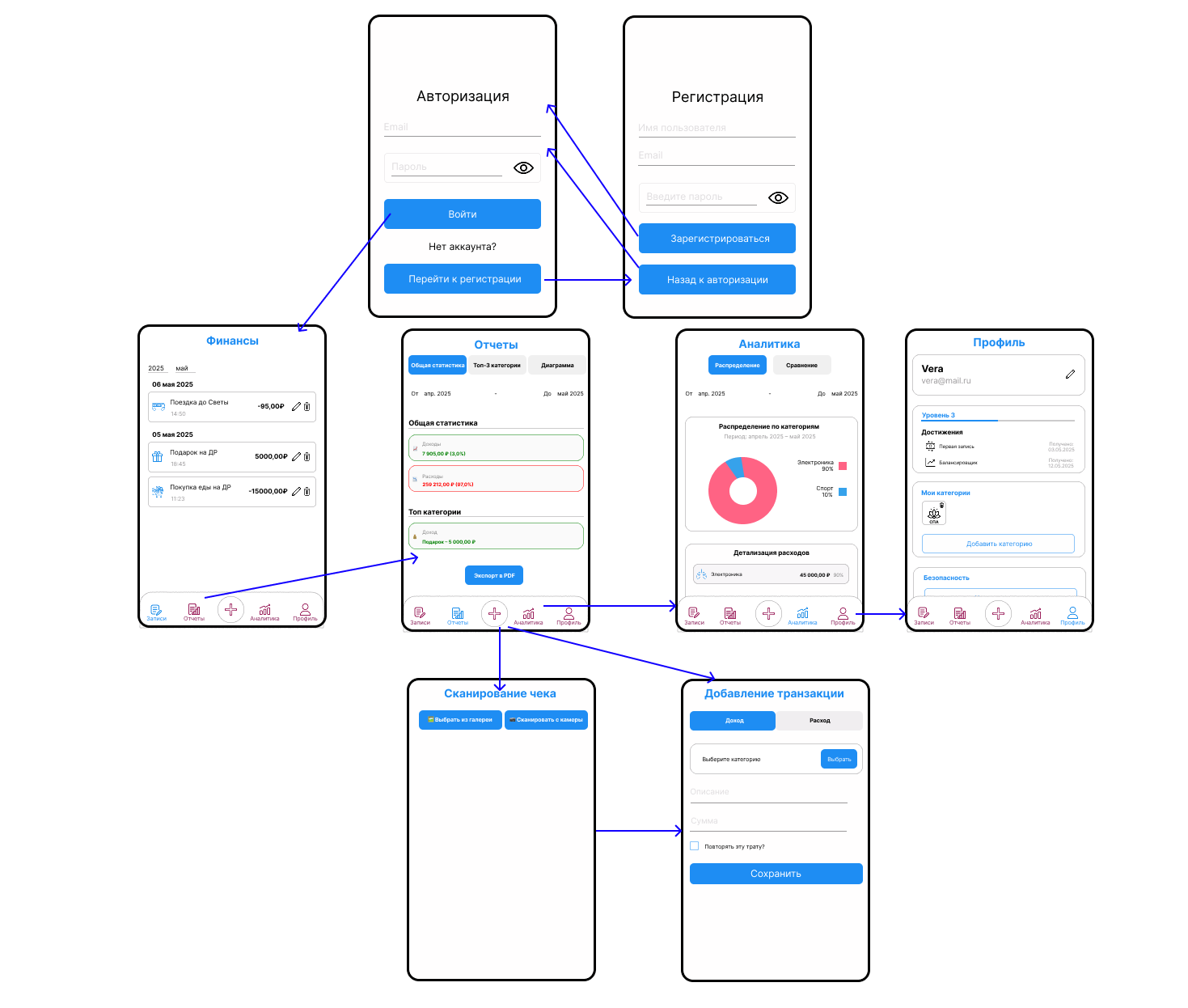


Рисунок 2 – Карта экранов и переходов

Интерфейс приложения, разработанный на основе определенных требований, интуитивно понятен, так как является легко настраиваемым и понятным пользователю без необходимости изучений правил его использования.

Элементы управления легко узнаваемые и располагаются логичным образом. Однотипные элементы управления работают одинаково во всех частях приложения. Навигация и элементы управления схожи с теми, которые используют другие приложения. Интерфейс минималистичный и не перегружает пользователей лишней информацией, затрудняющей понимание и использование.

1. **Выбор средств реализации**

Для разработки мобильного приложения были выбраны технологии и инструменты, обеспечивающие эффективную реализацию функциональных и нефункциональных требований.

* + 1. **Язык программирования**

В качестве основного языка разработки был выбран C#.   
C# является популярным кроссплатформенным языком программирования, позволяющий создавать программное обеспечение, которое работает на широком диапазоне устройств. Также он поддерживает единую кодовую базу для всех компонентов системы (клиент, сервер, логика) [9].

В сравнении с такими языками, как JavaScript или Dart, C# обеспечивает более тесную интеграцию с экосистемой .NET, что позволяет использовать единый стек технологий как для фронтенда, так и для бэкенда. Это упрощает архитектуру, ускоряет разработку и снижает риски. Также C# обеспечивает мощную поддержку Microsoft, активное сообщество и регулярные обновления, гарантируя устойчивость и перспективность решения.

* + 1. **Клиентская часть**

Для клиентской части был выбран .NET MAUI (Multi-platform App UI) как основная технология. Мультиплатформенный пользовательский интерфейс приложений .NET (.NET MAUI) — это кроссплатформенная платформа для создания собственных мобильных и классических приложений с помощью C# и XAML, которая является прямой эволюцией Xamarin.Forms с существенными улучшениями [10]. Основными причинами выбора являются:

* возможность разработки кроссплатформенного приложения для iOS и Android с единой кодовой базой,
* готовые компоненты для UI,
* поддержка нативных возможностей устройств (камера, файловая система),
* единая проектная система, использующая мультитаргетинг для платформ Android, iOS.

В сравнении, например, с тем же Xamarin.Forms, у .NET MAUI единый проект под разные целевые платформы, лучшая производительность и долгосрочная поддержка от Microsoft [11].

* + 1. **Серверная часть**

Для разработки серверной части был выбран ASP.NET Core. ASP.NET Core – это кроссплатформенный фреймворк с открытым исходным кодом для разработки веб-приложений, который можно использовать для быстрого создания динамических приложений с отрисовкой на стороне сервера.

В отличие от таких решений, как Node.js или Spring Boot, ASP.NET Core предлагает тесную интеграцию с инфраструктурой .NET и строго типизированную разработку. Кроме того, ASP.NET Core изначально проектировался как кроссплатформенный, масштабируемый и облачно-ориентированный, что делает его особенно подходящим для современных мобильных и веб-сервисов. Его также можно применять, чтобы создавать HTTP API для мобильных приложений. Также у фреймворка отличная поддержка БД через Entity Framework Core, встроенная поддержка аутентификации/авторизации и высокая производительность [12].

* + 1. **База данных**

PostgreSQL – объектно-реляционная система управления базами данных с открытым исходным кодом. На первом месте в ней стоит расширяемость, техническое совершенство и совместимость. PostgreSQL – кроссплатформенное ПО, к нему существуют интерфейсы из всех современных языков программирования.

Например, в сравнении с MySQL, PostgreSQL предлагает более высокую гибкость и возможности для вертикальной масштабируемости, что делает его предпочтительным выбором для рабочих задач в области хранения данных и бизнес-аналитики. В отличие от NoSQL-решений, таких как MongoDB, PostgreSQL сочетает надежность реляционных баз с гибкостью NoSQL, поддерживая как структурированные данные, так и JSON-документы, что делает его универсальным решением для работы с разными типами данных [13].

* + 1. **Дополнительные библиотеки**

Для уведомлений была выбрана библиотека Plugin.LocalNotification, позволяющая отправлять локальные push-уведомления, планируемые по времени и отправляющиеся с заданным интервалом. Она предоставляет простой и единообразный API как для Android, так и для iOS, что упрощает реализацию уведомлений без необходимости вручную взаимодействовать с платформенными SDK.

Для сканирования будет использоваться библиотека ZXing.Net.Maui – наиболее популярная библиотека для .NET. ZXing поддерживает iOS, Android и Windows через единый API, у нее высокая производительность и нативная интеграция с камерами устройства. В отличие от альтернатив, таких как MLKit от Google или нативные камеры SDK, ZXing.Net.Maui не требует сложной настройки и предоставляет достаточную точность и скорость работы, что делает её оптимальной для задач считывания чеков в мобильном приложении.

Для работы с базой данных используется EF Core – ORM-библиотека от Microsoft, обеспечивающая удобную и типобезопасную работу с реляционными данными через C#. EF Core позволяет описывать структуру базы в виде C#-классов, автоматически создавать таблицы и выполнять миграции. Благодаря использованию LINQ, доступ к данным реализован декларативно, без ручного написания SQL-запросов. EF Core упрощает реализацию CRUD-операций, ускоряет разработку и повышает читаемость кода [14].

Для добавления аутентификации через JWT-токены будет использоваться пакет Microsoft.AspNetCore.Authentication.JwtBearer, который также включает в себя валидацию токенов (подпись, срок жизни, издатель). С помощью этого пакета будет происходить защита API от неавторизованного доступа к функционалу приложения.

Для визуального представления данных будут использоваться следующие библиотеки:

1. Microcharts.Maui для отображения круговой диаграммы так как предлагает легковесное решение для базовых визуализаций, адаптированное под .NET MAUI. Она обеспечивает базовую визуализацию без перегрузки UI и подходит для отображения долей,
2. Syncfusion.Maui.Charts для построения столбчатых диаграмм. Библиотека поддерживает гибкую настройку осей, поддержку фильтрации, что делает ее более подходящей для представления временных рядов и сравнительного анализа.
3. **Разработка программы**
4. **Проектирование архитектуры программы**

Разрабатываемое мобильное приложение построено на клиент-серверной архитектуре с четким разделением ответственности между клиентской и серверной частями, а также БД. Основной целью данного подхода является упрощение поддержки, тестирования и масштабирования приложения.

Система включает в себя:

* мобильный клиент на платформе .NET MAUI, реализующий логику отображения и базовую валидацию данных. Обеспечивает взаимодействие с пользователем, отображение данных о транзакциях, аналитику в виде графиков и достижения,
* веб-сервер на ASP.NET Core, который обеспечивает обработку бизнес-логики, аутентификацию, авторизацию, а также взаимодействие с БД с помощью EF Core,
* базу данных PostgreSQL, где хранятся основные сущности (пользователи, транзакции, категории, достижения).

Клиентская часть построена по упрощённой архитектуре с разделением представления и логики. Пользовательский интерфейс описан в XAML-файлах, а логика обработки событий и взаимодействия с сервером реализована в code-behind (xaml.cs).

Серверная часть реализована в виде Web API, где обработка запросов осуществляется через контроллеры (Controller), а модель (Model) представляет данные и бизнес-логику. Представление (View) в рамках Web API отсутствует, так как сервер возвращает данные в виде JSON.

Для обмена данными между клиентом и сервером используется Representational State Transfer Application Programming Interface (далее – REST API). REST – это архитектура программного обеспечения для взаимодействия по протоколу HTTP, а REST API – механизм запроса от клиентского приложения к серверному программному обеспечению [15].

На рисунке 3 представлена диаграмма последовательности взаимодействия системы при авторизации, получении транзакций и добавлении транзакции.

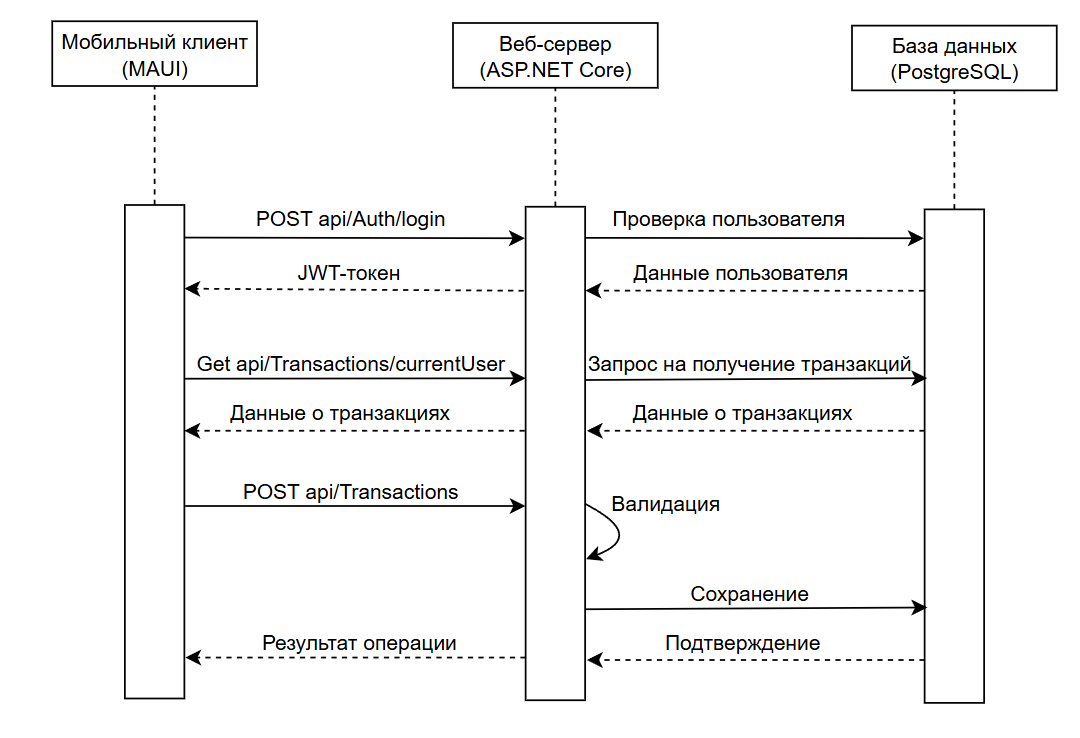


Рисунок 3 – Диаграмма последовательности взаимодействия системы

Для более наглядного представления потоков данных в системе и их взаимодействия между ключевыми компонентами (клиент, сервер, БД) была построена диаграмма потоков данных (DFD). DFD – это способ графического описания входящих и выходящих потоков данных.

Существует три уровня DFD-схем:

* контекстный (на этом уровне расписываются в общих чертах системы и процессы),
* логический (уровень, описывающий происходящие в системе процессы более подробно, указывая, какие входящие и выходящие данные для каждого из процессов),
* физический (является более детализированным логическим уровнем, где появляется подробное описание баз данных) [16].

На рисунках 4 – 6 показаны DFD-диаграммы.

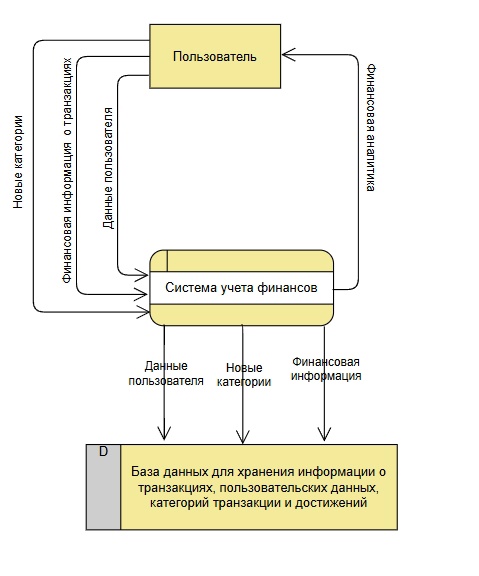


Рисунок 4 – DFD-диаграмма 0 уровня (контекстная схема)

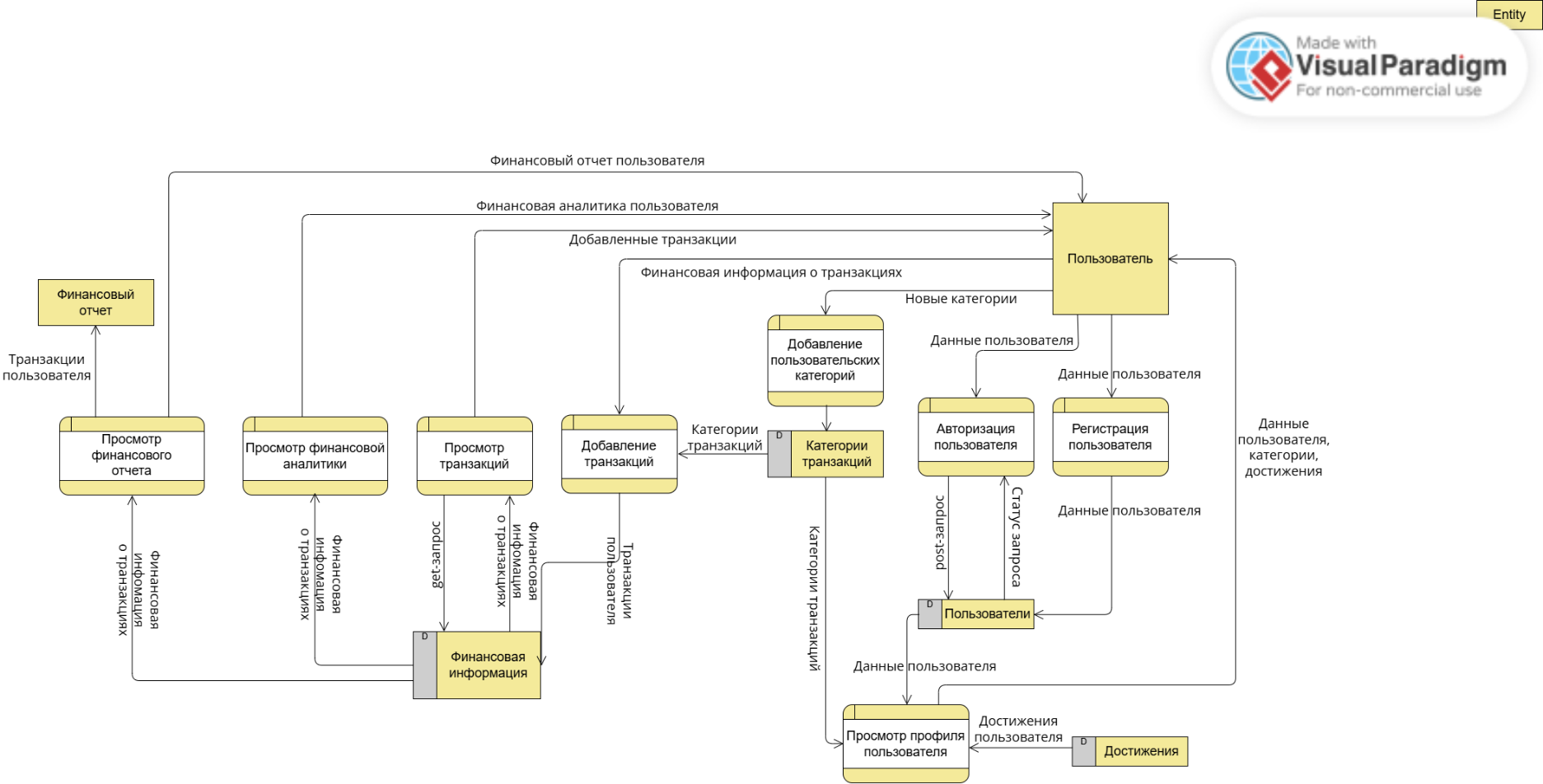


Рисунок 5 – DFD-диаграмма 1 уровня (логическая)



Рисунок 6 – DFD-диаграмма 1 уровня (физическая)

1. **Описание данных, структур данных, файлов**

Основным подходом при проектировании БД и информационных систем является модель типа Entity-Relationship (далее – ER). ER-диаграмма представляет собой тип блок-схемы, который показывает взаимосвязи между различными «сущностями» (такими как люди, объекты, концепции и т.д.) [17].

Для описания структур данных, содержащихся в мобильном приложении, выделяются следующие ключевые сущности:

* категории,
* пользователи,
* транзакции (основная финансовая операция),
* повторяющиеся транзакции (специальный тип операций),
* достижения,
* достижения пользователя.

На рисунке 7 представлена ER-диаграмма, визуализирующая описанные сущности и их взаимосвязи.

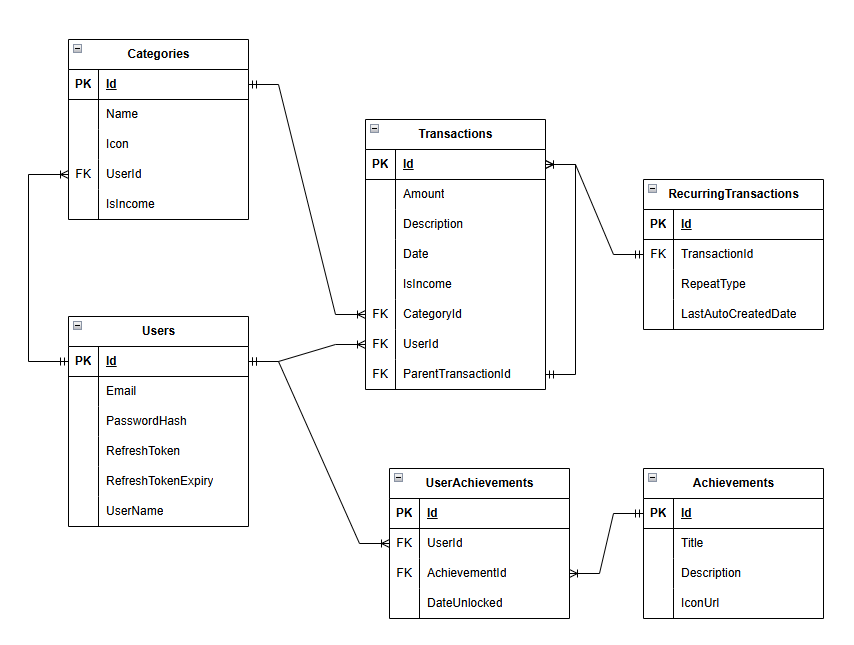


Рисунок 7 – ER-диаграмма

Поля и их значения таблиц, используемые в базе данных, приведены в словаре данных в таблице 2.

Таблица 2 – Словарь данных

|  |  |  |
| --- | --- | --- |
| Имя | Тип данных | Значение |
| Categories | | |
| Id | integer | Первичный ключ, значение автоинкрементируется, начиная с 1 |
| Name | varchar(255)t | Содержит название категории |
| Icon | text | Содержит путь к иконке категории |
| UserId | integer | (FK) Внешний ключ – null если категория стандартная; иначе, привязана к пользователю |
| IsIncome | boolean | true если категория относится к доходам |
| Users | | |
| Id | integer | Первичный ключ, значение автоинкрементируется, начиная с 1 |
| Email | varchar(255) | Содержит Email пользователя |
| PasswordHash | text | Содержит захешированный пароль |
| RefrehToken | text | Содержит refresh-токен |
| RefreshTokenExpiry | timestamp with timezone | Содержит время окончания refresh-токена |
| UserName | varchar(255) | Содержит имя пользователя |
| Transactions | | |
| Id | integer | Первичный ключ, значение автоинкрементируется, начиная с 1 |
| Amount | numeric | Сумма транзакции |
| Description | varchar(255) | Описание транзакции |
| Date | timestamp with timezone | Дата проведения транзакции |
| IsIncome | boolean | true если транзакция является доходом |
| CategoryId | integer | (FK) Внешний ключ – хранит Id категории |
| UserId | integer | (FK) Внешний ключ – хранит Id пользователя |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| Имя | Тип данных | Значение |
| ParentTransactionId | integer | (FK) Внешний ключ – null если транзакция родительская (от нее идут повторы) или не повторяющаяся, иначе привязана к родительской транзакции, от которой идет повтор |
| RecurringTransactions | | |
| Id | integer | Первичный ключ, значение автоинкрементируется, начиная с 1 |
| TransactionId | integer | (FK) Внешний ключ – хранит Id категории |
| RepeatType | integer | Содержит тип повтора:   * 0 – не повторяется, * 1 – ежедневно, * 2 – еженедельно, * 3 – ежемесячно. |
| LastAutoCreatedDate | timestamp with timezone | Содержит время последнего повтора |
| Achievements | | |
| Id | integer | Первичный ключ, значение автоинкрементируется, начиная с 1 |
| Title | varchar(100) | Содержит название достижения |
| Description | varchar(300) | Содержит описание получения достижения |
| IconUrl | text | Содержит путь к иконке достижения |
| UserAchievements | | |
| Id | integer | Первичный ключ, значение автоинкрементируется, начиная с 1 |
| UserId | integer | (FK) Внешний ключ – хранит Id пользователя |
| AchievementId | integer | (FK) Внешний ключ – хранит Id достижения |
| DateUnlocked | timestamp with timezone | Содержит время получения достижения |

Классы сущностей в приложении реализованы в виде C#-моделей, соответствующих структурам таблиц, представленных в ER-диаграмме (рисунок 7). Для взаимодействия с БД используется технология EF Core, которая позволяет задавать связи между сущностями и ограничения с помощью Fluent API в методе OnModelCreating класса контекста БД. Использование Fluent API обеспечивает большую гибкость в конфигурации моделей, особенно при определении сложных связей между сущностями.

Для того, чтобы перенести описанные модели и связи в физическую структуру БД EF Core использует механизм миграций. Он позволяет поэтапно изменять модели к БД. Созданная миграция БД, содержащая определения всех таблиц и связей, реализованных с помощью EF Core, представлена в Приложении А. Скрипт создания миграции был получен с помощью команды в консоли диспетчера пакетов: Add-Migration InitialCreate.

1. **Описание сервисов и окружения приложения**

Мобильный клиент реализован с использованием кроссплатформенной технологии .NET MAUI и построен на архитектуре с разделением слоев логики. Ключевая бизнес-логика и взаимодействие с сервером вынесены в отдельные сервисы. Все сетевые запросы выполняются через HttpClient, что позволяет едино управлять конфигурацией HTTP-запросов и поддерживать аутентификацию через JWT.

Основные клиентские сервисы:

* AuthService – сервис, отвечающий за аутентификацию и регистрацию пользователей. Поддерживает вход в систему, регистрацию, обновление JWT-токена и проверку текущего состояния авторизации,
* TransactionsService – сервис, обеспечивающий управление транзакциями: получение, добавление, обновление и удаление,
* CategoryService – реализует функции работы с категориями доходов и расходов: получение всех категорий, получение категорий текущего пользователя, добавление категорий и их удаление,
* UsersService – отвечает за обновление пользовательских данных и смену пароля,
* AchievementService – сервис, обеспечивающий получение списка достижений пользователя и проверку выполнения условий для получения новых достижений,
* ReportService – позволяет формировать и экспортировать отчеты в формате PDF. Сервис отправляет список транзакций и параметры фильтрации на сервер, где выполняется генерация отчета,
* NotificationService – локальный сервис, отвечающий за планирование и отображение ежедневных уведомлений пользователю.

Также реализованы вспомогательные классы, обеспечивающие корректную и безопасную работу с аутентификацией и HTTP-запросами:

* JwtHelper – статический вспомогательный класс, предназначенный для извлечения информации из JWT-токена. Использование класса позволяет исключить дублирование кода и централизовать работу с JWT-токенами,
* AuthTokenHandler – реализация DelegatingHandler (класс, позволяющий расширять обработку HTTP-запросов), добавляющая JWT-токен к каждому исходящему HTTP-запросу клиента.

Иконки категорий (стандартных и пользовательских), достижений, общих элементов (например, кнопки «Удалить», «Редактировать») и используемые для навигации в меню, хранятся в папке Resources/Image.

Серверная часть приложения, разработанная на платформе ASP.NET Core Web API, реализует REST API для обмена данными с мобильным клиентом.

Сервер включает в себя следующие контроллеры:

* AuthController – контроллер, реализующий регистрацию, вход в систему и обновление токенов,
* UsersController – управляет профилем пользователя, включая просмотр и обновление информации,
* TransactionsController – отвечает за создание, редактирование, удаление и получение финансовых транзакций,
* CategoriesController – обеспечивает работу с категориями доходов и расходов,
* ReportsController – экспортирует отчеты в PDF,
* AchievementController – обрабатывает получение достижений пользователя.

Каждый контроллер взаимодействует с соответствующим сервисом на клиенте.

Для удобства тестирования REST API серверной части реализована интеграция с Swagger UI, который доступен по адресу /swagger/index.html в режиме разработки.

Отдельным компонентом серверной архитектуры также является фоновый сервис RecurringTransactionsBackgroundService. Он запускается вместе с сервером и каждые два часа вызывает метод GenerateRecurringTransactionsAsync сервиса RecurringTransactionService для проверки всех повторяющихся транзакций пользователя. Логика сервиса включает в себя нахождение всех транзакций с типом повтора, у который еще не было создано копии на текущее время, вычисление следующей даты повторения, создание новой копии, сохранение новой транзакции в БД.

Доступ к БД осуществляется через AppDbContext, построенный на EF Core. Контекст содержит в себе таблицы: Users, Categories, Transactions, RecurringTransactions, Achievements, UserAchievements.

1. **Тестирование**

Для проверки работоспособности приложения выполняется функциональное тестирование на основе подготовленных сценариев использования и заявленного функционала.

1. **Тестирование регистрации и авторизации**

При первом открытии приложения система направляет пользователя на страницу авторизации. Если аккаунта нет, то пользователь может перейти на страницу регистрации и зарегистрироваться (рисунки 8, 9).

|  |  |
| --- | --- |
| C:\Users\VerON\Documents\Frame 2.png  Рисунок 8 – Форма регистрации | C:\Users\VerON\Documents\Frame 2.png  Рисунок 9 – Просмотр функциональности раскрытия пароля |

При попытке зарегистрировать пользователя, чей email уже был указан, приложение выдаст ошибку (рисунок 10).

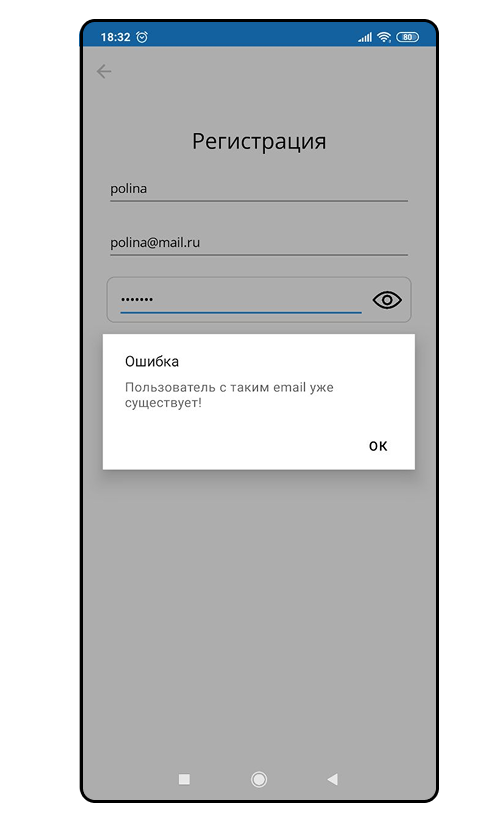


Рисунок 10 – Сообщение о повторном использовании email

При успешной регистрации пользователя переносит на страницу авторизации (рисунок 11), где необходимо внести корректные данные для входа, иначе система выдаст ошибку (рисунок 12).

|  |  |
| --- | --- |
| Рисунок 11 – Форма авторизации | Рисунок 12 – Ошибка о неверно введенных данных |

1. **Тестирование взаимодействия с транзакциями, получения достижений и страницы профиля**

После успешной авторизации пользователя переносит на страницу «Записи», на которой отображены транзакции текущего месяца (рисунок 13).

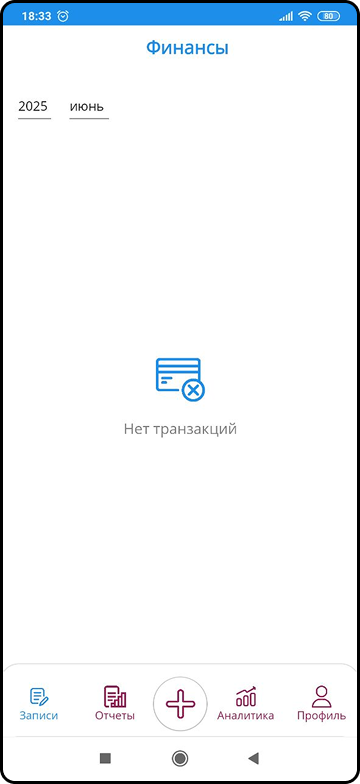


Рисунок 13 – Вкладка «Записи» при незаполненных транзакциях

С помощью кнопки «+» можно добавить новые транзакции. После нажатия приложение открывает модальное окно, в котором предлагает пользователю выбрать вариант добавления транзакции, а именно: ручное добавление данных, сканирование чека (рисунок 14).

Если выбрать добавление через ручное, то система закрывает модальное окно и переходит на новую страницу, где предлагает пользователю самостоятельно внести информацию о транзакции (рисунок 15):

* вид транзакции (доход/расход),
* выбрать категорию транзакции,
* описание транзакции,
* сумму транзакции.

В дополнение, приложение предлагает пользователю запомнить данную транзакцию и перевести ее в периодическую.

Для тестирования работы данного функционала добавим тестовую транзакцию расхода «Сдача анализов»:

* выберем вид транзакции – «Расход»,
* выберем категорию – «Здоровье»,
* внесем описание – «Сдача анализов»,
* внесем сумму транзакции – «5000».

После введенных данных сохраняем данную транзакцию нажатием на кнопку «Сохранить» (рисунок 16).

Возвращаемся в раздел «Записи», где отображена созданная транзакция (рисунок 17). Чтобы получить более детальную информацию о транзакции необходимо нажать на запись. После нажатия открывается модальное окно, в котором отражена подробная информация о транзакции (рисунок 18).

Добавим тестовую транзакцию дохода «Зарплата за пол месяца». Заполнение транзакции дохода заполняются аналогичным образом (рисунок 19). Пример выбора транзакции дохода представлен на рисунке 20.

|  |  |
| --- | --- |
| Рисунок 14 – Меню выбора добавления транзакции | Рисунок 15 – Страница добавления транзакции |

|  |  |
| --- | --- |
| Рисунок 16 – Заполненные данные на странице | Рисунок 17 – Просмотр добавленной транзакции |
| Рисунок 18 – Просмотр информации о транзакции | Рисунок 19 – Добавление дохода |

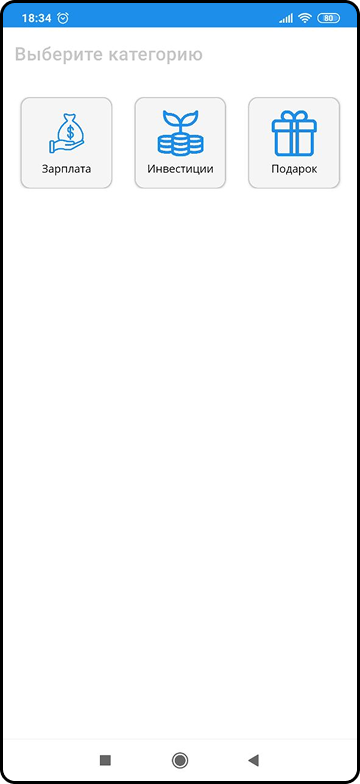


Рисунок 20– Окно выбора категорий при добавлении дохода

После добавления транзакций перейдем на вкладку «Профиль» и увидим, что появилось достижение «Первая запись» (рисунок 21).

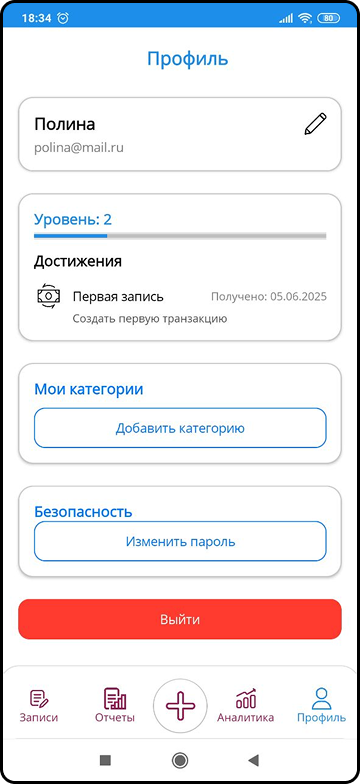


Рисунок 21 – Просмотр достижения «Первая запись»

Протестируем возможность редактирования пользовательских данных.

Изменим имя пользователя «Полина» на «Полинка», для этого нажмем на кнопку редактирования профиля в виде карандаша. После нажатия приложение дает возможность внести изменения в профиль пользователя, а именно изменить имя пользователя и электронную почту. После смены имени, необходимо нажать на кнопку подтверждения изменений (зеленая кнопка с «галочкой») в случае, если вносить изменения не нужно и необходимо отменить изменения, предусмотрена соответствующая кнопка (красная кнопка с «крестом») (рисунок 22). После подтверждения изменений система автоматически отображает новое имя пользователя в профиле (рисунок 23).

|  |  |
| --- | --- |
| Рисунок 22 – Активированный режим редактирования | Рисунок 23 – Просмотр смены имени |

Для тестирования функции добавления новых категорий создадим новую. Для этого нажмем на кнопку «Добавить категорию» в блоке «Мои категории» на странице профиля (рисунок 23).

После нажатия, система открывает новое окно создания категории «Добавить категорию» для создания категории необходимо выбрать из выпадающего списка вид транзакции (расход/доход), выбрать соответствующую иконку и ввести название категории, для сохранения категории необходимо нажать на кнопку «Сохранить» (рисунок 24).

После сохранния категории она будет отображаться в профиле   
(рисунок 25).

|  |  |
| --- | --- |
| Рисунок 24 – Добавление персонализированных категорий | Рисунок 25 – Просмотр добавленной категории и смена пароля |

Протестируем функцию смены пароля профиля. Для этого нажмем на кнопку «Изменить пароль» в блоке «Безопасность» в профиле пользователя (рисунок 25).

Для упешной смены пароля необходимо ввести текущий пароль от учетной записи, новый пароль и его повтор.

В случае, если текущий пароль не совпадает с тем, что сохранен в системе, приложение выдаст ошибку (рисунок 26). Для того, чтобы минимизировать проблемы входа в систему при смене пароля на новый, система просит дважды писать новый пароль при его смене. Если пароли не совпадают, то система выдает соответствующую ошибку (рисунок 27).

|  |  |
| --- | --- |
| Рисунок 26 – Ошибка при неверном вводе старого пароля | Рисунок 27 – Ошибка при несовпадении паролей |

В случае, если все поля были заполнены корректно, то система изменяет текущий пароль на новый и выдает соответсвующее уведомление  
(рисунок 28).

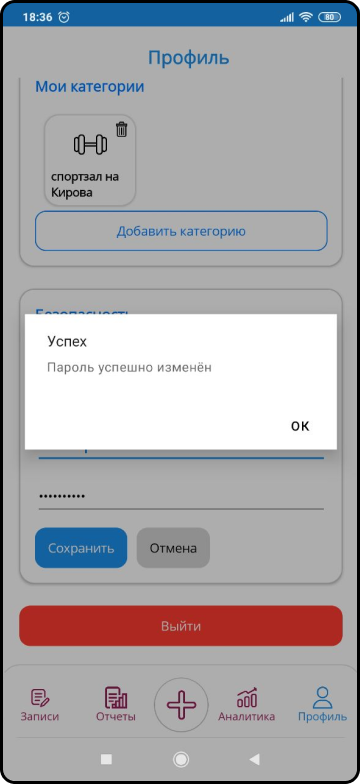


Рисунок 28 – Успешная смена пароля

Протестируем возможность добавления новой транзакции, через сканирование чека с использованием ранее добавленой пользовательской категории. С помощью кнопки «+» добавим новую транзакцию. После нажатия в модальном окне, выбраем вариант добавления транзакции через сканирование чека.

Система открывает новое окно «Сканирование чека» и предлагает выбрать пользователю откуда взять чек для скнирования, а именно: из галлереи или сканирование камерой телефона. Для данных опций предусмотрены соответствующие кнопки (рисунок 29).

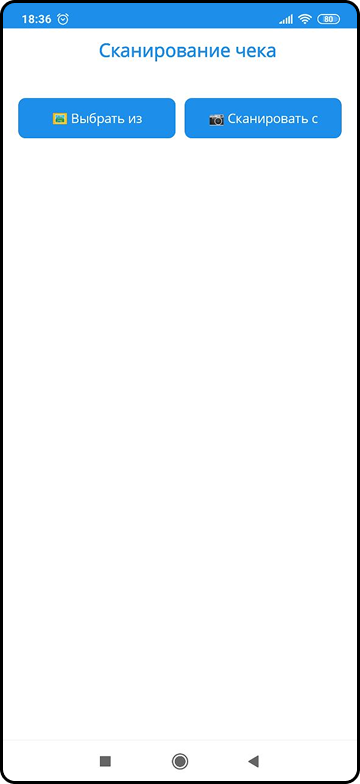


Рисунок 29 – Окно с выбором откуда сканировать чек

При выборе сканирования через камеру система выдаст запрос разрешений на использование камеры телефона для съемки фото или видео, а также открыть доступ к фото, мультимедиа и файлам на устройстве   
(рисунки 30, 31).

|  |  |
| --- | --- |
| Рисунок 30 – Разрешение на съемку фото и видео | Рисунок 31 – Разрешение на доступ к мультимедиа |

После предоставления доступа к съемке и мультимедиа, приложение откроет камеру мобильного устройства, с помощью которой будет произведено сканирование чека.

После сканирования чека система открывает окно «Добавление транзакции», где автоматически внесла распознанную информацию (рисунок 32). Для выбора транзакции нажмем соответствующую кнопку и в открытом окне выбора категории выберем созданную ранее. Пользовательские категории отличаются от системных цветом шрифта (системные – синий, пользовательские – черный) (рисунок 33).

В случае выбора сканирования чека через выбор соответствующей фотографии из галлереи устройста система работает аналогично.

|  |  |
| --- | --- |
| Рисунок 32 – Заполненные данные после сканирования | Рисунок 33 – Выбор категорий расхода и просмотр пользовательской категории |

После успешного добавления, система перенесет пользователя на изначальную страницу. Если перейти на вкладку «Записи», применив фильтр по дате, можно увидеть добавленную транзакцию от даты, заявленной в чеке   
(рисунок 34).

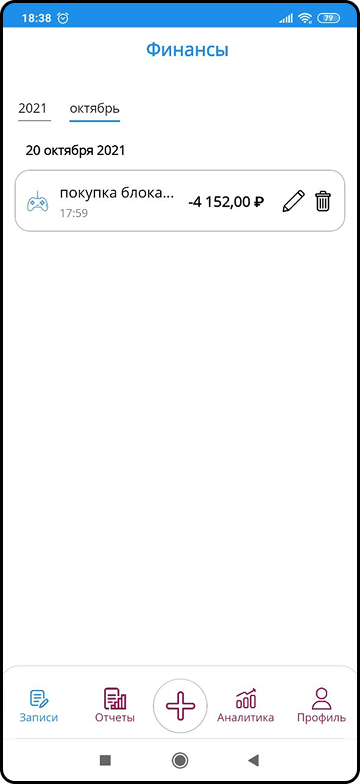


Рисунок 34 – Просмотр транзакции

Протестируем функцию удаления транзакций. Для этого перейдем в раздел «Записи» и в списке транзакций удалим транзакцию «Сдача анализов» нажмем на соответствующую кнопку «Мусорное ведро». После нажатия система в модальном окне запрашивает подтвеждение удаления транзакции (рисунок 36). Для отмены удаления имеется соответствующая кнопка. После подтверждения удаления система удаляет запись из транзакций (рисунок 37).

|  |  |
| --- | --- |
| Рисунок 36 – Удаление транзакции | Рисунок 37 – Отображение оставшихся транзакций |

Протестируем функцию редактирования транзакции. Для этого созданим тестовую расходную транзакцию «Посещение» и сделаем ее повторяемой, периодичность повторения выбирается из выпадающего списка после нажатия на чекбокс, выберем повторяемость «Каждую неделю» (рисунки 38, 39).

|  |  |
| --- | --- |
| Рисунок 38 – Создание повторяющейся транзакции | Рисунок 39 – Просмотр транзакций после добавления |

Для редактирования транзакции необходимо нажать на кнопку в виде карандаша и система откроет окно редактирования транзакции. Внесем изменения в стоимость и периодичность повторения транзакции. Сохраним изменения транзакции нажатием соответствующей кнопки «Сохранить» (рисунок 40).

Результат изменения сразу отоборажается в разделе «Записи»  
(рисунок 41).

|  |  |
| --- | --- |
| Рисунок 40 – Изменение стоимости | Рисунок 41 – Просмотр изменившейся информации |

1. **Тестирование аналитики и отчетов**

Для тестирования отчетов, перейдем на вкладку «Отчеты».

По умолчанию открывается общая статистика, на которой отображается процент соотношения доходов и расходов за указанный период, топ категорий, формирующийся на основе самой крупной транзакции (как для дохода, так и для расхода), а также пиковые дни, высчитывающиеся по самому большому расходу или доходу за день (рисунки 42, 43). Отчет можно экспортировать с помощью кнопки «Экспорт в PDF». Результат экспорта общей статистики показан на рисунке 44.

Если перейти на вкладку «Топ-3», то пользователю выведутся 3 категории, на которые было потрачено больше всего денег (рисунки 45, 46). Отчет также можно экспортировать в PDF (результат на рисунке 47).

Протестируем соотношение доходов и расходов, представленное в виде круговой диаграммы на вкладке «Диаграмма». На вкладке отображается как диаграмма, так и информация по категориям: название и общая сумма затрат или дохода (рисунки 48, 49). Информацию можно фильтровать по периодам: от месяца определенного года до месяца того же или другого года. Отчет можно экспортировать в PDF с помощью соответствующей кнопки «Экспорт в PDF» (рисунки 50, 51).

Все экспортируемые отчеты можно как сохранить на устройство, так и отправить на другое устройство (рисунок 50).

|  |  |
| --- | --- |
| Рисунок 42 – Отчет с общей информацией | Рисунок 43 – Отчет с общей информацией (низ страницы) |

|  |  |
| --- | --- |
| Рисунок 44 – Экспортированный отчет с общей информацией | Рисунок 45 – Отчет с Топ-3 категориями по расходам |
| Рисунок 46 – Отчет с Топ-3 после использования фильтрации | Рисунок 47 – Экспортированный отчет с топ-3 категориями |
| Рисунок 48 – Отчет с диаграммой соотношения доходов и расходов | Рисунок 49 – Детализация по отчету |

|  |  |
| --- | --- |
| Рисунок 50 – Окно с выбором, куда направить экспортированный отчет | Рисунок 51 – Экспортированный отчет с диаграммой |

Далее протестируем работу функций в разделе «Аналитика». По умолчанию открывается распределение расходов по категориям, где расположена круговая диаграмма с соотношением расходов и детализация расходов (рисунок 52). Данные можно фильтровать по периодам.

На вкладке «Сравнение» отображается столбчатая диаграмма сравнения доходов и расходов. Выводимую информацию можно фильтровать по месяцам или по годам (рисунки 53, 54).

|  |  |
| --- | --- |
| Рисунок 52 – Распределение расходов по категориям | Рисунок 53 – Сравнение расходов и доходов по месяцам |

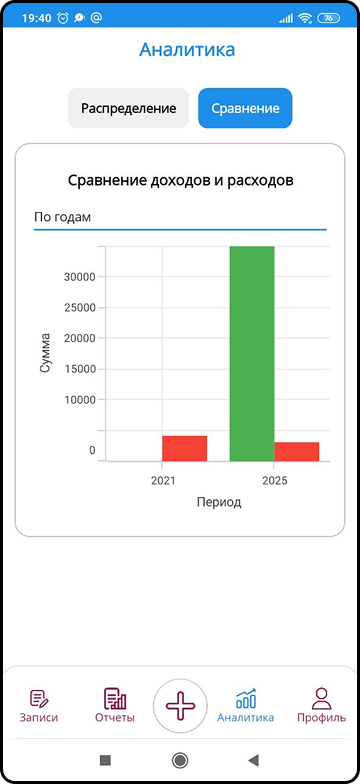


Рисунок 54 – Сравнение расходов и доходов по годам

* 1. **Тестирование ежедневного уведомления**

Протестируем работу системы уведомлений пользователя. Приложение отправляет уведомления пользователю автоматически с момента установки на устройство. Каждый день в 19:00 должно приходить уведомление, напоминающее пользователю о необходимости внести расходы и доходы за день (рисунок 55).

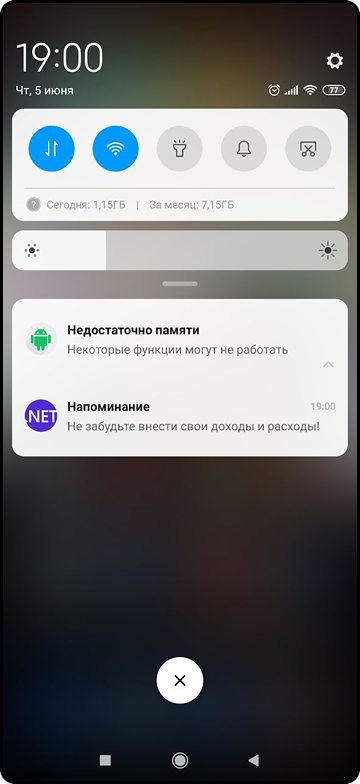


Рисунок 55 – Просмотр уведомления

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения выпускной квалификационной работы была достигнута поставленная цель – разработано мобильное приложение для ведения личного бюджета, обеспечивающее учет финансов, анализ расходов и доходов, а также визуализацию данных для повышения финансовой грамотности пользователей.

Основные результаты работы:

1. проведен анализ существующих аналогов (п. 1.1.2 пояснительной записки), на основе которого определены ключевые функциональные требования к приложению,
2. изучены теоретические основы разработки мобильных приложений (п. 2.4), включая выбор технологий, что позволило создать кроссплатформенное приложение,
3. реализовано полнофункциональное приложение (п. 3), включающее:

* учет операций;
* регистрацию и авторизацию пользователей;
* аналитику с графическим отображением;
* систему достижений для пользователей.

1. проведено комплексное тестирование (п. 4), подтвердившее корректную работу алгоритмов.

Все требования, сформулированные во введении, выполнены в полном объеме. Разработанное приложение позволяет пользователям:

* эффективно контролировать свои финансы,
* минимизировать необдуманные траты,
* формировать здоровые финансовые привычки.

Таким образом, решение соответствует поставленным задачам и может быть использовано в повседневной жизни для повышения финансовой дисциплины.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Чекашева Олеся Геннадьевна, Теплова Валерия Алексеевна Финансовая грамотность и потребительское поведение // Вестник ПензГУ. 2024. №3 (47). URL: https://cyberleninka.ru/article/n/finansovaya-gramotnost-i-potrebitelskoe-povedenie31 (дата обращения: 14.04.2025).
2. Словарь финансово-экономических терминов : словарь / А. В. Шаркова, А. А. Килячков, Е. В. Маркина, С. П. Солянникова. — 2-е изд. — Москва : Дашков и К, 2017. — 1168 с. — ISBN 978-5-394-02801-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/91226 (дата обращения: 14.04.2025).
3. Белоусова В.Ю., Скворцова М.В. Психологические факторы нерационального финансового поведения населения // Финансы и кредит. - 2020. - №5. - С. 7-9.
4. Топ-8 приложений для личных финансов //   
   Журнал Финуслуги URL: https://finuslugi.ru/navigator/nakopit-i-sohranit/stat\_top\_8\_prilozhenij\_dlya\_lichnyh\_finansov (дата обращения: 15.04.2025).
5. Остроух, А. В. Проектирование информационных систем : монография / А. В. Остроух, Н. Е. Суркова. — 2-е изд., стер. — Санкт-Петербург : Лань, 2021. — 164 с. — ISBN 978-5-8114-8377-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/175513 (дата обращения: 15.04.2025).
6. Функциональные и нефункциональные требования: различия и особенности // Блог Практикума URL: https://practicum.yandex.ru/ blog/funkcionalnye-i-nefunkcionalnye-trebovaniya/ (дата обращения: 03.05.2025).
7. Девицына С.Н., Пилькевич П.В., Удод Е.В. Способы улучшения защищенности сервисов, использующих JWT-токены // Экономика. Информатика. - 2023. - №50. - С. 144-151.
8. Вигерс Карл И., Битти Джой Разработка требований к программному обеспечению. - 3-е изд. - СПб: БХВ, 2024. - 736 с.
9. Прайс Марк Дж. C# 10 и .NET 6. Современная кроссплатформенная разработка. - 6-е изд. - СПб: Питер, 2023. - 848 с.
10. Документация по пользовательскому интерфейсу приложений для .NET с несколькими платформами // Mocrosoft URL: https://learn.microsoft.com/ru-ru/dotnet/maui/?view=net-maui-9.0 (дата обращения: 03.05.2025).
11. Эволюция Xamarin в .NET MAUI // Xamarin URL: https://xamarin.ru/knowledge-base/mobile-application-development/ehvolyuciya-xamarin-v-net-maui/ (дата обращения: 03.05.2025).
12. Лок, Э. ASP.Net Core в действии / Э. Лок; перевод с английского Д. А. Беликов. — Москва : ДМК Пресс, 2021. — 906 с.
13. Салахалдин Джуба, Андрей Волков Изучаем PostgreSQL 10. - 2-е изд. - Москва: ДМК Пресс, 2019. - 400 с.
14. Джон П. Смит Entity Framework Core в действии. - 2-е изд. - М.: ДМК, 2022. - 689 с.
15. Садовая Е. Н. Современные вызовы и угрозы информационной безопасности rest api и способы их предотвращения // Молодой исследователь Дона. 2023. №3 (42).
16. DFD: примеры и правила построения диаграмм потоков данных // Практикум URL: https://practicum.yandex.ru/blog/diagramma-potokov-dannyh-dfd/ (дата обращения: 01.06.2025).
17. В.В. Миронов, К.В. Миронов Концептуально-онтологические аспекты множественности предка в информационных моделях "сущность-связь" // Онтология проектирования. - 2024. - №14. - С. 493-503.

# **ПРИЛОЖЕНИЕ А (обязательное)**

**Код миграции**

using System;

using Microsoft.EntityFrameworkCore.Migrations;

using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

namespace FinanceApp.WebApi.Migrations

{

/// <inheritdoc />

public partial class InitialCreate : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Achievements",

columns: table => new

{

Id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

Title = table.Column<string>(type: "character varying(100)", maxLength: 100, nullable: false),

Description = table.Column<string>(type: "character varying(300)", maxLength: 300, nullable: false),

IconUrl = table.Column<string>(type: "text", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Achievements", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Categories",

columns: table => new

{

Id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

Name = table.Column<string>(type: "character varying(255)", maxLength: 255, nullable: false),

Icon = table.Column<string>(type: "text", nullable: false),

UserId = table.Column<int>(type: "integer", nullable: true),

IsIncome = table.Column<bool>(type: "boolean", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Categories", x => x.Id);

});

migrationBuilder.CreateTable(

name: "ReccuringTransactions",

columns: table => new

{

Id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

TransactionId = table.Column<int>(type: "integer", nullable: false),

RepeatType = table.Column<int>(type: "integer", nullable: false),

LastAutoCreatedDate = table.Column<DateTime>(type: "timestamp with time zone", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_ReccuringTransactions", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Users",

columns: table => new

{

Id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

UserName = table.Column<string>(type: "character varying(255)", maxLength: 255, nullable: false),

Email = table.Column<string>(type: "character varying(255)", maxLength: 255, nullable: false),

PasswordHash = table.Column<string>(type: "text", nullable: false),

RefreshToken = table.Column<string>(type: "text", nullable: false),

RefreshTokenExpiryTime = table.Column<DateTime>(type: "timestamp with time zone", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Users", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Transactions",

columns: table => new

{

Id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

Amount = table.Column<decimal>(type: "numeric", nullable: false),

Description = table.Column<string>(type: "character varying(255)", maxLength: 255, nullable: true),

Date = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),

IsIncome = table.Column<bool>(type: "boolean", nullable: false),

ParentTransactionId = table.Column<int>(type: "integer", nullable: true),

CategoryId = table.Column<int>(type: "integer", nullable: false),

UserId = table.Column<int>(type: "integer", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Transactions", x => x.Id);

table.ForeignKey(

name: "FK\_Transactions\_Categories\_CategoryId",

column: x => x.CategoryId,

principalTable: "Categories",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_Transactions\_Users\_UserId",

column: x => x.UserId,

principalTable: "Users",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "UserAchievements",

columns: table => new

{

Id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

UserId = table.Column<int>(type: "integer", nullable: false),

AchievementId = table.Column<int>(type: "integer", nullable: false),

DateUnlocked = table.Column<DateTime>(type: "timestamp with time zone", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_UserAchievements", x => x.Id);

table.ForeignKey(

name: "FK\_UserAchievements\_Achievements\_AchievementId",

column: x => x.AchievementId,

principalTable: "Achievements",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_UserAchievements\_Users\_UserId",

column: x => x.UserId,

principalTable: "Users",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "IX\_Transactions\_CategoryId",

table: "Transactions",

column: "CategoryId");

migrationBuilder.CreateIndex(

name: "IX\_Transactions\_UserId",

table: "Transactions",

column: "UserId");

migrationBuilder.CreateIndex(

name: "IX\_UserAchievements\_AchievementId",

table: "UserAchievements",

column: "AchievementId");

migrationBuilder.CreateIndex(

name: "IX\_UserAchievements\_UserId",

table: "UserAchievements",

column: "UserId");

}

}

}

# **ПРИЛОЖЕНИЕ Б (обязательное)**

**Основой код программы**

Листинг файла AchievementService.cs

using FinanceApp.Core.Models;

using System.Net.Http.Json;

namespace FinanceApp.Maui.Services

{

public class AchievementService

{

private readonly HttpClient \_httpClient;

public AchievementService(IHttpClientFactory httpClientFactory)

{

\_httpClient = httpClientFactory.CreateClient("FinanceApi");

}

public async Task<List<UserAchievement>> GetUserAchievementsAsync()

{

if (Connectivity.Current.NetworkAccess != NetworkAccess.Internet)

{

throw new Exception("No internet connection");

}

return await \_httpClient.GetFromJsonAsync<List<UserAchievement>>("api/Achievement/userAchievements");

}

public async Task<List<UserAchievement>> CheckAndUpdateAchievementsAsync()

{

if (Connectivity.Current.NetworkAccess != NetworkAccess.Internet)

{

throw new Exception("No internet connection");

}

return await \_httpClient.GetFromJsonAsync<List<UserAchievement>>("api/Achievement/checkAchievements");

}

}

}

Листинг файла AuthService.cs

using FinanceApp.Core.Models;

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Net.Http.Json;

namespace FinanceApp.Maui.Services

{

public class AuthService

{

private readonly HttpClient \_httpClient;

public AuthService(IHttpClientFactory httpClientFactory)

{

\_httpClient = httpClientFactory.CreateClient("FinanceApi");

}

public async Task<bool> RegisterAsync(string email, string username, string password)

{

var user = new

{

UserName = username,

PasswordHash = password,

Email = email,

RefreshToken = string.Empty,

RefreshTokenExpiryTime = DateTime.MinValue

};

var response = await \_httpClient.PostAsJsonAsync("api/Auth/register", user);

return response.IsSuccessStatusCode;

}

public async Task<bool> LoginAsync(string email, string password)

{

var user = new { Email = email, Password = password };

var response = await \_httpClient.PostAsJsonAsync("api/Auth/login", user);

if (!response.IsSuccessStatusCode) return false;

var result = await response.Content.ReadFromJsonAsync<Dictionary<string, string>>();

Preferences.Set("jwt\_token", result["token"]);

Preferences.Set("refresh\_token", result["refreshToken"]);

return true;

}

public async Task<bool> RefreshTokenAsync()

{

try

{

var jwtToken = Preferences.Get("jwt\_token", null);

var refreshToken = Preferences.Get("refresh\_token", null);

if (string.IsNullOrEmpty(jwtToken) || string.IsNullOrEmpty(refreshToken))

return false;

var tokens = new { Token = jwtToken, RefreshToken = refreshToken };

var response = await \_httpClient.PostAsJsonAsync("api/Auth/refresh", tokens);

Console.WriteLine($"Refresh status: {response.StatusCode}"); // Логирование

if (!response.IsSuccessStatusCode)

{

var errorContent = await response.Content.ReadAsStringAsync();

Console.WriteLine($"Refresh error: {errorContent}");

return false;

}

var result = await response.Content.ReadFromJsonAsync<Dictionary<string, string>>();

if (result == null || !result.ContainsKey("token") || !result.ContainsKey("refreshToken"))

{

Console.WriteLine("Invalid response format");

return false;

}

Preferences.Set("jwt\_token", result["token"]);

Preferences.Set("refresh\_token", result["refreshToken"]);

Console.WriteLine("Tokens updated successfully");

return true;

}

catch (Exception ex)

{

Console.WriteLine($"Refresh exception: {ex}");

return false;

}

}

public async Task<bool> IsAuthenticatedAsync()

{

var token = Preferences.Get("jwt\_token", null);

if (string.IsNullOrEmpty(token))

return false;

try

{

var tokenHandler = new JwtSecurityTokenHandler();

var jwtToken = tokenHandler.ReadJwtToken(token);

if (jwtToken.ValidTo >= DateTime.UtcNow)

return true;

return await RefreshTokenAsync();

}

catch

{

return false;

}

}

}

}

Листинг файла CategoryService.cs

using System.Net.Http.Json;

using FinanceApp.Core.Models;

public class CategoryService

{

private readonly HttpClient \_httpClient;

public CategoryService(IHttpClientFactory httpClientFactory)

{

\_httpClient = httpClientFactory.CreateClient("FinanceApi");

}

// Получение списка категорий

public async Task<List<Category>> GetCateroryAsync()

{

if (Connectivity.Current.NetworkAccess != NetworkAccess.Internet)

{

throw new Exception("No internet connection");

}

return await \_httpClient.GetFromJsonAsync<List<Category>>("api/Categories");

}

// Добавление новой категории

public async Task AddCategoryAsync(Category category)

{

await \_httpClient.PostAsJsonAsync("api/Categories", category);

}

// Получение списка категорий текущего пользователя

public async Task<List<Category>> GetCategoriesCurrentUserAsync()

{

if (Connectivity.Current.NetworkAccess != NetworkAccess.Internet)

{

throw new Exception("No internet connection");

}

return await \_httpClient.GetFromJsonAsync<List<Category>>("api/Categories/currentUser");

}

//Удаление категории

public async Task DeleteCategoryAsync(int id)

{

await \_httpClient.DeleteAsync($"api/Categories/{id}");

}

}

Листинг файла NotificationService.cs

using Plugin.LocalNotification;

namespace FinanceApp.Maui.Services

{

public class NotificationService

{

private readonly INotificationService \_notificationService;

public NotificationService(INotificationService notificationService)

{

\_notificationService = notificationService;

}

public void ScheduleDailyNotification(int hour = 9, int minute = 0)

{

var notification = new NotificationRequest

{

NotificationId = 100,

Title = "Напоминание",

Description = "Не забудьте внести свои доходы и расходы!",

Schedule = new NotificationRequestSchedule

{

NotifyTime = GetNextNotifyTime(hour, minute),

NotifyRepeatInterval = TimeSpan.FromDays(1),

RepeatType = NotificationRepeat.TimeInterval

}

};

\_notificationService.Show(notification);

}

private static DateTime GetNextNotifyTime(int hour, int minute)

{

var now = DateTime.Now;

var scheduledTime = new DateTime(now.Year, now.Month, now.Day, hour, minute, 0);

if (scheduledTime <= now)

scheduledTime = scheduledTime.AddDays(1);

return scheduledTime;

}

}

}

Листинг файла ReportService.cs

using FinanceApp.Core.Models;

using System.Net.Http.Json;

namespace FinanceApp.Maui.Services

{

public class ReportService

{

private readonly HttpClient \_httpClient;

public ReportService(IHttpClientFactory httpClientFactory)

{

\_httpClient = httpClientFactory.CreateClient("FinanceApi");

}

public async Task<byte[]> GenerateReportAsync(List<Transaction> transactions, DateTime fromDate, DateTime toDate, string tab)

{

var requestData = new

{

Transactions = transactions,

FromDate = fromDate,

ToDate = toDate,

Tab = tab

};

var response = await \_httpClient.PostAsJsonAsync("api/reports/export", requestData);

response.EnsureSuccessStatusCode();

return await response.Content.ReadAsByteArrayAsync();

}

}

}

Листинг файла TransactionsService.cs

using System.Net.Http.Json;

using FinanceApp.Core.Models;

public class TransactionsService

{

private readonly HttpClient \_httpClient;

public TransactionsService(IHttpClientFactory httpClientFactory)

{

\_httpClient = httpClientFactory.CreateClient("FinanceApi");

}

// Получение списка транзакций

public async Task<List<Transaction>> GetTransactionsAsync()

{

if (Connectivity.Current.NetworkAccess != NetworkAccess.Internet)

{

throw new Exception("No internet connection");

}

return await \_httpClient.GetFromJsonAsync<List<Transaction>>("api/Transactions");

}

public async Task<List<Transaction>> GetTransactionsCurrentUserAsync()

{

if (Connectivity.Current.NetworkAccess != NetworkAccess.Internet)

{

throw new Exception("No internet connection");

}

return await \_httpClient.GetFromJsonAsync<List<Transaction>>("api/Transactions/currentUser");

}

// Добавление новой транзакции

public async Task AddTransactionAsync(Transaction transaction)

{

await \_httpClient.PostAsJsonAsync("api/Transactions", transaction);

}

//Обновление информации о транзакции

public async Task UpdateTransactionAsync(Transaction transaction, int id)

{

await \_httpClient.PutAsJsonAsync($"api/Transactions/{id}", transaction);

}

//Удаление информации о транзакции

public async Task DeleteTransactionAsync(int id)

{

await \_httpClient.DeleteAsync($"api/Transactions/{id}");

}

}

Листинг файла UsersService.cs

using System.Net.Http.Json;

using FinanceApp.Core.Models;

namespace FinanceApp.Maui.Services

{

public class UsersService

{

private readonly HttpClient \_httpClient;

public UsersService(IHttpClientFactory httpClientFactory)

{

\_httpClient = httpClientFactory.CreateClient("FinanceApi");

}

public async Task<HttpResponseMessage> UpdateUsersAsync(UpdateUserDto user, int id)

{

return await \_httpClient.PutAsJsonAsync($"api/Users/{id}", user);

}

public async Task<HttpResponseMessage> ChangePasswordAsync(int id, string oldPassword, string newPassword)

{

var dto = new

{

CurrentPassword = oldPassword,

NewPassword = newPassword

};

return await \_httpClient.PutAsJsonAsync($"api/Users/{id}/change-password", dto);

}

}

}

Листинг файла JwtHelper.cs

using System.IdentityModel.Tokens.Jwt;

namespace FinanceApp.Maui

{

public static class JwtHelper

{

public static (string UserName, string Email, string UserId) ParseToken(string jwt)

{

var handler = new JwtSecurityTokenHandler();

var token = handler.ReadJwtToken(jwt);

var userName = token.Claims.FirstOrDefault(c => c.Type == "unique\_name")?.Value;

var email = token.Claims.FirstOrDefault(c => c.Type == "email")?.Value;

var userId = token.Claims.FirstOrDefault(c => c.Type == "nameid")?.Value;

return (userName, email, userId);

}

public static string? ParseTokenId(string jwt)

{

var handler = new JwtSecurityTokenHandler();

var token = handler.ReadJwtToken(jwt);

var userId = token.Claims.FirstOrDefault(c => c.Type == "nameid")?.Value;

return userId;

}

}

}

Листинг файла AuthTokenHandler.cs

namespace FinanceApp.Maui

{

public class AuthTokenHandler : DelegatingHandler

{

protected override async Task<HttpResponseMessage> SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)

{

var token = Preferences.Get("jwt\_token", null);

if (!string.IsNullOrEmpty(token))

{

request.Headers.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);

}

return await base.SendAsync(request, cancellationToken);

}

}

}

Листинг файла RegisterPage.xaml.cs

using FinanceApp.Maui.Services;

namespace FinanceApp.Maui;

public partial class RegisterPage : ContentPage

{

private readonly AuthService \_authService;

public RegisterPage(AuthService authService)

{

InitializeComponent();

\_authService = authService;

}

private async void OnRegisterClicked(object sender, EventArgs e)

{

bool success = await \_authService.RegisterAsync(EmailEntry.Text, UsernameEntry.Text, PasswordEntry.Text);

if (success)

{

await DisplayAlert("Успешно", "Вы зарегистрированы!", "OK");

await Navigation.PopAsync(); // назад на LoginPage после успешной регистрации

}

else

{

await DisplayAlert("Ошибка", "Пользователь с таким email уже существует!", "ОК");

}

}

private async void OnLoginRedirectClicked(object sender, EventArgs e)

{

await Navigation.PopAsync(); // возвращаемся к LoginPage

}

private void OnTogglePasswordClicked(object sender, EventArgs e)

{

PasswordEntry.IsPassword = !PasswordEntry.IsPassword;

TogglePasswordButton.Source = PasswordEntry.IsPassword ? "eye\_open.png" : "eye\_closed.png";

}

}

Листинг файла RegisterPage.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="FinanceApp.Maui.RegisterPage">

<VerticalStackLayout Padding="30" Spacing="15">

<Label Text="Регистрация" FontSize="24" HorizontalOptions="Center" />

<Entry x:Name="UsernameEntry" Placeholder="Имя пользователя" />

<Entry x:Name="EmailEntry" Placeholder="Email" />

<Frame CornerRadius="8"

Padding="0"

BorderColor="LightGray"

HasShadow="False"

HeightRequest="50">

<Grid>

<Entry x:Name="PasswordEntry"

Placeholder="Введите пароль"

IsPassword="True"

Margin="10,0,50,0"

VerticalOptions="Center"

BackgroundColor="Transparent"/>

<Image x:Name="TogglePasswordButton" Source="eye\_open.png"

HeightRequest="32"

WidthRequest="32"

HorizontalOptions="End"

Margin="0,0,10,0">

<Image.GestureRecognizers>

<TapGestureRecognizer Tapped="OnTogglePasswordClicked"/>

</Image.GestureRecognizers>

</Image>

</Grid>

</Frame>

<Button Text="Зарегистрироваться" Clicked="OnRegisterClicked" BackgroundColor="#1d8eea" />

<Button Text="Назад к авторизации" Clicked="OnLoginRedirectClicked" BackgroundColor="#1d8eea" />

</VerticalStackLayout>

</ContentPage>

Листинг файла LoginPage.xaml.cs

using FinanceApp.Maui.Services;

namespace FinanceApp.Maui;

public partial class LoginPage : ContentPage

{

private readonly AuthService \_authService;

private readonly TransactionsService \_transactionsService;

public LoginPage(AuthService authService, TransactionsService transactionsService)

{

InitializeComponent();

\_authService = authService;

\_transactionsService = transactionsService;

}

private async void OnLoginClicked(object sender, EventArgs e)

{

bool success = await \_authService.LoginAsync(EmailEntry.Text, PasswordEntry.Text);

if (success)

{

App.Current.MainPage = new NavigationPage(new MainPage());

}

else

{

await DisplayAlert("Ошибка", "Неверные данные!", "ОК");

}

}

private async void OnRegisterRedirectClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new RegisterPage(\_authService));

}

private void OnTogglePasswordClicked(object sender, EventArgs e)

{

PasswordEntry.IsPassword = !PasswordEntry.IsPassword;

TogglePasswordButton.Source = PasswordEntry.IsPassword ? "eye\_open.png" : "eye\_closed.png";

}

}

Листинг файла LoginPage.xaml

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="FinanceApp.Maui.LoginPage">

<VerticalStackLayout Padding="30" Spacing="15">

<Label Text="Авторизация" FontSize="24" HorizontalOptions="Center" />

<Entry x:Name="EmailEntry" Placeholder="Email" />

<Frame CornerRadius="8"

Padding="0"

BorderColor="LightGray"

HasShadow="False"

HeightRequest="50">

<Grid>

<Entry x:Name="PasswordEntry"

Placeholder="Пароль"

IsPassword="True"

Margin="10,0,50,0"

VerticalOptions="Center"

BackgroundColor="Transparent"/>

<Image x:Name="TogglePasswordButton" Source="eye\_open.png"

HeightRequest="32"

WidthRequest="32"

HorizontalOptions="End"

Margin="0,0,10,0">

<Image.GestureRecognizers>

<TapGestureRecognizer Tapped="OnTogglePasswordClicked"/>

</Image.GestureRecognizers>

</Image>

</Grid>

</Frame>

<Button Text="Войти" Clicked="OnLoginClicked" BackgroundColor="#1d8eea"/>

<Label Text="Нет аккаунта?" HorizontalOptions="Center" />

<Button Text="Перейти к регистрации" Clicked="OnRegisterRedirectClicked" BackgroundColor="#1d8eea"/>

</VerticalStackLayout>

</ContentPage>

Листинг файла MainPage.xaml.cs

using FinanceApp.Core.Models;

using FinanceApp.Maui.Services;

namespace FinanceApp.Maui

{

public partial class MainPage : ContentPage

{

private readonly TransactionsService \_transactionsService;

private bool \_suppressReload = false;

private int \_selectedYear;

private int \_selectedMonth;

protected override void OnAppearing()

{

base.OnAppearing();

var now = DateTime.Now;

if (YearPicker.ItemsSource == null)

{

var years = Enumerable.Range(now.Year - 5, 10).ToList();

YearPicker.ItemsSource = years;

}

if (YearPicker.SelectedItem == null)

YearPicker.SelectedItem = now.Year;

if (MonthPicker.ItemsSource == null)

{

var months = Enumerable.Range(1, 12)

.Select(m => new DateTime(2000, m, 1).ToString("MMMM"))

.ToList();

MonthPicker.ItemsSource = months;

}

if (MonthPicker.SelectedIndex == -1)

MonthPicker.SelectedIndex = now.Month - 1;

\_selectedYear = (int)YearPicker.SelectedItem;

\_selectedMonth = MonthPicker.SelectedIndex + 1;

if (!\_suppressReload)

LoadTransactions();

\_suppressReload = false;

// Устанавливаем текущую страницу для BottomMenu

if (this.FindParent<ContentPage>() is ContentPage page &&

page.FindByName<BottomMenu>("BottomMenu") is BottomMenu menu)

{

menu.CurrentPage = "records";

}

}

public MainPage()

{

InitializeComponent();

\_transactionsService = App.Current.Handler.MauiContext.Services.GetService<TransactionsService>();

LoadTransactions();

}

private async void OnTransactionTapped(object sender, TappedEventArgs e)

{

if (e.Parameter is Transaction transaction)

{

\_suppressReload = true;

await Navigation.PushModalAsync(new TransactionDetailsPage(transaction));

}

}

public async void LoadTransactions()

{

var transactions = await \_transactionsService.GetTransactionsCurrentUserAsync();

foreach (var t in transactions)

{

if (t.Date.Kind == DateTimeKind.Utc)

t.Date = t.Date.ToLocalTime();

}

var groupedTransactions = transactions

.Where(t => t.Date.Year == \_selectedYear && t.Date.Month == \_selectedMonth)

.OrderByDescending(t => t.Date)

.GroupBy(t => t.Date.Date)

.Select(g => new TransactionGroup(g.Key, g.ToList()))

.ToList();

TransactionsList.ItemsSource = groupedTransactions;

bool isEmpty = !groupedTransactions.Any();

TransactionsList.IsVisible = !isEmpty;

EmptyStateContainer.IsVisible = isEmpty;

}

private async void OnAddTransactionClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new AddTransactionPage(\_transactionsService));

}

private async void OnScanButtonClicked(object sender, EventArgs e)

{

await Navigation.PushAsync(new ScanReceiptPage(\_transactionsService));

}

private async void OnEditClicked(object sender, TappedEventArgs e)

{

if (e.Parameter is Transaction transaction)

{

await Navigation.PushAsync(new AddTransactionPage(\_transactionsService, transaction));

}

}

private async void OnDeleteClicked(object sender, TappedEventArgs e)

{

if (e.Parameter is not Transaction transaction)

return;

bool answer = await DisplayAlert(

"Подтверждение удаления",

$"Вы точно хотите удалить транзакцию \"{transaction.Description}\" на сумму {transaction.Amount:C}?",

"Да, удалить",

"Отмена");

if (answer)

{

try

{

await \_transactionsService.DeleteTransactionAsync(transaction.Id);

LoadTransactions(); // Обновление списка после удаления

await DisplayAlert("Успешно", "Транзакция удалена", "OK");

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось удалить транзакцию: {ex.Message}", "OK");

}

}

}

private void OnFilterChanged(object sender, EventArgs e)

{

if (YearPicker.SelectedItem is int year && MonthPicker.SelectedIndex >= 0)

{

\_selectedYear = year;

\_selectedMonth = MonthPicker.SelectedIndex + 1;

LoadTransactions();

}

}

}

}

public class TransactionGroup : List<Transaction>

{

public DateTime Date { get; }

public decimal TotalAmount { get; }

public TransactionGroup(DateTime date, List<Transaction> transactions) : base(transactions)

{

Date = date;

}

}

Листинг файла ProfilePage.xaml.cs

using FinanceApp.Core.Models;

using FinanceApp.Maui.Services;

using System.IdentityModel.Tokens.Jwt;

using System.Net.Http.Json;

using System.Diagnostics;

using FinanceApp.Maui.ModalWindow;

using System.Collections.ObjectModel;

using System.Windows.Input;

namespace FinanceApp.Maui;

public partial class ProfilePage : ContentPage

{

private readonly TransactionsService \_transactionsService;

private readonly AuthService \_authService;

private readonly UsersService \_usersService;

private readonly CategoryService \_categoryService;

private readonly AchievementService \_achievementService;

private string \_originalName;

private string \_originalEmail;

private int \_userId;

private const int TotalAchievements = 4; // Всего достижений в системе

public ObservableCollection<Category> Categories { get; set; } = new ObservableCollection<Category>();

public ObservableCollection<UserAchievement> UserAchievements { get; set; } = new ObservableCollection<UserAchievement>();

public ProfilePage()

{

InitializeComponent();

BindingContext = this;

\_transactionsService = App.Current.Handler.MauiContext.Services.GetService<TransactionsService>();

\_authService = App.Current.Handler.MauiContext.Services.GetService<AuthService>();

\_usersService = App.Current.Handler.MauiContext.Services.GetService<UsersService>();

\_categoryService = App.Current.Handler.MauiContext.Services.GetService<CategoryService>();

\_achievementService = App.Current.Handler.MauiContext.Services.GetService<AchievementService>();

}

protected override async void OnAppearing()

{

base.OnAppearing();

if (this.FindParent<ContentPage>() is ContentPage page &&

page.FindByName<BottomMenu>("BottomMenu") is BottomMenu menu)

{

menu.CurrentPage = "profile";

}

var token = Preferences.Get("jwt\_token", null);

if (!string.IsNullOrWhiteSpace(token))

{

var (userName, email, userId) = JwtHelper.ParseToken(token);

UserNameLabel.Text = userName;

UserNameEntry.Text = userName;

EmailLabel.Text = email;

EmailEntry.Text = email;

\_userId = int.Parse(userId);

await LoadCategories();

await LoadUserAchievementsAndLevel();

}

}

private async Task LoadCategories()

{

try

{

var categories = await \_categoryService.GetCategoriesCurrentUserAsync();

MainThread.BeginInvokeOnMainThread(() =>

{

Categories.Clear();

foreach (var category in categories)

{

Categories.Add(category);

}

});

}

catch (Exception ex)

{

Debug.WriteLine($"Ошибка загрузки категорий: {ex.Message}");

await DisplayAlert("Ошибка", "Не удалось загрузить категории", "OK");

}

}

private async Task LoadUserAchievementsAndLevel()

{

try

{

var userAchievements = await \_achievementService.GetUserAchievementsAsync();

MainThread.BeginInvokeOnMainThread(() =>

{

UserAchievements.Clear();

foreach (var ua in userAchievements)

{

UserAchievements.Add(ua);

}

// Обновляем CollectionView

AchievementsCollectionView.ItemsSource = UserAchievements;

// Обработка отсутствия достижений

NoAchievementsLabel.IsVisible = UserAchievements.Count == 0;

AchievementsCollectionView.IsVisible = UserAchievements.Count > 0;

// Обновляем уровень и прогресс

int achievedCount = UserAchievements.Count;

int level = CalculateLevel(achievedCount);

double progress = TotalAchievements > 0

? (double)achievedCount / TotalAchievements

: 0;

// Обновляем UI

LevelProgressBar.Progress = progress;

LevelLabel.Text = $"Уровень: {level}";

});

}

catch (Exception ex)

{

Debug.WriteLine($"Ошибка загрузки достижений: {ex.Message}");

}

}

// Логика расчёта уровня

private int CalculateLevel(int achievedCount)

{

return 1 + achievedCount;

}

public ICommand DeleteCategoryCommand => new Command<Category>(async (category) =>

{

if (category == null) return;

bool confirm = await Application.Current.MainPage.DisplayAlert(

"Удалить категорию",

$"Вы уверены, что хотите удалить категорию \"{category.Name}\"?",

"Удалить", "Отмена");

if (confirm)

{

try

{

await \_categoryService.DeleteCategoryAsync(category.Id);

Categories.Remove(category);

}

catch (Exception ex)

{

Debug.WriteLine($"Ошибка удаления категории: {ex.Message}");

}

}

});

private async void OnLogoutClicked(object sender, EventArgs e)

{

Preferences.Remove("jwt\_token");

Preferences.Remove("refresh\_token");

Application.Current.MainPage = new NavigationPage(new LoginPage(\_authService, \_transactionsService));

}

private void OnEditTapped(object sender, EventArgs e)

{

\_originalName = UserNameLabel.Text;

\_originalEmail = EmailLabel.Text;

UserNameLabel.IsVisible = false;

EmailLabel.IsVisible = false;

UserNameEntry.IsVisible = true;

EmailEntry.IsVisible = true;

EditIcon.IsVisible = false;

EditButtons.IsVisible = true;

}

private void OnCancelTapped(object sender, EventArgs e)

{

UserNameEntry.Text = \_originalName;

EmailEntry.Text = \_originalEmail;

ExitEditMode();

}

private async void OnSaveTapped(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(UserNameEntry.Text) ||

string.IsNullOrWhiteSpace(EmailEntry.Text))

{

await DisplayAlert("Ошибка", "Имя и email не должны быть пустыми", "ОК");

return;

}

try

{

var dto = new UpdateUserDto

{

UserName = UserNameEntry.Text,

Email = EmailEntry.Text

};

var response = await \_usersService.UpdateUsersAsync(dto, \_userId);

if (response.IsSuccessStatusCode)

{

var result = await response.Content.ReadFromJsonAsync<Dictionary<string, string>>();

Preferences.Set("jwt\_token", result["token"]);

Preferences.Set("refresh\_token", result["refreshToken"]);

UserNameLabel.Text = dto.UserName;

EmailLabel.Text = dto.Email;

ExitEditMode();

}

else

{

await DisplayAlert("Ошибка", "Не удалось сохранить данные", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось сохранить данные: {ex.Message}", "ОК");

}

}

private void ExitEditMode()

{

UserNameLabel.IsVisible = true;

EmailLabel.IsVisible = true;

UserNameEntry.IsVisible = false;

EmailEntry.IsVisible = false;

EditIcon.IsVisible = true;

EditButtons.IsVisible = false;

}

private void OnToggleChangePasswordClicked(object sender, EventArgs e)

{

ChangePasswordBlock.IsVisible = !ChangePasswordBlock.IsVisible;

}

private void OnChangePasswordCancelClicked(object sender, EventArgs e)

{

ChangePasswordBlock.IsVisible = false;

OldPasswordEntry.Text = "";

NewPasswordEntry.Text = "";

RepeatPasswordEntry.Text = "";

}

private async void OnChangePasswordSaveClicked(object sender, EventArgs e)

{

var oldPassword = OldPasswordEntry.Text;

var newPassword = NewPasswordEntry.Text;

var repeatPassword = RepeatPasswordEntry.Text;

if (string.IsNullOrWhiteSpace(oldPassword) ||

string.IsNullOrWhiteSpace(newPassword) ||

string.IsNullOrWhiteSpace(repeatPassword))

{

await DisplayAlert("Ошибка", "Все поля обязательны", "ОК");

return;

}

if (newPassword != repeatPassword)

{

await DisplayAlert("Ошибка", "Пароли не совпадают", "ОК");

return;

}

try

{

var response = await \_usersService.ChangePasswordAsync(\_userId, oldPassword, newPassword);

if (response.IsSuccessStatusCode)

{

await DisplayAlert("Успех", "Пароль успешно изменён", "ОК");

OnChangePasswordCancelClicked(null, null);

}

else

{

await DisplayAlert("Ошибка", "Старый пароль неверен или произошла ошибка", "ОК");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Не удалось изменить пароль: {ex.Message}", "ОК");

}

}

private async void AddCategoryClicked(object sender, EventArgs e)

{

await Navigation.PushModalAsync(new AddCategoryPage());

}

}

Листинг файла ReportsPage.xaml.cs

using Microcharts;

using SkiaSharp;

using System.Reflection.Metadata;

using System.IO;

using Microcharts.Maui;

using System.Globalization;

using FinanceApp.Core.Models;

using MauiColors = Microsoft.Maui.Graphics.Colors;

using MauiColor = Microsoft.Maui.Graphics.Color;

using System.Diagnostics;

using FinanceApp.Maui.Services;

namespace FinanceApp.Maui.Pages;

public partial class ReportsPage : ContentPage

{

private readonly TransactionsService \_transactionsService;

private readonly ReportService \_reportService;

private string \_currentTab = "General";

private List<Transaction> \_transactions = new();

private DateTime \_fromDate = DateTime.Now.AddMonths(-1);

private DateTime \_toDate = DateTime.Now;

public ReportsPage()

{

InitializeComponent();

\_transactionsService = App.Current.Handler.MauiContext.Services.GetService<TransactionsService>();

\_reportService = App.Current.Handler.MauiContext.Services.GetService<ReportService>();

}

protected override void OnAppearing()

{

base.OnAppearing();

InitializePeriodPickers();

UpdatePeriodButtonsText();

LoadTransactions();

BottomMenu.CurrentPage = "reports";

}

private void InitializePeriodPickers()

{

// Инициализация годов

var years = Enumerable.Range(DateTime.Now.Year - 5, 6).Reverse().ToList();

FromYearPicker.ItemsSource = years;

ToYearPicker.ItemsSource = years;

FromYearPicker.SelectedItem = \_fromDate.Year;

ToYearPicker.SelectedItem = \_toDate.Year;

// Инициализация месяцев

InitializeMonths(FromMonthContainer, \_fromDate.Month);

InitializeMonths(ToMonthContainer, \_toDate.Month);

}

private void InitializeMonths(FlexLayout container, int selectedMonth)

{

container.Children.Clear();

for (int i = 1; i <= 12; i++)

{

var monthName = CultureInfo.CurrentCulture.DateTimeFormat.GetAbbreviatedMonthName(i);

var button = new Button

{

Text = monthName,

WidthRequest = 70,

HeightRequest = 40,

Margin = 5,

FontSize = 14,

CornerRadius = 5,

BindingContext = i

};

if (i == selectedMonth)

{

button.BackgroundColor = MauiColor.FromArgb("#1d8eea");

button.TextColor = Colors.White;

}

else

{

button.BackgroundColor = Colors.Transparent;

button.TextColor = Colors.Black;

}

button.Clicked += (sender, e) => OnMonthSelected(sender, e, container == FromMonthContainer);

container.Children.Add(button);

}

}

private void OnMonthSelected(object sender, EventArgs e, bool isFromPeriod)

{

if (sender is Button button && button.BindingContext is int month)

{

var yearPicker = isFromPeriod ? FromYearPicker : ToYearPicker;

if (yearPicker.SelectedItem is int year)

{

if (isFromPeriod)

{

\_fromDate = new DateTime(year, month, 1);

}

else

{

\_toDate = new DateTime(year, month, 1).AddMonths(1).AddDays(-1);

}

// Обновляем стили кнопок

var container = isFromPeriod ? FromMonthContainer : ToMonthContainer;

foreach (var child in container.Children)

{

if (child is Button monthButton)

{

bool isSelected = monthButton.BindingContext is int m && m == month;

monthButton.BackgroundColor = isSelected ? MauiColor.FromArgb("#1d8eea") : Colors.Transparent;

monthButton.TextColor = isSelected ? Colors.White : Colors.Black;

}

}

UpdatePeriodButtonsText();

LoadTransactions();

}

}

}

private void UpdateSelectedPeriod(bool isFromPeriod)

{

var yearPicker = isFromPeriod ? FromYearPicker : ToYearPicker;

if (yearPicker.SelectedItem is int year)

{

if (isFromPeriod)

{

\_fromDate = new DateTime(year, \_fromDate.Month, 1);

}

else

{

\_toDate = new DateTime(year, \_toDate.Month, 1).AddMonths(1).AddDays(-1);

}

UpdatePeriodButtonsText();

LoadTransactions();

}

}

private void UpdatePeriodButtonsText()

{

FromPeriodButton.Text = \_fromDate.ToString("MMM yyyy", CultureInfo.CurrentCulture);

ToPeriodButton.Text = \_toDate.ToString("MMM yyyy", CultureInfo.CurrentCulture);

}

private async void LoadTransactions()

{

\_transactions = await \_transactionsService.GetTransactionsCurrentUserAsync();

\_transactions = \_transactions

.Where(t => t.Date >= \_fromDate && t.Date <= \_toDate)

.ToList();

switch (\_currentTab)

{

case "General":

ShowGeneralReport();

break;

case "Top":

ShowTopCategoriesReport();

break;

case "Pie":

ShowPieChartReport();

break;

}

}

private void ShowFromPeriodPicker(object sender, EventArgs e)

{

FromPeriodPopup.IsVisible = true;

ToPeriodPopup.IsVisible = false;

}

private void ShowToPeriodPicker(object sender, EventArgs e)

{

ToPeriodPopup.IsVisible = true;

FromPeriodPopup.IsVisible = false;

}

private void HideFromPeriodPicker(object sender, EventArgs e)

{

FromPeriodPopup.IsVisible = false;

UpdateSelectedPeriod(true);

}

private void HideToPeriodPicker(object sender, EventArgs e)

{

ToPeriodPopup.IsVisible = false;

UpdateSelectedPeriod(false);

}

private void OnTabClicked(object sender, EventArgs e)

{

var button = (Button)sender;

GeneralTabButton.BackgroundColor = TopCategoriesTabButton.BackgroundColor = PieChartTabButton.BackgroundColor = MauiColor.FromArgb("#f0f0f0");

GeneralTabButton.TextColor = TopCategoriesTabButton.TextColor = PieChartTabButton.TextColor = MauiColors.Black;

button.BackgroundColor = MauiColor.FromArgb("#1d8eea");

button.TextColor = MauiColors.White;

\_currentTab = button.Text switch

{

"Общая статистика" => "General",

"Топ-3 категории" => "Top",

"Диаграмма" => "Pie",

\_ => "General"

};

switch (\_currentTab)

{

case "General":

ShowGeneralReport();

break;

case "Top":

ShowTopCategoriesReport();

break;

case "Pie":

ShowPieChartReport();

break;

}

}

private void ShowGeneralReport()

{

var filteredTransactions = \_transactions

.Where(t => t.Date >= \_fromDate && t.Date <= \_toDate)

.ToList();

var income = filteredTransactions.Where(t => t.Category.IsIncome).Sum(t => t.Amount);

var expense = filteredTransactions.Where(t => !t.Category.IsIncome).Sum(t => t.Amount);

var total = income + expense;

var incomePercent = total == 0 ? 0 : income / total \* 100;

var expensePercent = total == 0 ? 0 : expense / total \* 100;

var topIncome = filteredTransactions

.Where(t => t.Category.IsIncome)

.OrderByDescending(t => t.Amount)

.FirstOrDefault();

var topExpense = filteredTransactions

.Where(t => !t.Category.IsIncome)

.OrderByDescending(t => t.Amount)

.FirstOrDefault();

var mostIncomeDay = filteredTransactions

.Where(t => t.Category.IsIncome)

.GroupBy(t => t.Date.Date)

.Select(g => new { Date = g.Key, Total = g.Sum(t => t.Amount) })

.OrderByDescending(x => x.Total)

.FirstOrDefault();

var mostExpenseDay = filteredTransactions

.Where(t => !t.Category.IsIncome)

.GroupBy(t => t.Date.Date)

.Select(g => new { Date = g.Key, Total = g.Sum(t => t.Amount) })

.OrderByDescending(x => x.Total)

.FirstOrDefault();

View CreateCard(string title, string value, Color color, string icon = "📊")

{

var grid = new Grid

{

RowDefinitions =

{

new RowDefinition { Height = GridLength.Auto },

new RowDefinition { Height = GridLength.Auto }

},

ColumnDefinitions =

{

new ColumnDefinition { Width = 30 },

new ColumnDefinition { Width = GridLength.Star }

}

};

var iconLabel = new Label

{

Text = icon,

FontSize = 20,

VerticalOptions = LayoutOptions.Center

};

Grid.SetRowSpan(iconLabel, 2);

Grid.SetRow(iconLabel, 0);

Grid.SetColumn(iconLabel, 0);

var titleLabel = new Label

{

Text = title,

FontSize = 14,

TextColor = Colors.Gray,

VerticalOptions = LayoutOptions.End

};

Grid.SetRow(titleLabel, 0);

Grid.SetColumn(titleLabel, 1);

var valueLabel = new Label

{

Text = value,

FontSize = 20,

FontAttributes = FontAttributes.Bold,

TextColor = color,

VerticalOptions = LayoutOptions.Start

};

Grid.SetRow(valueLabel, 1);

Grid.SetColumn(valueLabel, 1);

grid.Children.Add(iconLabel);

grid.Children.Add(titleLabel);

grid.Children.Add(valueLabel);

return new Frame

{

CornerRadius = 16,

BackgroundColor = Color.FromArgb("#FAFAFA"),

BorderColor = color,

HasShadow = true,

Padding = new Thickness(16),

Margin = new Thickness(0, 8),

Content = grid

};

}

View CreateSectionHeader(string text)

{

return new VerticalStackLayout

{

Margin = new Thickness(0, 24, 0, 4),

Children =

{

new Label

{

Text = text,

FontSize = 18,

FontAttributes = FontAttributes.Bold,

TextColor = Colors.Black

},

new BoxView

{

HeightRequest = 1,

BackgroundColor = Colors.LightGray,

HorizontalOptions = LayoutOptions.Fill

}

}

};

}

var layout = new VerticalStackLayout

{

Padding = 20,

Spacing = 0,

Children =

{

CreateSectionHeader("Общая статистика"),

CreateCard("Доходы", $"{income:C} ({incomePercent:F1}%)", MauiColors.Green, "📈"),

CreateCard("Расходы", $"{expense:C} ({expensePercent:F1}%)", MauiColors.Red, "📉"),

CreateSectionHeader("Топ категории"),

CreateCard("Доход", $"{topIncome?.Category?.Name ?? "—"} — {topIncome?.Amount:C}", MauiColors.Green, "💰"),

CreateCard("Расход", $"{topExpense?.Category?.Name ?? "—"} — {topExpense?.Amount:C}", MauiColors.Red, "💸"),

CreateSectionHeader("Пиковые дни"),

CreateCard("Доход", $"{mostIncomeDay?.Date:dd.MM.yyyy} — {mostIncomeDay?.Total:C}", MauiColors.Green, "📅"),

CreateCard("Расход", $"{mostExpenseDay?.Date:dd.MM.yyyy} — {mostExpenseDay?.Total:C}", MauiColors.Red, "🗓️")

}

};

ContentContainer.Content = new ScrollView { Content = layout };

}

private void ShowTopCategoriesReport()

{

var filteredTransactions = \_transactions

.Where(t => t.Date >= \_fromDate && t.Date <= \_toDate)

.ToList();

var totalExpenses = filteredTransactions

.Where(t => !t.Category.IsIncome)

.Sum(t => t.Amount);

var topCategories = filteredTransactions

.Where(t => !t.Category.IsIncome)

.GroupBy(t => t.Category.Name)

.Select(g => new

{

Category = g.Key,

Total = g.Sum(x => x.Amount),

Transactions = g.OrderByDescending(t => t.Date).ToList()

})

.OrderByDescending(x => x.Total)

.Take(3)

.ToList();

var layout = new VerticalStackLayout

{

Padding = 20,

Spacing = 16

};

foreach (var item in topCategories)

{

double percent = totalExpenses == 0 ? 0 : (double)(item.Total / totalExpenses) \* 100;

var categoryHeader = new HorizontalStackLayout

{

VerticalOptions = LayoutOptions.Center,

Spacing = 8,

Children =

{

new Label

{

Text = item.Category,

FontSize = 20,

FontAttributes = FontAttributes.Bold,

TextColor = Colors.Black,

HorizontalOptions = LayoutOptions.StartAndExpand

},

new Label

{

Text = $"{item.Total:C}",

FontSize = 18,

TextColor = Colors.Red,

HorizontalOptions = LayoutOptions.End

}

}

};

var progressBar = new Grid

{

HeightRequest = 6,

WidthRequest = 200,

BackgroundColor = Colors.LightGray,

HorizontalOptions = LayoutOptions.Start,

Margin = new Thickness(0, 4, 0, 8),

Children =

{

new BoxView

{

Color = Colors.Red,

HorizontalOptions = LayoutOptions.Start,

WidthRequest = 200 \* (percent / 100),

HeightRequest = 6,

CornerRadius = 3

}

}

};

var transactionsLayout = new VerticalStackLayout

{

Spacing = 4,

Padding = new Thickness(8, 0, 0, 0)

};

foreach (var transaction in item.Transactions)

{

var transactionRow = new HorizontalStackLayout

{

Spacing = 6,

Children =

{

new BoxView

{

Color = Colors.Gray,

WidthRequest = 6,

HeightRequest = 6,

CornerRadius = 3,

VerticalOptions = LayoutOptions.Center

},

new Label

{

Text = $"{transaction.Date:dd.MM.yyyy}: {transaction.Description}",

FontSize = 14,

TextColor = Colors.Gray,

HorizontalOptions = LayoutOptions.StartAndExpand

},

new Label

{

Text = $"{transaction.Amount:C}",

FontSize = 14,

TextColor = Colors.Red,

HorizontalOptions = LayoutOptions.End

}

}

};

transactionsLayout.Children.Add(transactionRow);

}

var card = new Frame

{

CornerRadius = 12,

Padding = new Thickness(16),

Margin = new Thickness(0, 0, 0, 16),

BackgroundColor = Colors.White,

HasShadow = true,

Content = new VerticalStackLayout

{

Spacing = 4,

Children =

{

categoryHeader,

progressBar,

new Label

{

Text = $"Транзакций: {item.Transactions.Count}",

FontSize = 14,

TextColor = Colors.DarkGray,

Margin = new Thickness(0,0,0,8)

},

transactionsLayout

}

}

};

layout.Children.Add(card);

}

ContentContainer.Content = new ScrollView { Content = layout };

}

private void ShowPieChartReport()

{

var filteredTransactions = \_transactions

.Where(t => t.Date >= \_fromDate && t.Date <= \_toDate)

.ToList();

var palette = new[]

{

"#FF6F61", "#6B5B95", "#88B04B", "#F7CAC9", "#92A8D1",

"#955251", "#B565A7", "#009B77", "#DD4124", "#45B8AC"

};

var incomePalette = new[]

{

"#4CAF50", "#81C784", "#388E3C", "#66BB6A", "#2E7D32"

};

var rnd = new Random();

var grouped = filteredTransactions

.GroupBy(t => new { t.Category.Name, t.Category.IsIncome })

.Select(g =>

{

var sum = g.Sum(x => x.Amount);

string colorHex = g.Key.IsIncome

? incomePalette[rnd.Next(incomePalette.Length)]

: palette[rnd.Next(palette.Length)];

return new

{

Category = g.Key.Name,

IsIncome = g.Key.IsIncome,

Total = sum,

Color = SKColor.Parse(colorHex),

ColorHex = colorHex

};

})

.OrderByDescending(x => x.Total)

.ToList();

var entries = grouped.Select(g => new ChartEntry((float)g.Total)

{

Label = g.Category,

ValueLabel = g.Total.ToString("C0", CultureInfo.CurrentCulture),

Color = g.Color

}).ToList();

var pieChart = new PieChart

{

Entries = entries,

BackgroundColor = SKColors.Transparent,

HoleRadius = 0.4f,

LabelTextSize = 0,

LabelColor = SKColors.Transparent

};

View CreateLegendItem(dynamic item)

{

return new Frame

{

Padding = new Thickness(12, 8),

Margin = new Thickness(0, 4),

CornerRadius = 12,

HasShadow = false,

BackgroundColor = Color.FromArgb("#F0F0F0"),

Content = new HorizontalStackLayout

{

Spacing = 12,

VerticalOptions = LayoutOptions.Center,

Children =

{

new BoxView

{

Color = Color.FromArgb(item.ColorHex),

WidthRequest = 16,

HeightRequest = 16,

CornerRadius = 4,

VerticalOptions = LayoutOptions.Center

},

new Label

{

Text = $"{item.Category}: {item.Total:C0}",

FontSize = 16,

TextColor = Colors.Black,

VerticalOptions = LayoutOptions.Center

}

}

}

};

}

View CreateSectionHeader(string text, Color color)

{

return new VerticalStackLayout

{

Margin = new Thickness(0, 20, 0, 4),

Children =

{

new Label

{

Text = text,

FontSize = 18,

FontAttributes = FontAttributes.Bold,

TextColor = color

},

new BoxView

{

HeightRequest = 1,

BackgroundColor = Colors.LightGray,

HorizontalOptions = LayoutOptions.Fill

}

}

};

}

var legendLayout = new VerticalStackLayout

{

Spacing = 4,

Margin = new Thickness(20, 10)

};

var incomes = grouped.Where(x => x.IsIncome).ToList();

if (incomes.Any())

{

legendLayout.Children.Add(CreateSectionHeader("Доходы", MauiColors.Green));

foreach (var item in incomes)

legendLayout.Children.Add(CreateLegendItem(item));

}

var expenses = grouped.Where(x => !x.IsIncome).ToList();

if (expenses.Any())

{

legendLayout.Children.Add(CreateSectionHeader("Расходы", MauiColors.Red));

foreach (var item in expenses)

legendLayout.Children.Add(CreateLegendItem(item));

}

var mainLayout = new VerticalStackLayout

{

Padding = 20,

Children =

{

new ChartView

{

Chart = pieChart,

HeightRequest = 300,

Margin = new Thickness(0, 0, 0, 20)

},

legendLayout

}

};

ContentContainer.Content = new ScrollView { Content = mainLayout };

}

private async void OnExportClicked(object sender, EventArgs e)

{

try

{

var pdfBytes = await \_reportService.GenerateReportAsync(\_transactions, \_fromDate, \_toDate, \_currentTab);

var fileName = $"report\_{DateTime.Now:yyyyMMdd\_HHmmss}.pdf";

var filePath = Path.Combine(FileSystem.CacheDirectory, fileName);

await File.WriteAllBytesAsync(filePath, pdfBytes);

// Открыть системное окно "Поделиться"

await Share.RequestAsync(new ShareFileRequest

{

Title = "Поделиться PDF-отчётом",

File = new ShareFile(filePath)

});

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", $"Ошибка при создании PDF:\n{ex.Message}", "OK");

Console.WriteLine(ex);

}

}

private string RandomColor()

{

var random = new Random();

return $"#{random.Next(0x1000000):X6}";

}

private SKBitmap RenderChartToBitmap(Chart chart, int width, int height)

{

var bitmap = new SKBitmap(width, height);

using var canvas = new SKCanvas(bitmap);

canvas.Clear(SKColors.White);

try

{

chart.Draw(canvas, width, height);

}

catch (Exception ex)

{

Debug.WriteLine($"Ошибка при рисовании графика: {ex.Message}");

}

return bitmap;

}

}

Листинг файла ScanReceiptPage.xaml.cs

using SkiaSharp;

using System.Text.Json;

using System.Web;

using ZXing.Common;

using ZXing;

using FinanceApp.Core.Models;

using System.Text.RegularExpressions;

namespace FinanceApp.Maui;

public partial class ScanReceiptPage : ContentPage

{

private readonly TransactionsService \_transactionsService;

public ScanReceiptPage(TransactionsService transactionsService)

{

InitializeComponent();

\_transactionsService = transactionsService;

}

private async void OnPickFromGallery(object sender, EventArgs e)

{

try

{

var file = await FilePicker.PickAsync(new PickOptions

{

PickerTitle = "Выберите изображение с QR-кодом",

FileTypes = FilePickerFileType.Images

});

if (file != null)

{

using var stream = await file.OpenReadAsync();

await ProcessQrFromStream(stream);

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка", ex.Message, "ОК");

}

}

private async void OnScanWithCamera(object sender, EventArgs e)

{

if (MediaPicker.Default.IsCaptureSupported)

{

var photo = await MediaPicker.Default.CapturePhotoAsync();

if (photo != null)

{

using var stream = await photo.OpenReadAsync();

await ProcessQrFromStream(stream);

}

}

else

{

await DisplayAlert("Ошибка", "Съёмка не поддерживается на этом устройстве.", "ОК");

}

}

private async Task ProcessQrFromStream(Stream stream)

{

try

{

using var bitmap = SKBitmap.Decode(stream);

var source = new ZXing.SkiaSharp.SKBitmapLuminanceSource(bitmap);

var binarizer = new HybridBinarizer(source);

var binaryBitmap = new BinaryBitmap(binarizer);

var reader = new ZXing.MultiFormatReader();

var result = reader.decode(binaryBitmap);

if (result != null)

ParseQrData(result.Text);

else

ResultLabel.Text = "QR-код не найден на изображении.";

}

catch (Exception ex)

{

await DisplayAlert("Ошибка обработки", ex.Message, "OK");

}

}

private async void ParseQrData(string qrData)

{

try

{

var data = JsonSerializer.Deserialize<MyQrData>(qrData);

if (data != null && data.sum != 0)

{

var transaction = new Transaction

{

Description = null,

Amount = data.sum,

Date = DateTime.Parse(data.date),

UserId = 1,

CategoryId = 1,

IsIncome = false,

};

await \_transactionsService.AddTransactionAsync(transaction);

await Navigation.PopAsync();

return;

}

}

catch { }

if (qrData.Contains("t=") && qrData.Contains("&s="))

{

var parts = System.Web.HttpUtility.ParseQueryString(qrData);

string dateRaw = parts["t"];

string sum = parts["s"];

DateTime parsedDate = DateTime.Now;

try

{

if (!string.IsNullOrEmpty(dateRaw))

{

string dateStr = dateRaw.Replace("T", "");

parsedDate = DateTime.ParseExact(dateStr, "yyyyMMddHHmm", null);

}

}

catch

{

parsedDate = DateTime.Now;

}

await Navigation.PushAsync(new AddTransactionPage(\_transactionsService, Regex.Replace(sum, @"[^\d.,]", "").Replace('.', ','), parsedDate));

return;

}

ResultLabel.Text = $"Тип: Текст\nСодержание:\n{qrData}";

}

}

Листинг файла AnalyticsPage.xaml.cs

using FinanceApp.Core.Models;

using Microcharts;

using Microcharts.Maui;

using SkiaSharp;

using Syncfusion.Maui.Charts;

using System.Globalization;

namespace FinanceApp.Maui.Pages;

public partial class AnalyticsPage : ContentPage

{

private readonly TransactionsService \_transactionsService;

private string \_currentTab = "categories";

private DateTime \_fromDate = DateTime.Now.AddMonths(-1);

private DateTime \_toDate = DateTime.Now;

private bool \_barChartInitialized = false;

public static List<string> ComparisonModes => new() { "По месяцам", "По годам" };

public AnalyticsPage()

{

InitializeComponent();

\_transactionsService = App.Current.Handler.MauiContext.Services.GetService<TransactionsService>();

}

protected override void OnAppearing()

{

base.OnAppearing();

InitializePeriodPickers();

UpdatePeriodButtonsText();

LoadStatistics();

BottomMenu.CurrentPage = "analytics";

ComparisonModePicker.SelectedIndex = 0;

}

private void InitializePeriodPickers()

{

// Инициализация годов

var years = Enumerable.Range(DateTime.Now.Year - 5, 6).Reverse().ToList();

FromYearPicker.ItemsSource = years;

ToYearPicker.ItemsSource = years;

FromYearPicker.SelectedItem = \_fromDate.Year;

ToYearPicker.SelectedItem = \_toDate.Year;

// Инициализация месяцев

InitializeMonths(FromMonthContainer, \_fromDate.Month);

InitializeMonths(ToMonthContainer, \_toDate.Month);

}

private void InitializeMonths(FlexLayout container, int selectedMonth)

{

container.Children.Clear();

for (int i = 1; i <= 12; i++)

{

var monthName = CultureInfo.CurrentCulture.DateTimeFormat.GetAbbreviatedMonthName(i);

var button = new Button

{

Text = monthName,

WidthRequest = 70,

HeightRequest = 40,

Margin = 5,

FontSize = 14,

CornerRadius = 5,

BindingContext = i

};

if (i == selectedMonth)

{

button.BackgroundColor = Color.FromArgb("#1d8eea");

button.TextColor = Colors.White;

}

else

{

button.BackgroundColor = Colors.Transparent;

button.TextColor = Colors.Black;

}

button.Clicked += (sender, e) => OnMonthSelected(sender, e, container == FromMonthContainer);

container.Children.Add(button);

}

}

private void ShowFromPeriodPicker(object sender, EventArgs e)

{

FromPeriodPopup.IsVisible = true;

ToPeriodPopup.IsVisible = false;

}

private void ShowToPeriodPicker(object sender, EventArgs e)

{

ToPeriodPopup.IsVisible = true;

FromPeriodPopup.IsVisible = false;

}

private void HideFromPeriodPicker(object sender, EventArgs e)

{

FromPeriodPopup.IsVisible = false;

UpdateSelectedPeriod(true);

}

private void HideToPeriodPicker(object sender, EventArgs e)

{

ToPeriodPopup.IsVisible = false;

UpdateSelectedPeriod(false);

}

private void OnMonthSelected(object sender, EventArgs e, bool isFromPeriod)

{

if (sender is Button button && button.BindingContext is int month)

{

var yearPicker = isFromPeriod ? FromYearPicker : ToYearPicker;

if (yearPicker.SelectedItem is int year)

{

if (isFromPeriod)

{

\_fromDate = new DateTime(year, month, 1);

}

else

{

\_toDate = new DateTime(year, month, 1).AddMonths(1).AddDays(-1);

}

// Обновляем стили кнопок

var container = isFromPeriod ? FromMonthContainer : ToMonthContainer;

foreach (var child in container.Children)

{

if (child is Button monthButton)

{

bool isSelected = monthButton.BindingContext is int m && m == month;

monthButton.BackgroundColor = isSelected ? Color.FromArgb("#1d8eea") : Colors.Transparent;

monthButton.TextColor = isSelected ? Colors.White : Colors.Black;

}

}

UpdatePeriodButtonsText();

}

}

}

private void UpdateSelectedPeriod(bool isFromPeriod)

{

var yearPicker = isFromPeriod ? FromYearPicker : ToYearPicker;

if (yearPicker.SelectedItem is int year)

{

if (isFromPeriod)

{

\_fromDate = new DateTime(year, \_fromDate.Month, 1);

}

else

{

\_toDate = new DateTime(year, \_toDate.Month, 1).AddMonths(1).AddDays(-1);

}

UpdatePeriodButtonsText();

LoadStatistics();

}

}

private void UpdatePeriodButtonsText()

{

FromPeriodButton.Text = \_fromDate.ToString("MMM yyyy", CultureInfo.CurrentCulture);

ToPeriodButton.Text = \_toDate.ToString("MMM yyyy", CultureInfo.CurrentCulture);

PeriodLabel.Text = $"Период: {\_fromDate:MMMM yyyy} — {\_toDate:MMMM yyyy}";

}

private void OnTabClicked(object sender, EventArgs e)

{

\_currentTab = sender == CategoriesTabButton ? "categories" : "comparison";

CategoriesContent.IsVisible = \_currentTab == "categories";

ComparisonContent.IsVisible = \_currentTab == "comparison";

CategoriesTabButton.BackgroundColor = \_currentTab == "categories" ? Color.FromArgb("#1d8eea") : Color.FromArgb("#F0F0F0");

ComparisonTabButton.BackgroundColor = \_currentTab == "comparison" ? Color.FromArgb("#1d8eea") : Color.FromArgb("#F0F0F0");

CategoriesTabButton.TextColor = \_currentTab == "categories" ? Color.FromArgb("#fff") : Color.FromArgb("#000");

ComparisonTabButton.TextColor = \_currentTab == "comparison" ? Color.FromArgb("#fff") : Color.FromArgb("#000");

if (\_currentTab == "comparison")

{

if (!\_barChartInitialized)

{

BarChartView.SizeChanged += OnBarChartViewSizeChanged;

}

else

{

LoadComparisonChart();

}

}

}

private void OnBarChartViewSizeChanged(object sender, EventArgs e)

{

if (BarChartView.Width > 0 && BarChartView.Height > 0)

{

BarChartView.SizeChanged -= OnBarChartViewSizeChanged;

\_barChartInitialized = true;

LoadComparisonChart();

}

}

private async void LoadStatistics()

{

PeriodLabel.Text = $"Период: {\_fromDate:MMMM yyyy} — {\_toDate:MMMM yyyy}";

var transactions = await \_transactionsService.GetTransactionsCurrentUserAsync();

var expenses = transactions

.Where(t => !t.Category.IsIncome && t.Date >= \_fromDate && t.Date <= \_toDate)

.ToList();

var categoryGroups = expenses

.GroupBy(t => t.Category)

.Select(g => new CategoryStatistic

{

Category = g.Key,

Amount = g.Sum(t => t.Amount)

})

.OrderByDescending(g => g.Amount)

.ToList();

var totalAmount = categoryGroups.Sum(g => g.Amount);

foreach (var g in categoryGroups)

g.Percentage = totalAmount > 0 ? Math.Round((double)(g.Amount / totalAmount) \* 100, 1) : 0;

CategoriesList.ItemsSource = categoryGroups;

UpdatePieChart(categoryGroups);

}

private void UpdatePieChart(List<CategoryStatistic> categories)

{

PieChartContainer.IsVisible = categories.Count > 0;

if (!PieChartContainer.IsVisible)

{

PieChartContainer.Content = null;

return;

}

var colors = new[]

{

SKColor.Parse("#FF6384"), SKColor.Parse("#36A2EB"), SKColor.Parse("#FFCE56"),

SKColor.Parse("#4BC0C0"), SKColor.Parse("#9966FF"), SKColor.Parse("#FF9F40")

};

var rnd = new Random();

var entries = categories.Select((cat, i) => new ChartEntry((float)cat.Amount)

{

Label = cat.Category.Name,

ValueLabel = $"{cat.Percentage}%",

Color = i < colors.Length ? colors[i] : SKColor.Parse($"#{rnd.Next(0x1000000):X6}"),

TextColor = SKColors.Black

}).ToList();

var chartView = new ChartView

{

Chart = new PieChart

{

Entries = entries,

BackgroundColor = SKColors.Transparent,

LabelTextSize = 24,

LabelMode = LabelMode.RightOnly,

HoleRadius = 0.4f

},

HeightRequest = 200

};

PieChartContainer.Content = chartView;

}

private void OnComparisonModeChanged(object sender, EventArgs e)

{

if (\_currentTab == "comparison")

LoadComparisonChart();

}

private async void LoadComparisonChart()

{

var transactions = await \_transactionsService.GetTransactionsCurrentUserAsync();

var groupByYear = ComparisonModePicker.SelectedIndex == 1;

var grouped = groupByYear

? transactions.GroupBy(t => t.Date.Year.ToString())

: transactions.GroupBy(t => $"{t.Date.Month:D2}.{t.Date.Year}");

var data = grouped

.OrderBy(g => groupByYear ? g.Key : $"{g.First().Date:yyyyMM}")

.Select(g => new ChartData

{

Label = g.Key,

Income = (float)g.Where(t => t.Category.IsIncome).Sum(t => t.Amount),

Expense = (float)g.Where(t => !t.Category.IsIncome).Sum(t => t.Amount)

})

.ToList();

BarChartView.Series.Clear();

var incomeSeries = new ColumnSeries

{

ItemsSource = data,

XBindingPath = "Label",

YBindingPath = "Income",

Label = "Доходы",

Fill = Color.FromArgb("#4CAF50")

};

var expenseSeries = new ColumnSeries

{

ItemsSource = data,

XBindingPath = "Label",

YBindingPath = "Expense",

Label = "Расходы",

Fill = Color.FromArgb("#F44336")

};

BarChartView.Series.Add(incomeSeries);

BarChartView.Series.Add(expenseSeries);

}

}

public class ChartData

{

public string Label { get; set; }

public float Income { get; set; }

public float Expense { get; set; }

}

public class CategoryStatistic

{

public Category Category { get; set; }

public decimal Amount { get; set; }

public double Percentage { get; set; }

}

Листинг файла AddTransactionPage.xaml.cs

namespace FinanceApp.Maui;

using FinanceApp.Core.Models;

using FinanceApp.Maui.Services;

public partial class AddTransactionPage : ContentPage

{

private readonly TransactionsService \_transactionsService;

private readonly AchievementService \_achievementService;

bool isIncome = true;

DateTime? transactionDate = null;

Transaction \_transaction;

private readonly CategoryService \_categoryService;

private Category \_selectedCategory;

public AddTransactionPage(TransactionsService transactionsService)

{

InitializeComponent();

\_transactionsService = transactionsService;

\_categoryService = App.Current.Handler.MauiContext.Services.GetService<CategoryService>();

\_achievementService = App.Current.Handler.MauiContext.Services.GetService<AchievementService>();

}

public AddTransactionPage(TransactionsService transactionsService, string amount, DateTime date)

{

InitializeComponent();

\_transactionsService = transactionsService;

AmountEntry.Text = amount;

transactionDate = date;

\_categoryService = App.Current.Handler.MauiContext.Services.GetService<CategoryService>();

\_achievementService = App.Current.Handler.MauiContext.Services.GetService<AchievementService>();

ChangeTab(SpendTab, IncomeTab, false);

IncomeTab.IsEnabled = false;

SpendTab.IsEnabled = false;

}

public AddTransactionPage(TransactionsService transactionsService, Transaction transaction)

{

InitializeComponent();

\_transaction = transaction;

\_transactionsService = transactionsService;

\_categoryService = App.Current.Handler.MauiContext.Services.GetService<CategoryService>();

\_achievementService = App.Current.Handler.MauiContext.Services.GetService<AchievementService>();

AmountEntry.Text = \_transaction.Amount.ToString();

DescriptionEntry.Text = \_transaction.Description;

if(\_transaction.RepeatType != 0)

{

RepeatCheckBox.IsChecked = true;

}

if(\_transaction.IsIncome == true)

{

ChangeTab(IncomeTab, SpendTab, true);

}

else

{

ChangeTab(SpendTab, IncomeTab, false);

}

UpdateCategoryUI(transaction.Category.Icon, transaction.Category.Name);

}

private async void OnSaveClicked(object sender, EventArgs e)

{

var repeatType = RepeatType.None;

if (RepeatCheckBox.IsChecked)

{

repeatType = RepeatIntervalPicker.SelectedIndex switch

{

0 => RepeatType.Daily,

1 => RepeatType.Weekly,

2 => RepeatType.Monthly,

\_ => RepeatType.None

};

}

if (\_transaction != null)

{

\_transaction.RepeatType = repeatType;

\_transaction.Description = DescriptionEntry.Text;

\_transaction.Amount = decimal.Parse(AmountEntry.Text);

\_transaction.IsIncome = isIncome;

\_transaction.CategoryId = \_selectedCategory.Id;

await \_transactionsService.UpdateTransactionAsync(\_transaction, \_transaction.Id);

}

else

{

\_transaction = new Transaction

{

Description = DescriptionEntry.Text,

Amount = decimal.Parse(AmountEntry.Text),

Date = transactionDate ?? DateTime.UtcNow,

UserId = 0,

CategoryId = \_selectedCategory.Id,

IsIncome = isIncome,

RepeatType = repeatType

};

await \_transactionsService.AddTransactionAsync(\_transaction);

}

var updatedAchievements = await \_achievementService.CheckAndUpdateAchievementsAsync();

await Navigation.PopAsync();

}

private void OnRepeatCheckedChanged(object sender, CheckedChangedEventArgs e)

{

RepeatIntervalPicker.IsVisible = e.Value;

if (e.Value && RepeatIntervalPicker.Items.Count > 0)

{

RepeatIntervalPicker.SelectedIndex = 0;

}

else

{

RepeatIntervalPicker.SelectedIndex = -1;

}

}

private async void OnIncomeTabClicked(object sender, EventArgs e)

{

ChangeTab(IncomeTab, SpendTab, true);

}

private async void OnSpendTabClicked(object sender, EventArgs e)

{

ChangeTab(SpendTab, IncomeTab, false);

}

private async void ChangeTab(Button buttonActive, Button buttonInactive, bool isIncomeValue)

{

isIncome = isIncomeValue;

buttonActive.BackgroundColor = Color.FromArgb("#1d8eea");

buttonInactive.BackgroundColor = Color.FromArgb("#F0F0F0");

buttonActive.TextColor = Color.FromArgb("#fff");

buttonInactive.TextColor = Color.FromArgb("#000");

CategoryIcon.Source = null;

CategoryLabel.Text = "Выберите категорию";

}

private async void OnSelectCategoryClicked(object sender, EventArgs e)

{

var categories = await \_categoryService.GetCateroryAsync();

var token = Preferences.Get("jwt\_token", null);

var userId = JwtHelper.ParseTokenId(token);

var filteredCategories = categories.Where(c => (c.IsIncome == isIncome && c.IsDefault)

|| (c.IsIncome == isIncome && c.UserId == int.Parse(userId))).ToList();

var modalPage = new SelectCategoryPage(filteredCategories);

await Navigation.PushModalAsync(new NavigationPage(modalPage));

var result = await modalPage.Result;

if (result != null)

{

\_selectedCategory = result;

UpdateCategoryUI(\_selectedCategory.Icon, \_selectedCategory.Name);

}

}

private void UpdateCategoryUI(string IconCategory, string nameCategory)

{

CategoryIcon.Source = IconCategory;

CategoryLabel.Text = nameCategory;

}

}

Листинг файла BottomMenu.xaml.cs

using FinanceApp.Maui.Pages;

using System.Windows.Input;

namespace FinanceApp.Maui;

public partial class BottomMenu : Frame

{

private readonly TransactionsService \_transactionsService;

public static readonly BindableProperty CurrentPageProperty =

BindableProperty.Create(nameof(CurrentPage), typeof(string), typeof(BottomMenu),

default(string), propertyChanged: OnCurrentPageChanged);

public static readonly BindableProperty MenuCommandProperty =

BindableProperty.Create(nameof(MenuCommand), typeof(ICommand), typeof(BottomMenu), null);

public BottomMenu()

{

InitializeComponent();

\_transactionsService = App.Current.Handler.MauiContext.Services.GetService<TransactionsService>();

}

public string CurrentPage

{

get => (string)GetValue(CurrentPageProperty);

set => SetValue(CurrentPageProperty, value);

}

public ICommand MenuCommand

{

get => (ICommand)GetValue(MenuCommandProperty);

set => SetValue(MenuCommandProperty, value);

}

private static void OnCurrentPageChanged(BindableObject bindable, object oldValue, object newValue)

{

var menu = (BottomMenu)bindable;

menu.UpdateActiveTab();

}

private void UpdateActiveTab()

{

UpdateTab("records", RecordsIcon, RecordsLabel);

UpdateTab("reports", ReportsIcon, ReportsLabel);

UpdateTab("analytics", AnalyticsIcon, AnalyticsLabel);

UpdateTab("profile", ProfileIcon, ProfileLabel);

}

private void UpdateTab(string pageName, Image icon, Label label)

{

bool isActive = CurrentPage == pageName;

label.TextColor = isActive ? Color.FromArgb("#1d8eea") : Color.FromArgb("#9a1e5a");

icon.Source = isActive ? $"menu\_{pageName}\_active.png" : $"menu\_{pageName}\_inactive.png";

}

private async void OnMenuClicked(object sender, EventArgs e)

{

var parentPage = this.FindParent<Page>();

if (parentPage == null) return;

var actionSheet = new AddTransactionActionSheet();

await Navigation.PushModalAsync(actionSheet);

string result = await actionSheet.ActionResult.Task;

switch (result)

{

case "manual":

await parentPage.Navigation.PushAsync(new AddTransactionPage(\_transactionsService));

break;

case "scan":

var page = Handler.MauiContext.Services.GetRequiredService<ScanReceiptPage>();

await parentPage.Navigation.PushAsync(page);

break;

}

}

private async void OnTabClicked(object sender, EventArgs e)

{

string pageName = null;

if (e is TappedEventArgs tappedEventArgs)

{

pageName = tappedEventArgs.Parameter as string;

}

else if (sender is ImageButton button)

{

pageName = button.CommandParameter as string;

}

if (!string.IsNullOrEmpty(pageName))

{

CurrentPage = pageName;

// Если команда установлена, выполняем её

if (MenuCommand != null && MenuCommand.CanExecute(pageName))

{

MenuCommand.Execute(pageName);

}

else // Иначе выполняем навигацию напрямую

{

var parentPage = this.FindParent<Page>();

if (parentPage == null) return;

Page targetPage = pageName switch

{

"records" => new MainPage(),

"reports" => new ReportsPage(),

"analytics" => new AnalyticsPage(),

"profile" => new ProfilePage(),

\_ => null

};

if (targetPage != null)

{

await parentPage.Navigation.PushAsync(targetPage);

Navigation.RemovePage(Navigation.NavigationStack[Navigation.NavigationStack.Count - 2]);

}

}

}

}

}

public static class ViewExtensions

{

public static T FindParent<T>(this Element element) where T : Element

{

while (element != null)

{

if (element is T parent)

return parent;

element = element.Parent;

}

return null;

}

}

Листинг файла AddCategoryPage.xaml.cs

using FinanceApp.Core.Models;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Globalization;

using System.Runtime.CompilerServices;

namespace FinanceApp.Maui.ModalWindow;

public partial class AddCategoryPage : ContentPage

{

private readonly CategoryService \_categoryService;

public AddCategoryPage()

{

InitializeComponent();

BindingContext = this;

\_categoryService = App.Current.Handler.MauiContext.Services.GetService<CategoryService>();

}

public ObservableCollection<CategoryIcon> SpendIconList { get; set; } = new()

{

new CategoryIcon { Icon = "category\_beauty.png" },

new CategoryIcon { Icon = "category\_children.png" },

new CategoryIcon { Icon = "category\_cinema.png" },

new CategoryIcon { Icon = "category\_coffee.png" },

new CategoryIcon { Icon = "category\_concert.png" },

new CategoryIcon { Icon = "category\_delivery.png" },

new CategoryIcon { Icon = "category\_flight.png" },

new CategoryIcon { Icon = "category\_repair.png" },

new CategoryIcon { Icon = "category\_sport.png" },

new CategoryIcon { Icon = "category\_entertainments.png" }

};

protected override async void OnAppearing()

{

base.OnAppearing();

await PopupContainer.FadeTo(1, 200);

await PopupContainer.ScaleTo(1, 200, Easing.SinOut);

TypePicker.SelectedIndex = 0;

}

private async void OnSaveClicked(object sender, EventArgs e)

{

var type = TypePicker.SelectedItem?.ToString();

bool isIncome = type == "Доход" ? true : false;

var icon = (CategoryIcon)IconsCollectionView.SelectedItem;

var name = CategoryNameEntry.Text;

if (string.IsNullOrWhiteSpace(name) || string.IsNullOrEmpty(type) || icon == null)

{

await DisplayAlert("Ошибка", "Пожалуйста, заполните все поля", "Ок");

return;

}

var token = Preferences.Get("jwt\_token", null);

var userId = JwtHelper.ParseTokenId(token);

var category = new Category

{

IsIncome = isIncome,

Name = name,

Icon = icon.Icon,

IsDefault = false,

UserId = int.Parse(userId)

};

// Отправка на сервер

await \_categoryService.AddCategoryAsync(category);

// Закрыть popup

await Navigation.PopModalAsync();

}

private async void OnOverlayTapped(object sender, EventArgs e)

{

await Navigation.PopModalAsync();

}

private void OnIconSelected(object sender, SelectionChangedEventArgs e)

{

// Сбрасываем выделение у всех элементов

foreach (var item in SpendIconList)

{

item.IsSelected = false;

}

// Устанавливаем выделение для выбранного элемента

if (IconsCollectionView.SelectedItem is CategoryIcon selectedIcon)

{

selectedIcon.IsSelected = true;

}

}

}

public class CategoryIcon : INotifyPropertyChanged

{

private string \_icon;

private bool \_isSelected;

public string Icon

{

get => \_icon;

set

{

\_icon = value;

OnPropertyChanged();

}

}

public bool IsSelected

{

get => \_isSelected;

set

{

if (\_isSelected != value)

{

\_isSelected = value;

OnPropertyChanged();

}

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

public class BoolToColorConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

return (bool)value ? Color.FromArgb("#4C6EF5") : Colors.Transparent;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

Листинг файла AchievementController.cs

using FinanceApp.Core.Models;

using FinanceApp.WebApi.Data;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

using Microsoft.EntityFrameworkCore;

namespace FinanceApp.WebApi.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class AchievementController : ControllerBase

{

private readonly AppDbContext \_db;

public AchievementController(AppDbContext db)

{

\_db = db;

}

[HttpGet("userAchievements")]

public async Task<IActionResult> GetUserAchievements()

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value);

var userAchievements = await \_db.UserAchievements

.Include(ua => ua.Achievement)

.Where(ua => ua.UserId == userId)

.ToListAsync();

return Ok(userAchievements);

}

[HttpGet("checkAchievements")]

public async Task<IActionResult> CheckAndUpdateAchievements()

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value);

// Проверяем все возможные достижения

await CheckFirstTransactionAchievement(userId);

await CheckThreeCategoriesAchievement(userId);

await CheckSevenDaysAchievement(userId);

await CheckTwoMonthsProfitAchievement(userId);

// Возвращаем обновленный список достижений пользователя

var updatedAchievements = await \_db.UserAchievements

.Include(ua => ua.Achievement)

.Where(ua => ua.UserId == userId)

.ToListAsync();

return Ok(updatedAchievements);

}

private async Task CheckFirstTransactionAchievement(int userId)

{

// Проверяем, есть ли у пользователя хотя бы одна транзакция

var hasTransactions = await \_db.Transactions.AnyAsync(t => t.UserId == userId);

var hasAchievement = await \_db.UserAchievements

.AnyAsync(ua => ua.UserId == userId && ua.AchievementId == 1);

if (hasTransactions && !hasAchievement)

{

await AddAchievementToUser(userId, 1);

}

}

private async Task CheckThreeCategoriesAchievement(int userId)

{

// Проверяем, использовал ли пользователь 3 разные категории расходов

var expenseCategoriesCount = await \_db.Transactions

.Where(t => t.UserId == userId && t.IsIncome == false)

.Select(t => t.CategoryId)

.Distinct()

.CountAsync();

var hasAchievement = await \_db.UserAchievements

.AnyAsync(ua => ua.UserId == userId && ua.AchievementId == 2);

if (expenseCategoriesCount >= 3 && !hasAchievement)

{

await AddAchievementToUser(userId, 2);

}

}

private async Task CheckSevenDaysAchievement(int userId)

{

// Проверяем, вносил ли пользователь транзакции 7 дней подряд

var transactionDates = await \_db.Transactions

.Where(t => t.UserId == userId)

.Select(t => t.Date.Date)

.Distinct()

.OrderBy(d => d)

.ToListAsync();

int consecutiveDays = 0;

DateTime? previousDate = null;

foreach (var date in transactionDates)

{

if (previousDate == null || date == previousDate.Value.AddDays(1))

{

consecutiveDays++;

if (consecutiveDays >= 7) break;

}

else

{

consecutiveDays = 1;

}

previousDate = date;

}

var hasAchievement = await \_db.UserAchievements

.AnyAsync(ua => ua.UserId == userId && ua.AchievementId == 3);

if (consecutiveDays >= 7 && !hasAchievement)

{

await AddAchievementToUser(userId, 3);

}

}

private async Task CheckTwoMonthsProfitAchievement(int userId)

{

// Получаем текущий и предыдущий месяц

var currentDate = DateTime.UtcNow;

var currentMonthStart = new DateTime(currentDate.Year, currentDate.Month, 1, 0, 0, 0, DateTimeKind.Utc);

var previousMonthStart = currentMonthStart.AddMonths(-1);

// Проверяем доходы и расходы за два месяца

var currentMonthTransactions = await \_db.Transactions

.Where(t => t.UserId == userId &&

t.Date >= currentMonthStart &&

t.Date < currentMonthStart.AddMonths(1))

.ToListAsync();

var previousMonthTransactions = await \_db.Transactions

.Where(t => t.UserId == userId &&

t.Date >= previousMonthStart &&

t.Date < currentMonthStart)

.ToListAsync();

var currentMonthIncome = currentMonthTransactions.Where(t => t.Amount > 0).Sum(t => t.Amount);

var currentMonthExpenses = Math.Abs(currentMonthTransactions.Where(t => t.Amount < 0).Sum(t => t.Amount));

var previousMonthIncome = previousMonthTransactions.Where(t => t.Amount > 0).Sum(t => t.Amount);

var previousMonthExpenses = Math.Abs(previousMonthTransactions.Where(t => t.Amount < 0).Sum(t => t.Amount));

var currentMonthProfit = currentMonthIncome > currentMonthExpenses;

var previousMonthProfit = previousMonthIncome > previousMonthExpenses;

var hasAchievement = await \_db.UserAchievements

.AnyAsync(ua => ua.UserId == userId && ua.AchievementId == 4);

if (currentMonthProfit && previousMonthProfit && !hasAchievement)

{

await AddAchievementToUser(userId, 4);

}

}

private async Task AddAchievementToUser(int userId, int achievementId)

{

var userAchievement = new UserAchievement

{

UserId = userId,

AchievementId = achievementId,

DateUnlocked = DateTime.UtcNow

};

\_db.UserAchievements.Add(userAchievement);

await \_db.SaveChangesAsync();

}

}

}

Листинг файла AuthController.cs

using FinanceApp.Core.Models;

using FinanceApp.WebApi.Data;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Security.Cryptography;

using System.Text;

namespace FinanceApp.WebApi.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class AuthController : ControllerBase

{

private readonly AppDbContext \_context;

private readonly PasswordHasher<User> \_hasher = new PasswordHasher<User>();

public AuthController(AppDbContext context) => \_context = context;

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] User user)

{

if (await \_context.Users.AnyAsync(u => u.Email == user.Email))

return BadRequest("User already exists");

user.PasswordHash = \_hasher.HashPassword(user, user.PasswordHash);

\_context.Users.Add(user);

await \_context.SaveChangesAsync();

return Ok("Registered");

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginRequest request)

{

var existing = await \_context.Users.FirstOrDefaultAsync(x => x.Email == request.Email);

if (existing == null) return Unauthorized();

var result = \_hasher.VerifyHashedPassword(existing, existing.PasswordHash, request.Password);

if (result != PasswordVerificationResult.Success) return Unauthorized();

var token = GenerateJwtToken(existing);

var refreshToken = GenerateRefreshToken();

existing.RefreshToken = refreshToken;

existing.RefreshTokenExpiryTime = DateTime.UtcNow.AddDays(14);

await \_context.SaveChangesAsync();

return Ok(new { Token = token, RefreshToken = refreshToken });

}

[HttpPost("refresh")]

public async Task<IActionResult> Refresh([FromBody] TokenRefreshReq request)

{

try

{

Console.WriteLine($"Received refresh request. Token: {request.Token?.Substring(0, 10)}..., RefreshToken: {request.RefreshToken}");

var principal = GetPrincipalFromExpiredToken(request.Token);

var email = principal.Identity.Name;

var user = await \_context.Users.FirstOrDefaultAsync(u => u.Email == email);

Console.WriteLine($"User found: {user != null}, DB RefreshToken: {user?.RefreshToken}, Request RefreshToken: {request.RefreshToken}, Expiry: {user?.RefreshTokenExpiryTime}");

if (user == null || user.RefreshToken != request.RefreshToken || user.RefreshTokenExpiryTime <= DateTime.UtcNow)

return Unauthorized();

var newToken = GenerateJwtToken(user);

var newRefreshToken = GenerateRefreshToken();

user.RefreshToken = newRefreshToken;

user.RefreshTokenExpiryTime = DateTime.UtcNow.AddDays(14);

await \_context.SaveChangesAsync();

return Ok(new { Token = newToken, RefreshToken = newRefreshToken });

}

catch (Exception ex)

{

Console.WriteLine($"Controller exception: {ex}");

throw;

}

}

private ClaimsPrincipal GetPrincipalFromExpiredToken(string token)

{

var tokenValidationParameters = new TokenValidationParameters

{

ValidateAudience = false,

ValidateIssuer = false,

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes("SuperSecretKey123!SuperSecretKey123!")),

ValidateLifetime = false

};

var tokenHandler = new JwtSecurityTokenHandler();

var principal = tokenHandler.ValidateToken(token, tokenValidationParameters, out var securityToken);

if (securityToken is not JwtSecurityToken jwtSecurityToken ||

!jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256, StringComparison.InvariantCultureIgnoreCase))

throw new SecurityTokenException("Invalid token");

return principal;

}

private string GenerateJwtToken(User user)

{

try

{

if (string.IsNullOrEmpty(user.Email))

throw new Exception("Email is NULL или пустой!");

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes("SuperSecretKey123!SuperSecretKey123!");

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new[] {

new Claim(ClaimTypes.Name, user.UserName),

new Claim(ClaimTypes.Email, user.Email),

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

catch (Exception ex)

{

Console.WriteLine($"TOKEN ERROR: {ex}");

throw;

}

}

private string GenerateRefreshToken() =>

Convert.ToBase64String(RandomNumberGenerator.GetBytes(64));

}

public class RefreshRequest

{

public string RefreshToken { get; set; }

}

}

Листинг файла CategoriesController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using FinanceApp.Core.Models;

using FinanceApp.WebApi.Data;

using System.Security.Claims;

namespace FinanceApp.WebApi.Controllers;

[ApiController]

[Route("api/[controller]")]

public class CategoriesController : ControllerBase

{

private readonly AppDbContext \_db;

public CategoriesController(AppDbContext db)

{

\_db = db;

}

[HttpGet]

public async Task<IActionResult> Get()

{

var category = await \_db.Categories.ToListAsync();

return Ok(category);

}

[HttpGet("{id}")]

public async Task<IActionResult> Get(int id)

{

var category = await \_db.Categories.FindAsync(id);

return category == null ? NotFound() : Ok(category);

}

[HttpPost]

public async Task<IActionResult> Post([FromBody] Category category)

{

var maxId = await \_db.Categories.MaxAsync(c => (int?)c.Id) ?? 0;

category.Id = maxId + 1;

await \_db.Categories.AddAsync(category);

await \_db.SaveChangesAsync();

return CreatedAtAction(nameof(Get), new { id = category.Id }, category);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var category = await \_db.Categories.FindAsync(id);

if (category == null)

return NotFound();

\_db.Categories.Remove(category);

await \_db.SaveChangesAsync();

return NoContent();

}

[HttpGet("currentUser")]

public async Task<IActionResult> GetByCurrentUser()

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value);

var categories = await \_db.Categories.Where(t => t.UserId == userId).ToListAsync();

return Ok(categories);

}

}

Листинг файла TransactionsController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using FinanceApp.Core.Models;

using FinanceApp.WebApi.Data;

using System.Security.Claims;

namespace FinanceApp.WebApi.Controllers;

[ApiController]

[Route("api/[controller]")]

public class TransactionsController : ControllerBase

{

private readonly AppDbContext \_db;

public TransactionsController(AppDbContext db)

{

\_db = db;

}

[HttpGet]

public async Task<IActionResult> Get()

{

var transactions = await \_db.Transactions.ToListAsync();

return Ok(transactions);

}

[HttpGet("currentUser")]

public async Task<IActionResult> GetByCurrentUser()

{

var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value);

var transactions = await \_db.Transactions.Where(t => t.UserId == userId).Include(t => t.Category).OrderByDescending(t => t.Date).ToListAsync();

return Ok(transactions);

}

[HttpGet("{id}")]

public async Task<IActionResult> Get(int id)

{

var transaction = await \_db.Transactions.FindAsync(id);

return transaction == null ? NotFound() : Ok(transaction);

}

[HttpPost]

public async Task<IActionResult> Post([FromBody] Transaction transaction)

{

var category = await \_db.Categories.FindAsync(transaction.CategoryId);

var user = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value);

transaction.Date = transaction.Date.ToUniversalTime();

if (category == null || user == null)

return BadRequest("Invalid CategoryId or UserId");

transaction.UserId = user;

await \_db.Transactions.AddAsync(transaction);

await \_db.SaveChangesAsync();

return CreatedAtAction(nameof(Get), new { id = transaction.Id }, transaction);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var transaction = await \_db.Transactions.FindAsync(id);

if (transaction == null)

return NotFound();

\_db.Transactions.Remove(transaction);

await \_db.SaveChangesAsync();

return NoContent();

}

[HttpPut("{id}")]

public async Task<IActionResult> Put(int id, [FromBody] Transaction inputTransaction)

{

if (id != inputTransaction.Id)

return BadRequest();

\_db.Entry(inputTransaction).State = EntityState.Modified;

await \_db.SaveChangesAsync();

return NoContent();

}

[HttpGet("by-date")]

public async Task<IActionResult> GetByDate([FromQuery] DateTime startDate, [FromQuery] DateTime endDate)

{

var transactions = await \_db.Transactions

.Where(t => t.Date >= startDate && t.Date <= endDate)

.ToListAsync();

return Ok(transactions);

}

[HttpGet("stats")]

public async Task<IActionResult> GetStats()

{

var stats = await \_db.Transactions

.GroupBy(t => t.IsIncome)

.Select(g => new {

Type = g.Key ? "Income" : "Expense",

Total = g.Sum(t => t.Amount)

})

.ToListAsync();

return Ok(stats);

}

[HttpGet("by-user/{userId}")]

public async Task<IActionResult> GetByUser(int userId)

{

var transactions = await \_db.Transactions

.Where(t => t.UserId == userId)

.Include(t => t.Category) // Подгружаем категории

.ToListAsync();

return Ok(transactions);

}

[HttpGet("by-category/{categoryId}")]

public async Task<IActionResult> GetByCategory(int categoryId)

{

var transactions = await \_db.Transactions

.Where(t => t.CategoryId == categoryId)

.Include(t => t.User) // Подгружаем пользователей

.ToListAsync();

return Ok(transactions);

}

}

Листинг файла UsersController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using FinanceApp.Core.Models;

using FinanceApp.WebApi.Data;

using System.IdentityModel.Tokens.Jwt;

using System.Text;

using Microsoft.IdentityModel.Tokens;

using System.Security.Claims;

using System.Security.Cryptography;

using Microsoft.AspNetCore.Identity;

namespace FinanceApp.WebApi.Controllers;

[ApiController]

[Route("api/[controller]")]

public class UsersController : ControllerBase

{

private readonly AppDbContext \_db;

public UsersController(AppDbContext db)

{

\_db = db;

}

[HttpGet]

public async Task<IActionResult> Get()

{

var user = await \_db.Users.ToListAsync();

return Ok(user);

}

[HttpGet("{id}")]

public async Task<IActionResult> Get(int id)

{

var user = await \_db.Users.FindAsync(id);

return user == null ? NotFound() : Ok(user);

}

[HttpPost]

public async Task<IActionResult> Post([FromBody] User user)

{

await \_db.Users.AddAsync(user);

await \_db.SaveChangesAsync();

return CreatedAtAction(nameof(Get), new { id = user.Id }, user);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var user = await \_db.Users.FindAsync(id);

if (user == null)

return NotFound();

\_db.Users.Remove(user);

await \_db.SaveChangesAsync();

return NoContent();

}

[HttpPut("{id}/change-password")]

public async Task<IActionResult> ChangePassword(int id, [FromBody] ChangePasswordDto dto)

{

var user = await \_db.Users.FindAsync(id);

if (user == null)

return NotFound("Пользователь не найден");

var hasher = new PasswordHasher<User>();

var verificationResult = hasher.VerifyHashedPassword(user, user.PasswordHash, dto.CurrentPassword);

if (verificationResult != PasswordVerificationResult.Success)

return BadRequest("Неверный текущий пароль");

user.PasswordHash = hasher.HashPassword(user, dto.NewPassword);

await \_db.SaveChangesAsync();

return Ok(new { Message = "Пароль успешно изменён" });

}

[HttpPut("{id}")]

public async Task<IActionResult> Put(int id, [FromBody] UpdateUserDto dto)

{

var user = await \_db.Users.FindAsync(id);

if (user == null)

return NotFound();

var oldUserName = user.UserName;

user.UserName = dto.UserName;

user.Email = dto.Email;

await \_db.SaveChangesAsync();

// Всегда генерируем новые токены при обновлении данных

var newToken = GenerateJwtToken(user);

var newRefreshToken = GenerateRefreshToken();

user.RefreshToken = newRefreshToken;

user.RefreshTokenExpiryTime = DateTime.UtcNow.AddDays(14);

await \_db.SaveChangesAsync();

return Ok(new

{

Message = "User updated successfully",

Token = newToken,

RefreshToken = newRefreshToken

});

}

private string GenerateJwtToken(User user)

{

try

{

if (string.IsNullOrEmpty(user.Email))

throw new Exception("Email is NULL или пустой!");

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes("SuperSecretKey123!SuperSecretKey123!");

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new[] {

new Claim(ClaimTypes.Name, user.UserName),

new Claim(ClaimTypes.Email, user.Email),

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

catch (Exception ex)

{

Console.WriteLine($"TOKEN ERROR: {ex}");

throw;

}

}

private string GenerateRefreshToken() =>

Convert.ToBase64String(RandomNumberGenerator.GetBytes(64));

}

Листинг файла RecurringTransactionsBackgroundService.cs

namespace FinanceApp.WebApi

{

public class RecurringTransactionsBackgroundService : BackgroundService

{

private readonly IServiceProvider \_services;

private readonly ILogger<RecurringTransactionsBackgroundService> \_logger;

public RecurringTransactionsBackgroundService(

IServiceProvider services,

ILogger<RecurringTransactionsBackgroundService> logger)

{

\_services = services;

\_logger = logger;

}

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

while (!stoppingToken.IsCancellationRequested)

{

try

{

using (var scope = \_services.CreateScope())

{

var service = scope.ServiceProvider.GetRequiredService<RecurringTransactionService>();

await service.GenerateRecurringTransactionsAsync();

}

}

catch (Exception ex)

{

\_logger.LogError(ex, "Ошибка в фоновой задаче повторяющихся транзакций");

}

await Task.Delay(TimeSpan.FromHours(2), stoppingToken);

}

}

}

}

Листинг файла ReportsController.cs

using FinanceApp.Core.Models;

using Microsoft.AspNetCore.Mvc;

using QuestPDF.Fluent;

using QuestPDF.Helpers;

using QuestPDF.Infrastructure;

using SkiaSharp;

namespace FinanceApp.WebApi.Controllers

{

[ApiController]

[Route("api/reports")]

public class ReportsController : ControllerBase

{

public class ReportRequest

{

public List<Transaction> Transactions { get; set; }

public DateTime FromDate { get; set; }

public DateTime ToDate { get; set; }

public string Tab { get; set; }

}

[HttpPost("export")]

public IActionResult ExportReport([FromBody] ReportRequest request)

{

if (request.Transactions == null || request.Transactions.Count == 0)

return BadRequest("Список транзакций пуст.");

var pdfBytes = CreatePdfReport(request.Transactions, request.FromDate, request.ToDate, request.Tab);

return File(pdfBytes, "application/pdf", $"report\_{DateTime.Now:yyyyMMdd\_HHmm}.pdf");

}

private byte[] CreatePdfReport(List<Transaction> transactions, DateTime fromDate, DateTime toDate, string tab)

{

using var ms = new MemoryStream();

QuestPDF.Settings.License = QuestPDF.Infrastructure.LicenseType.Community;

QuestPDF.Fluent.Document.Create(document =>

{

document.Page(page =>

{

page.Size(QuestPDF.Helpers.PageSizes.A4);

page.Margin(40);

page.DefaultTextStyle(style => style.FontSize(14).FontFamily("Arial"));

page.Header()

.AlignCenter()

.Text($"Отчет от {DateTime.Now:dd.MM.yyyy}")

.Style(TextStyle.Default.FontSize(20).Bold());

page.Content()

.Column(column =>

{

column.Item().PaddingBottom(25).PaddingTop(25).Text($"Период: {fromDate:dd.MM.yyyy} - {toDate:dd.MM.yyyy}");

switch (tab)

{

case "General":

{

var filtered = transactions

.Where(t => t.Date >= fromDate && t.Date <= toDate)

.ToList();

var income = filtered.Where(t => t.Category.IsIncome).Sum(t => t.Amount);

var expense = filtered.Where(t => !t.Category.IsIncome).Sum(t => t.Amount);

var total = income + expense;

var incomePercent = total == 0 ? 0 : income / total \* 100;

var expensePercent = total == 0 ? 0 : expense / total \* 100;

var topIncome = filtered

.Where(t => t.Category.IsIncome)

.OrderByDescending(t => t.Amount)

.FirstOrDefault();

var topExpense = filtered

.Where(t => !t.Category.IsIncome)

.OrderByDescending(t => t.Amount)

.FirstOrDefault();

var mostIncomeDay = filtered

.Where(t => t.Category.IsIncome)

.GroupBy(t => t.Date.Date)

.Select(g => new { Date = g.Key, Total = g.Sum(t => t.Amount) })

.OrderByDescending(x => x.Total)

.FirstOrDefault();

var mostExpenseDay = filtered

.Where(t => !t.Category.IsIncome)

.GroupBy(t => t.Date.Date)

.Select(g => new { Date = g.Key, Total = g.Sum(t => t.Amount) })

.OrderByDescending(x => x.Total)

.FirstOrDefault();

column.Item().PaddingBottom(10).Text("Общая статистика").Style(TextStyle.Default.Size(16).Bold());

column.Item().Table(table =>

{

table.ColumnsDefinition(columns =>

{

columns.RelativeColumn(); // Название

columns.ConstantColumn(150); // Значение

});

void AddRow(string label, string value)

{

table.Cell().Element(CellStyle).Text(label).FontSize(14);

table.Cell().Element(CellStyle).AlignRight().Text(value).FontSize(14);

}

static IContainer CellStyle(IContainer container)

=> container.PaddingVertical(5).BorderBottom(1).BorderColor(Colors.Grey.Lighten2);

AddRow("Доходы", $"{income:F2}");

AddRow("Расходы", $"{expense:F2}");

AddRow("Доходы (%)", $"{incomePercent:F2}%");

AddRow("Расходы (%)", $"{expensePercent:F2}%");

});

column.Item().PaddingTop(20).PaddingBottom(10).Text("Ключевые показатели").Style(TextStyle.Default.Size(16).Bold());

column.Item().Table(table =>

{

table.ColumnsDefinition(columns =>

{

columns.RelativeColumn();

columns.ConstantColumn(200);

});

void AddKeyMetric(string label, string value)

{

table.Cell().Element(container => CellStyle(container).AlignMiddle()).Text(label).LineHeight(1.5f);

table.Cell().Element(container => CellStyle(container).AlignMiddle()).AlignRight().Text(value).LineHeight(1.5f);

}

static IContainer CellStyle(IContainer container)

=> container.PaddingVertical(5).BorderBottom(1).BorderColor(Colors.Grey.Lighten3);

if (topIncome != null)

AddKeyMetric("Самый крупный доход", $"{topIncome.Category.Name} – {topIncome.Amount:F2} ({topIncome.Date:dd.MM.yyyy})");

if (topExpense != null)

AddKeyMetric("Самый крупный расход", $"{topExpense.Category.Name} – {topExpense.Amount:F2} ({topExpense.Date:dd.MM.yyyy})");

if (mostIncomeDay != null)

AddKeyMetric("День с наибольшими доходами", $"{mostIncomeDay.Date:dd.MM.yyyy} – {mostIncomeDay.Total:F2}");

if (mostExpenseDay != null)

AddKeyMetric("День с наибольшими расходами", $"{mostExpenseDay.Date:dd.MM.yyyy} – {mostExpenseDay.Total:F2}");

});

}

break;

case "Top":

{

var filteredTransactions = transactions

.Where(t => t.Date >= fromDate && t.Date <= toDate)

.ToList();

var totalExpenses = filteredTransactions

.Where(t => !t.Category.IsIncome)

.Sum(t => t.Amount);

var topCategories = filteredTransactions

.Where(t => !t.Category.IsIncome)

.GroupBy(t => t.Category.Name)

.Select(g => new

{

Category = g.Key,

Total = g.Sum(x => x.Amount),

Transactions = g.OrderByDescending(t => t.Date).ToList()

})

.OrderByDescending(x => x.Total)

.Take(3)

.ToList();

column.Item().PaddingBottom(15).Text("Топ 3 категории расходов").FontSize(18).Bold();

foreach (var item in topCategories)

{

double percent = totalExpenses == 0 ? 0 : (double)(item.Total / totalExpenses) \* 100;

column.Item().PaddingBottom(5).Text($"{item.Category} — {item.Total:C} ({percent:F2}%)").FontSize(14).Bold();

// Прогрессбар

column.Item().PaddingBottom(8).Element(container =>

{

container

.Height(10)

.Border(1)

.BorderColor(Colors.Grey.Lighten2)

.Background(Colors.Grey.Lighten3)

.PaddingLeft(0)

.PaddingRight(0)

.PaddingVertical(1)

.Element(inner =>

{

inner.Row(row =>

{

row.ConstantItem((float)(200 \* (percent / 100)))

.Height(8)

.Background(Colors.Red.Medium)

.ExtendHorizontal()

.AlignMiddle();

row.RelativeItem()

.Height(8)

.Background(Colors.Transparent); // пустой "хвост"

});

});

});

// Список транзакций

foreach (var transaction in item.Transactions)

{

column.Item().PaddingBottom(2).Row(row =>

{

row.ConstantItem(13).Element(square =>

{

square

.AlignMiddle()

.PaddingRight(5)

.Element(inner => inner

.Height(8)

.Background(Colors.Grey.Medium)

);

});

row.RelativeItem()

.AlignMiddle()

.Text($"{transaction.Date:dd.MM.yyyy}: {transaction.Description}")

.FontSize(12)

.FontColor(Colors.Grey.Darken1);

row.ConstantItem(80)

.AlignMiddle()

.Text($"{transaction.Amount:C}")

.FontSize(12)

.FontColor(Colors.Red.Medium)

.AlignRight();

});

}

column.Item().PaddingBottom(15);

}

}

break;

case "Pie":

var pieImage = GeneratePieChartImage(transactions);

column.Item().PaddingBottom(10).Text("Распределение расходов и доходов по категориям:").Bold().AlignCenter();

column.Item().Element(container =>

{

container

.Height(300)

.AlignCenter()

.Image(pieImage, ImageScaling.FitArea);

});

break;

}

});

});

}).GeneratePdf(ms);

return ms.ToArray();

}

private byte[] GeneratePieChartImage(List<Transaction> transactions)

{

var grouped = transactions

.GroupBy(t => new { t.Category.Name, t.Category.IsIncome })

.Select(g => new

{

Category = g.Key.Name,

IsIncome = g.Key.IsIncome,

Total = g.Sum(x => x.Amount)

})

.Where(x => x.Total > 0)

.OrderByDescending(x => x.Total)

.ToList();

if (!grouped.Any())

return Array.Empty<byte>();

var width = 600;

var height = 500;

var center = new SKPoint(width / 2f, height \* 0.4f);

var radius = Math.Min(width, height) \* 0.3f;

var incomeColors = new[]

{

SKColors.DarkGreen, SKColors.ForestGreen, SKColors.LimeGreen,

SKColors.MediumSeaGreen, SKColors.SeaGreen

};

var expenseColors = new[]

{

SKColors.DarkRed, SKColors.IndianRed, SKColors.Firebrick,

SKColors.Crimson, SKColors.Red

};

using var surface = SKSurface.Create(new SKImageInfo(width, height));

var canvas = surface.Canvas;

canvas.Clear(SKColors.White);

float startAngle = 0;

int incomeColorIndex = 0;

int expenseColorIndex = 0;

float total = (float)grouped.Sum(x => (double)x.Total);

if (total == 0) total = 1;

// Рисуем сектора диаграммы

foreach (var item in grouped)

{

float sweepAngle = (float)((double)item.Total / total \* 360);

var color = item.IsIncome

? incomeColors[incomeColorIndex++ % incomeColors.Length]

: expenseColors[expenseColorIndex++ % expenseColors.Length];

using var paint = new SKPaint

{

Style = SKPaintStyle.Fill,

Color = color,

IsAntialias = true

};

using var path = new SKPath();

path.MoveTo(center);

path.ArcTo(new SKRect(center.X - radius, center.Y - radius, center.X + radius, center.Y + radius),

startAngle, sweepAngle, false);

path.Close();

canvas.DrawPath(path, paint);

float midAngle = startAngle + sweepAngle / 2;

float labelRadius = radius \* 0.7f;

float labelX = center.X + labelRadius \* (float)Math.Cos(midAngle \* Math.PI / 180);

float labelY = center.Y + labelRadius \* (float)Math.Sin(midAngle \* Math.PI / 180);

var labelPaint = new SKPaint

{

Color = SKColors.Black,

TextSize = 18,

IsAntialias = true,

TextAlign = SKTextAlign.Center

};

float percent = (float)((double)item.Total / total \* 100);

if (percent >= 5)

{

canvas.DrawText($"{percent:0.#}%", labelX, labelY, labelPaint);

}

startAngle += sweepAngle;

}

// Легенда

var legendMarginTop = height \* 0.8f;

var legendItemHeight = 28f;

var legendBoxSize = 20f;

var incomes = grouped.Where(x => x.IsIncome).ToList();

var expenses = grouped.Where(x => !x.IsIncome).ToList();

// Настроим ширину колонок и отступы, чтобы легенда была по центру

float columnWidth = 250;

float totalLegendWidth = columnWidth \* 2 + 40;

float legendStartX = (width - totalLegendWidth) / 2f;

float leftColumnX = legendStartX;

float rightColumnX = legendStartX + columnWidth + 40;

var legendTextPaint = new SKPaint

{

Color = SKColors.Black,

TextSize = 18,

IsAntialias = true,

TextAlign = SKTextAlign.Left

};

var legendTitlePaint = new SKPaint

{

Color = SKColors.Black,

TextSize = 22,

IsAntialias = true,

Typeface = SKTypeface.FromFamilyName(null, SKFontStyle.Bold)

};

// Заголовки колонок

canvas.DrawText("Доходы", leftColumnX, legendMarginTop, legendTitlePaint);

canvas.DrawText("Расходы", rightColumnX, legendMarginTop, legendTitlePaint);

// Функция для рисования легенды в колонке

void DrawLegendColumn(List<dynamic> items, float startX, float startY, SKColor[] colors)

{

for (int i = 0; i < items.Count; i++)

{

var item = items[i];

var color = item.IsIncome

? incomeColors[i % incomeColors.Length]

: expenseColors[i % expenseColors.Length];

float y = startY + legendItemHeight \* (i + 1);

using var paint = new SKPaint

{

Style = SKPaintStyle.Fill,

Color = color,

IsAntialias = true

};

// Рисуем квадрат цвета

canvas.DrawRect(startX, y - legendBoxSize + 9, legendBoxSize, legendBoxSize, paint);

// Текст по вертикали по центру квадрата

var textY = y + legendBoxSize / 2 - 2;

string text = $"{item.Category}: {item.Total:C0}";

canvas.DrawText(text, startX + legendBoxSize + 10, textY, legendTextPaint);

}

}

DrawLegendColumn(incomes.Cast<dynamic>().ToList(), leftColumnX, legendMarginTop, incomeColors);

DrawLegendColumn(expenses.Cast<dynamic>().ToList(), rightColumnX, legendMarginTop, expenseColors);

using var image = surface.Snapshot();

using var data = image.Encode(SKEncodedImageFormat.Png, 100);

return data.ToArray();

}

}

}