МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»**

Кафедра математического и компьютерного моделирования

Разработка информационной системы по учёту финансов на микросервисной архитектуре

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента 3 курса 381 группы

направление 09.04.03 — Прикладная информатика

механико-математического факультета

Трегубова Романа Эдуардовича

Научный руководитель
доцент, к.ф.-м.н. И.В. Плаксина

Зав. кафедрой
зав. каф., д.ф.-м.н., доцент Ю.А. Блинков

Саратов 2024

Введение. В современном мире управление финансами занимает важное место в жизни каждого человека. Эффективный учет доходов и расходов позволяет не только осуществлять контроль над финансовыми потоками, но и принимать обоснованные решения для достижения поставленных финансовых целей. В условиях активного развития цифровых технологий пользователи предъявляют высокие требования к информационным системам, ожидая от них удобства, функциональности, а также высокой надежности и стабильности. Цель работы — разработка системы, удовлетворяющей эти требования. Задачи включают анализ существующих систем, изучение технологий, проектирование архитектуры и реализацию прототипа. Работа актуальна для цифрового общества, содержит теоретические и практические разделы, итоги и предложения по развитию.

Анализ требований к системе учета личных финансов. Современные системы учета финансов обеспечивают ключевые инструменты для управления личными средствами, такие как отслеживание доходов и расходов, и планирование бюджета. Были проанализированы текущие решения, чтобы выявить их сильные стороны, такие как удобные интерфейсы и автоматизация, и слабые стороны, включая ограниченную настройку и проблемы масштабируемости. Новая система совершенствует эти аспекты, предлагая более гибкие и интегрированные возможности для пользователей.

Разрабатываемая система учета финансов должна предоставлять пользователям следующие возможности:

1. Учет доходов и расходов по категориям — система должна позволять пользователю фиксировать все операции, классифицируя их по заранее заданным или пользовательским категориям, например, «Продукты», «Транспорт», «Развлечения»;
2. Отслеживание операций по различным критериям —- предоставление фильтров для поиска операций по дате, категории, сумме или другим параметрам;
3. Создание отчетов —- формирование отчетов за определенные временные периоды (например, за месяц или год) для анализа доходов и расходов.

Эти функции обеспечат пользователю удобство управления финансами, анализ данных и возможности для улучшения финансового планирования.

Для эффективного управления своими финансами пользователи будут взаимодействовать с системой через интуитивно понятный интерфейс. Основные сценарии использования включают добавление и просмотр операций, фильтрация данных и управление категориями. Рассмотрены основные сценарии использования и представлены с помощью диаграммы UML в соответствии с рисунком 1.

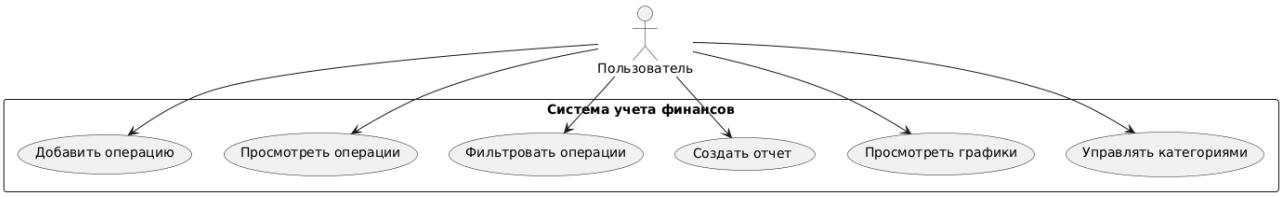

Рисунок 1 — Диаграмма основных сценариев использования

Методы технической реализации задачи. Одной из ключевых задач при разработке информационной системы является выбор архитектурного подхода, который обеспечит требуемую производительность, надежность и масштабируемость. В условиях роста объемов данных и повышения нагрузки на систему традиционные монолитные архитектуры часто оказываются недостаточно гибкими для удовлетворения современных требований. В связи с этим все более актуальным становится использование микросервисной архитектуры.

Рассмотрены способы взаимодействия микросервисов между собой, а также паттерны микросервисной архитектуры.

Далее рассмотрены причины выбора языка программирования C# и платформы .NET для серверной части информационной системы. Эти технологии обеспечивают высокую производительность, гибкость и удобство разработки, что особенно важно для микросервисной архитектуры.

Для реализации информационной системы по учёту финансов была выбрана реляционная система управления базами данных (СУБД) PostgreSQL. Это решение было принято на основе множества факторов, таких как производительность, поддержка сложных запросов, расширяемость и надёжность.

Выбор языка программирования и фреймворка для клиентской части информационной системы. Для разработки клиентской части информационной системы был выбран язык программирования JavaScript и фреймворк React, в сочетании с библиотекой компонентов Ant Design. Эти технологии обеспечивают высокую производительность, удобство в разработке и интеграции с другими сервисами, что делает их оптимальным выбором для нашего проекта.

Далее рассмотрены следующие ключевые аспекты разработки информационной системы: выбор языка программирования и фреймворка для клиентской части информационной системы, системы авторизации и аутентификации, использование контейнеризации и оркестрации, кэширование, файловый сервер S3, мониторинг и логирование.

Рассмотрен выбор оптимального языка программирования и фреймворка для разработки клиентской части системы, с учетом производительности, удобства разработки и поддержки современных веб-технологий. Рассмотрены различные подходы и инструменты для реализации надежной системы авторизации и аутентификации, обеспечивающей безопасность и удобство для пользователей. Анализ преимуществ и особенностей использования контейнеризации (например, Docker) и оркестрации (например, Kubernetes) для развертывания и управления микросервисами. Выбор подходящих технологий и инструментов для кэширования данных, что позволит повысить производительность системы и снизить нагрузку на серверы. Рассмотрение использования файлового сервера S3 для хранения и управления большими объемами данных, обеспечивая надежность и масштабируемость. Обсуждение инструментов и методик для мониторинга и логирования работы системы, что позволит своевременно выявлять и устранять проблемы, а также улучшать производительность и надежность системы.

Проектирование архитектуры информационной системы Эффективная архитектура информационной системы является фундаментом для обеспечения её производительности, надежности и способности адаптироваться к изменениям. На этапе проектирования закладываются принципы, которые определяют, насколько успешно система будет справляться с возло-

женными на неё задачами, а также обеспечивать удовлетворение потребностей пользователей в долгосрочной перспективе.

При проектировании информационной системы учета личных финансов была разработана архитектурная схема, которая отражает основные сервисы, базы данных, сопутствующие компоненты и способы их взаимодействия. Эта схема обеспечивает наглядное представление логической структуры системы и взаимодействия между её элементами. Общая схема архитектуры представлена в соответствии с рисунком 2.

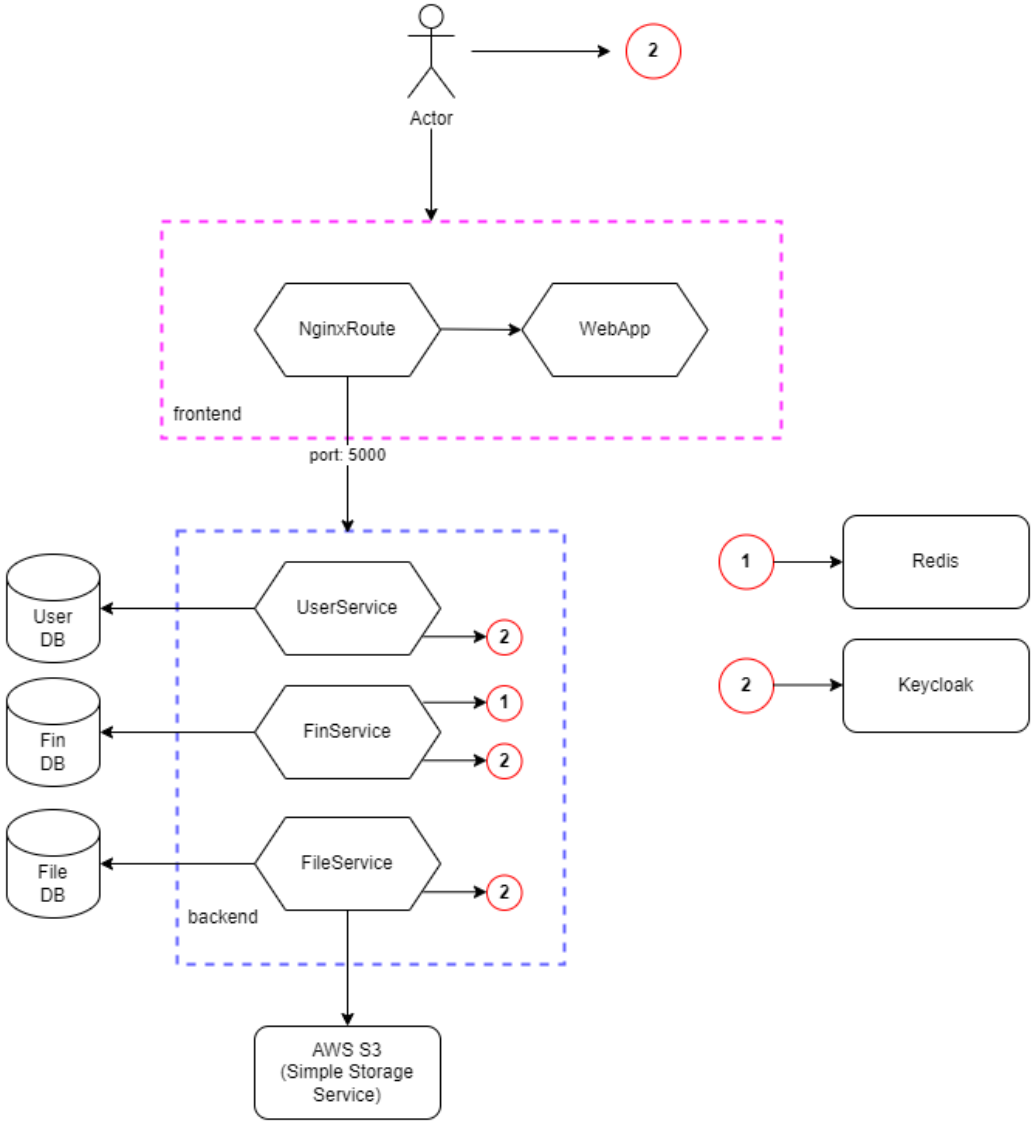

Рисунок 2 — Основная схема системы

ER диаграммы. Проектирование схем баз данных — это процесс создания логической и физической структуры базы данных, которая будет эффективно поддержи- вать хранение, управление и извлечение данных в соответ-

ствии с требовани- ями приложения и пользователей. Этот процесс включает в себя анализ данных, определение сущностей и их взаимосвязей, выбор моделей данных и создание схемы, обеспечивающей оптимальную производительность и целостность данных.

База данных «UserDB». Представим схему базы данных по работе с пользователями в виде ER-диаграммы в соответствии с рисунком 3.

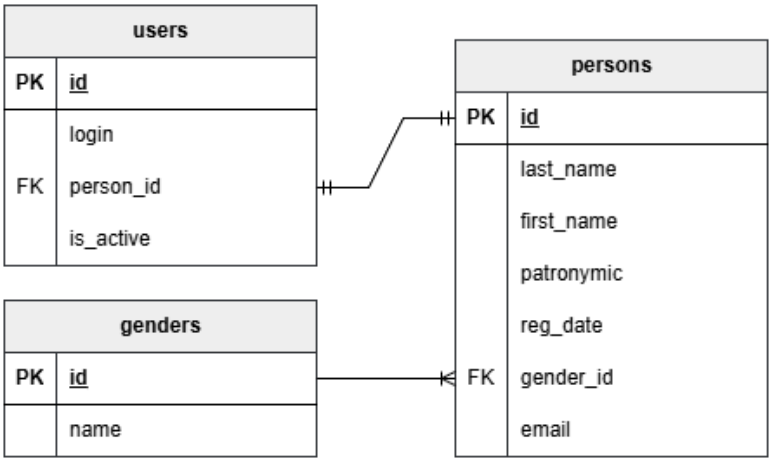

Рисунок 3 — Схема базы данных «UserDB»

База данных «FinDB». Представим схему базы данных по работе с финансовыми операциями в виде ER-диаграммы в соответствии с рисунком 4.

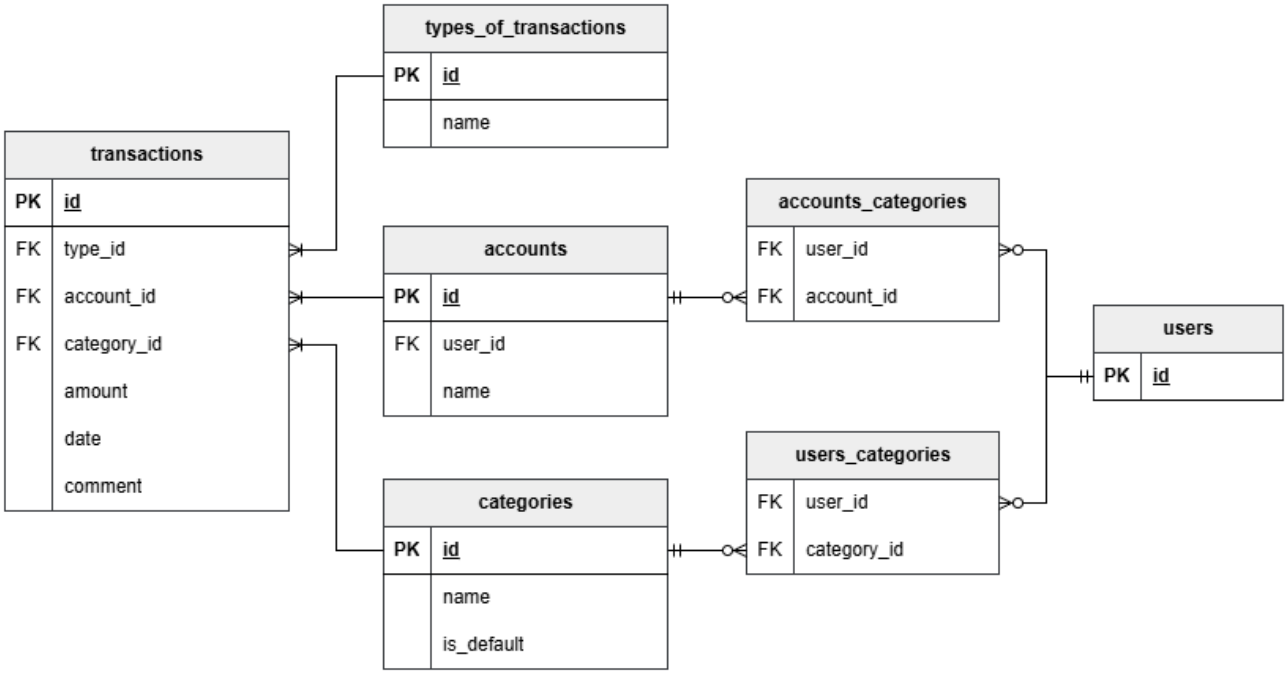

Рисунок 4 — Схема базы данных «FinDB»

База данных «FileDB». Представим схему базы данных по работе с файлами в виде ER-диаграммы в соответствии с рисунком 5.

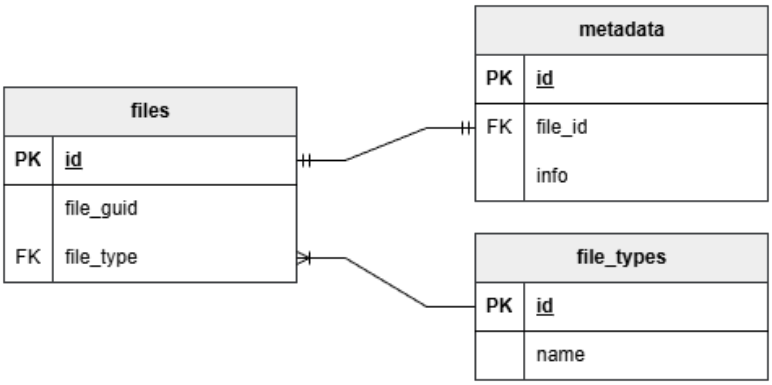

Рисунок 5 — Схема базы данных «FileDB»

Определение API для взаимодействия с микросервисами. Эндпоинты API — специфические URL-адреса, через которые клиенты взаимодействуют с сервером в рамках API. Они представляют собой точки входа в систему, где доступны определенные ресурсы и действия, определенные API. Каждый эндпоинт ассоциируется с определенным путем и, как правило, с HTTP-методом (например, GET, POST, PUT, DELETE), определяющим тип операции, которую клиент может выполнить.

Для каждого из микросервисов представлены ключевые эндпоинты:

Управление пользователями:

- POST /api/auth/register: Регистрация нового пользователя;
- POST /api/auth/login: Вход пользователя в систему;
- POST /api/auth/logout: Выход пользователя из системы;
- GET /api/users/{userId}: Получить информацию о пользователе;
- PUT /api/users/{userId}: Обновить информацию о пользователе;
- DELETE /api/users/{userId}: Удалить пользователя.

Управление финансовыми транзакциями:

- GET /api/transactions: Получить список всех транзакций пользователя;
- GET /api/transactions/{transactionId}: Получить информацию о конкретной транзакции;
- POST /api/transactions: Создать новую транзакцию;
- PUT /api/transactions/{transactionId}: Обновить информацию о транзакции;

- DELETE /api/transactions/{transactionId}: Удалить транзакцию.

Управление счетами:

- GET /api/accounts: Получить список всех счетов пользователя;
- GET /api/accounts/{accountId}: Получить информацию о конкретном счете;
- POST /api/accounts: Создать новый счет;
- PUT /api/accounts/{accountId}: Обновить информацию о счете;
- DELETE /api/accounts/{accountId}: Удалить счет.

Управление категориями трат и доходов:

- GET /api/categories: Получить список всех категорий транзакций пользователя;
- GET /api/categories/{categoryId}: Получить информацию о конкретной категории;
- POST /api/categories: Создать новую категорию;
- PUT /api/categories/{categoryId}: Обновить информацию о категории;
- DELETE /api/categories/{categoryId}: Удалить категорию.

Управление файлами:

- GET /api/files/{fileId}: Получить файл по идентификатору;
- POST /api/files: Загрузить новый файл;
- DELETE /api/files/{fileId}: Удалить файл по идентификатору.

Эти эндпоинты представляют основную функциональность каждого микросервиса и могут быть дополнены или изменены в зависимости от требований к системе учета финансов.

Wireframe интерфейса. Перед созданием пользовательского интерфейса необходимо выполнить его проектирование. На этапе проектирования важно продумать расположение элементов таким образом, чтобы взаимодействие пользователей с интерфейсом было интуитивным и удобным. Особое внимание следует уделить размещению наиболее значимых виджетов, обеспечив их нахождение в центре визуального восприятия пользователя.

Применение вайрфрейма в процессе проектирования интерфейса способствует предварительной оценке его эффективности, выявлению потенциальных проблем взаимодействия с пользователями и, как следствие, улучшению качества разрабатываемого программного продукта.

Использование Pencil в процессе проектирования пользовательского интерфейса способствует не только оптимизации разработки, но и улучшению конечного продукта за счёт предварительного анализа структуры и функциональности системы.

Окно статистики — основное окно, которое открывается при запуске приложения. На нем содержатся данные с которым пользователь взаимодействует наиболее часто. Примеры окон статистики с заполненными данными для доходов и расходов представлены в соответствии с рисунком 6. В окнах содержатся следующие элементы:

- список вкладок, которые позволяют переключать текущий режим просмотра для доходов или расходов;
- список радиокнопок, позволяющие переключать текущий диопазон дат для просматриваемых операций;
- общая статистика по доходам/расходам для выбранного периода, которые показывают наблюдаемую дату или диопазон, сумму за этот период;
- кнопка добавления новой операции, позволяет добавить новую операцию дохода/расхода;
- список доходов/расходов по категориям, так же отображает процентное соотношение и общую сумму по категории.

Элементы располагаются в порядке наиболее приоритетной информации сверху вниз. Это соответствует паттерну дизайна «туннелирование», предназначенному для направления пользователя к последовательному и неуклонному совершению конкретного действия.

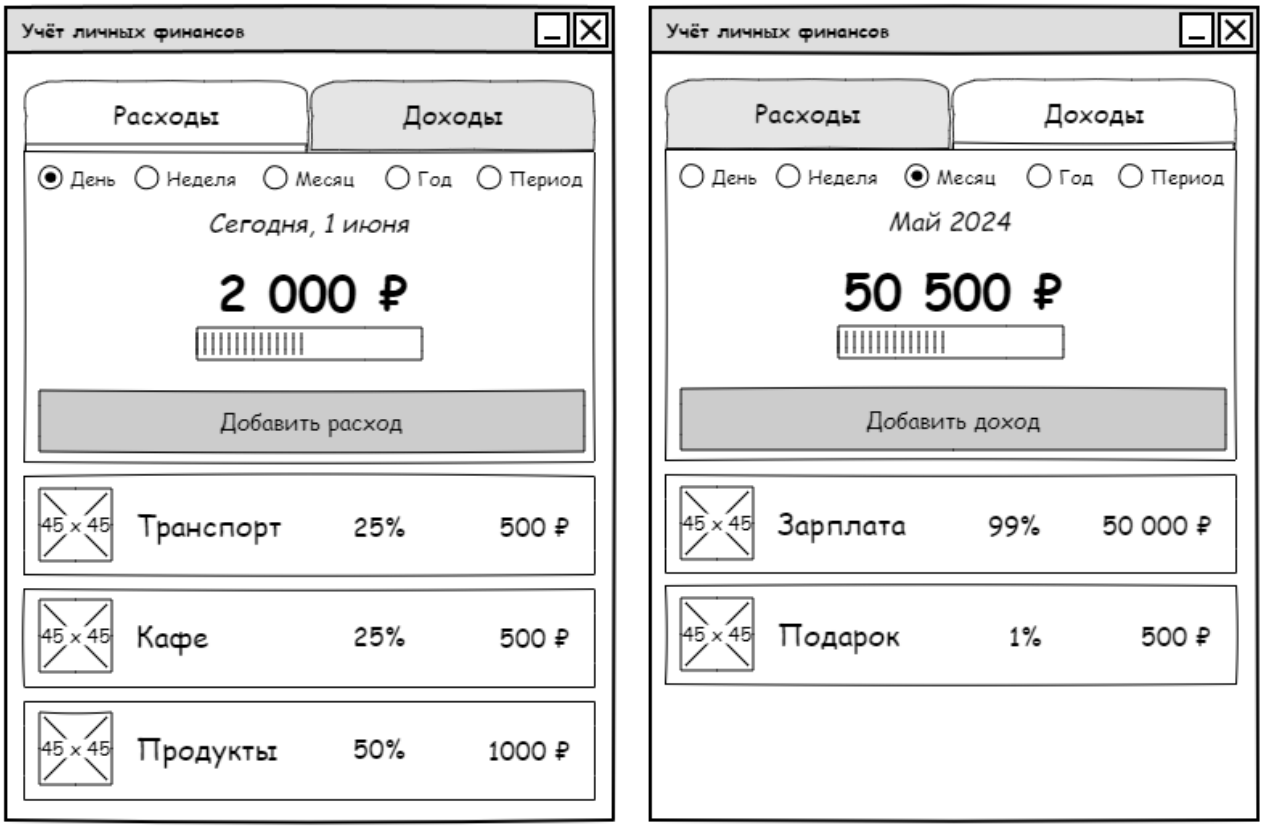

Рисунок 6 — Вайрфрейм окна статистики по доходам и расходам

Реализация информационной системы. На основании разработанной архитектурной модели и определенных функциональных требований было спроектировано приложение, которое представляет собой модульную систему, построенную на основе микросервисного подхода. Завершение этапа проектирования создало основу для перехода к реализации системы, включающей разработку программных компонентов и настройку инфраструктуры.

Реализация началась с последовательного создания ключевых элементов системы, таких как подсистемы хранения данных, серверная логика и пользовательский интерфейс. Особое внимание уделялось обеспечению согласованности взаимодействия между компонентами, стандартизации интерфейсов и соблюдению принципов отказоустойчивости и масштабируемости.

Для реализации информационной системы было выбрано и настроено программное обеспечение, обеспечивающее полноценную разработку, тестирование и развертывание проекта. В данном разделе приведён перечень исполь-

зуемых инструментов, их функциональное назначение, а также этапы установки и настройки.

Реализация микросервисов. В данном разделе будет показана реализация микросервиса с использованием платформы .NET, фреймворка ASP.NET Core, библиотеки FluentValidation для валидации данных и Dapper для работы с базой данных. Провайдером ADO.NET выступает npgsql. Этот микросервис предназначен для выполнения операций с финансовыми данными в рамках информационной системы.

Микросервис состоит из нескольких ключевых компонентов:

- **PL (Presentation Layer) — слой представления**, в него входят контроллеры, которые обрабатывают HTTP-запросы и валидация ViewModel объектов, которые представляют собой самый верхний слой объектов которые попадают в приложение от пользователя;
- **BL (Business Layer) — слой бизнес-логики**, содержит логику, специфичную для конкретного бизнеса или домена приложения. Он отвечает за выполнение сложных операций, которые не могут быть реализованы на уровне представления или доступа к данным;
- **DAL (Data Access Layer) — слой доступа к данным**, отвечает за взаимодействие с базой данных или другими источниками данных. Он абстрагирует детали работы с данными от остальной части приложения.

Реализация фронтенда. Для реализации фронтенд-части информационной системы был выбран фреймворк React, который позволяет создавать динамичные и отзывчивые пользовательские интерфейсы. В качестве компонента для визуального оформления был использован библиотека Ant Design, предоставляющая набор готовых UI-компонентов. Взаимодействие с бэкенд-микросервисом происходило через REST API, что позволяло организовать эффективное обмен данными между фронтендом и сервером.

Был создан компонент окна статистики. Для реализации данного компонента пользовательского интерфейса использовалась библиотека компонентов Ant Design, которая предоставляет множество готовых и адаптируемых элементов пользовательского интерфейса, таких как формы, кнопки, модальные окна и другие. Использование этих компонентов позволило значитель-

но ускорить процесс разработки и обеспечить высокое качество интерфейса. Скриншот окна представлен в соответствии с рисунком 7.

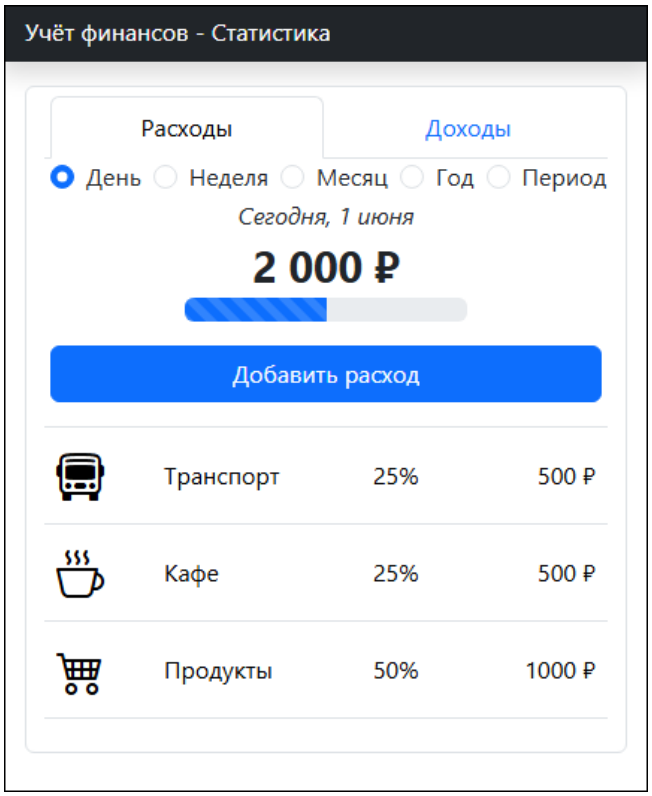

Рисунок 7 — Окно статистики

Взаимодействие между микросервисами. Одной из ключевых задач при реализации микросервисной архитектуры является организация взаимодействия между компонентами системы. В данном разделе рассмотрено использование **gRPC** для обеспечения высокопроизводительной и типобезопасной коммуникации между микросервисами системы учета личных финансов.

gRPC стал эффективным решением для реализации взаимодействия между микросервисами, обеспечив надежность, производительность и удобство дальнейшего масштабирования системы.

Интеграция .NET Aspire в микросервисную систему. В рамках данного исследования была проведена интеграция платформы .NET Aspire в микросервисную архитектуру системы учёта финансов. Использование .NET Aspire позволяет упростить процесс оркестрации микросервисов, обеспечивая удобное управление зависимостями, взаимодействием между сервисами и их конфигурацией.

Интеграция .NET Aspire обеспечила автоматизированное управление зависимостями между микросервисами, упростила их конфигурацию и повысила масштабируемость системы. Использование данной платформы подтвердило её эффективность для реализации облачных решений. В дальнейшем планируется интеграция дополнительных функциональных возможностей и улучшение аналитических инструментов.

Заключение. В рамках магистерской работы была разработана информационная система учета личных финансов на основе микросервисной архитектуры. Основная цель исследования — создание удобной и эффективной системы управления финансовыми потоками пользователей с высокой производительностью, надежностью и масштабируемостью.

Анализ существующих систем учета финансов выявил их основные преимущества и недостатки. Исследование современных подходов к проектированию информационных систем обосновало выбор микросервисной архитектуры. Были сформулированы функциональные требования, спроектирована архитектура системы, включая микросервисы, схемы баз данных и методы взаимодействия компонентов.

В ходе реализации были созданы основные программные компоненты и настроена инфраструктура для их развертывания. Результаты подтвердили эффективность выбранной архитектуры и технологий. Система демонстрирует гибкость и надежность, способную адаптироваться к увеличению нагрузки и изменению потребностей пользователей.

Внедрение системы может значительно улучшить управление личными финансами, предлагая удобный и функциональный инструмент. Дальнейшее развитие включает интеграцию с внешними сервисами, расширение аналитических модулей и оптимизацию интерфейса. Работа вносит вклад в методы проектирования информационных систем на основе микросервисной архитектуры и может быть полезна для дальнейших исследований и разработок.