

Содержание

Введение	5
Постановка задачи и требования к системе	7
2.1 Требования к приложению	7
Средства реализации	8
3.1 Клиентская часть	8
3.2 Серверная часть	9
3.3 Обоснование выбора технологий	9
Архитектура системы	10
4.1 Общая архитектура приложения	10
4.2 Структура базы данных	11
4.3 API-routers	12
4.4 Клиентские компоненты	14
Реализация	16
5.1 Система аутентификации	16
5.2 Управление группами	16
5.3 Учёт расходов	19
5.4 Регистрация платежей	20
5.5 Система событий	20
Тестирование	22
Дальнейшее развитие системы	24

Заключение	25
Список литературы	26
Приложение	27

Введение

Организация совместных поездок и мероприятий с друзьями и знакомыми стала обыденным явлением. При этом одной из наиболее трудоёмких задач является справедливое разделение общих расходов между участниками. В процессе путешествия или группового мероприятия возникают различные траты: проживание, питание, транспорт, развлечения. Зачастую один человек оплачивает расходы за всю группу, а затем возникает необходимость вести учёт и производить взаиморасчёты.

Проблема учёта и разделения групповых расходов имеет несколько важных аспектов. Во-первых, в процессе совместного мероприятия возникает множество различных трат: бронирование жилья, покупка билетов, оплата питания, расходы на развлечения и транспорт. Во-вторых, часто один или несколько участников оплачивают расходы за всю группу, что требует последующих взаиморасчётов. В-третьих, необходимо обеспечить прозрачность и справедливость распределения расходов с учётом того, что разные участники могут иметь различный уровень участия в тех или иных тратах.

Традиционные методы ведения учёта групповых расходов включают использование бумажных блокнотов, текстовых сообщений в мессенджерах или электронных таблиц. Однако такие подходы имеют существенные недостатки:

- Высокая вероятность ошибок при ручном подсчёте
- Сложность отслеживания изменений и истории операций
- Отсутствие автоматического расчёта итоговых долгов
- Необходимость дублирования информации между участниками

Существующие решения для разделения расходов, такие как Splitwise,

Tricount и Settle Up, решают эти проблемы, хоть и имеют свои недостатки: зависимость от постоянного интернет-соединения, хранение персональных финансовых данных на внешних серверах, наличие платных функций для базовых возможностей, а также сложность интерфейса в некоторых приложениях.

Актуальность работы определяется необходимостью создания решения, которое сочетало бы простоту использования, локальное хранение данных, возможность работы в условиях ограниченного доступа к сети и полный контроль пользователей над своей информацией.

Цель данной работы – разработка веб-приложения для управления групповыми расходами, которое обеспечивает автоматизацию расчётов, прозрачность финансовых операций и возможность работы в оффлайн-режиме.

Задачи:

1. Разработать архитектуру веб-приложения на основе современных технологий
2. Спроектировать структуру базы данных для хранения информации о пользователях, группах, расходах и платежах
3. Разработать алгоритм оптимизации количества транзакций при расчёте взаиморасчётов между участниками
4. Создать интуитивный пользовательский интерфейс, адаптированный для различных устройств
5. Реализовать функциональность Progressive Web App
6. Провести тестирование разработанного приложения

Постановка задачи и требования к системе

Система должна обеспечивать следующие основные функции:

- Управление пользователями и их аутентификация;
- Создание и администрирование групп участников;
- Учёт расходов с различными способами разделения сумм между участниками;
- Автоматический расчёт балансов и оптимизация схем взаиморасчётов;
- Регистрация выполненных платежей между участниками.

2.1 Требования к приложению

Управление пользователями:

- Регистрация новых пользователей с использованием email и пароля;
- Аутентификация пользователей.

Управление группами:

- Создание групп;
- Генерация уникальных кодов приглашений;
- Присоединение пользователей к группам по коду приглашения.

Учёт расходов:

- Добавление расхода с указанием описания, суммы, даты и плательщика;
- Выбор участников, между которыми разделяется расход;

- Поддержка различных способов разделения;
- Редактирование и удаление ранее добавленных расходов;

Расчёт балансов между участниками:

- Автоматический расчёт баланса каждого участника;
- Оптимизация количества необходимых транзакций;
- Отображение схемы взаиморасчётов (кто кому и сколько должен).

Регистрация платежей:

- Отметка о фактически выполненном переводе между участниками;
- Автоматический пересчёт балансов с учётом зарегистрированных платежей;

Средства реализации

3.1 Клиентская часть

Для разработки клиентской части приложения использовались следующие технологии:

React 19 [1] – современная JavaScript-библиотека для создания пользовательских интерфейсов на основе компонентного подхода.

Next.js 16 – React-фреймворк с поддержкой серверного рендеринга. Используется для: API Routes для создания backend-функционала, file-based роутинга для удобной навигации.

TypeScript 5 [2] – типизированный JavaScript с поддержкой статической проверки типов. Обеспечивает раннее выявление ошибок на этапе разработки и улучшение поддерживаемости кода.

Tailwind CSS 4 – utility-first CSS фреймворк. Позволяет определять стили прямо в HTML-коде, используя предопределённые классы-утилиты.

JavaScript [3] – язык программирования, используемый для реализации логики приложения как на клиентской, так и на серверной стороне.

3.2 Серверная часть

Node.js [4] – серверная платформа на базе V8 JavaScript-движка. Обеспечивает единый язык для клиентской и серверной частей приложения, а также доступ к обширной экосистеме пакетов через менеджер npm.

Next.js API Routes – встроенная возможность создания RESTful API [5, 6]. Файлы в директории `app/api/` автоматически становятся HTTP-эндпоинтами.

SQLite [7] – встраиваемая реляционная СУБД. Используется библиотека Better-SQLite3 для синхронного взаимодействия с базой данных, что обеспечивает простоту работы.

Custom HTTPS Server – собственный сервер на базе Node.js `https` модуля [?] (`server.js`). Так как у меня iPhone он отказывается открывать http сайты из локальной сети. Поэтому были созданы самоподписанные сертификаты и соответственно для этого нужен такой «костыль».

3.3 Обоснование выбора технологий

Главным критерием выбора такого стека – единая экосистема Next.js. Это позволяет использовать один и тот же язык для фронта и бэка, что упрощает разработку и позволяет переиспользовать код (типы данных, утилиты).

Архитектура системы

4.1 Общая архитектура приложения

Приложение TripF построено по классической клиент-серверной архитектуре с использованием паттерна MVC (Model-View-Controller), адаптированного для современных веб-приложений.

Архитектурные слои:

1. **Слой представления (View)** – React-компоненты, отвечающие за отображение интерфейса и взаимодействие с пользователем (`app/components/`).
2. **Слой маршрутизации (Controller)** – Next.js страницы (`app/[route]/page.tsx`) и API Routes (`app/api/[route]/route.ts`).
3. **Бизнес-логика** – функции для работы с данными, расчётов, валидации (`app/lib/`).
4. **Слой данных (Model)** – взаимодействие с SQLite через библиотеку Better-SQLite3. Схема БД и функции доступа в `app/lib/db.ts`.

Поток данных:

1. Пользователь взаимодействует с React-компонентом
2. Компонент вызывает API-эндпоинт через `fetch`/`SWR`
3. API Route проверяет авторизацию, валидирует данные
4. Бизнес-логика обрабатывает запрос, работает с БД
5. Результат возвращается в виде JSON
6. `SWR` кэширует ответ и обновляет компонент
7. React перерисовывает интерфейс

4.2 Структура базы данных

База данных спроектирована в соответствии с принципами реляционной модели данных [8] и состоит из 8 таблиц.

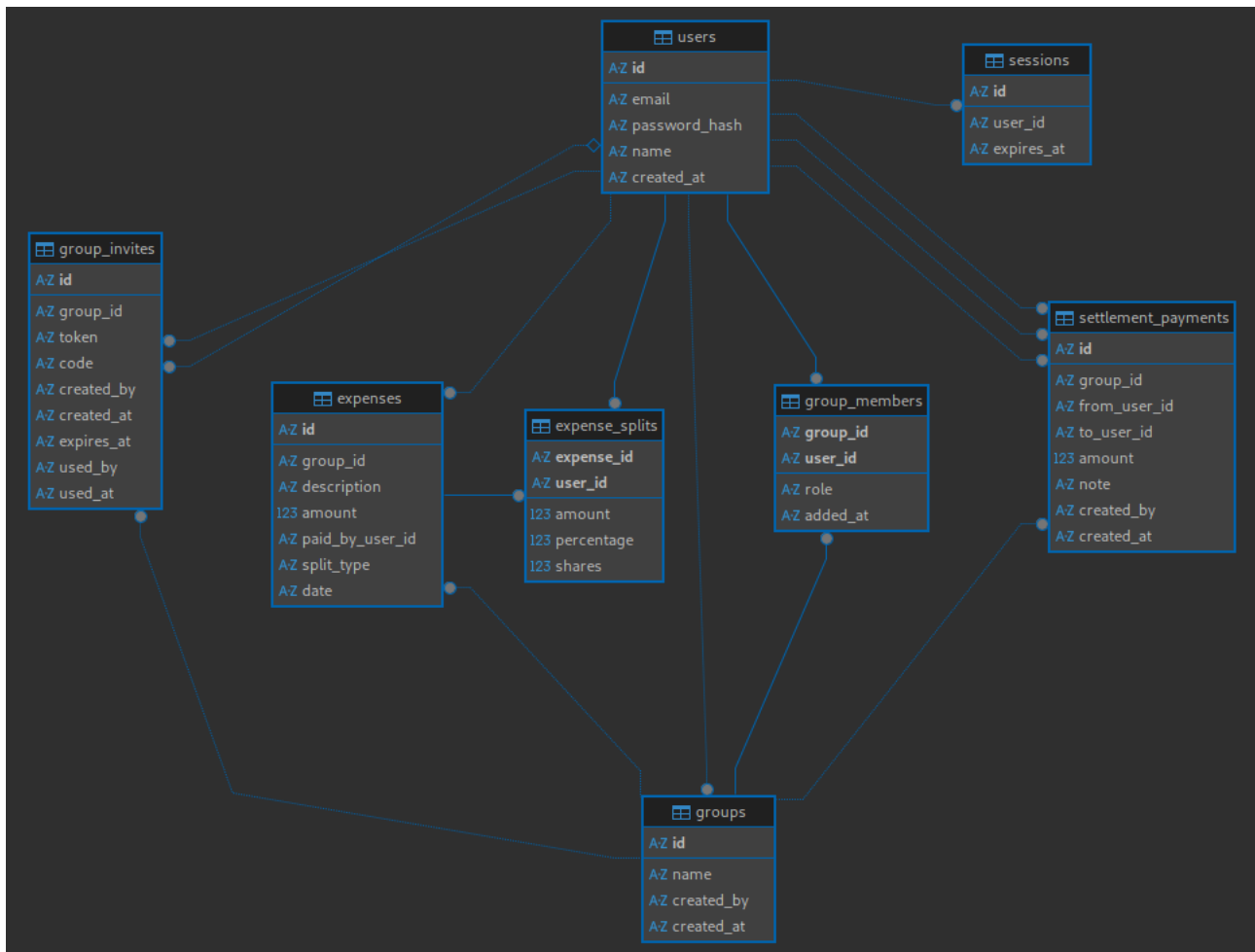


Рис. 1: Схема базы данных

Основные сущности: `users` – таблица с пользователями, `groups` – таблица с группами, `expenses` – таблица с расходами, `settlement_payments` – таблица с платежами.

Вспомогательные сущности: `group_members`, `expense_splits` – для связи многие ко многим, `group_invites` – таблица с приглашениями в группу, `sessions` – таблица с сессиями пользователей.

4.3 API-routers

Аутентификация (/api/auth/*):

- POST /api/auth/register – регистрация нового пользователя. Создает пользователя, хеширует пароль, создает сессию.
- POST /api/auth/login – вход в систему. Проверяет пароль, создает сессию, устанавливает cookie.
- POST /api/auth/logout – выход из системы. Удаляет сессию и cookie.
- GET /api/auth/me – получение информации о текущем пользователе.

Управление группами (/api/groups/*):

- GET /api/groups – получение списка групп текущего пользователя
- POST /api/groups – создание новой группы.
-
- GET /api/groups/[id] – получение информации о группе с участниками
- DELETE /api/groups/[id] – удаление группы.

Расходы (/api/groups/[id]/expenses/*):

- GET /api/groups/[id]/expenses – список всех расходов группы с разбивкой
- POST /api/groups/[id]/expenses – создание расхода. После создания генерируется событие `expense:created`.
- GET /api/groups/[id]/expenses/[expenseId] – получение конкретного расхода

- PUT /api/groups/[id]/expenses/[expenseId] – редактирование расхода. Удаляет старые splits и создаёт новые.
- DELETE /api/groups/[id]/expenses/[expenseId] – удаление расхода. Каскадно удаляет splits благодаря FOREIGN KEY.

Платежи (/api/groups/[id]/payments/*):

- GET /api/groups/[id]/payments – список всех платежей группы
- POST /api/groups/[id]/payments – регистрация платежа. Вставляет запись в settlement_payments. Генерирует событие payment:created.
- DELETE /api/groups/[id]/payments/[paymentId] – удаление платежа

Приглашения (/api/groups/[id]/invites/*):

- GET /api/groups/[id]/invites – список активных приглашений группы
- POST /api/groups/[id]/invites – создание нового приглашения. Генерирует уникальный 6-символьный код и 48-символьный токен.
- POST /api/invites/accept – принятие приглашения. Добавляет пользователя в group_members, отмечает приглашение использованным.

Обработка ошибок:

Все API-маршруты используют единообразную обработку ошибок: [9]

- 401 Unauthorized – нет сессии или сессия истекла
- 403 Forbidden – нет доступа к ресурсу
- 404 Not Found – ресурс не найден
- 400 Bad Request – невалидные данные

- 409 Conflict – конфликт (например, email уже существует)
- 500 Internal Server Error – ошибка сервера

Все ошибки возвращаются в формате `{"error": "[Описание ошибки]"}`.

4.4 Клиентские компоненты

React-компоненты организованы в папке `app/components/`

AuthGuard.tsx – компонент-обёртка для проверки наличия сессии, при её отсутствии перенаправляет на страницу входа.

GroupCard.tsx – карточка группы на главной странице. Отображает название, количество участников, общую сумму расходов.

GroupView.tsx – главный компонент страницы группы.

ExpenseList.tsx – список расходов группы. Отображает каждый расход с описанием, суммой, плательщиком, датой. Кнопки редактирования и удаления.

ExpenseForm.tsx – форма добавления/редактирования расхода.

SettlementList.tsx – список расчётов. Отображает кто кому и сколько должен. Кнопки для регистрации платежа. Показывает историю выполненных платежей.

ParticipantList.tsx – список участников группы с отображением их балансов (переплата/долг).

GroupInvitePanel.tsx – панель управления приглашениями. Показывает активные приглашения, позволяет создать новое, копировать код/ссылку в буфер обмена.

JoinGroupForm.tsx – форма для присоединения к группе по коду приглашения.

UserMenu.tsx – header приложения с отображением логотипа, имени и

кнопкой выхода.

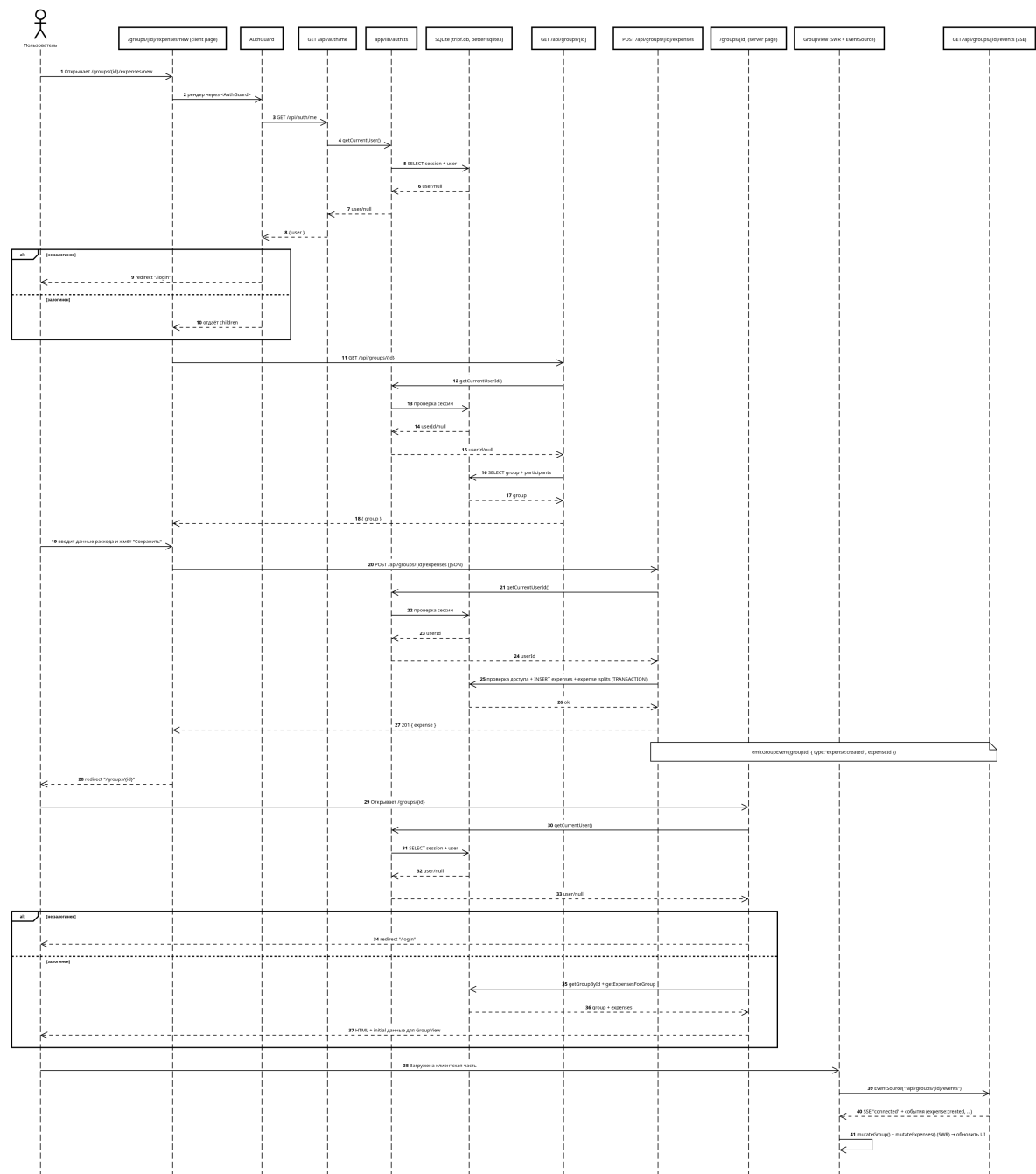


Рис. 2: Диаграмма классов

Реализация

5.1 Система аутентификации

Система аутентификации реализована в файле `app/lib/auth.ts`. Обработывает регистрацию, вход, управление сессиями и защиту паролей.

Управление сессиями

Сессии хранятся в таблице `sessions` БД. При входе создаётся новая сессия: реализация функции `createSession` приведена в листинге 1.

Время жизни сессии – 30 дней. При каждом запросе проверяется актуальность сессии.

Идентификатор сессии хранится в HTTP-only cookie, откуда он и извлекается при необходимости валидировать сессию: пример функции установки cookie приведён в листинге 3.

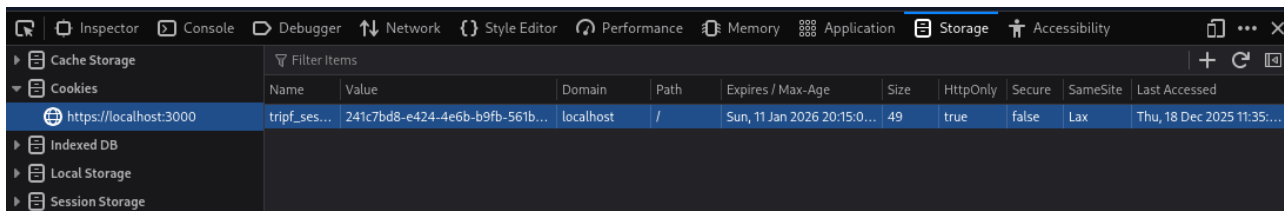


Рис. 3: Cookie с идентификатором сессии

5.2 Управление группами

Функции для работы с группами находятся в `app/lib/groups.ts` и `app/lib/invites.ts`.

Система приглашений

Для присоединения к группе используются два типа идентификаторов:

- **Токен** – для ссылок-приглашений вида `/invite/[token]`

- **Код** – короткий код типа «A3K9M2» для ручного ввода

Генерация уникального кода (`app/lib/invites.ts`): реализация генерации уникального кода показана в листинге 4.

Реализация генерации приглашения приведена в листинге 5.

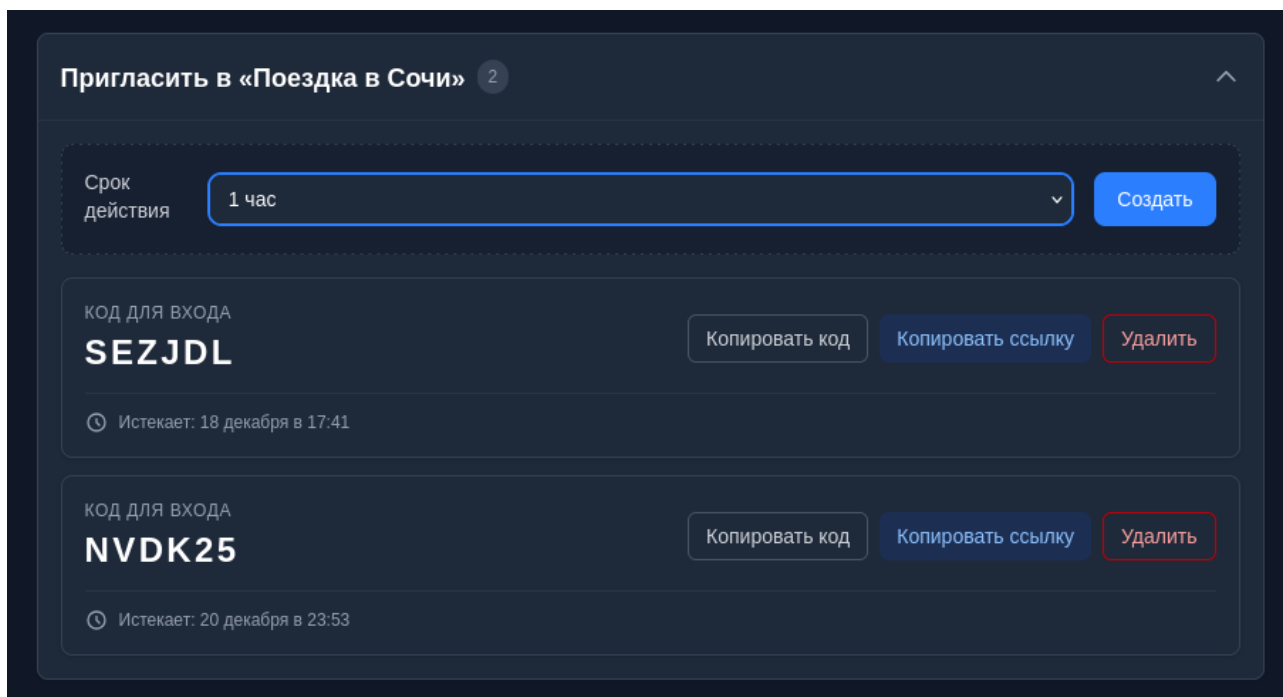


Рис. 4: Примеры приглашений

Примеры ссылки приглашения:

`localhost:3000/invite/558130fc7f7d34e9b4650578bc1fa6cb5dce7823c0e8d6c7`

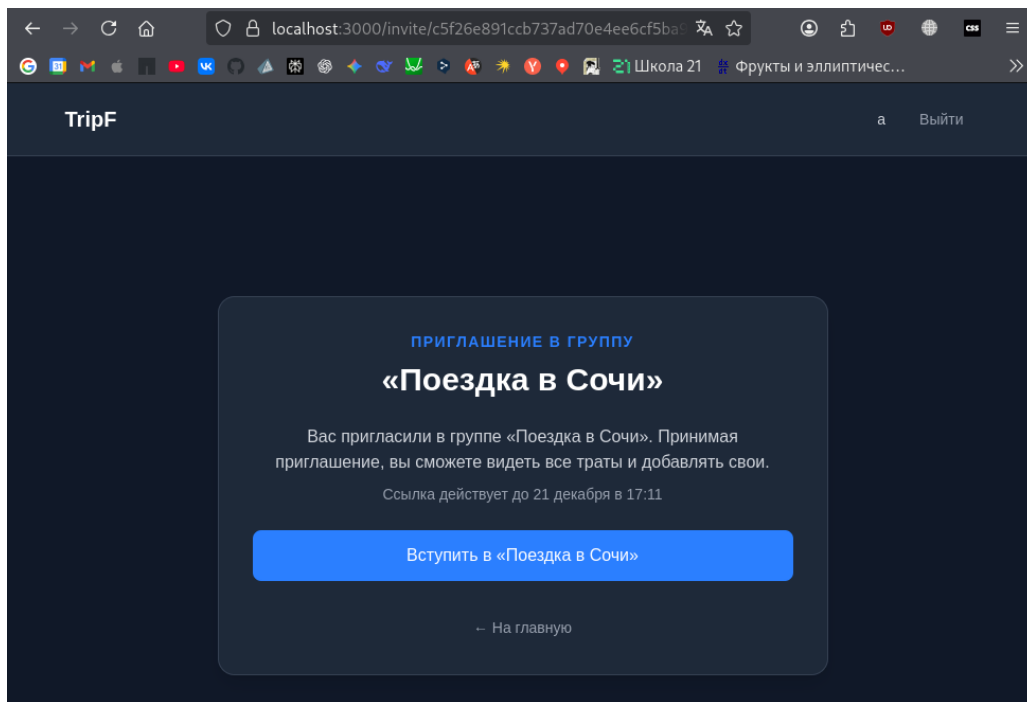


Рис. 5: Пример перехода по ссылке приглашения

При повторном переходе по ссылке приглашения будет отображаться ошибка:

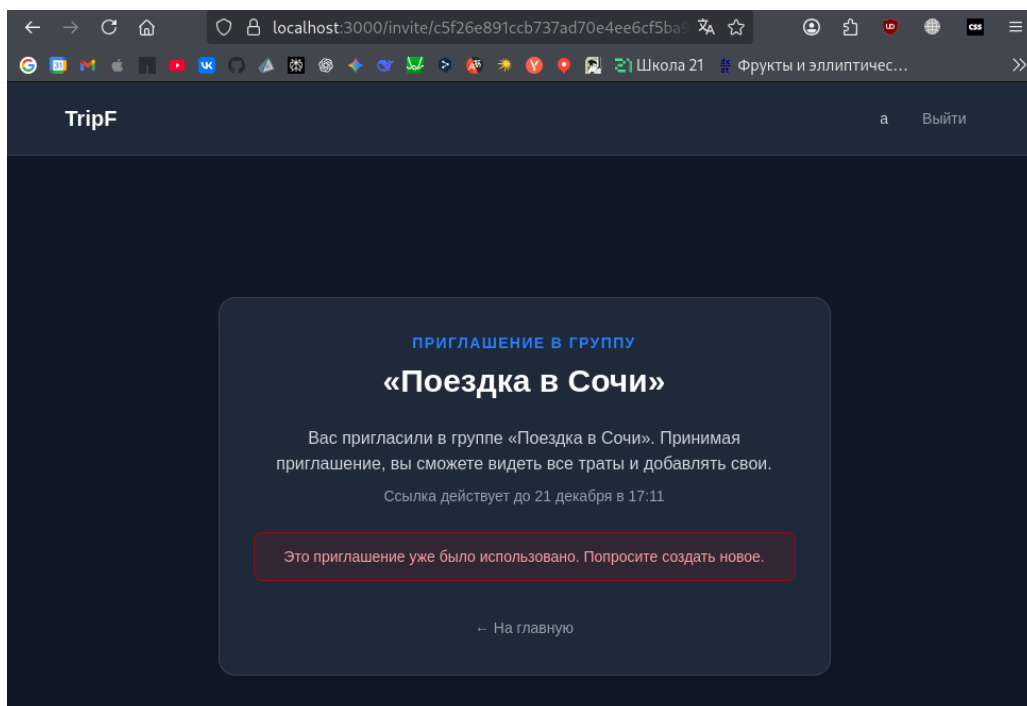


Рис. 6: Ошибка перехода по ссылке приглашения

5.3 Учёт расходов

Создание расхода

API-маршрут `api/groups/[id]/expenses/route.ts` принимает JSON формата указанного в листинге 6.

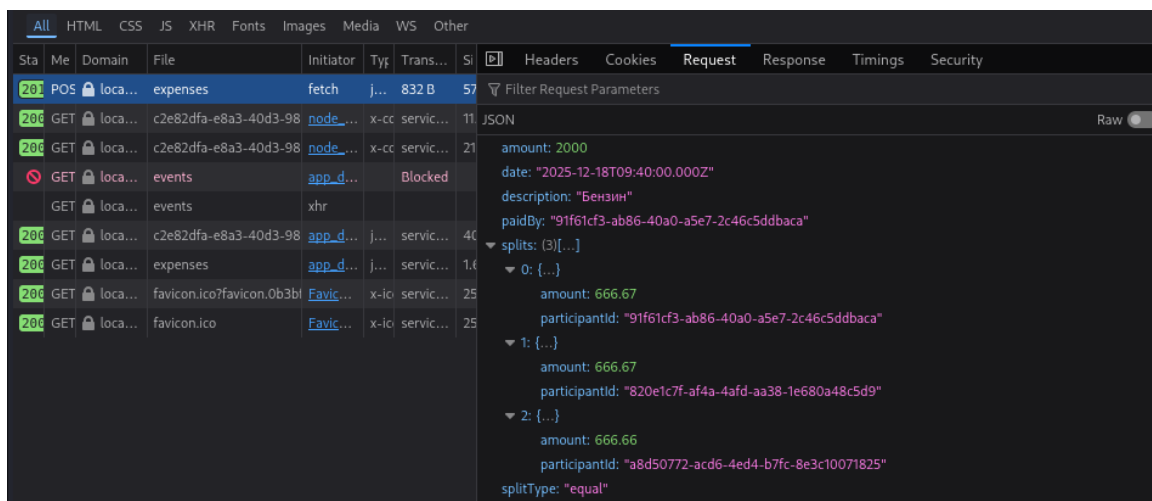


Рис. 7: Запрос на создание расхода

Реализация в `api/groups/[id]/expenses/route.ts`: фрагмент кода вставки расхода и разбивки приведён в листинге 7.

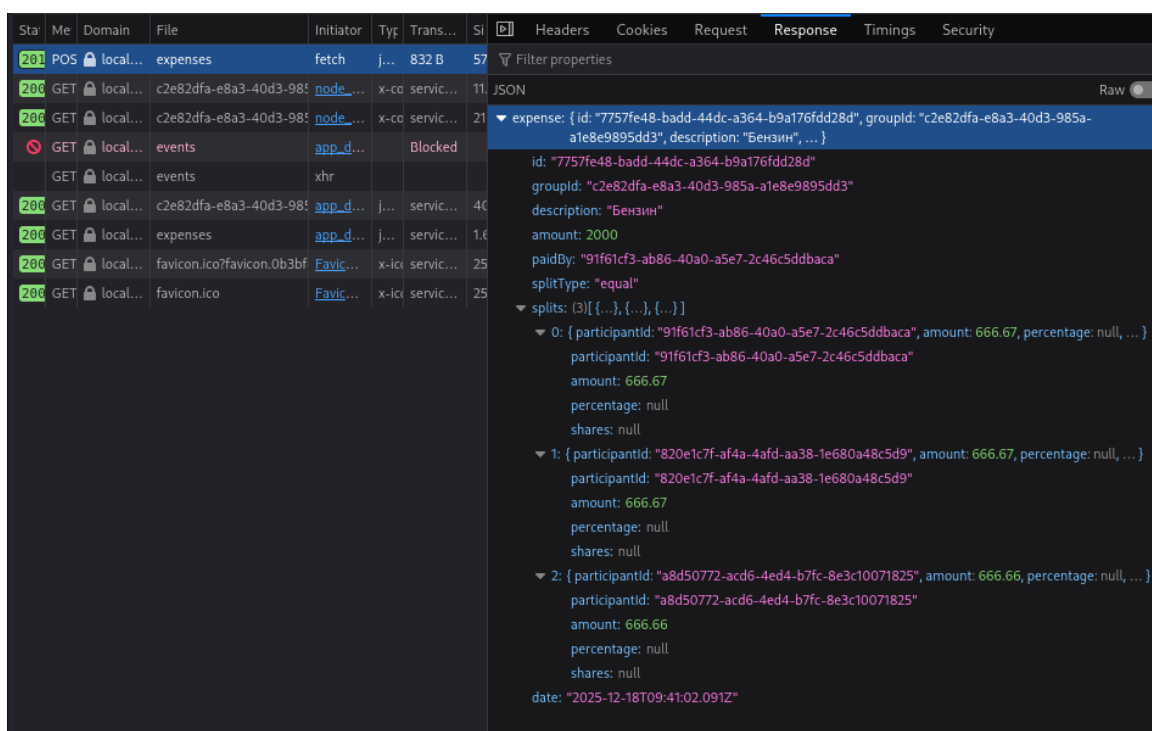


Рис. 8: Ответ о создании расхода

Затем генерируем событие для обновления UI: пример генерации события об изменении данных показан в листинге 8.

5.4 Регистрация платежей

Когда участники фактически производят переводы друг другу (через банк, наличными), они регистрируют это в приложении для пересчёта балансов.

Создание платежа

API-маршрут `POST api/groups/[id]/payments/route.ts`: полный обработчик создания платежа приведён в листинге 9.

201	POST	localhost:...	payments	fetch	json	565 B	3...	Filter Request Parameters
200	GET	localhost:...	c2e82dfa-e8a3-40d3-985a-a1e8e9f...	app_d6f9...	json	service ...	4...	JSON Raw
200	GET	localhost:...	payments	app_d6f9...	json	service ...	3...	amount: 4933.34 from: "91f61cf3-ab86-40a0-a5e7-2c46c5ddbaca" note: null to: "a8d50772-acd6-4ed4-b7fc-8e3c10071825"
200	GET	localhost:...	expenses	app_d6f9...	json	service ...	1...	
200	GET	localhost:...	payments	app_d6f9...	json	service ...	3...	
200	GET	localhost:...	new?_rsc=lvn0n	node_mo...	x-c-...	service ...	9...	
⛔	GET	localhost:...	events	xhr	Blocked			

Рис. 9: Запрос на создание платежа

Sta...	Me...	Domain	File	Initiator	Type	Transfer...	Size	🔍 Headers	Cookies	Request	Response	Timings	Security
201	POST	🔒 localhost...	payments	fetch	json	565 B	3...	🔍 Filter properties					
200	GET	🔒 localhost...	c2e82dfa-e8a3-40d3-985a-a1e8e9...	app_d6f9...	json	service ...	4...	JSON					Raw
200	GET	🔒 localhost...	payments	app_d6f9...	json	service ...	3...	<div>▼ payment: { id: "8df86b96-e966-404e-a6e3-06c7e5f7bce0", groupId: "c2e82dfa-e8a3-40d3-985a-a1e8e9895dd3", from: "91f61cf3-ab86-40a0-a5e7-2c46c5ddbaca", ... } id: "8df86b96-e966-404e-a6e3-06c7e5f7bce0" groupId: "c2e82dfa-e8a3-40d3-985a-a1e8e9895dd3" from: "91f61cf3-ab86-40a0-a5e7-2c46c5ddbaca" to: "a8d50772-acd6-4ed4-b7fc-8e3c10071825" amount: 4933.34 note: null createdBy: "91f61cf3-ab86-40a0-a5e7-2c46c5ddbaca" createdAt: "2025-12-18 11:49:44"</div>					
200	GET	🔒 localhost...	expenses	app_d6f9...	json	service ...	1...						
200	GET	🔒 localhost...	payments	app_d6f9...	json	service ...	3...						
200	GET	🔒 localhost...	new?_rsc=lvmon	node_mo...	x-c...	service ...	9...						
🚫	GET	🔒 localhost...	events	xhr	Blocked								
200	GET	🔒 localhost...	me	app_fd391...	json	service ...	1...						
200	GET	🔒 localhost...	c2e82dfa-e8a3-40d3-985a-a1e8e9...	app_fd391...	json	service ...	4...						
200	GET	🔒 localhost...	me	app_fd391...	json	service ...	1...						
200	GET	🔒 localhost...	c2e82dfa-e8a3-40d3-985a-a1e8e9...	app_fd391...	json	service ...	4...						

Рис. 10: Ответ на создание платежа

5.5 Система событий

Для синхронизации интерфейса при изменении данных используется простая реализация паттерна Observer в `app/lib/eventBus.ts`.

Реализация `EventBus` вынесены в листинг 10.

Использование в компоненте:

В `GroupView.tsx` компонент подписывается на события группы: пример использования `EventBus` на клиенте в компоненте `GroupView` приведён в листинге 11.

Функция `mutate` из `SWR` инвалидирует кэш и перезапрашивает данные с сервера. Это позволяет обновлять данные в `UI` при изменении данных на сервере, у всех пользователей в группе. Но такая система не сохранит события при перезагрузке сервера.

Для production-версии в идеале переписать на Redis Pub/Sub или WebSocket.

Тестирование

Допустим есть 4 человека, которые хотят поехать в путешествие. Они регистрируются в приложении. Один из них создаёт группу и приглашает трёх других. Пользователи вводят коды приглашений и попадают в группу. Далее каждый вносит расходы в группу. Затем происходит автоматический расчёт долгов между участниками. После этого пользователи отмечают платежи как оплаченные и система автоматически рассчитывает балансы между участниками.

The screenshot displays the TripF application interface for a trip group named "Поездка в Сочи". The interface is dark-themed and includes the following sections:

- Header:** "TripF" logo, user initials "a", and a "Выйти" (Logout) button.
- Navigation:** A link "← Назад к группам" (Back to groups).
- Trip Group:** "Поездка в Сочи" with a total expense of "Всего расходов: 14 400 Р". Buttons for "+ Добавить расход" (Add expense) and "Удалить группу" (Delete group) are present.
- Участники (Participants):** Three participants labeled "a", "b", and "c", each with a "Удалить" (Delete) button.
- Пригласить в «Поездка в Сочи» (Invite to 'Trip to Sochi'):** A dropdown menu.
- Расходы (Expenses):**
 - Жильё (Accommodation):** 12 000 Р, paid on 17 Dec 2025. Split between a (4 000 Р), b (4 000 Р), and c (4 000 Р). Buttons: "Редактировать" (Edit), "Удалить" (Delete).
 - Бенз (Gas):** 2 400 Р, paid on 17 Dec 2025. Split between a (800 Р), b (800 Р), and c (800 Р). Buttons: "Редактировать" (Edit), "Удалить" (Delete).
- Рекомендуемые расчеты (Recommended calculations):**
 - Participant a:** Should pay 4 800 Р, should receive from c. Button: "Отметить оплату" (Mark payment).
 - Participant b:** Should pay 2 400 Р, should receive from c. Button: "Отметить оплату" (Mark payment).
- Платежи (Payments):**
 - Fields for "Кто платит" (Who pays) and "Кто получает" (Who receives).
 - Fields for "Сумма" (Amount) and "Комментарий (необязательно)" (Commentary, optional).
 - Buttons: "Добавить платёж" (Add payment), "Платежей пока нет" (No payments yet).

Рис. 11: Интерфейс приложения

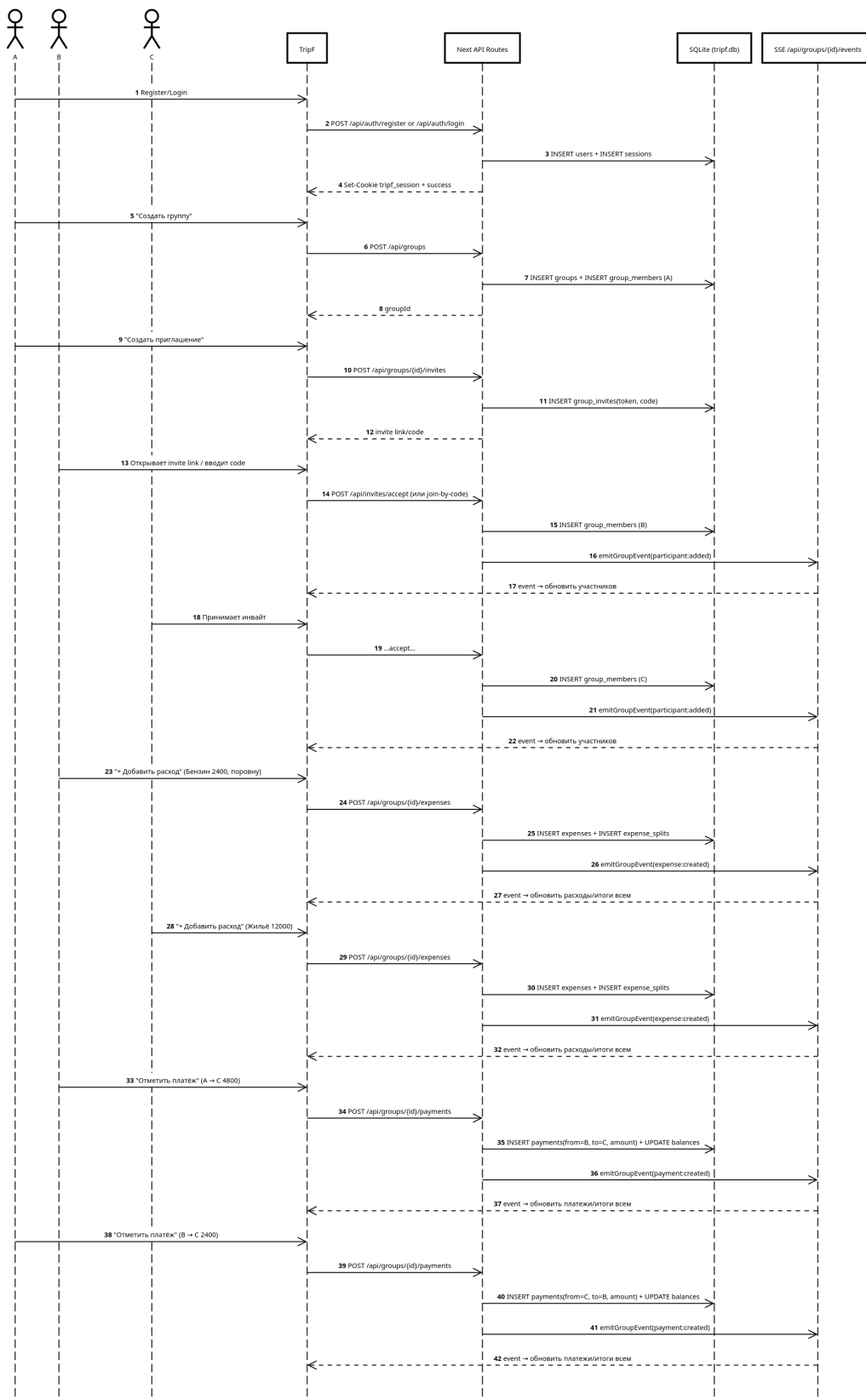


Рис. 12: Сценарий использования

Дальнейшее развитие системы

В текущей версии реализованы базовые функции для учёта групповых расходов. Для расширения возможностей приложения можно добавить возможность разделения расхода не только поровну, но и по процентам, суммам, долям.

Добавить мультивалютность, то есть возможность использования нескольких валют в одной группе. Сделать интеграцию с API курсов валют и пересчитывать суммы расходов в основную валюту группы.

Сделать категоризацию расходов (гостиницы, такси, питание, развлечения, ...).

Сделать дашборд с графиком расходов по категориям, участникам. Средний расход на человека и тому подобное.

Реализовать интеграцию с платёжными системами, например генерировать QR-код для переводов по СБП.

Возможность загружать фотографии чеков к расходам.

Заключение

В ходе выполнения курсовой работы была достигнута поставленная цель – разработано функциональное веб-приложение TripF для управления групповыми расходами.

Основные результаты работы:

1. Разработана архитектура веб-приложения на основе клиент-серверной модели с использованием React, Next.js и SQLite
2. Разработан и протестирован жадный алгоритм минимизации количества транзакций при расчёте взаиморасчётов между участниками группы
3. Создан адаптивный пользовательский интерфейс, работающий на различных устройствах
4. Реализована функциональность Progressive Web App
5. Проведено функциональное тестирование, подтвердившее корректность работы приложения

Дальнейшее развитие системы:

Перспективными направлениями развития системы являются: интеграция с платёжными системами для автоматизации переводов, реализация поддержки множественных валют с автоматической конвертацией, внедрение машинного обучения для категоризации расходов.

Список литературы

- [1] Alex Banks and Eve Porcello. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2nd edition, 2022.
- [2] Gavin M Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In ECOOP 2014–Object-Oriented Programming, pages 257–281. Springer, 2014.
- [3] David Flanagan. JavaScript: The Definitive Guide. O'Reilly Media, 7th edition, 2020.
- [4] Mike Cantelon, Marc Harter, TJ Holowaychuk, and Nathan Rajlich. Node.js in Action. Manning Publications, 2nd edition, 2017.
- [5] Stefan Tilkov, Martin Eigenbrodt, Silvia Schreier, and Oliver Wolf. REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web. dpunkt.verlag, 3rd edition, 2015.
- [6] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"web services: making the right architectural decision. In Proceedings of the 17th international conference on World Wide Web, pages 805–814. ACM, 2008.
- [7] Mike Owens. The Definitive Guide to SQLite. Apress, 2006.
- [8] Abraham Silberschatz, Henry F Korth, and S Sudarshan. Database System Concepts. McGraw-Hill, 6th edition, 2010.
- [9] Roy Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, RFC Editor, June 2014.

Приложение

Ссылка на исходный код

Исходный код приложения доступен по ссылке: <https://github.com/arpefly/tripf>

Листинги кода

Листинг 1: Создание сессии пользователя

```
export function createSession(userId: string): string {
  const sessionId = randomUUID();
  const expiresAt = new Date(
    Date.now() + 30 * 24 * 60 * 60 * 1000
  ).toISOString();
  db.prepare(
    'INSERT INTO sessions (id, user_id, expires_at)
    VALUES (?, ?, ?)'
  ).run(sessionId, userId, expiresAt);

  return sessionId;
}
```

Листинг 2: Проверка актуальности сессии

```
export function getSession(sessionId: string):
  Session | null {
  const session = db.prepare(
    'SELECT * FROM sessions
    WHERE id = ? AND expires_at > datetime('now')'
  ).get(sessionId);

  return session || null;
}
```

Листинг 3: Установка cookie с идентификатором сессии

```
export async function setSessionCookie(  
  sessionId: string  
) : Promise<void> {  
  const cookieStore = await cookies();  
  cookieStore.set('tripf_session', sessionId, {  
    httpOnly: true,  
    sameSite: 'lax',  
    maxAge: 30 * 24 * 60 * 60,  
    path: '/',  
  });  
}
```

Листинг 4: Генерация уникального кода приглашения

```
const CODE_ALPHABET =  
  'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890';  
const CODE_LENGTH = 6;  
  
function generateUniqueCode() {  
  let code: string;  
  do {  
    code = '';  
    for (let i = 0; i < CODE_LENGTH; i++) {  
      const index = Math.floor(  
        Math.random() * CODE_ALPHABET.length  
      );  
      code += CODE_ALPHABET[index];  
    }  
  } while (  
    db.prepare(  
      'SELECT 1 FROM group_invites WHERE code = ?'  
    ).get(code)  
  );  
  return code;  
}
```

```
}
```

Листинг 5: Создание приглашения в группу

```
export function createGroupInvite(  
  groupId: string,  
  createdBy: string,  
  expiresInHours: number = 72  
) : GroupInvite {  
  const inviteId = randomUUID();  
  const token = randomBytes(24).toString('hex');  
  const code = generateUniqueCode();  
  const expiresAt = new Date(  
    Date.now() + expiresInHours * 60 * 60 * 1000  
  );  
  
  db.prepare(  
    'INSERT INTO group_invites  
      (id, group_id, token, code, created_by,  
       expires_at)  
    VALUES (?, ?, ?, ?, ?, ?)'  
  ).run(inviteId, groupId, token, code, createdBy,  
    expiresAt);  
  
  return getInviteById(inviteId);  
}
```

Листинг 6: Структура запроса на создание расхода

```
{  
  "description": "desc",  
  "amount": 0000,  
  "paidBy": "user-uuid",  
  "splitType": "equal",  
  "splits": [  
    { "participantId": "user1-uuid", "amount": 0000 },  
    { "participantId": "user2-uuid", "amount": 0000 },  
  ],  
}
```

```

    ...
]
}

```

Листинг 7: Вставка расхода и разбивки по участникам

```

const expenseId = randomUUID();
const expenseDate = new Date().toISOString();

const insertExpense = db.transaction(() => {
  // Создаём расход
  db.prepare(
    'INSERT INTO expenses
      (id, group_id, description, amount,
       paid_by_user_id, split_type, date)
      VALUES (?, ?, ?, ?, ?, ?, ?)'
  ).run(expenseId, groupId, description, amount,
        paidBy, splitType, expenseDate);

  // Создаём разбивку
  const insertSplit = db.prepare(
    'INSERT INTO expense_splits
      (expense_id, user_id, amount)
      VALUES (?, ?, ?)'
  );

  for (const split of splits) {
    insertSplit.run(
      expenseId,
      split.participantId,
      split.amount
    );
  }
});

insertExpense();

```

Листинг 8: Генерация события о создании расхода

```
emitGroupEvent(groupId, {  
  type: 'expense:created',  
  expenseId  
});
```

Листинг 9: Обработчик создания платежа

```
export async function POST(request, { params }) {  
  const userId = await getCurrentUserId();  
  const { id: groupId } = await params;  
  const { from, to, amount, note } =  
    await request.json();  
  
  // Валидация  
  if (!from || !to || !amount || amount <= 0) {  
    return NextResponse.json(  
      { error: 'Неверные_данные' },  
      { status: 400 }  
    );  
  }  
  
  if (from === to) {  
    return NextResponse.json(  
      { error: 'Нельзя_платить_самому_себе' },  
      { status: 400 }  
    );  
  }  
  
  const paymentId = randomUUID();  
  const createdAt = new Date().toISOString();  
  
  db.prepare(  
    'INSERT INTO settlement_payments  
      (id, group_id, from_user_id, to_user_id,
```

```

        amount, note, created_by, created_at)
VALUES (?, ?, ?, ?, ?, ?, ?, ?)'
).run(paymentId, groupId, from, to, amount,
      note, userId, createdAt);

// Генерируем событие
emitGroupEvent(groupId, {
  type: 'payment:created',
  paymentId
});

return NextResponse.json({
  payment: { id: paymentId, from, to, amount,
            note, createdBy: userId, createdAt }
}, { status: 201 });
}

```

Листинг 10: Простая реализация EventBus

```

type Listener = (payload: unknown) => void;

const listeners =
  new Map<string, Set<Listener>>>();

export function subscribeToGroup(
  groupId: string,
  listener: Listener
): () => void {
  const groupListeners =
    listeners.get(groupId) ?? new Set<Listener>();
  groupListeners.add(listener);
  listeners.set(groupId, groupListeners);

  // Возвращаем функцию отписки
  return () => {
    const current = listeners.get(groupId);

```

```

    if (!current) return;
    current.delete(listener);
    if (current.size === 0) {
        listeners.delete(groupId);
    }
};
}

export function emitGroupEvent(
    groupId: string,
    payload: unknown
): void {
    const groupListeners = listeners.get(groupId);
    if (!groupListeners) return;

    for (const listener of groupListeners) {
        try {
            listener(payload);
        } catch (error) {
            console.error('Event listener error:', error);
        }
    }
}

```

Листинг 11: Подписка компонента GroupView на события группы

```

useEffect(() => {
    const unsubscribe = subscribeToGroup(
        groupId,
        (payload) => {
            // При любом изменении перезагружаем данные
            mutate(`/api/groups/${groupId}`);
            mutate(`/api/groups/${groupId}/expenses`);
            mutate(`/api/groups/${groupId}/payments`);
        }
    );
});

```

```
    return unsubscribe; // Отписка при размонтировании  
}, [groupId]);
```

Скриншоты интерфейса

The screenshot shows a dark-themed web interface for a service called "TripF". At the top left is the logo "TripF" and at the top right is a link "Войти". In the center is a registration form titled "Регистрация". The form contains three input fields: "Имя" (Name) with the value "Арсений", "Email" with the value "arsenii@gmail.com", and "Пароль" (Password) with masked characters "•••••". Below these fields is a blue button labeled "Зарегистрироваться" and a link "Войти" below it.

TripF

Войти

Регистрация

Имя

Арсений

Email

arsenii@gmail.com

Пароль

•••••

Зарегистрироваться

Войти

Рис. 13: Форма регистрации

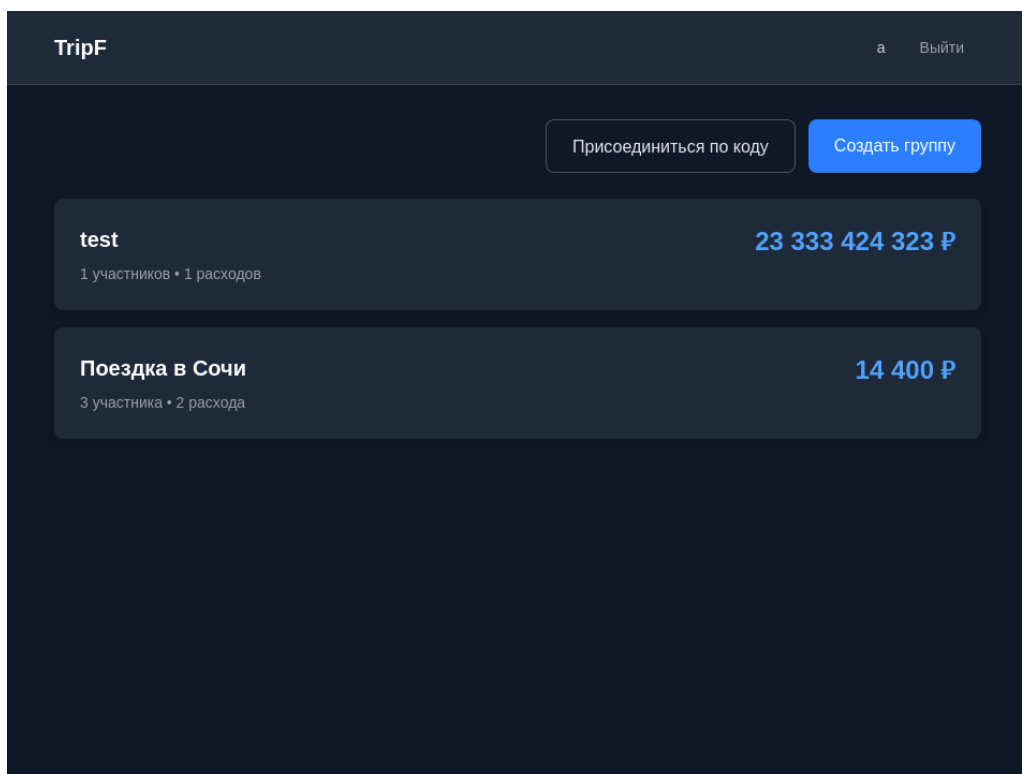


Рис. 14: Главная страница

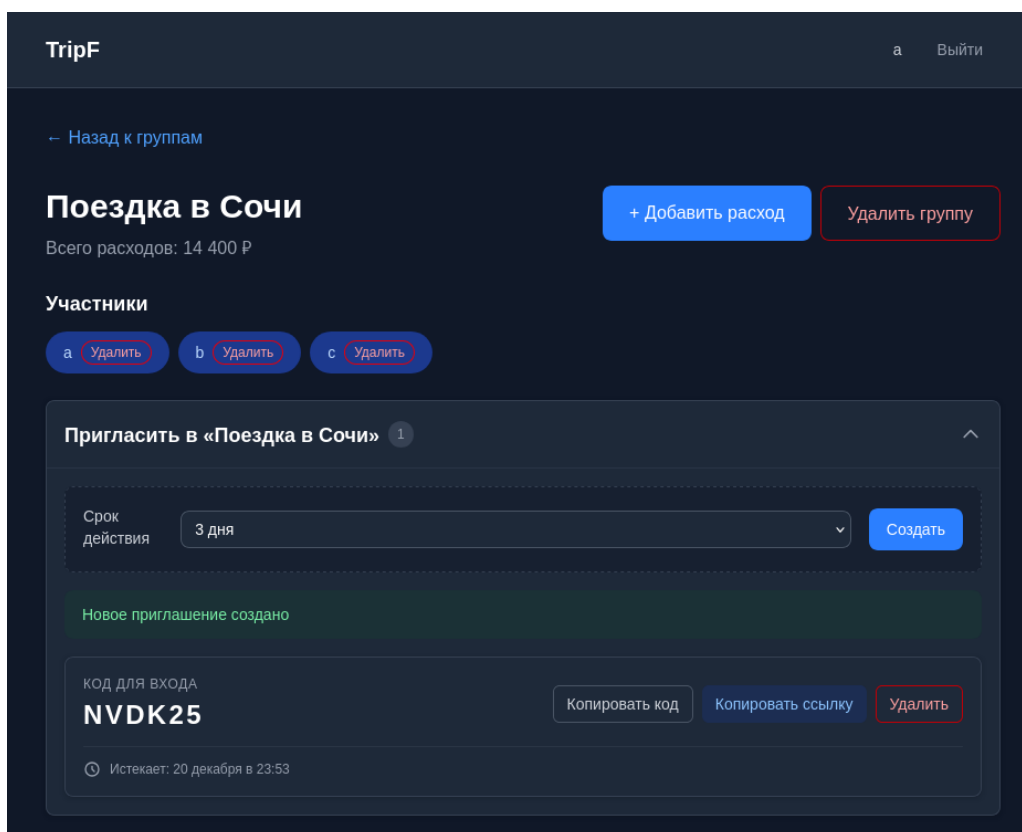


Рис. 15: Панель создания приглашения

TripF a Выйти

[← Назад к группе](#)

Новый расход

Поездка в Сочи

Дата

12/17/2025, 11:48 PM

Описание

Burn

Сумма (₽)

150

Оплатил

a

Способ разделения

Поровну

Участники

a b c

Отмена Добавить

Рис. 16: Форма добавления расхода

Расходы

Жильё

Оплатил: c

12 000 ₽

17 дек. 2025 г.

Разделено между: a (4 000 ₽), b (4 000 ₽), c (4 000 ₽)

Редактировать Удалить

Бенз

Оплатил: b

2 400 ₽

17 дек. 2025 г.

Разделено между: a (800 ₽), b (800 ₽), c (800 ₽)

Редактировать Удалить

Рекомендуемые расчеты

A a 3 800 ₽ должен заплатить должен получить c C

Отметить оплату

B b 2 400 ₽ должен заплатить должен получить c C

Отметить оплату

Платежи

Платеж добавлен

Кто платит

Кто получает

Сумма

Комментарий (необязательно)

0

Например, СБП

Добавить платёж

a → c

17.12.2025, 18:54:56

1 000 ₽

Отменить

Рис. 17: Список расчётов