IDR:

Цели и задачи проекта:

1. Предоставление пользователям возможности вести подробный трекинг расходов и доходов, создавать категории расходов, управлять бюджетом.
2. Реализовать безопасную и отказоустойчивую систему для операций записи и чтения данных.
4. Интегрировать брокер сообщений (Kafka) для асинхронной обработки записей, обеспечивая гарантии доставки событий.
5. Внедрить схему реплицирования БД, чтобы обеспечить отказоустойчивость
6. Организовать контейнеризацию компонентов с помощью Docker и docker-compose для упрощения развертывания.

Основные компоненты системы:

1. Gateway:
    - Роль: Единая точка входа для клиентских запросов. Выполняет предварительную аутентификацию, маршрутизацию запросов и базовое логирование.
    - Технологии: через WEB
2. Write Publisher & WriteService:
    - Write Publisher: Принимает запросы на запись, проводит валидацию и публикует сообщения в Apache Kafka.
    - WriteService: Подписывается на топики Kafka, обрабатывает полученные сообщения, выполняет бизнес-логику и осуществляет запись в основное хранилище.

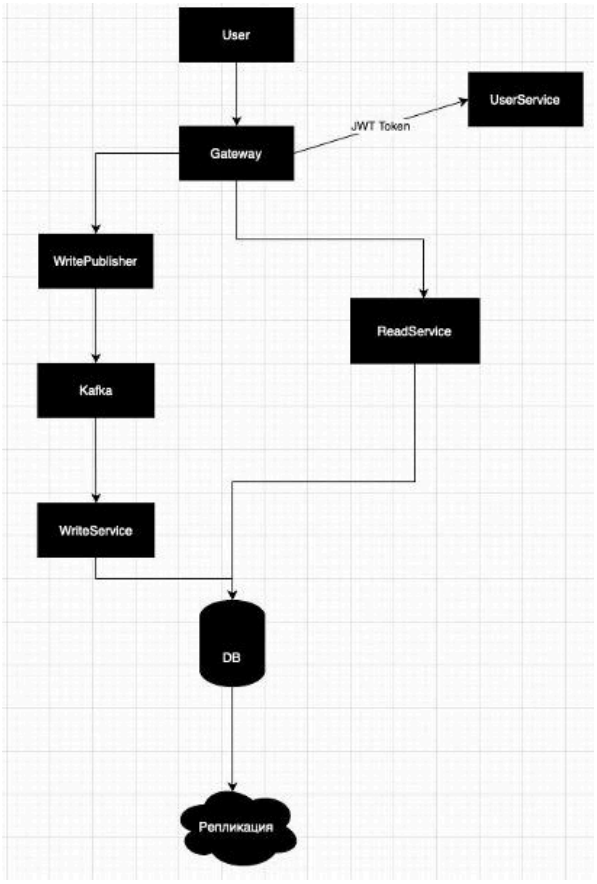
    - Преимущества: Асинхронная обработка транзакций, гарантированная доставка сообщений, масштабируемость.
3. ReadService:
    - Роль: получает запросы, ходит в бд и возвращается с ответом.
4. База данных:
    - Хранилище: Основное хранилище финансовых транзакций пользователей.
    - Особенности: Поддержка реплицирования.
5. Kafka:
    - Роль: Обеспечение асинхронной передачи сообщений между компонентами для записи транзакций.
    - Преимущества: Высокая пропускная способность и отказоустойчивость.
6. Контейнеризация и оркестрация:
    - Контейнеризация: Использование Docker для упаковки и изоляции микросервисов.
    - Инструменты: Файл `docker-compose.yml` (и Dockerfile для каждого сервиса) предоставляет возможность локального развёртывания и тестирования.
    - Оркестрация: Возможное использование Kubernetes для масштабирования и управления контейнерами на продакшен-среде (в будущем).

Сценарии взаимодействия компонентов

1. Запись данных (Write flow):
    - Клиент отправляет запрос через API Gateway.
    - Запрос направляется к Write Publisher, где происходит валидация JWT – токена и публикация сообщения в Apache Kafka.
    - WriteService получает сообщение из Kafka, осуществляет необходимую обработку и сохраняет данные в базу данных.
2. Чтение данных (Read flow):
    - Клиент делает запрос через Gateway
    - Gateway обращается в ReadService, валидирует токен и получает userId
    - Потом ReadService получает данные из БД и через Gateway возвращает пользователю

3. Gateway Контейнеризация:
    - Разработка и тестирование проходят локально с использованием docker-compose.

Схема:



Архитектурные решения (ADR)
Ключевые решения:

- ADR-1: Выбор Apache Kafka как брокера сообщений
  Обоснование: Kafka обеспечивает высокую пропускную способность, масштабируемость и гарантированную доставку сообщений, что критично для обработки большого количества транзакций.
- ADR-2: Реплицирование базы данных
  Обоснование: Шардирование позволяет равномерно распределять нагрузку по базе данных, что особенно важно при росте числа пользователей.

- ADR-3: Использование API Gateway как центральной точки входа
  Обоснование: Централизация маршрутизации запросов упрощает аутентификацию, логирование и управление трафиком.
- ADR-4: Контейнеризация с Docker и использование docker-compose
  Обоснование: Применение Docker упрощает развёртывание и тестирование сервисов, а docker-compose позволяет быстро запускать всю систему локально или в тестовой среде.

Функциональные требования

1. Авторизация и регистрация
   - Пользователь может зарегистрироваться, указав:
   Email (валидный формат, уникальность)
   Пароль (минимум 9 символов)
   - Пользователь может войти в систему, указав email и пароль.
   - При ошибках ввода (неверный email/пароль) система отображает сообщение об ошибке.
   - Пользователь может выйти из системы.

2. Создание чека
   - Пользователь может создать чек, указав:
     Название (обязательное поле)
     Сумму (число > 0, обязательное поле)
     Категорию (выбор из списка или создание новой)
     Дату (по умолчанию текущая)
   - После успешного создания чека система сохраняет его и отображает в списке.
   - При ошибке (например, не заполнено обязательное поле) система отображает сообщение.

3. Просмотр списка чеков
   - Пользователь видит список всех своих чеков по категории.
   - В списке отображается:
     Название чека
     Сумма
     Дата
4. Управление категориями
   - Пользователь видит список всех своих категорий.
   - Пользователь может нажать на категорию, чтобы увидеть все чеки в ней.

5. Обработка ошибок и загрузка

- При загрузке данных (например, списка чеков) отображается индикатор загрузки.
- При ошибке (нет сети, сервер недоступен) отображается сообщение с возможностью повторить запрос.

6. Дополнительные требования
   - Валидация полей (например, сумма должна быть числом).

Нефункциональные требования

Прочность (Нагрузочная устойчивость):

- Система должна выдерживать одновременную работу до 1000 пользователей без деградации производительности.
- При пиковой нагрузке (до 2000 запросов в минуту) время отклика не должно превышать 2 секунды.

Количество пользователей:

- Приложение должно поддерживать не менее 50 000 зарегистрированных пользователей.
- Система должна масштабироваться горизонтально при увеличении числа пользователей.

Скорость отклика (Производительность)

- Время отклика системы на стандартные запросы не должно превышать 0.2 секунды в 95% случаев.

Надёжность (Отказоустойчивость и доступность)

- Система должна обеспечивать доступность 99,9% (uptime) в течении тестового периода.
- В случае сбоя восстановление работы должно занимать не более 15 минут.

Производительность:

- Время отклика: Среднее время ответа для операций чтения должно составлять менее 100–200 мс.
- Пропускная способность: система обрабатывает 500–1000 транзакций в секунду

Масштабирование

- Горизонтальное масштабирование: Все микросервисы (Gateway, WriteService, ReadService) готовы к масштабированию через запуск дополнительных контейнеров.
- Репликация базы данных: Распределение данных по `userId` позволяет добавлять реплики по мере необходимости.

Надёжность и отказоустойчивость

- Кластеризация Kafka и резервирование: Настроены механизмы отказоустойчивости через кластеризацию Apache Kafka.
- Контейнеризация и оркестрация: Использование Docker (и потенциально Kubernetes для продакшн-среды) обеспечивает быстрый перезапуск и отказоустойчивость сервисов.

Безопасность

- Аутентификация и авторизация: Реализованы механизмы проверки JWT-токенами для управления доступом к сервисам.

Анализ рисков и меры смягчения

Основные риски

- Перегрузка Kafka или базы данных: Резкое увеличение числа транзакций может привести к снижению производительности.

Меры по снижению рисков:

- Шардирование вместо репликации.