Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Курсовая работа

по дисциплине “Конструирование программного обеспечения ”

Выполнил

Студент гр. 5130904/20103 Ильченко А. А.

Руководитель Иванов А. С.

Санкт-Петербург
2025 г.

Оглавление

Задание

Задача нашего курса сделать проект и пройти большинство классических стадий создания ПО, поэтому в меньшей степени важна тема курсовика и в большей степени работоспособность и оформление. Хорошим проектом для курсовика является приложение, где есть база данных, серверная часть, внешняя зависимость и UI. Такой набор компонент нужен, чтобы архитектурные схемы, которые вы будете рисовать в рамках курсовика не были вырожденными. Перечисленные компоненты - это ограничение снизу, если вы хотите использовать в своем проекте больше компонентов, например, брокер сообщений, балансировщик и т.д., то вы можете это сделать. Язык для написания серверно части - любой База данных - PostgreSQL, MySQL (если хотите использовать что-то другое, согласуйте свой выбор с преподавателем) UI - Web, Mobile, Telegram (UI через бота), TUI (terminal UI) Внешняя зависимость - обычно здесь используется какой-то внешний поставщик данных, можно использовать список открытых API, можно использовать локальные API политеха, например, расписание занятий, можно использовать локальные для РФ источники публичной информации, например, API Росреестра, ГАР (ФИАС)

Определение проблемы

В наше время людям все сложнее следить за своими тратами. Из-за преимущественно безналичного расчета пропала осязаемость денег, а подписка на любимый онлайн сервис списывается даже без непосредственного участия пользователя. Так же большой объем информаций, который необходимо удерживать в голове для рабочих или же учебных целей часто вытесняется данные о стоимости чашке кофе, булочке или аренде самоката. Говоря про сервисы аренды, они же, в свою очередь, только способствуют неразберихе путем списания «залога», засоряя статью расходов в приложении банка. Как итог вышеперечисленного люди могут встретить «утечку» денег и дыры в бюджете.

Выработка требований

Для ращения проблемы предлагается разработать Web-приложение:” Трекер финансов”.

Я иметь возможность самостоятельно прописать трату за сегодня чтобы уделить внимание даже, на первый взгляд, незначительными вещам.
Так же я хочу видеть список всех трат за сегодня с общей суммой трат чтобы отслеживать глобальное количество потраченных денег. Важно получать диаграмму которая бы отражала куда ушли потраченные средства, например, позволяла бы заметит, что больше половины бюджета уходит на развлечения.
И в конце необходимо получаться актуальную сводку курса валют для приведения разного рода трат к одному общему знаменателю.

Предполагается, что приложением будут пользоваться не меньше 10000 человек, данные (траты) которых будут храниться не меньше 5 лет.

Разработка архитектуры и детальное проектирование

Характер нагрузки на сервис

Соотношение R/W нагрузки

Тип операции	Описание	Оценка %
Чтение (Read)	Получение списка трат, конвертация валют	~70%
Запись (Write)	Добавление трат, регистрация пользователей, удаление трат	~30%

Основная часть нагрузки — это получение уже сохранённых данных (например, просмотр списка трат, загрузка диаграмм и конвертации), а запись происходит сравнительно реже.

Оценка объёмов трафика (на начальном этапе)

Параметр	Значение (прибл.)
Среднее число активных пользователей в сутки	10000
Количество транзакций на пользователя в день	100
Количество обращений к API в сутки	1000
Средний размер запроса	~1-2 KB
Суточный трафик	~2 GB
Суточный дисковый прирост	~500 MB

Объемы дисковой системы

Компонент	Объем
Пользователи	~1-2 KB на пользователя
Траты	~1 KB на каждую трату
Пример для 1 года и 10000 пользователей	~5-10 GB

База данных остаётся компактной, так как хранит только текстовые и числовые данные.

Диаграммы C4 Модели

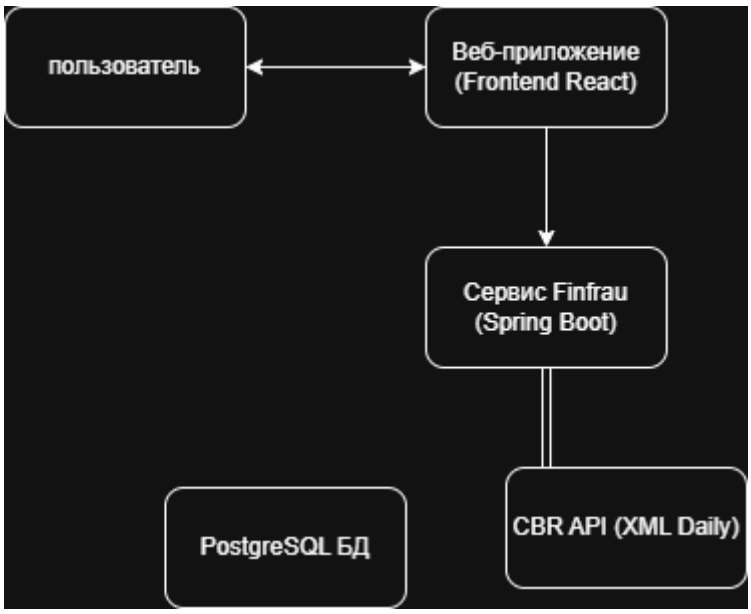

Рисунок 1 Диграмма C4 level 1

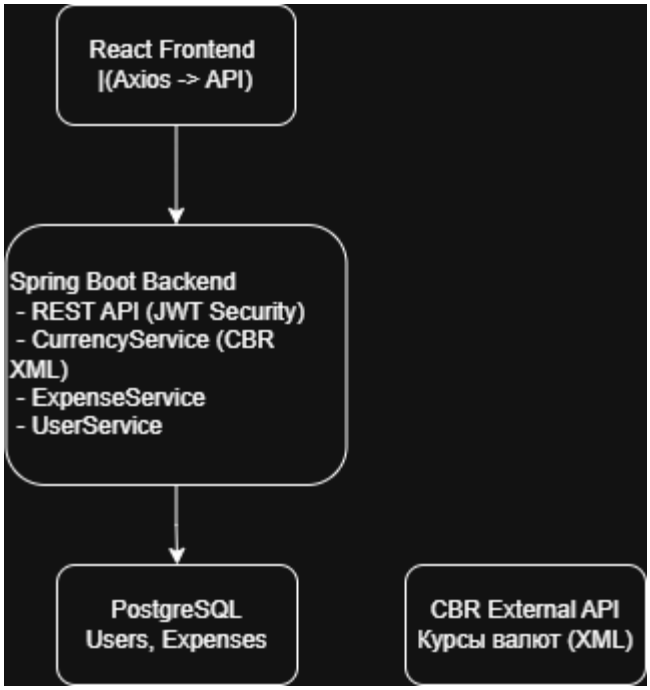

Рисунок 2 Диграмма C4 level 2

Контракты API (REST)

Регистрация пользователя:

- POST /register
- Request: { "username": "user", "password": "pass" }
- Response: 200 OK или 409 Conflict

Логин:

- POST /login

- Request: { "username": "user", "password": "pass" }
- Response: token (string)

Работа с тратами:

- POST /expenses (добавить трату)
- GET /expenses (список трат)
- DELETE /expenses/{id} (удалить трату)

Конвертация валюты:

- POST /cnv
- Request: { "from": "USD", "to": "RUB", "amount": 100 }
- Response: { converted amount }

Нефункциональные требования

Параметр	Требование
Время отклика API	< 500 мс для чтения, < 1 сек для записи
Надёжность	99.9% uptime
Масштабируемость	Горизонтальная
Безопасность	JWT, HTTPS, пароль в bcrypt

Обновление курсов валют 1 раз в день (при первом запросе в день)

Схема базы данных

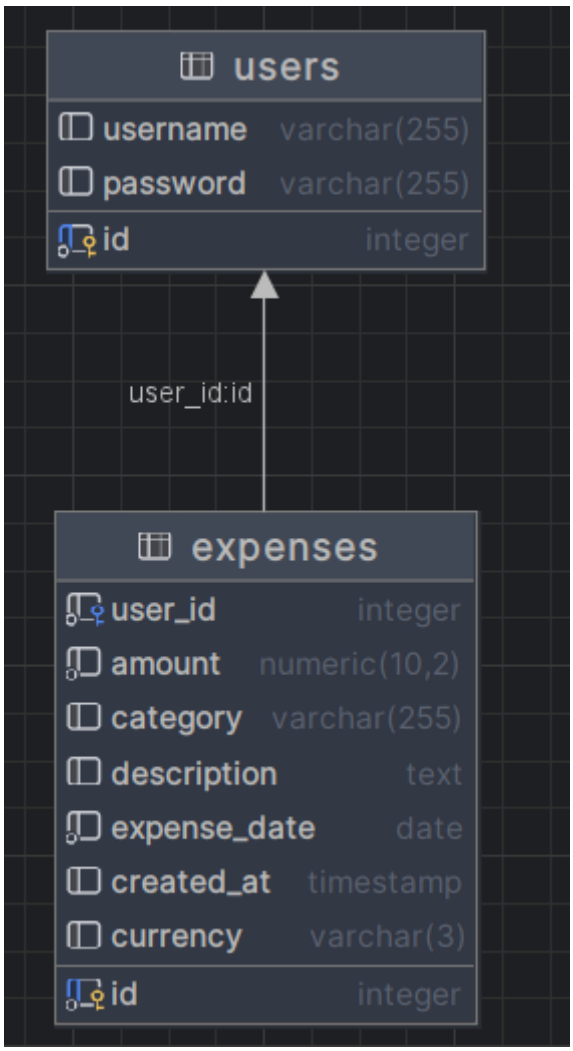

Рисунок 3 схема базы данных

Почему выдержит нагрузку:

- Все индексы по user_id, expense_date позволяют быстро фильтровать и агрегировать данные.
- Данных немного (никаких больших текстов, бинарных файлов, BLOB).
- PostgreSQL отлично масштабируется вертикально и горизонтально.

Масштабирование при росте нагрузки в 10 раз

Компонент	*Изменения*
Бэкенд	Несколько инстансов Spring Boot (Kubernetes / Docker Swarm)
База	Перенос на управляемый кластер PostgreSQL (например, AWS RDS)
Фронтенд	Статическое хранение на CDN (например, Cloudflare, S3)
Логирование	Централизованное логирование (например, ELK Stack)
Мониторинг	Prometheus + Grafana
Балансировка	Nginx + Load Balancer

Кодирование и отладка

Код проекта в github репозитории: https://github.com/eshsupman/financefrau

Unit тестирование

Тестирование обмена валют

Для тестирования обмена валют был применен Mocking и выработано 4 теста.
В коде тесты расположены в следующем порядке:

1. обычный тест конвертации (testConvert_withMockedRates). Ожидается успешная конвертации по заданному курсу
2. конвертация одинаковой валюты (RUB -> RUB) (testConvert_sameCurrency). Ожидается отсутствие конвертации, количество денег не изменится
3. конвертация неизвестной валюты (testConvert_unknownCurrency). Ожидается отсутствие конвертации, количество денег не изменится.
4. тест кэширования (testCachingWorks). Ожидается отсутствие запроса к API ЦБ РФ в случае, если котировки на сегодняшний лень уже есть

Тестирование JWT

Для тестирования системы json web token были разработаны следующие тесты

1. генерация токена (testGenerateAndExtractToken). Ожидается генерация токена с подписью, именем пользователя и сроком годности 24 часа.
2. Валидация токена (testValidateToken_Valid). Ожидается успешная валидация токена.
3. Валидация неисправного токена (testValidateToken_Invalid). Ожидается ошибка валидации.

Тестирование UserDetails

Следующая группа тестов была разработана для проверки метолов отвечающих за извлечение информации о пользователе.

1. Получить информацию по имени пользователя (testLoadUserByUsername_UserFound). Ожидается нахоД пользователя и получение зашифрованного пароля и id.
2. Получить информацию о несуществующем пользователе (testLoadUserByUsername_UserNotFound). Ожидается ошибка поиска пользователя.

Интеграционное тестирование

В рамках интеграционного теста был протестирован следующий сценарий:
создание нового пользователя -> авторизация -> добавление траты -> получение списка трат за сегодня.

Тест: ExpenseIntegrationTest

Сборка

Для сборки и запуска тестов запустить run_all.sh