Домашняя Бухгалтерия — План Проекта

Цель: Научиться fullstack-разработке на реальном проекте, который будешь использовать сам.

Что получится в итоге: Веб-приложение, куда ты загружаешь скриншот из банковского приложения → AI распознаёт сумму и описание → сохраняет в базу → показывает отчёты по тратам.

Фаза 0: Подготовка (1 день)

Что делаем

- Устанавливаем всё необходимое
- Проверяем что всё работает

Инструменты

Что	Зачем
Python 3.11+	Бекенд (серверная часть)
Node.js 20+	Фронтенд (то что видит пользователь)
VS Code или IntelliJ IDEA	Редактор кода
Git	Контроль версий (история изменений)
Docker	Контейнеры (изолированные среды)
Claude Code	AI-помощник для написания кода

Проверка

```bash
python --version    # должно быть 3.11+
node --version      # должно быть 20+
git --version       # любая версия
docker --version    # любая версия
```

Фаза 1: Проектирование (2-3 дня)

> **Главное правило:** Сначала думаем, потом пишем код. Это из книги — "define errors out of existence".

Шаг 1.1: Описание функций (что делает приложение)

Напиши простым языком что должно уметь приложение. Например:

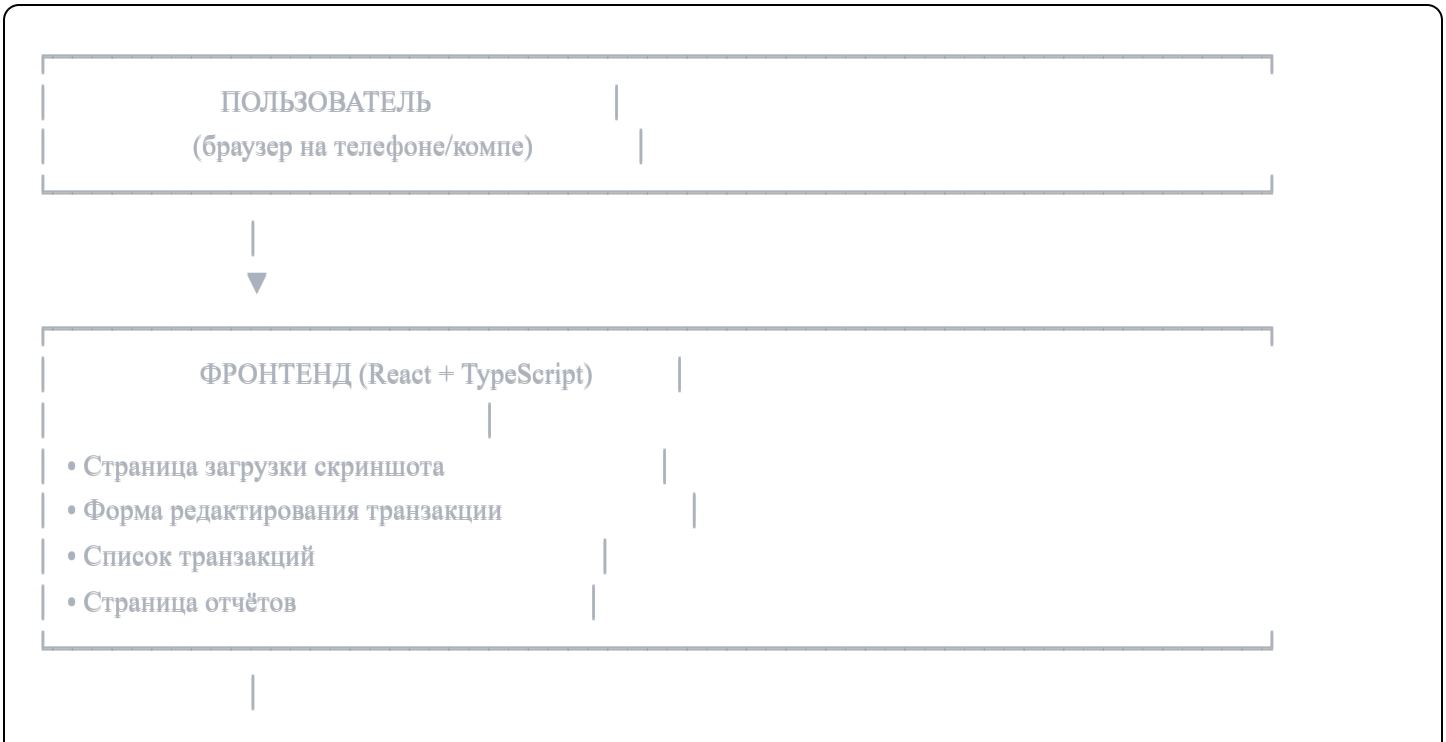
Основные функции (MVP — минимальный продукт):

1. Загрузить скриншот из банковского приложения
2. AI распознаёт: сумму, описание, дату
3. Пользователь может поправить если AI ошибся
4. Сохранить транзакцию в базу
5. Показать список всех транзакций
6. Показать сумму трат за месяц

Дополнительные функции (потом):

- Категории (еда, транспорт, развлечения)
- Автоматическое определение категории
- Графики трат по месяцам
- Бюджеты и лимиты
- Напоминания

Шаг 1.2: Архитектура (как устроено внутри)



Шаг 1.3: Модель данных (что хранится в базе)

```
Таблица: transactions
├── id          — уникальный номер (автоматически)
├── amount      — сумма (число с копейками)
├── description — описание ("Пятёрочка", "Яндекс.Такси")
├── category    — категория (пока можно null)
├── date        — дата транзакции
├── image_path  — путь к сохранённому скриншоту
├── raw_text    — что распознал AI (для отладки)
├── created_at  — когда добавлено в базу
└── updated_at  — когда последний раз менялось
```

Шаг 1.4: API endpoints (точки входа)

Метод	URL	Что делает
POST	`/api/upload`	Загрузить скриншот, получить распознанные данные
POST	`/api/transactions`	Создать транзакцию
GET	`/api/transactions`	Получить список транзакций
GET	`/api/transactions/{id}`	Получить одну транзакцию
PUT	`/api/transactions/{id}`	Обновить транзакцию
DELETE	`/api/transactions/{id}`	Удалить транзакцию
GET	`/api/reports/monthly`	Отчёт по месяцам

Фаза 2: MVP Бекенд (1 неделя)

Шаг 2.1: Структура проекта

```
home-finance/
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py          # точка входа FastAPI
│   │   ├── config.py        # настройки
│   │   ├── database.py      # подключение к БД
│   │   ├── models.py        # модели данных (ORM)
│   │   ├── schemas.py       # схемы для API
│   │   ├── routers/
│   │   │   ├── transactions.py
│   │   │   └── upload.py
│   │   └── services/
│   │       └── ocr_service.py  # работа с AI
│   ├── requirements.txt
│   └── Dockerfile
├── frontend/
│   └── (потом)
├── docker-compose.yml
└── README.md
```

Шаг 2.2: Задачи по порядку

День 1-2: Базовый API без AI

1. Создать проект FastAPI
2. Подключить PostgreSQL через SQLAlchemy
3. Сделать CRUD для транзакций (создать/читать/обновить/удалить)
4. Проверить через Swagger UI (автоматически на `/docs`)

День 3-4: Добавляем AI

1. Подключить Anthropic API (Claude)
2. Написать сервис распознавания изображений
3. Сделать endpoint загрузки скриншотов
4. Протестировать на реальных скриншотах

День 5-7: Docker и тесты

1. Написать Dockerfile для бекенда
2. Написать docker-compose.yml (бекенд + postgres)
3. Написать базовые тесты
4. Проверить что всё поднимается одной командой

Шаг 2.3: Технологии бекенда

Что	Зачем	Почему именно это
FastAPI	Веб-фреймворк	Быстрый, современный, автодокументация
SQLAlchemy	Работа с БД	Стандарт в Python, не пишем SQL руками
Pydantic	Валидация данных	Встроен в FastAPI, проверяет что данные правильные
PostgreSQL	База данных	Надёжная, бесплатная, стандарт индустрии
httpx	HTTP клиент	Для запросов к Claude API

Фаза 3: MVP Фронтенд (1 неделя)

Шаг 3.1: Структура фронтенда

```
frontend/
├── src/
│   ├── components/
│   │   ├── UploadForm.tsx      # форма загрузки
│   │   ├── TransactionList.tsx # список транзакций
│   │   ├── TransactionCard.tsx # карточка одной транзакции
│   │   └── MonthlyReport.tsx   # отчёт за месяц
│   ├── pages/
│   │   ├── HomePage.tsx
│   │   ├── UploadPage.tsx
│   │   └── ReportsPage.tsx
│   ├── api/
│   │   └── client.ts           # запросы к бекенду
│   ├── types/
│   │   └── index.ts            # типы TypeScript
│   ├── App.tsx
│   └── main.tsx
├── package.json
├── tsconfig.json
├── vite.config.ts
└── Dockerfile
```

Шаг 3.2: Задачи по порядку

День 1-2: Базовая структура

1. Создать проект через Vite + React + TypeScript
2. Настроить роутинг (react-router)
3. Сделать базовый layout (шапка, навигация)

День 3-4: Основные страницы

1. Страница загрузки скриншота
2. Форма редактирования распознанных данных
3. Список транзакций

День 5-7: Отчёты и стили

1. Страница отчётов (сумма за месяц)

2. Базовые стили (можно TailwindCSS)
3. Адаптив для мобилок
4. Docker для фронтенда

Шаг 3.3: Технологии фронтенда

Что	Зачем
React	Библиотека для UI
TypeScript	Типизация (меньше багов)
Vite	Сборщик (быстрый)
TailwindCSS	Стили (быстро и удобно)
React Query	Кеширование запросов

Фаза 4: Интеграция и деплой (3-5 дней)

Шаг 4.1: Всё вместе

1. Обновить docker-compose: фронт + бек + БД
2. Настроить nginx как reverse proxy
3. Проверить что всё работает локально

Шаг 4.2: Деплой (опционально)

- Можно задеплоить на VPS (DigitalOcean, Hetzner)
- Или запускать локально — для учёбы достаточно

Фаза 5: Улучшения (по желанию)

После MVP можно добавлять:

1. **Автокатегоризация** — AI сам определяет категорию
2. **Графики** — визуализация трат (Chart.js или Recharts)
3. **Бюджеты** — установить лимит на категорию, получать алерты

4. **Регулярные траты** — автоматически добавлять аренду/подписки
5. **Экспорт** — выгрузка в Excel/CSV
6. **Мультивалютность** — если есть траты в разных валютах
7. **PWA** — чтобы работало как приложение на телефоне

Словарь терминов

Термин	Что это
API	Интерфейс для общения между программами
Endpoint	Конкретный URL в API
CRUD	Create, Read, Update, Delete — базовые операции
ORM	Библиотека для работы с БД через объекты
Docker	Программа для запуска приложений в контейнерах
Контейнер	Изолированная среда со всеми зависимостями
MVP	Minimum Viable Product — минимальная рабочая версия
Фронтенд	То что видит пользователь (браузер)
Бекенд	Серверная часть (логика, БД)
TypeScript	JavaScript с типами
FastAPI	Python-фреймворк для создания API

Чек-лист готовности к следующей фазе

После Фазы 1 (Проектирование)

- [ ] Понимаю что делает каждый компонент
- [ ] Понимаю как данные текут от пользователя до базы
- [ ] Могу объяснить зачем нужен каждый endpoint

После Фазы 2 (Бекенд)

- [ ] API работает, могу создать/получить транзакцию через Swagger
- [ ] AI распознаёт скриншоты
- [ ] Docker поднимает всё одной командой

После Фазы 3 (Фронтенд)

- [ ] Могу загрузить скриншот через интерфейс
- [ ] Вижу список своих транзакций
- [ ] Вижу сумму трат за месяц

После Фазы 4 (Интеграция)

- [ ] Всё работает вместе
- [ ] Могу пользоваться приложением каждый день

Полезные ресурсы

- FastAPI документация
- React документация
- TypeScript Handbook
- Docker Get Started
- A Philosophy of Software Design — книга которую уже читаешь

Последнее обновление: Февраль 2026