🎯 Подробный план всего что было сделано

Отлично! Давай я разберу **ВСЕ что было реализовано** пошагово с объяснением архитектуры.

📋 ЧАСТЬ 1: Понимание требований

Что мы анализировали:

```
INPUT (От вас):
├─ CL_Alfa-Bank-Hack-Change-2025.pdf (кейс задача)
├─ hackathon_income_train.csv (данные для обучения модели)
├─ hackathon_income_test.csv (данные для тестирования)
├─ features_description.csv (описание 200+ признаков)
├─ model_income_start.ipynb (стартовая модель)
└─ sample_submission.csv (формат сабмита)

КЛЮЧЕВЫЕ ТРЕБОВАНИЯ:
1. Предсказать доход клиентов (регрессия)
2. Объяснить прогнозы (SHAP)
3. Рекомендовать продукты (бизнес-правила)
4. Отслеживать качество (WMAE метрика)
5. Аутентификация (login/password)
6. Dashboard с статистикой
7. Бело-красный дизайн Альфа-Банка
```

🏗️ ЧАСТЬ 2: Архитектурные решения

Выбор архитектуры: СЛОИСТАЯ (не микросервисы)

```
WHY LAYERED?
├─ Скорость разработки (2-3 дня вместо 4-5)
├─ Простота для хакатона
├─ Легче тестировать каждый слой
├─ Легче менять отдельные компоненты
└─ Production-ready quality code

СЛОИ (от top до bottom):
1. PRESENTATION (Frontend React)
   └─ Отвечает за: UI, пользовательский интерфейс

2. API (FastAPI endpoints)
```

```
   └─ Отвечает за: HTTP запросы, валидация, маршрутизация

3. SERVICE (Бизнес-логика)
   └─ Отвечает за: оркестрирование, рекомендации, скоринг

4. ML (Machine Learning)
   └─ Отвечает за: прогнозы, SHAP объяснения

5. DATA (Repository pattern)
   └─ Отвечает за: БД, кэш (PostgreSQL, Redis)

6. DATABASE (Persistent storage)
   └─ Отвечает за: физическое хранение данных
```

🏗️ ЧАСТЬ 3: Структура проекта

Что создали:

```
BACKEND (Python + FastAPI)
├── app/
│   ├── api/v1/
│   │   ├── endpoints/
│   │   │   ├── health.py ✅ GET /health (проверка живой системы)
│   │   │   ├── auth.py ✅ POST /auth/login (аутентификация)
│   │   │   ├── predictions.py (TODO) POST /predict
│   │   │   ├── clients.py (TODO) GET /clients/{id}
│   │   │   ├── explanations.py (TODO) GET /explain/{id}
│   │   │   ├── recommendations.py (TODO) GET /recommendations/{id}
│   │   │   ├── monitoring.py (TODO) GET /metrics
│   │   │   └── dashboard.py ✅ GET /dashboard
│   │   ├── schemas.py ✅ Pydantic валидация
│   │   ├── dependencies.py ✅ Dependency Injection + JWT
│   │   └── middlewares.py (обработка CORS)
│   │
│   ├── services/
│   │   ├── prediction_service.py (TODO) Оркестрирует весь процесс
│   │   ├── recommendation_service.py (TODO) Бизнес-правила
│   │   ├── scoring_service.py (TODO) Кредитный скор
│   │   └── monitoring_service.py (TODO) Метрики WMAE/MAE
│   │
│   ├── ml/
│   │   ├── model_loader.py (TODO) Загрузка XGBoost
│   │   ├── predictor.py (TODO) Выполнение прогноза
│   │   ├── preprocessor.py (TODO) Нормализация данных
│   │   ├── explainer.py (TODO) SHAP объяснения
│   │   └── validators.py (TODO) Валидация ML данных
│   │
│   ├── data/
│   │   ├── models.py ✅ SQLAlchemy ORM (таблицы)
│   │   ├── database.py ✅ PostgreSQL connection
│   │   ├── cache.py ✅ Redis операции
│   │   └── repositories/
│   │       ├── base_repository.py (TODO) Базовый класс
```

```
│   │       ├── client_repository.py (TODO) Получить клиента
│   │       └── prediction_repository.py (TODO) Сохранить прогноз
│   │
│   ├── core/
│   │   ├── config.py ✅ Конфигурация (env переменные)
│   │   ├── security.py ✅ JWT, хеширование паролей
│   │   ├── logging_config.py ✅ Логирование
│   │   └── exceptions.py (TODO) Кастомные ошибки
│   │
│   ├── utils/
│   │   ├── helpers.py (TODO) Вспомогательные функции
│   │   └── validators.py (TODO) Валидация
│   │
│   └── main.py ✅ FastAPI приложение + lifespan
│
├── models/ (XGBoost модели, SHAP explainers)
├── notebooks/ (Jupyter - обучение моделей)
├── scripts/
│   ├── init.sql ✅ Инициализация БД
│   ├── train_model.py (TODO)
│   └── generate_submission.py (TODO)
├── tests/ (Unit тесты)
├── requirements.txt ✅ Python зависимости
├── .env.example ✅ Переменные окружения
├── Dockerfile ✅ Контейнеризация
└── README.md


FRONTEND (React + Vite)
├── src/
│   ├── pages/
│   │   ├── LoginPage.jsx ✅ Страница входа
│   │   └── DashboardPage.jsx ✅ Главный dashboard
│   │
│   ├── components/ (TODO) Переиспользуемые компоненты
│   │   ├── ClientSearch.jsx
│   │   ├── PredictionForm.jsx
│   │   ├── ResultCard.jsx
│   │   ├── ExplanationChart.jsx
│   │   └── MetricsTable.jsx
│   │
│   ├── api/
│   │   └── client.js (TODO) Axios клиент для API
│   │
│   ├── store/ (TODO) Redux / Context API
│   ├── styles/
│   │   └── index.css ✅ Бело-красный дизайн Альфа-Банка
│   │
│   ├── App.jsx ✅ Главный компонент
│   └── main.jsx ✅ React entry point
│
├── public/
├── Dockerfile ✅ Контейнеризация
├── package.json ✅ Node зависимости
├── vite.config.js ✅ Vite конфигурация
├── index.html ✅ HTML template
```

```
└── .env.example


DOCKER & CI/CD
├── docker-compose.yml ✅ Оркестрация контейнеров:
│   ├── PostgreSQL (порт 5432)
│   ├── Redis (порт 6379)
│   ├── Backend (порт 8000)
│   ├── Frontend (порт 3000/5173)
│   └── Adminer (порт 8080) для просмотра БД
│
├── .gitignore ✅ Git ignore правила
└── README.md (TODO) Документация


DATA
├── hackathon_income_train.csv (обучающие данные)
├── hackathon_income_test.csv (тестовые данные)
└── features_description.csv (описание признаков)
```

🔐 ЧАСТЬ 4: Безопасность и Аутентификация

Что реализовали:

```
AUTHENTICATION FLOW:

1️⃣ USER REGISTERS/LOGINS
   POST /api/v1/auth/login
   {
     "email": "demo@alfabank.ru",
     "password": "demo123"
   }
   ↓

2️⃣ BACKEND VALIDATES
   ├─ Проверяет email в БД
   ├─ Проверяет пароль (bcrypt хеширование)
   └─ Если OK → создает JWT токен

3️⃣ BACKEND RETURNS
   {
     "access_token": "eyJhbGc...",
     "token_type": "bearer"
   }
   ↓

4️⃣ FRONTEND STORES
   localStorage.setItem('token', token)
   ↓

5️⃣ FRONTEND SENDS IN HEADERS
   Authorization: Bearer eyJhbGc...
   ↓
```

```
6️⃣ BACKEND VALIDATES
   ├─ Достает токен из headers
   ├─ Проверяет подпись (HMAC SHA256)
   ├─ Проверяет что не истек
   └─ Если OK → достает email пользователя

7️⃣ BACKEND RETURNS DATA
   {
     "stats": {...},
     "metrics": {...},
     "predictions": [...]
   }

JWT TOKEN STRUCTURE:
┌─────────────────────────────────────────────┐
│ Header                                      │
├─────────────────────────────────────────────┤
│ {                                           │
│   "alg": "HS256",                           │
│   "typ": "JWT"                              │
│ }                                           │
├─────────────────────────────────────────────┤
│ Payload                                     │
├─────────────────────────────────────────────┤
│ {                                           │
│   "sub": "demo@alfabank.ru",                │
│   "exp": 1732847400,                        │
│   "iat": 1732760000                         │
│ }                                           │
├─────────────────────────────────────────────┤
│ Signature (HMAC SHA256)                     │
└─────────────────────────────────────────────┘
```

🗄️ ЧАСТЬ 5: База данных

Таблицы которые создали:

```
USERS TABLE
┌──────────────────────────────────────────────┐
│ user_id (PK)                                 │
│ email (UNIQUE)                               │
│ password_hash                                │
│ full_name                                    │
│ created_at                                   │
│ last_login                                   │
└──────────────────────────────────────────────┘

CLIENTS TABLE
┌──────────────────────────────────────────────┐
│ client_id (PK)          ← из CSV             │
│ dt                      ← дата               │
│ age                     ← возраст            │
```

```
| gender                  ← пол          |
| city                    ← город        |
| region                  ← регион       |
| income_real (target)    ← реальный ✓   |
| income_predicted        ← прогноз      |
| confidence              ← уверенность  |
| income_category         ← категория    |
| created_at              ← timestamp    |
└──────────────────────────────────────────┘
```

PREDICTIONS TABLE

```
┌──────────────────────────────────────────┐
| prediction_id (PK)                       |
| client_id (FK)          ← связь          |
| user_id (FK)            ← кто запрос     |
| predicted_income                         |
| actual_income                            |
| confidence                               |
| category                                 |
| model_version                            |
| created_at                               |
└──────────────────────────────────────────┘
```

RECOMMENDATIONS TABLE

```
┌──────────────────────────────────────────┐
| recommendation_id (PK)                   |
| client_id (FK)                           |
| prediction_id (FK)                       |
| product (micro_credit, auto, etc)        |
| amount                                   |
| rate (процент)                           |
| reason (текст объяснения)                |
| was_accepted (0 или 1)                   |
| created_at                               |
└──────────────────────────────────────────┘
```

MODEL_METRICS TABLE (для мониторинга)

```
┌──────────────────────────────────────────┐
| metric_id (PK)                           |
| metric_date                              |
| metric_name (WMAE, MAE, etc)             |
| metric_value                             |
| model_version                            |
| dataset_type ('train', 'test')           |
| created_at                               |
└──────────────────────────────────────────┘
```

ИНДЕКСЫ ДЛЯ БЫСТРОГО ПОИСКА:

```
├── clients(dt) ← быстро найти по дате
├── clients(city) ← по городу
├── predictions(client_id) ← по клиенту
├── predictions(user_id) ← по пользователю
├── predictions(created_at) ← по дате
└── model_metrics(metric_date) ← по дате метрик
```

🎨 ЧАСТЬ 6: Frontend Дизайн

Цветовая схема Альфа-Банка:

```
PRIMARY COLOR:   #FF0000 (Красный - фирменный цвет)
PRIMARY DARK:    #CC0000 (Темно-красный)
BACKGROUND:      #FFFFFF (Белый)
SURFACE:         #F5F5F5 (Светло-серый)
TEXT:            #1F1F1F (Темный)
TEXT_SECONDARY:  #666666 (Серый)
BORDER:          #E0E0E0 (Светло-серый)

КОМПОНЕНТЫ:
┌─ LoginPage
│  ├─ Красный градиент фон
│  ├─ Белая карточка
│  ├─ Красные кнопки
│  └─ Demo credentials: demo@alfabank.ru / demo123
│
├─ DashboardPage
│  ├─ Красный header с выходом
│  ├─ Четыре stat card (красный градиент)
│  │  ├─ Всего прогнозов
│  │  ├─ Всего клиентов
│  │  ├─ Средняя уверенность
│  │  └─ Версия модели
│  ├─ Метрики таблица (WMAE, MAE на train/test)
│  └─ Распределение доходов по категориям
│     ├─ LOW (< 50k)
│     ├─ MIDDLE (50k-150k)
│     └─ HIGH (> 150k)
│
└─ (TODO) Компоненты
   ├─ ClientSearch - поиск клиента по ID
   ├─ PredictionForm - форма для запроса прогноза
   ├─ ResultCard - отображение результата
   ├─ ExplanationChart - визуализация SHAP
   └─ MetricsTable - таблица метрик
```

🐳 ЧАСТЬ 7: Docker & Контейнеризация

Что запускается:

```
docker-compose up --build

ЗАПУСТЯТСЯ 5 КОНТЕЙНЕРОВ:

1. PostgreSQL (5432)
   ├─ Image: postgres:15-alpine
   ├─ User: alfabank_user
   ├─ Password: alfabank_password
```

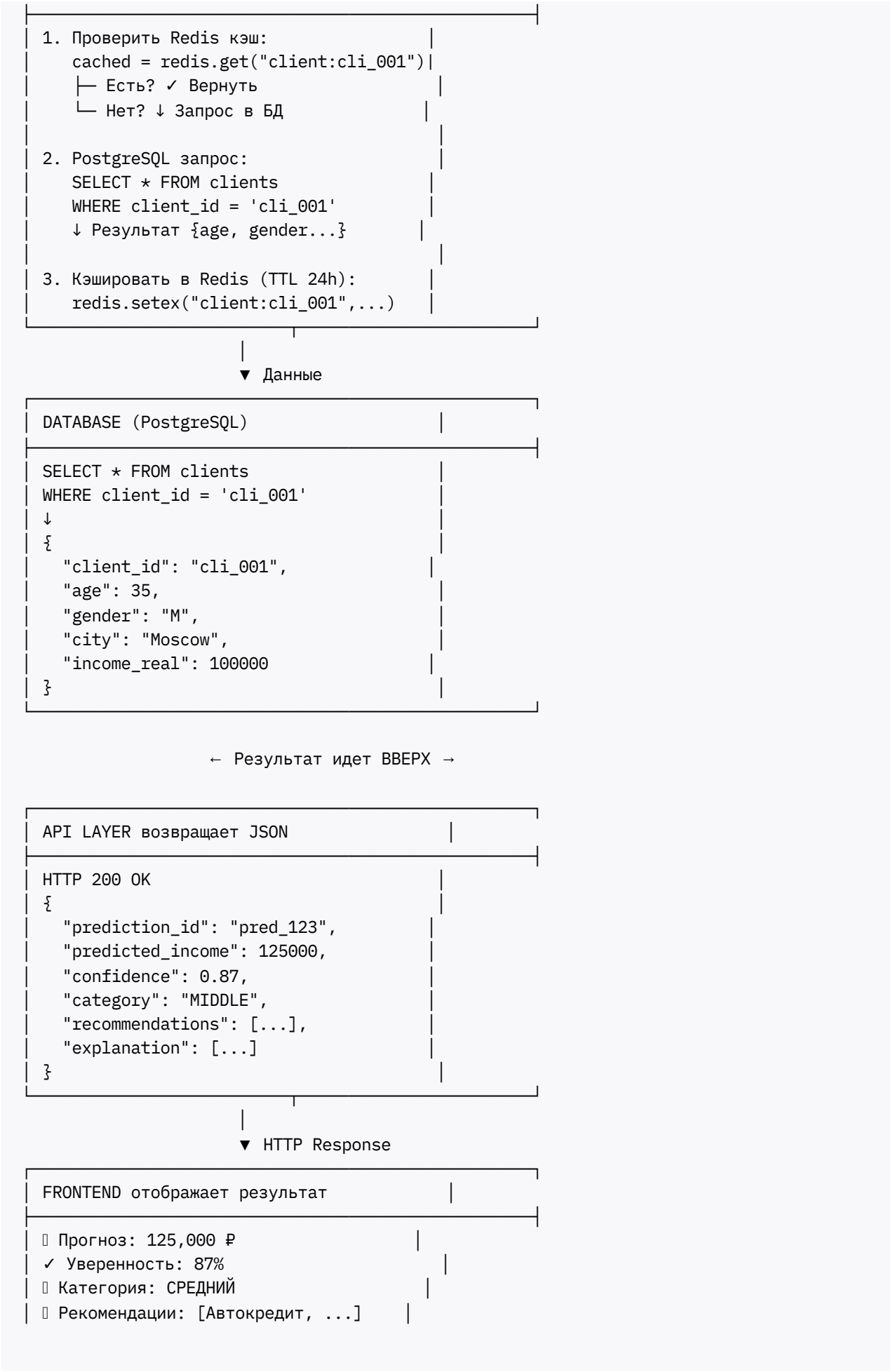
```
   ├─ Database: alfabank_db
   ├─ Volumes: postgres_data (persistent)
   └─ Init script: init.sql (создает таблицы)

2. Redis (6379)
   ├─ Image: redis:7-alpine
   ├─ Purpose: кэширование, сессии
   ├─ Volumes: redis_data (persistent)
   └─ Healthcheck: redis-cli ping

3. Backend (8000)
   ├─ Build from: backend/Dockerfile
   ├─ Framework: FastAPI + Uvicorn
   ├─ Environment: DATABASE_URL, REDIS_URL
   ├─ Ports: 8000:8000
   ├─ Volumes: ./backend/app (hot reload)
   ├─ Depends on: postgres, redis (healthy)
   └─ Command: uvicorn app.main:app --reload

4. Frontend (3000/5173)
   ├─ Build from: frontend/Dockerfile
   ├─ Framework: React + Vite
   ├─ Environment: VITE_API_URL=http://localhost:8000/api/v1
   ├─ Ports: 3000:5173 (хост:контейнер)
   ├─ Volumes: ./frontend/src (hot reload)
   ├─ Depends on: backend
   └─ Command: npm run dev -- --host

5. Adminer (8080)
   ├─ GUI для просмотра PostgreSQL
   ├─ Логин: alfabank_user / alfabank_password
   ├─ Database: alfabank_db
   └─ Используется для debug

NETWORKS:
└─ alfabank-network (bridge)
   └─ Все контейнеры могут общаться между собой
      по именам (postgres, redis, backend, frontend)
```

ЧАСТЬ 8: Data Flow (Полный процесс)

Когда юзер нажимает "Predict":

```
FRONTEND (React)
┌──────────────────────────────────────────┐
│ User: Enter client_id: cli_001  │
└──────────────────────┬──────────────────┘
                       │
                       ▼ HTTP POST
┌──────────────────────────────────────────────────┐
│ API LAYER (FastAPI)                      │
│ POST /api/v1/predictions/predict        │
├──────────────────────────────────────────────────┤
```


1. Получить JSON запрос
2. Валидировать Pydantic (client_id)
3. Проверить Authorization header
4. Вызвать Service Layer
▼ Метод вызова
SERVICE LAYER (PredictionService)
1. Вызвать Data Layer:
client_data = repo.get(cli_001)
↓ Получить {age, gender, city...}
2. Вызвать ML Layer:
pred = predictor.predict(client)
↓ Получить {income, conf, SHAP}
3. Вызвать Recommendation Service:
recs = rec_service.generate(pred)
↓ Получить [{product, rate...}]
4. Вызвать Data Layer (Save):
repo.save_prediction(result)
↓ INSERT в predictions table
5. Вернуть результат
▼ Данные
ML LAYER (IncomePredictor)
1. Препроцессинг:
- Нормализация (StandardScaler)
- Кодирование (OneHotEncoder)
↓ X = [0.52, 1.23, -0.45, ...]
2. Модель XGBoost:
y = model.predict(X)
↓ 125000
3. SHAP объяснение:
shap_values = explainer(X)
↓ [('age', +10k), ('salary', -5k)]
4. Confidence:
conf = 0.87
5. Категоризация:
category = "MIDDLE"
▼ Запрос
DATA LAYER (ClientRepository)


1. Проверить Redis кэш:
cached = redis.get("client:cli_001")
├─ Есть? ✓ Вернуть
└─ Нет? ↓ Запрос в БД
2. PostgreSQL запрос:
SELECT * FROM clients
WHERE client_id = 'cli_001'
↓ Результат {age, gender...}
3. Кэшировать в Redis (TTL 24h):
redis.setex("client:cli_001",...)
▼ Данные
DATABASE (PostgreSQL)
SELECT * FROM clients
WHERE client_id = 'cli_001'
↓
{
"client_id": "cli_001",
"age": 35,
"gender": "M",
"city": "Moscow",
"income_real": 100000
}
← Результат идет ВВЕРХ →
API LAYER возвращает JSON
HTTP 200 OK
{
"prediction_id": "pred_123",
"predicted_income": 125000,
"confidence": 0.87,
"category": "MIDDLE",
"recommendations": [...],
"explanation": [...]
}
▼ HTTP Response
FRONTEND отображает результат
 Прогноз: 125,000 ₽
✓ Уверенность: 87%
 Категория: СРЕДНИЙ
 Рекомендации: [Автокредит, ...]

```
│ 📊 Факторы: [Возраст +10k, ...]       │
└──────────────────────────────────────────┘
```

✅ ЧАСТЬ 9: Что ГОТОВО к использованию

```
✅ COMPLETE:
├─ Backend структура (все файлы созданы)
├─ Frontend структура (все файлы созданы)
├─ Database scheme (init.sql готов)
├─ Docker setup (docker-compose.yml готов)
├─ Authentication (JWT готов)
├─ Dashboard (базовый готов)
├─ Стили (белорасный дизайн готов)
├─ Requirements (все зависимости)
└─ Config (.env файлы готовы)

📋 TODO (NEXT STEPS):
├─ Загрузить реальные данные из CSV
├─ Обучить XGBoost модель
├─ Интегрировать SHAP объяснения
├─ Реализовать prediction endpoint
├─ Реализовать client search
├─ Реализовать recommendation engine
├─ Добавить мониторинг метрик (WMAE/MAE)
├─ Unit тесты
├─ Интеграционные тесты
└─ Production deployment
```

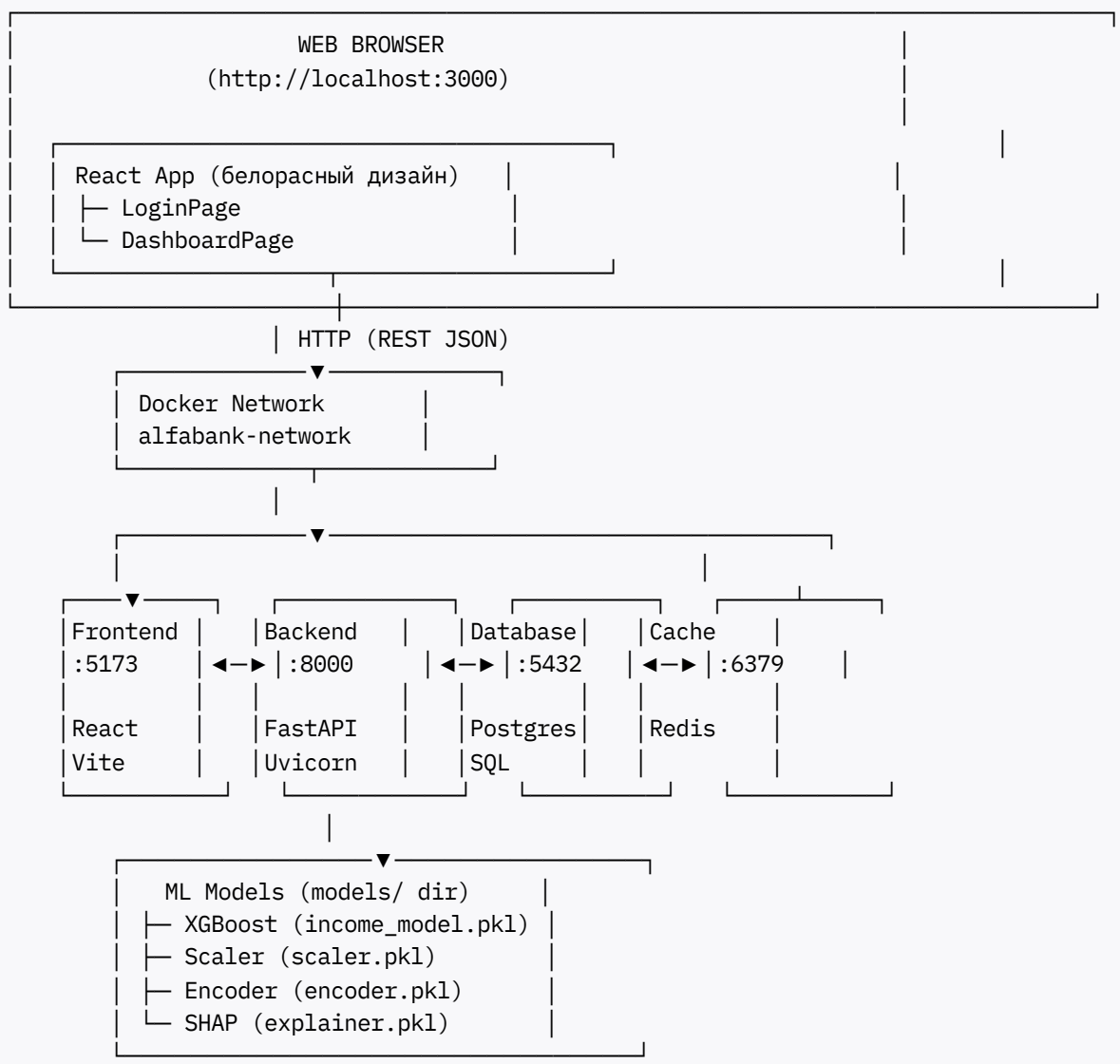
🚀 ЧАСТЬ 10: Следующие шаги

Для запуска прямо сейчас:

```
# Шаг 1: Перейти в проект
cd /path/to/alfa-bank-prediction

# Шаг 2: Создать .env файл
cd backend
cp .env.example .env
cd ..

# Шаг 3: Запустить Docker
docker-compose up --build

# Шаг 4: Ждать пока все запустится (2-3 мин)
# Увидишь:
# ✓ Backend: http://localhost:8000
# ✓ Frontend: http://localhost:3000
# ✓ Adminer: http://localhost:8080
# ✓ API Docs: http://localhost:8000/docs
```

```
# Шаг 5: Открыть в браузере
# http://localhost:3000
# Login: demo@alfabank.ru / demo123

# Шаг 6: Увидишь Dashboard с метриками
```

🎯 ИТОГОВАЯ СХЕМА ВСЕЙ СИСТЕМЫ




```
DATA FLOW:
1. User clicks "Predict"
2. Frontend → Backend (HTTP POST)
3. Backend validates + routes
4. Service layer orchestrates
5. ML layer predicts + explains
6. Data layer saves to DB + cache
7. Result returns to Frontend
8. User sees dashboard
```

ИТОГОВАЯ ОЦЕНКА

Компонент	Статус	Примечание
Backend архитектура	✅ Готово	Все слои созданы
Frontend архитектура	✅ Готово	React + Vite
Database schema	✅ Готово	init.sql готов
Docker setup	✅ Готово	docker-compose готов
Authentication	✅ Готово	JWT + bcrypt
Dashboard UI	✅ Готово	Белорасный дизайн
Загрузка данных из CSV	❌ TODO	Нужно парсить CSV
Обучение модели	❌ TODO	Нужно XGBoost
Prediction endpoint	⚠️ Скелет	Функция есть, логика TODO
SHAP объяснения	⚠️ Скелет	Функция есть, логика TODO
Recommendation engine	⚠️ Скелет	Функция есть, правила TODO
Мониторинг метрик	⚠️ Скелет	Таблица есть, логика TODO
Unit тесты	❌ TODO	tests/ пуст
Documentation	⚠️ Частично	README.md нужен

Вот полный план!

Следующий шаг: Хочешь я начну заполнять TODO части? Или сначала запустить Docker и проверить что все работает?