**Сервис для записи на фотосъемку: автоматизированная платформа для клиентов и фотографов**

**Ключевые возможности**

✅ **Для клиентов**:

* Запись через **веб**, **мобильное приложение** или **Telegram-бота**
* Личный кабинет с историей бронирований
* Push-уведомления и напоминания
* Чат с фотографом

✅ **Для фотографов и студий**:

* Управление расписанием через **веб, мобильное приложение или Telegram**
* Интеграция с календарями (Google Calendar)
* Автоматические уведомления о новых заявках
* CRM с аналитикой загрузки и доходов

✅ **Автоматизация процессов**:

* Подтверждение бронирований без участия менеджера
* Оповещения клиентов и фотографов
* Интеграция с платежными системами

**1. Компоненты системы**

| **Компонент** | **Технологии** | **Описание** |
| --- | --- | --- |
| **API Gateway** | Golang (Gin/Echo) + gRPC | Единая точка входа, маршрутизация запросов, аутентификация |
| **Booking Service** | Golang (PostgreSQL) | Обработка записей, проверка доступности слотов |
| **Schedule Service** | Golang (Redis + PostgreSQL) | Управление расписанием фотографов, кэширование свободных слотов |
| **Notification Service** | Golang (Kafka + FCM/Telegram) | Отправка email, SMS, push-уведомлений |
| **Chat Service** | Golang (WebSocket + MongoDB) | Реалтайм-чат между клиентом и фотографом |
| **Payment Service** | Golang (Stripe/YooMoney SDK) | Обработка платежей, вебхуки |
| **CRM Dashboard** | React.js + Java (Spring Boot) | Аналитика, управление заявками, отчеты |
| **Mobile App** | React Native (iOS/Android) | Запись, уведомления, чат |
| **Telegram Bot** | Python/Node.js | Альтернативный интерфейс для записи и управления |

**2. Технологический стек**

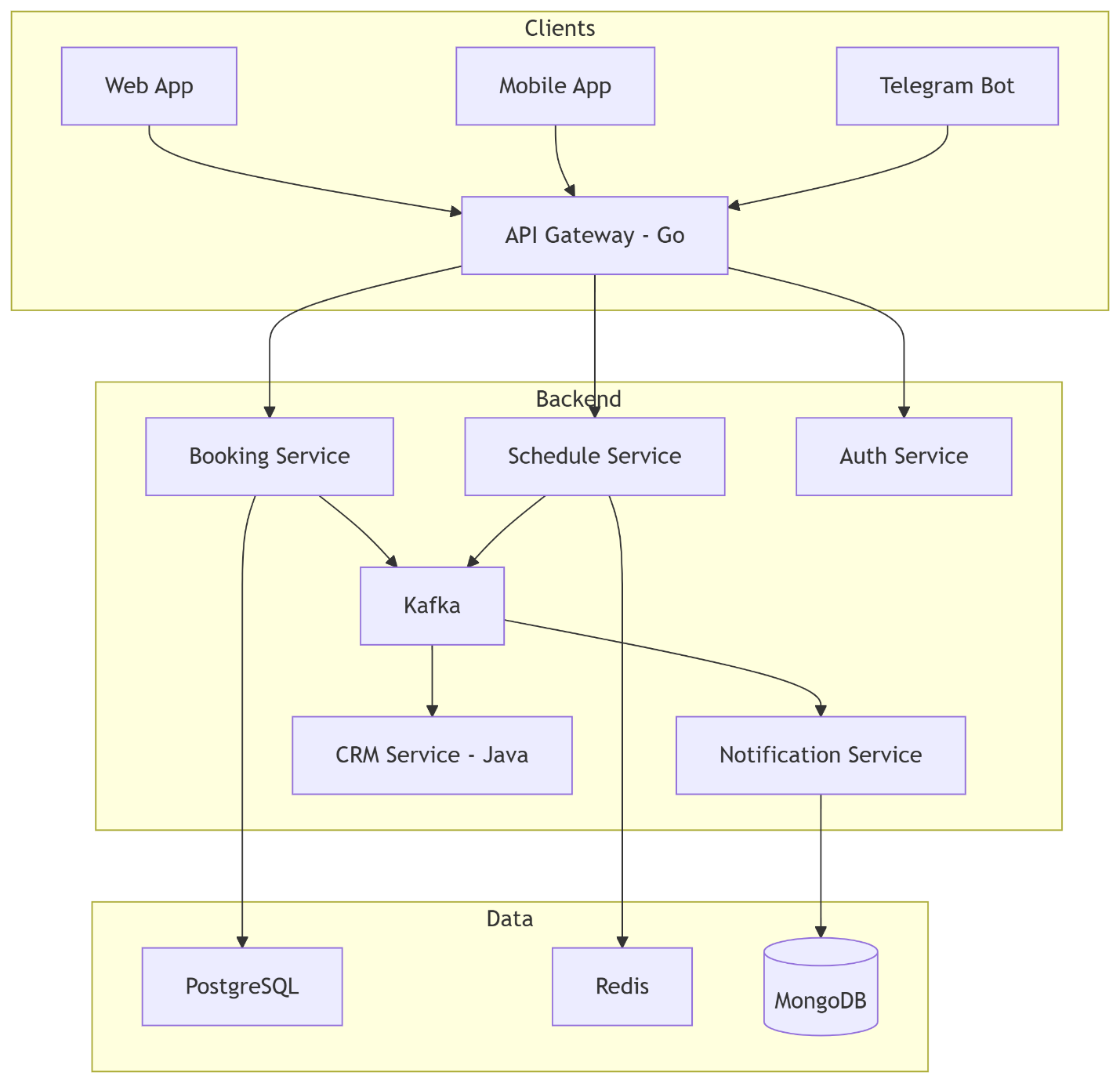
| **Категория** | **Технологии** |
| --- | --- |
| **Backend** | Golang (Gin, Echo, gRPC), Java (Spring Boot для CRM) |
| **Frontend** | React.js (веб), React Native (мобильное приложение) |
| **Базы данных** | PostgreSQL (основные данные), MongoDB (чат), Redis (кэш) |
| **Брокер сообщений** | Apache Kafka (асинхронные события) |
| **DevOps** | Docker, Kubernetes, GitHub Actions (CI/CD) |
| **Уведомления** | Firebase Cloud Messaging (FCM), Telegram Bot API, SendGrid (email) |
| **Платежи** | Stripe, YooMoney |
| **Мониторинг** | Prometheus + Grafana |

**3. Преимущества выбранных технологий**

| **Технология** | **Преимущества** |
| --- | --- |
| **Golang** | 🔹 Высокая производительность (горутины вместо потоков) 🔹 Минимальное потребление памяти 🔹 Простота развертывания (статический бинарник) |
| **Kafka** | 🔹 Масштабируемая обработка событий 🔹 Гарантированная доставка сообщений 🔹 Поддержка потоковой обработки |
| **PostgreSQL** | 🔹 Надежность и ACID-совместимость 🔹 Поддержка JSON и сложных запросов 🔹 Репликация и шардинг |
| **React Native** | 🔹 Кроссплатформенность (iOS + Android) 🔹 Готовые решения для push-уведомлений 🔹 Быстрая разработка |
| **Docker/K8s** | 🔹 Автоматическое масштабирование 🔹 Минимальные простои при обновлениях 🔹 Изолированность сервисов |

**4. Схемы**

**4.1. Компонентная схема**



flowchart TD

subgraph Clients

Web[Web App] --> API

Mobile[Mobile App] --> API

Telegram[Telegram Bot] --> API

end

subgraph Backend

API[API Gateway - Go] --> Booking[Booking Service]

API --> Schedule[Schedule Service]

API --> Auth[Auth Service]

Booking --> Kafka

Schedule --> Kafka

Kafka --> Notify[Notification Service]

Kafka --> CRM[CRM Service - Java]

end

subgraph Data

Booking --> PostgreSQL

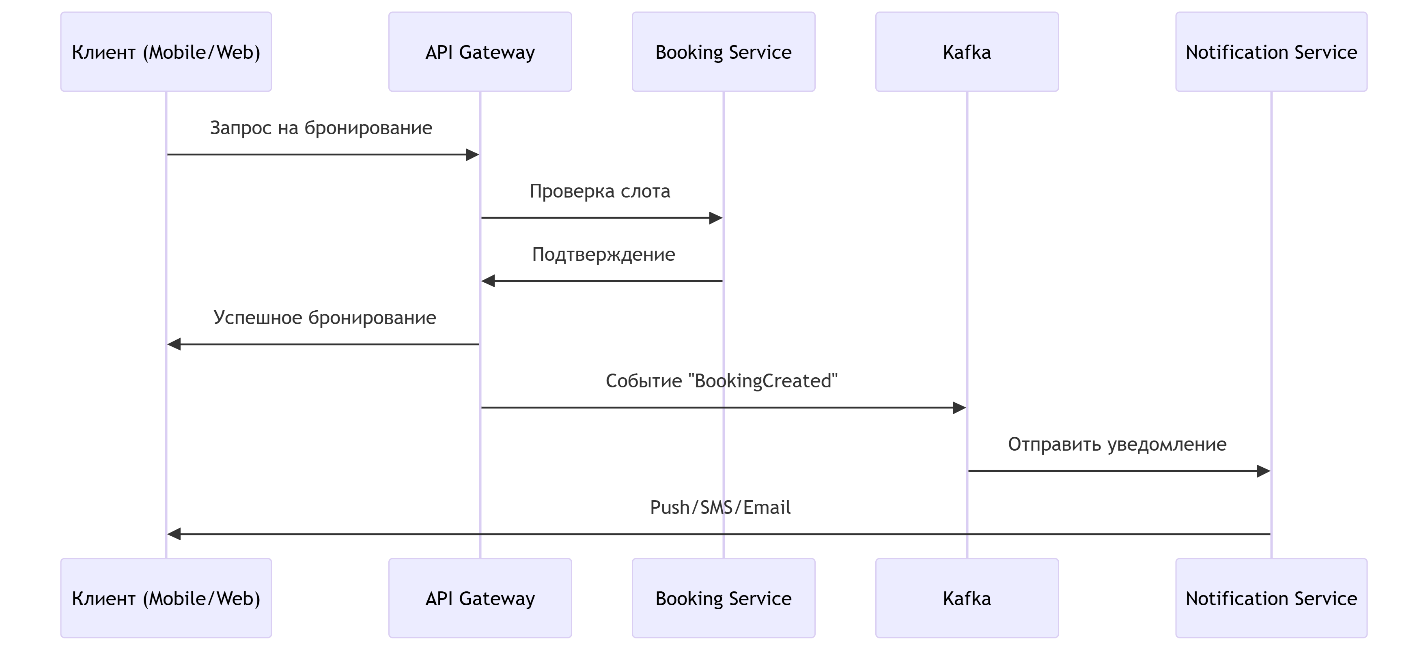
Schedule --> Redis

Notify --> MongoDB[(MongoDB)]

end

**4.2. Схема последовательности (запись клиента)**

Diagram



sequenceDiagram

participant Client as Клиент (Mobile/Web)

participant API as API Gateway

participant Booking as Booking Service

participant Kafka as Kafka

participant Notify as Notification Service

Client->>API: Запрос на бронирование

API->>Booking: Проверка слота

Booking->>API: Подтверждение

API->>Client: Успешное бронирование

API->>Kafka: Событие "BookingCreated"

Kafka->>Notify: Отправить уведомление

Notify->>Client: Push/SMS/Email

**4.3. Информационная модель БД (PostgreSQL)**

Diagram

erDiagram

users ||--o{ bookings : "имеет"

photographers ||--o{ schedule\_slots : "создает"

studios ||--o{ schedule\_slots : "предоставляет"

bookings }|--|| schedule\_slots : "использует"

users {

uuid id PK

string email

string role

}

photographers {

uuid id PK

uuid user\_id FK

}

studios {

uuid id PK

string location

}

schedule\_slots {

uuid id PK

timestamp start\_time

timestamp end\_time

uuid photographer\_id FK

uuid studio\_id FK

}

bookings {

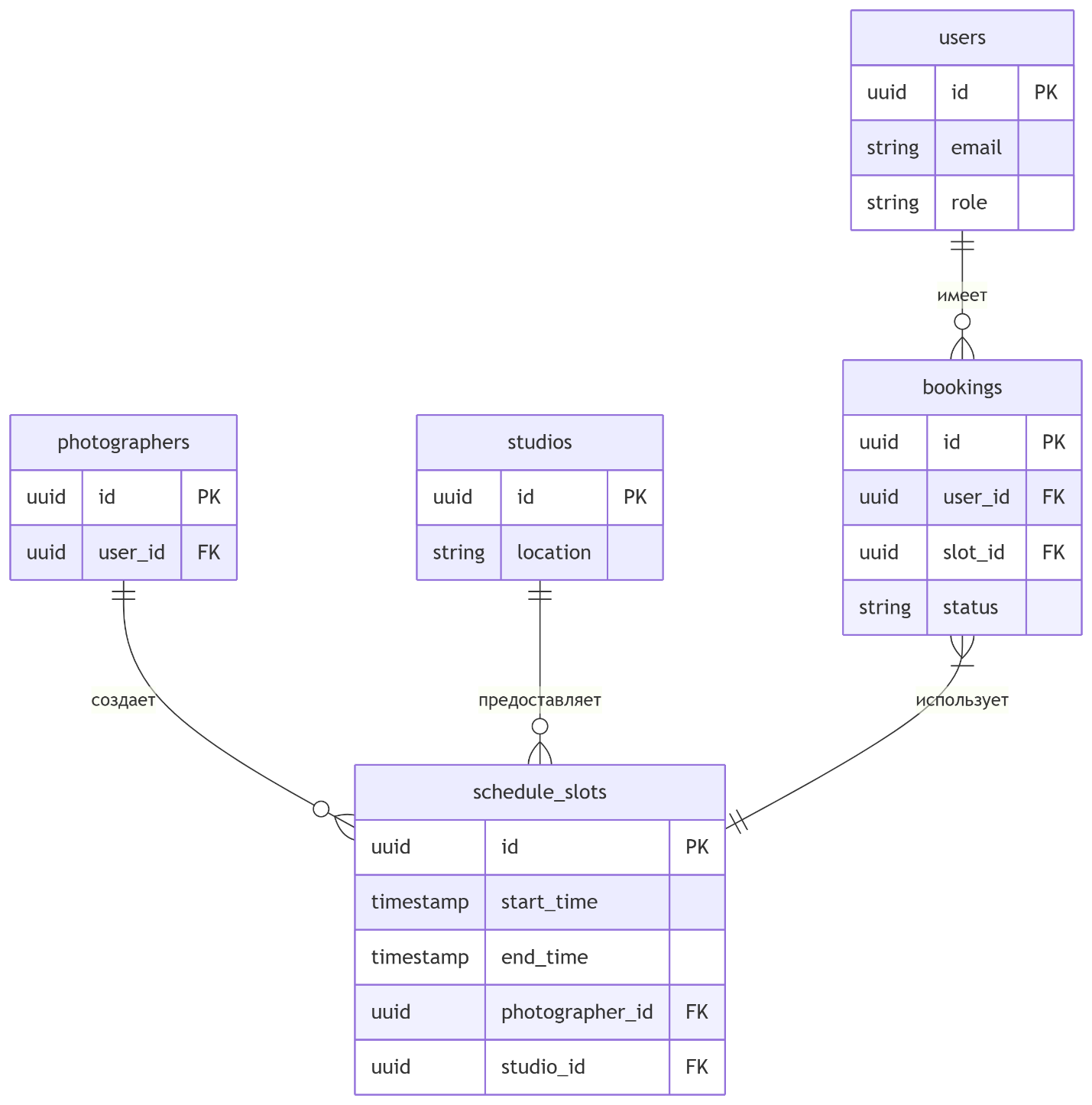
uuid id PK

uuid user\_id FK

uuid slot\_id FK

string status

}



**5. Автоматизация (минимизация участия разработчиков)**

🔹 **CI/CD (GitHub Actions)**

* Автоматические тесты и деплой при пуше в main
* Rollback при ошибках

🔹 **Мониторинг (Prometheus + Grafana)**

* Автооповещение о проблемах (500 ошибки, высокая загрузка CPU)

🔹 **Self-Service для фотографов**

* Настройка расписания без участия менеджера

🔹 **Автоматические уведомления**

* Напоминания клиентам за 24/1 час до съемки
* Уведомления фотографам о новых заявках

🔹 **Платежи**

* Автоподтверждение после успешной оплаты

**Итог**

Сервис полностью автоматизирует процесс записи, минимизируя ручное управление.  
**Golang** обеспечивает высокую производительность, **Kafka** — надежную обработку событий, **React Native** — кроссплатформенность.  
Система готова к масштабированию и требует вмешательства разработчиков только при критических изменениях.

**Поэтапный запуск сервиса для записи на фотосъемку**

Для развертывания сервиса потребуются **инфраструктура**, **настройка компонентов** и **интеграции**. Вот детальный план:

**Этап 1: Подготовка инфраструктуры**

**1.1. Серверная часть**

| **Ресурс** | **Минимальные требования** | **Рекомендации (для масштабирования)** |
| --- | --- | --- |
| **Сервер (VPS/Cloud)** | 2 CPU, 4 GB RAM, 50 GB SSD (например, Hetzner, AWS Lightsail) | 4+ CPU, 8 GB RAM, автоскейлинг (AWS EC2, Kubernetes) |
| **Базы данных** | PostgreSQL (10 GB), MongoDB (5 GB) | Управляемые БД (AWS RDS, MongoDB Atlas) |
| **Брокер сообщений** | Kafka (или Redis для MVP) | Managed Kafka (Confluent Cloud) |
| **Хранилище файлов** | S3-совместимое (MinIO, AWS S3) | AWS S3 + CloudFront (CDN для изображений) |

**1.2. Домены и SSL**

* Зарегистрировать домен (например, photo-booking.com).
* Настроить HTTPS (Let’s Encrypt или Cloudflare).

**Этап 2: Развертывание компонентов**

**2.1. Бэкенд (API и микросервисы)**

**Технологии**: Golang, Java (CRM), Docker.  
**Шаги**:

1. **Собрать образы** для каждого сервиса:

bash

Copy

Download

docker build -t booking-service -f ./booking/Dockerfile .

docker build -t notification-service -f ./notification/Dockerfile .

1. **Запустить в Kubernetes или через Docker Compose**:

yaml

Copy

Download

*# docker-compose.yml (упрощенный пример)*

services:

postgres:

image: postgres:14

volumes:

- pg\_data:/var/lib/postgresql/data

api:

image: api-gateway

ports:

- "80:8080"

depends\_on:

- postgres

**2.2. Фронтенд (веб и мобильное приложение)**

* **Веб**: Разместить статику (React) на S3 + CloudFront или Vercel.
* **Мобильное приложение**: Сборка через React Native (expo build:android / expo build:ios).

**2.3. Базы данных**

1. **PostgreSQL**:

sql

Copy

Download

CREATE DATABASE photo\_booking;

CREATE USER app\_user WITH PASSWORD 'secure\_password';

GRANT ALL PRIVILEGES ON DATABASE photo\_booking TO app\_user;

1. **MongoDB**:
   * Настроить репликасет (для отказоустойчивости).

**Этап 3: Интеграции**

**3.1. Платежные системы**

* **Stripe**:

go

Copy

Download

stripe.Key = "sk\_test\_..."

params := &stripe.PaymentIntentParams{

Amount: stripe.Int64(1000),

Currency: stripe.String("usd"),

}

pi, \_ := paymentintent.New(params)

* **ЮKassa**: API для RUB-платежей.

**3.2. Уведомления**

* **Telegram Bot**:

python

Copy

Download

import telebot

bot = telebot.TeleBot("TOKEN")

bot.send\_message(chat\_id, "Новая запись!")

* **FCM (Push)**:

go

Copy

Download

client, \_ := app.Messaging(ctx)

client.Send(ctx, &fcm.Message{Token: deviceToken, Notification: &fcm.Notification{Title: "Фотосессия!"}})

**3.3. Геолокация**

* **Google Maps API**: Для поиска ближайших студий.

**Этап 4: Запуск и мониторинг**

**4.1. CI/CD (автоматизация деплоя)**

* **GitHub Actions**:

yaml

Copy

Download

name: Deploy

on: push

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- run: docker-compose up -d --build

**4.2. Мониторинг**

* **Prometheus + Grafana**: Сбор метрик (CPU, RAM, ошибки API).
* **Sentry**: Логирование ошибок в реальном времени.

**Этап 5: Тестирование и вывод в прод**

**5.1. Тесты**

1. **Юнит-тесты** (Go):

go

Copy

Download

func TestBooking(t \*testing.T) {

req := BookingRequest{UserID: "123", SlotID: "456"}

assert.True(t, validateRequest(req))

}

1. **Нагрузочное тестирование** (Locust):

python

Copy

Download

from locust import HttpUser, task

class BookingUser(HttpUser):

@task

def book\_slot(self):

self.client.post("/api/book", json={"user\_id": 1, "slot\_id": 2})

**5.2. Плавный вывод**

1. **Canary-развертывание**:
   * 5% трафика на новую версию API.
2. **Откат при проблемах**:

bash

Copy

Download

kubectl rollout undo deployment/api

**Что нужно для запуска?**

1. **Инфраструктура**:
   * Серверы (или облако: AWS/GCP).
   * Домен + SSL.
2. **Код**:
   * Репозитории с бэкендом (Go/Java), фронтендом (React), мобильным приложением.
3. **Интеграции**:
   * Платежи (Stripe), уведомления (Telegram/FCM), карты (Google Maps).
4. **DevOps-инструменты**:
   * Docker, Kubernetes, CI/CD (GitHub Actions).

**Схема развертывания**

Diagram

Code

Download

flowchart TD

A[Подготовка серверов] --> B[Настройка БД]

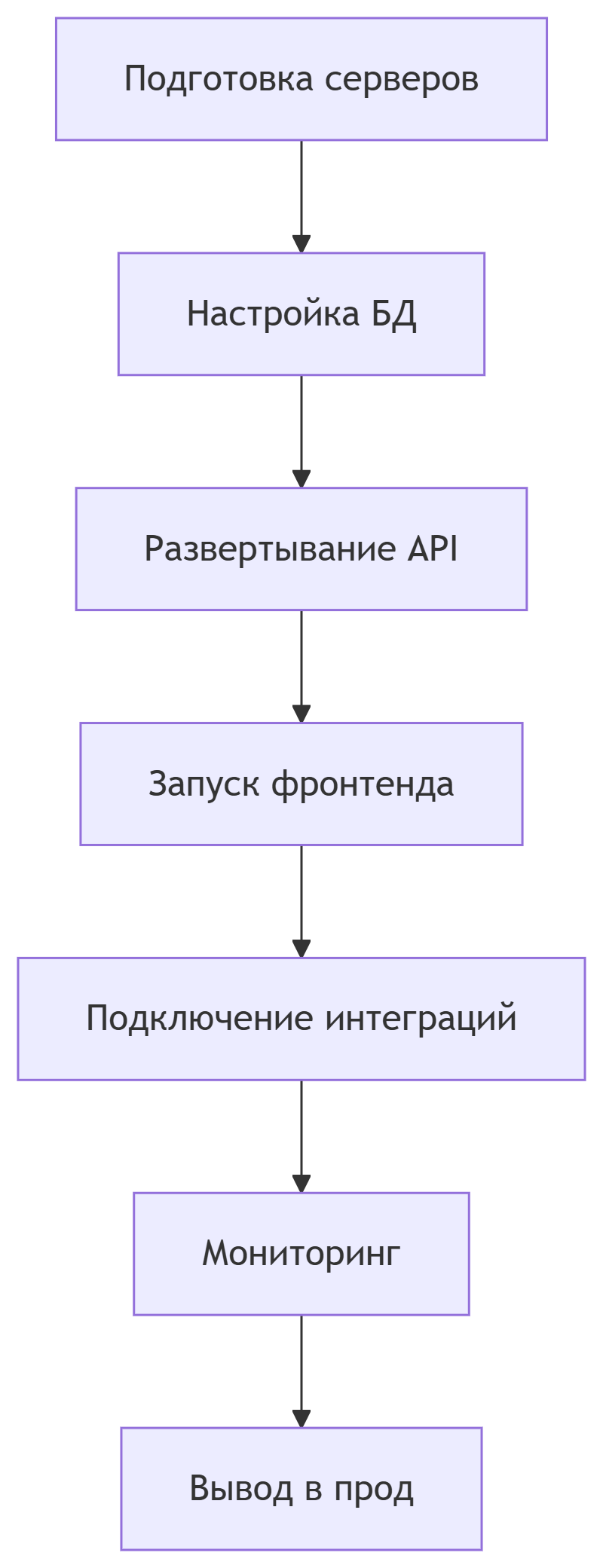
B --> C[Развертывание API]

C --> D[Запуск фронтенда]

D --> E[Подключение интеграций]

E --> F[Мониторинг]

F --> G[Вывод в прод]



**Итог**:

* Запуск MVP возможен за **2-4 недели** (на одном сервере с Docker).
* Полная версия с масштабированием — **1-2 месяца**.
* Минимальное участие разработчиков после настройки CI/CD и мониторинга.

**Дополнительные компоненты и нюансы для надежного и масштабируемого сервиса**

Чтобы сервис работал бесперебойно и покрывал все потребности пользователей, нужно предусмотреть:

**1. Безопасность**

**1.1. Защита данных**

* **Шифрование**:
  + TLS 1.3 для всех API.
  + Шифрование чувствительных данных в БД (например, платежных реквизитов) с помощью **AES-256** или **Vault**.
* **Аутентификация и авторизация**:
  + OAuth 2.0 / OpenID Connect (Google, Apple, VK).
  + Двухфакторная аутентификация (2FA) для фотографов и менеджеров.
* **Защита от атак**:
  + Rate Limiting (например, через **Redis**).
  + WAF (Cloudflare, AWS WAF) от SQL-инъекций и XSS.

**1.2. Резервное копирование и восстановление**

* **Базы данных**:
  + Ежедневные снепшоты PostgreSQL/MongoDB (например, через **AWS RDS Snapshots**).
  + Репликация в другой регион (Geo-Redundancy).
* **Файлы (изображения)**:
  + Versioning в S3 + кросс-регионная репликация.

**2. Юридические и финансовые аспекты**

**2.1. Документы**

* **Политика конфиденциальности** (GDPR/ФЗ-152).
* **Публичная оферта** (условия бронирования и возврата денег).
* **Согласие на обработку персональных данных**.

**2.2. Финансы**

* **Интеграция с бухгалтерией**:
  + Экспорт данных в **1С** или **QuickBooks**.
* **Отчетность**:
  + Автоматическая генерация чеков (ФЗ-54 для РФ).
* **Возвраты платежей**:
  + Механизм частичного/полного возврата через Stripe/YooMoney API.

**3. Пользовательский опыт (UX)**

**3.1. Дополнительные фичи**

* **Система рейтингов и отзывов**:
  + Оценка фотографов клиентами (хранить в PostgreSQL).
* **Отмена/перенос брони**:
  + Логика штрафов за позднюю отмену.
* **Гибкие пакеты услуг**:
  + Предоплаченные часы съемки, подарочные сертификаты.

**3.2. Аналитика поведения**

* **Google Analytics** / **Amplitude**:
  + Трекинг популярных студий, времени бронирования.
* **A/B-тестирование**:
  + Сравнение конверсии для разных вариантов интерфейса.

**4. Администрирование и поддержка**

**4.1. Инструменты для команды**

* **Internal Admin Panel**:
  + Возможность ручного редактирования броней (например, через **Retool**).
* **Чат-бот для сотрудников**:
  + Уведомления в Telegram о критичных ошибках.

**4.2. Логирование и аудит**

* **Centralized Logs**:
  + ELK-стек (Elasticsearch + Logstash + Kibana) или **Grafana Loki**.
* **Аудит действий**:
  + Таблица audit\_log в PostgreSQL с записью изменений (кто, когда, что изменил).

**5. Подготовка к пиковым нагрузкам**

**5.1. Автомасштабирование**

* **Для API**:
  + Kubernetes HPA (Horizontal Pod Autoscaler) на основе CPU/метрик Prometheus.
* **Для БД**:
  + Чтение из реплик PostgreSQL (например, через **PgBouncer**).

**5.2. Кэширование**

* **Redis**:
  + Кэш расписаний фотографов, чтобы снизить нагрузку на БД.
  + Сессии пользователей.

**6. Потенциальные проблемы и их решение**

| **Проблема** | **Решение** |
| --- | --- |
| **Двойное бронирование** | Оптимистичные блокировки (SELECT FOR UPDATE в PostgreSQL). |
| **Потеря соединения с платежным шлюзом** | Retry-логика + уведомление менеджеру. |
| **Сбой уведомлений** | Dead Letter Queue (DLQ) в Kafka для повторной отправки. |
| **Атака ботов** | reCAPTCHA v3 + лимит запросов с IP. |

## ****7. Информационная модель (дополнение к предыдущей схеме)****

erDiagram

users ||--o{ reviews : "оставляет"

bookings ||--o{ transactions : "имеет"

studios ||--o{ equipment : "содержит"

reviews {

uuid id PK

uuid user\_id FK

uuid photographer\_id FK

integer rating

text comment

}

transactions {

uuid id PK

uuid booking\_id FK

decimal amount

string status

timestamp created\_at

}

equipment {

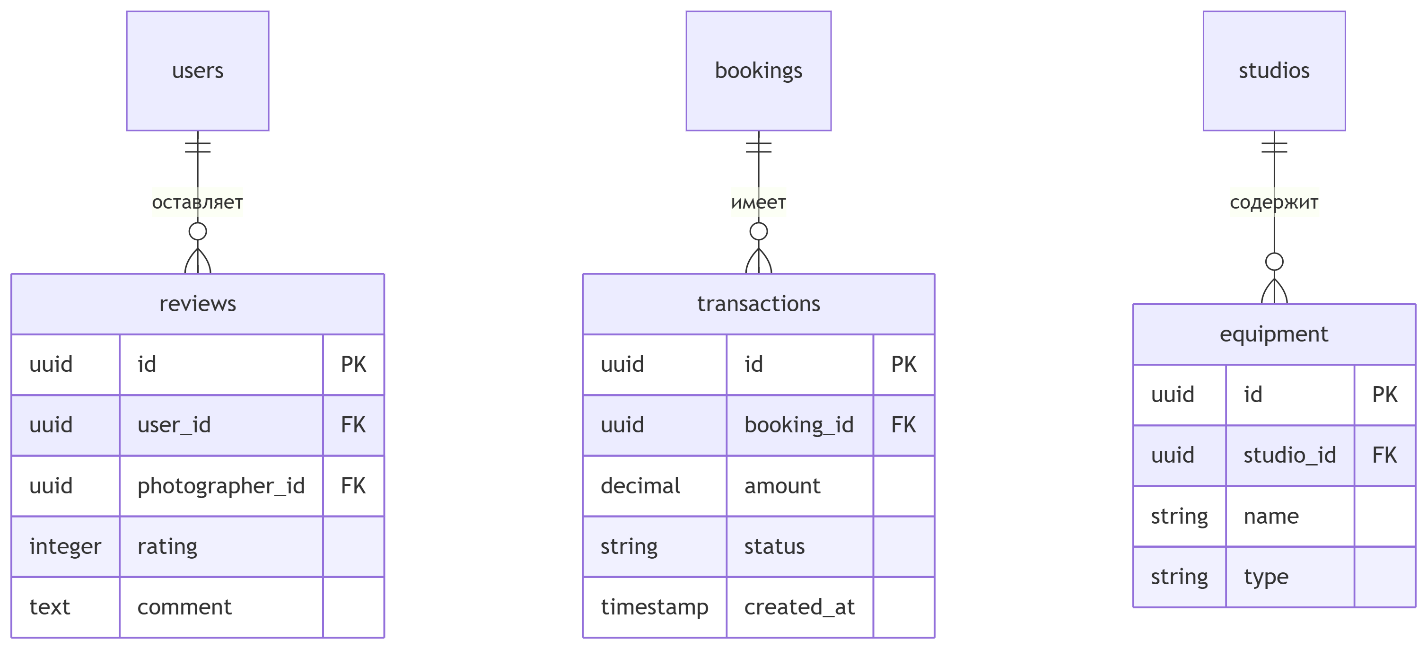
uuid id PK

uuid studio\_id FK

string name

string type

}



## ****Итог: что добавить к текущей архитектуре?****

1. **Безопасность**:
   * Шифрование, WAF, 2FA.
2. **Юридическая база**:
   * Оферта, политика конфиденциальности.
3. **Финансы**:
   * Интеграция с бухгалтерией, возвраты.
4. **UX-улучшения**:
   * Рейтинги, A/B-тесты, аналитика.
5. **Администрирование**:
   * Логирование, аудит, инструменты для поддержки.
6. **Надежность**:
   * DLQ, retry-логика, автомасштабирование.

Это превратит сервис из MVP в профессиональное решение, минимизирующее ручную работу и риски.