**Распознанная задача:**  
Создать локального Telegram-бота на C# (используя библиотеку Minimal APIs), который:

1. **Принимает название города на английском языке** и возвращает текущую погоду.
2. **Использует бесплатный погодный API** (который я найду), работающий по названию города.
3. **Хранит историю запросов** в базе данных (например, SQLite для простоты).
4. **Предоставляет команду**/stats, которая показывает:
   * Самые часто запрашиваемые города.
   * Статистику по погоде (средняя температура, классификация "холодно"/"тепло" и т.д.).
5. **Запускается локально,** без публикации в облако.
6. *(На будущее)* Будет готов к добавлению функции отправки картинки (погодной иконки).

**План выполнения**

**Этап 1: Подготовка и настройка проекта**

1. **Создать новый проект.**
   * dotnet new web -n WeatherTelegramBot
   * cd WeatherTelegramBot
2. **Добавить необходимые NuGet-пакеты:**
   * dotnet add package Telegram.Bot (основная библиотека для бота)
   * dotnet add package Microsoft.EntityFrameworkCore.Design (для работы с БД)
   * dotnet add package Microsoft.EntityFrameworkCore.Sqlite (БД SQLite)
   * dotnet add package HttpClient (для запросов к погодному API)

**Этап 2: Выбор и интеграция погодного API**

1. **Найти бесплатный API.** Отличный кандидат — **OpenWeatherMap**. У них есть бесплатный тариф "Current Weather Data", который идеально подходит.
   * Регистрируюсь на [openweathermap.org](https://openweathermap.org/api).

(5d539df439fbce04ed7bfa0a2516e38f

* + api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=5d539df439fbce04ed7bfa0a2516e38f)
  + Получаю API Key (он понадобится для запросов).

1. **Создать модель данных** для ответа от OpenWeatherMap (нас интересуют поля main.temp, weather[0].description и т.д.).
2. **Написать сервис (**WeatherService**)**. Это будет класс, который с помощью HttpClient делает запрос к api.openweathermap.org и возвращает удобный объект с погодой.

**Этап 3: Проектирование и настройка базы данных**

1. **Создать модель данных для хранения запросов.** Например, класс WeatherRequest:
   * Id (int, первичный ключ)
   * CityName (string)
   * Temperature (double)
   * Description (string)
   * RequestDate (DateTime)
2. **Создать контекст базы данных (**ApplicationDbContext**)** с DbSet<WeatherRequest>.
3. **Настроить подключение к SQLite** в Program.cs.

**Этап 4: Разработка логики Telegram-бота**

1. **Создать бота через @BotFather** в Telegram и получить его токен.
2. **Настроить обработчики команд в**Program.cs с помощью TelegramBotClient.
   * **Команда**/start: Приветственное сообщение.
   * **Обработка текстовых сообщений**: Если сообщение не команда, считать его названием города.
     + Передать город в WeatherService.
     + Полученные данные о погоде сохранить в базу данных.
     + Отправить пользователю текстовый ответ (например, "Погода в Москве: +15°C, облачно").
   * **Команда**/stats:
     + Запрос к БД для получения топ-N запрашиваемых городов (группировка по CityName).
     + Расчет средней температуры, определение самой частой погоды (например, "солнечно", "дождь").

**Этап 5: Сборка и локальный запуск**

1. **Написать инструкцию по запуску** (куда вставить токен бота и API-ключ погоды, команда dotnet run).
2. **Протестировать все функции:** запрос погоды, проверка статистики.

Это пошаговый план, который я начну реализовывать. Первым делом я изучу OpenWeatherMap API и подготовлю проект.

можешь схему плана работы нарисовать

Конечно! Вот схема плана работы в виде блок-схемы:

**Пояснения к схеме:**

**Этап 1** (Синий): Базовая настройка проекта C# с NuGet пакетами

**Этап 2** (Оранжевый): Работа с погодным API (OpenWeatherMap)

* Получение API ключа
* Настройка HttpClient
* Парсинг JSON ответа

**Этап 3** (Фиолетовый): Настройка базы данных SQLite

* Создание моделей данных
* Миграции БД
* Конфигурация EF Core

**Этап 4** (Голубой): Основная логика бота:

* /start - приветствие
* Запрос погоды по городу → сохранение в БД
* /stats - анализ статистики из БД

**Этап 5**: Финальное тестирование и локальный запуск

Сейчас я приступаю к реализации по этой схеме! Начинаю с Этапа 1.

и схему архитектуры приложения

Конечно! Вот схема архитектуры приложения:

**Компоненты архитектуры:**

**1. Внешние сервисы (External Services)**

* **Telegram Bot API** - получает/отправляет сообщения
* **OpenWeatherMap API** - предоставляет данные о погоде

**2. Основное приложение (.NET Minimal API)**

* **Program.cs** - точка входа, конфигурация DI контейнера
* **Endpoints** - обработчики HTTP запросов (webhook)

**3. Сервисный слой (Service Layer)**

* **Telegram Service** - обработка команд бота, форматирование ответов
* **Weather Service** - запросы к погодному API, парсинг данных
* **Statistics Service** - анализ данных из БД, генерация статистики

**4. Слой данных (Data Layer)**

* **Database Context** (EF Core) - работа с базой данных
* **SQLite Database** - хранение истории запросов

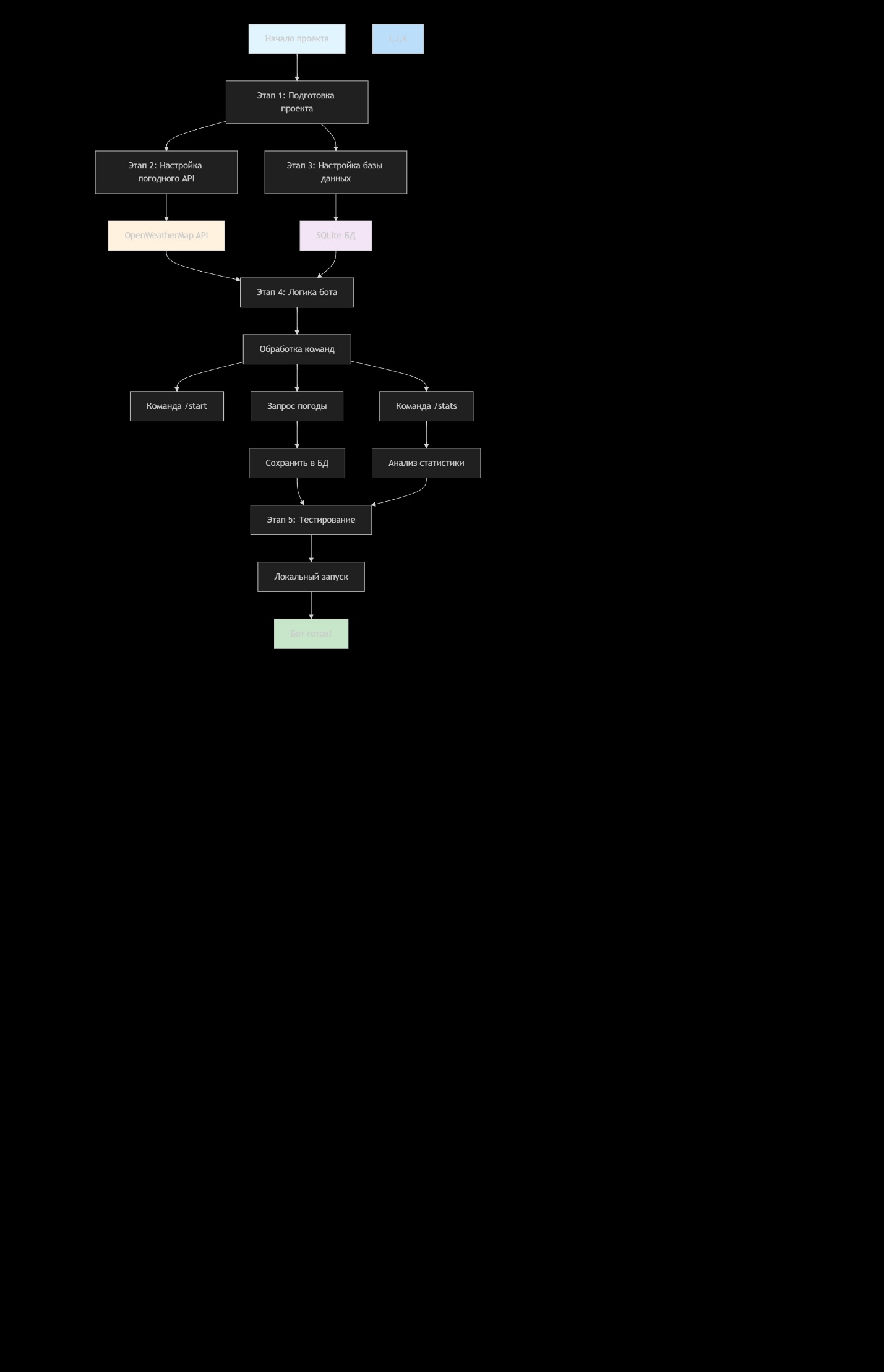
**5. Инфраструктура (Infrastructure)**

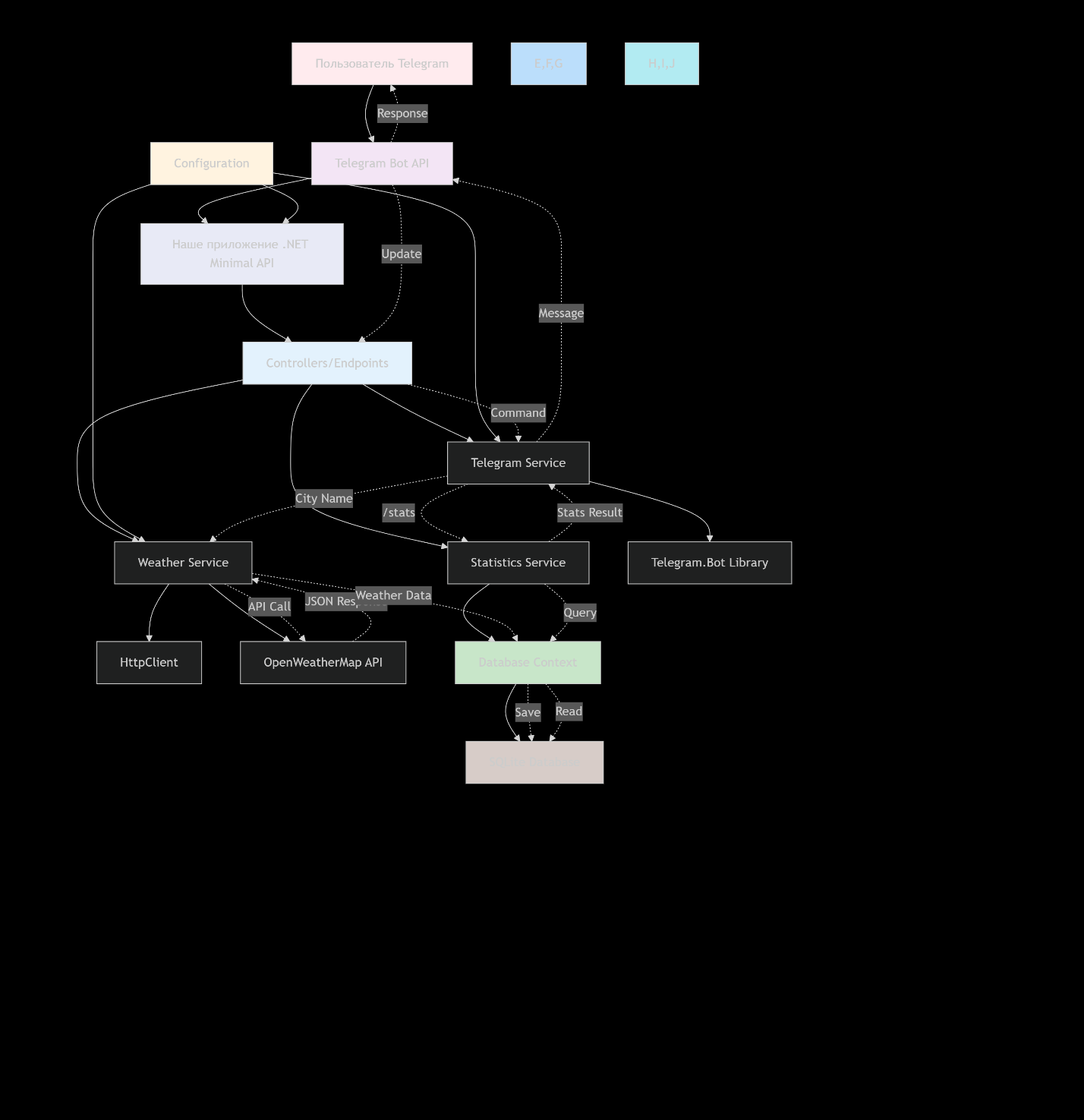
* **Configuration** - настройки (API ключи, токены бота)
* **HttpClient** - для внешних HTTP запросов
* **Telegram.Bot Library** - официальная библиотека для Telegram API

**Поток данных:**

1. **Пользователь** → /weather Moscow → **Telegram API**
2. **Telegram API** → Webhook → **Наше приложение**
3. **Endpoints** → распознает команду → **Telegram Service**
4. **Telegram Service** → извлекает город → **Weather Service**
5. **Weather Service** → запрос → **OpenWeatherMap API**
6. **OpenWeatherMap API** → погодные данные → **Weather Service**
7. **Weather Service** → сохраняет данные → **Database** → возвращает результат
8. **Telegram Service** → форматирует ответ → **Telegram API**
9. **Telegram API** → показывает погоду → **Пользователю**

Такая архитектура обеспечивает разделение ответственности и легкую поддержку кода!





api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=5d539df439fbce04ed7bfa0a2516e38f