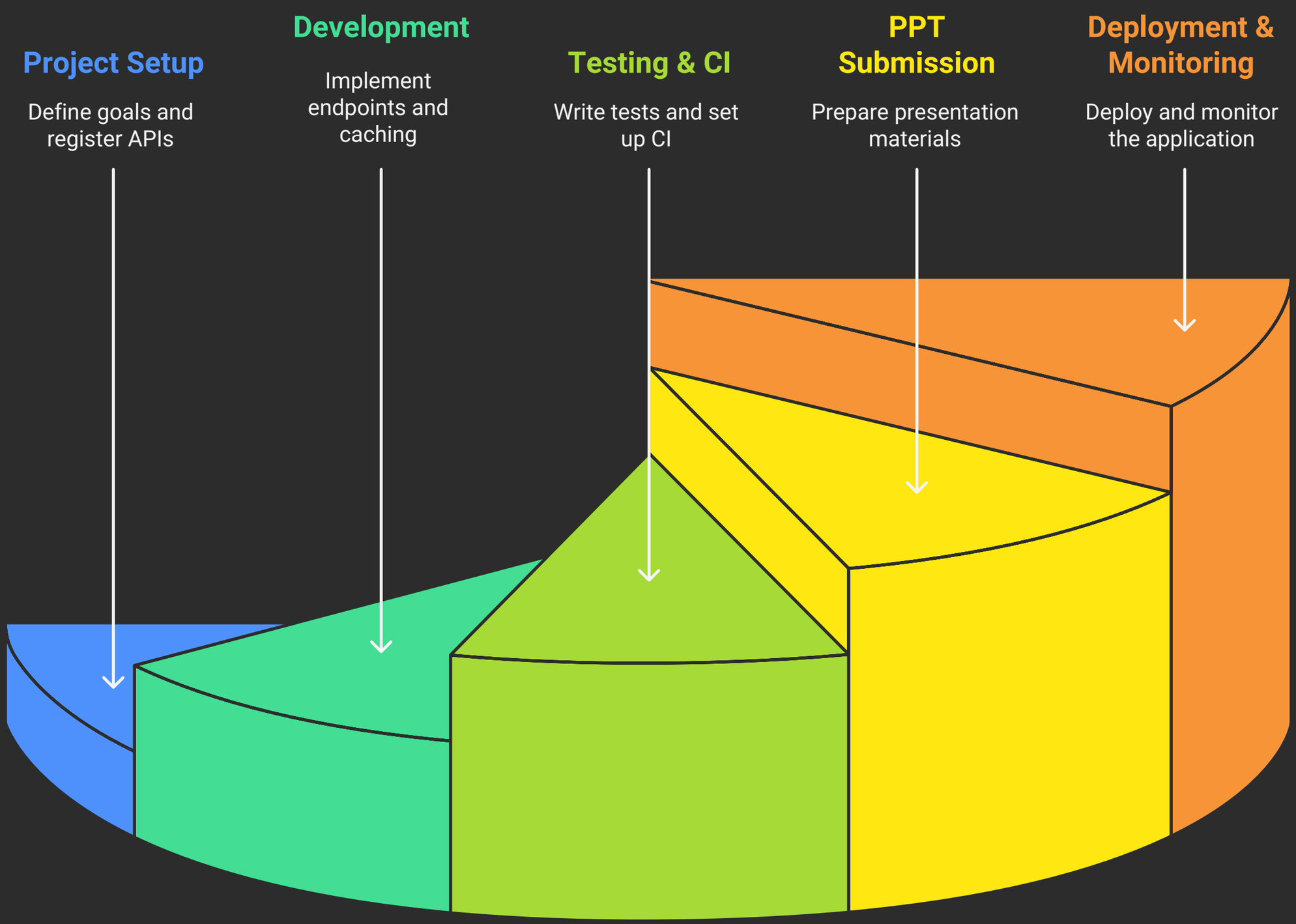
Delhi Data APIs: Project Lifecycle

This document outlines the complete project lifecycle for the "Delhi Data APIs" FastAPI project, from initial setup to deployment and monitoring. It provides a visual workflow chart with swimlanes representing key phases, designed for clarity and ease of understanding, particularly for presentation purposes.

```
graph LR
    subgraph Project Setup
        A[Idea & Goals <br> Define problem statement & objectives] -->
B(Register external APIs <br> Calendarific, WeatherAPI, ElectricityMaps);
        B --> C{Create repo & README <br> (FastAPI-focused)};
    end

    subgraph Development
        C --> D[Local env setup <br> Python venv / virtualenv];
        D --> E{requirements.txt <br> "pip install -r requirements.txt"};
        E --> F{Project scaffold <br> main.py, routes, config (ENV vars)};
        F --> G[Implement endpoints <br> /holidays, /weather, /carbon-intensity
<br> (include query param examples)];
        G --> H{Add basic caching layer <br> (in-memory or lru_cache)};
        H --> I{Add Swagger metadata <br> (tags, summaries) so /docs is clean};
    end

    subgraph Testing & CI
        I --> J[Write simple smoke tests <br> (pytest) for each endpoint <br>
(mock APIs)];
        J --> K{Create GitHub Actions workflow (CI) <br> run tests, lint,
build};
        K --> L((Optional: auto-update requirements check <br> or manual pip
freeze step));
    end

    subgraph PPT Submission (Hackathon)
        L --> M[Prepare one-slide summary <br> problem, solution, endpoints,
tech stack];
        M --> N{Add architecture diagram screenshot <br> (from this chart)};
        N --> O{Include demo screenshots of /docs};
        O --> P{Keep README and requirements.txt updated <br> (required for
reviewers)};
    end

    subgraph Deployment & Monitoring
        P --> Q[Deploy to lightweight host <br> Render / Railway / Heroku /
Vercel <br> one-click or simple Procfile];
        Q --> R{Add env vars on host <br> for API keys};
        R --> S{Health-check endpoint <br> /health};
        S --> T{Basic logging + error alerts <br> (stdout logs + platform
logs)};
        T --> U((Optionally set up simple uptime monitoring));
    end

    style B fill:#f9f,stroke:#333,stroke-width:2px
    style F fill:#ccf,stroke:#333,stroke-width:2px
    style R fill:#f9f,stroke:#333,stroke-width:2px

    linkStyle 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
stroke:#333,stroke-width:2px

    note right of F "Store API keys in ENV"
    note right of H "Rate limits: cache responses during demo"
    note right of N "PPT-ready assets: architecture + /docs screenshots"
    note right of E "Run pip freeze > requirements.txt"
```



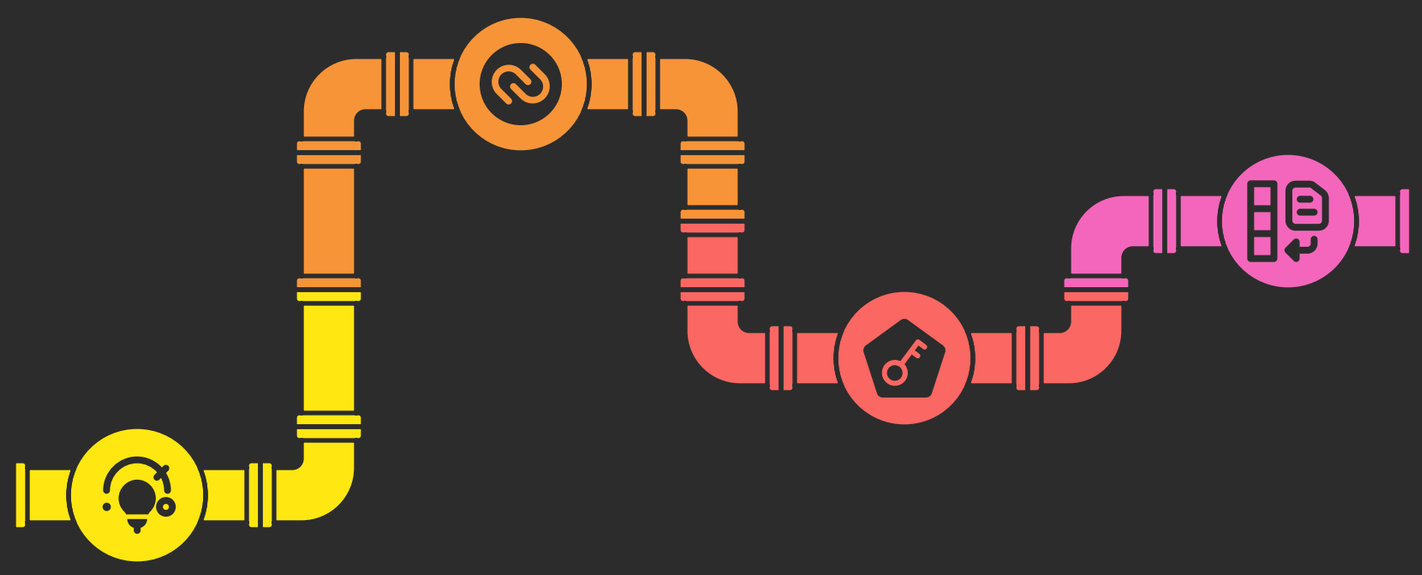
Detailed Breakdown

Project Setup

This initial phase focuses on defining the project's scope and setting up the necessary infrastructure.

- **Idea & Goals:** Clearly define the problem the API aims to solve and the specific objectives it should achieve.
- **Register External APIs:** Obtain API keys from Calendarific, WeatherAPI, and ElectricityMaps. **Important:** Store these keys securely using environment variables or a secrets management system.
- **Create Repo & README:** Initialize a new Git repository and create a comprehensive README file, emphasizing the project's FastAPI-based architecture.

API Project Setup Sequence



01

Define Idea & Goals

Clearly articulate the problem and objectives

02

Register External APIs

Obtain API keys from various services

03

Secure API Keys

Store keys securely using environment variables

04

Create Repo & README

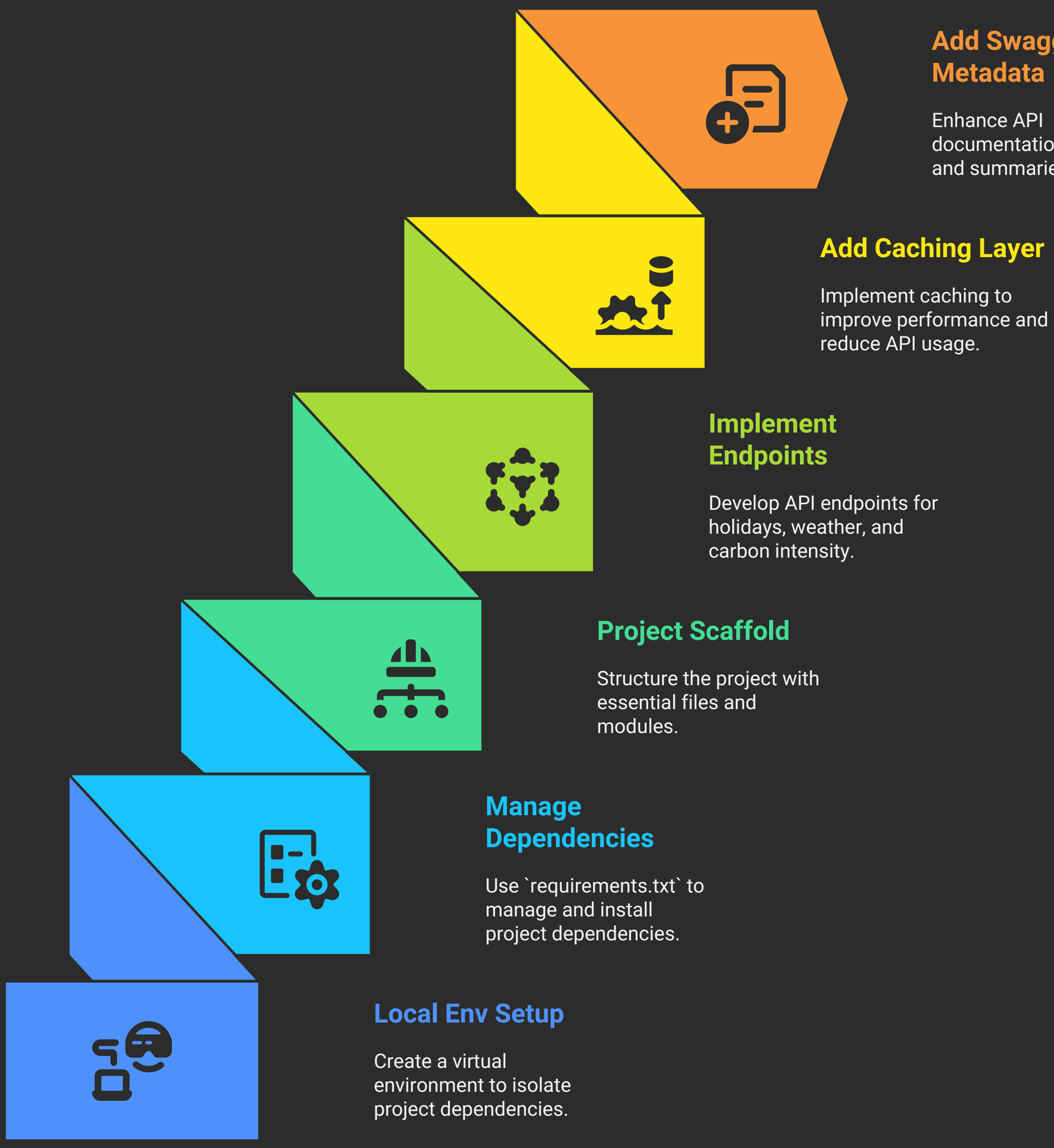
Initialize Git repository and document project

Development

The core development phase involves building the API endpoints and implementing essential features.

- **Local Env Setup:** Create a virtual environment (using venv or virtualenv) to isolate project dependencies.
- **requirements.txt:** Manage project dependencies using requirements.txt. Use pip install -r requirements.txt to install dependencies and pip freeze > requirements.txt to update the list.
- **Project Scaffold:** Structure the project with main.py, separate route files, and a configuration module to handle environment variables.
- **Implement Endpoints:** Develop the /holidays, /weather, and /carbon-intensity endpoints, including support for relevant query parameters.
- **Add Basic Caching Layer:** Implement a caching mechanism (e.g., in-memory or lru_cache) to improve performance and reduce API usage, especially during demonstrations.
- **Add Swagger Metadata:** Enhance the API documentation by adding tags and summaries to the endpoints, ensuring a clean and informative /docs interface.

Building Delhi Data APIs



Add Swagger Metadata

Enhance API documentation with tags and summaries.

Add Caching Layer

Implement caching to improve performance and reduce API usage.

Implement Endpoints

Develop API endpoints for holidays, weather, and carbon intensity.

Project Scaffold

Structure the project with essential files and modules.

Manage Dependencies

Use `requirements.txt` to manage and install project dependencies.

Local Env Setup

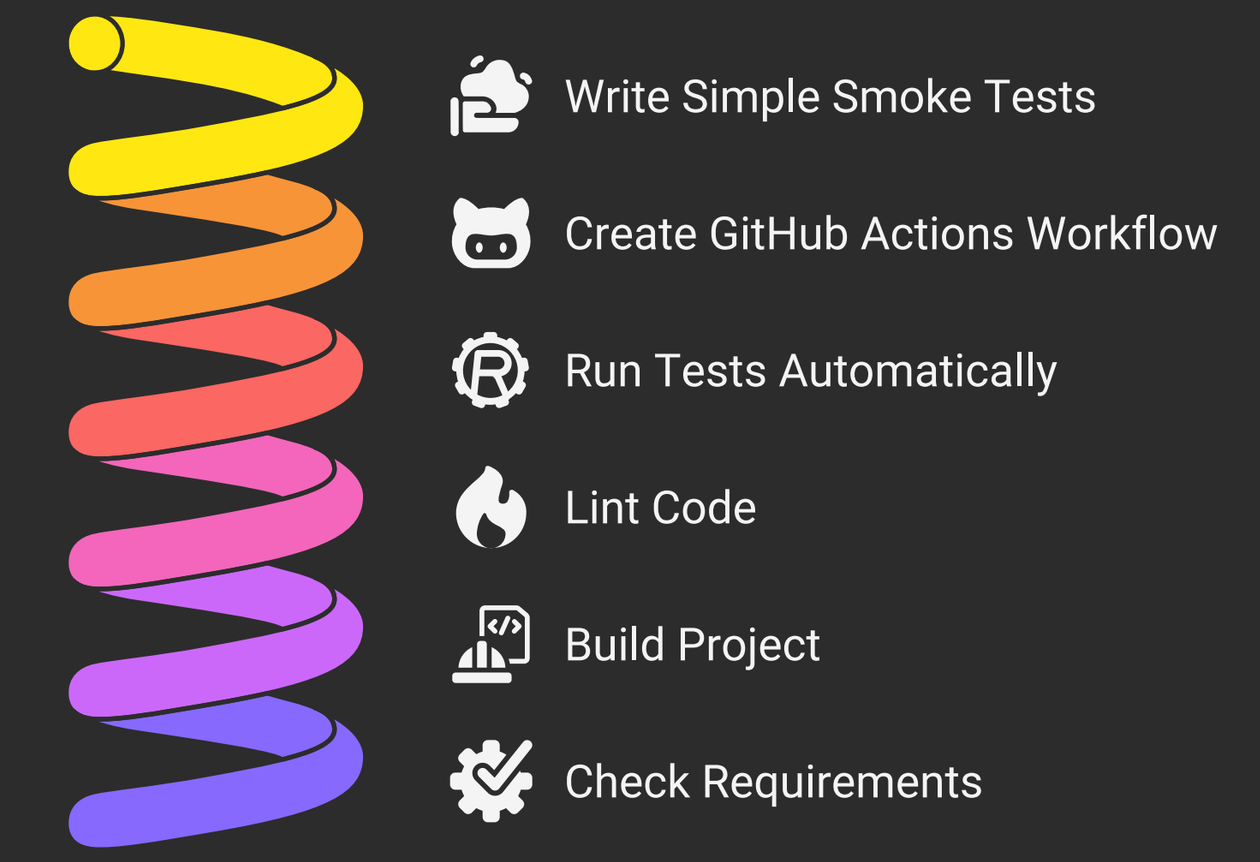
Create a virtual environment to isolate project dependencies.

Testing & CI

This phase focuses on ensuring the API's reliability and maintainability through automated testing and continuous integration.

- **Write Simple Smoke Tests:** Create basic tests (using pytest) for each endpoint, mocking external API responses to ensure consistent results.

- **Create GitHub Actions Workflow:** Set up a CI workflow that automatically runs tests, lints the code, and builds the project on every commit.
- **Optional: Auto-update requirements check:** Implement a check to ensure requirements.txt is up-to-date, or manually run pip freeze > requirements.txt before deployment.

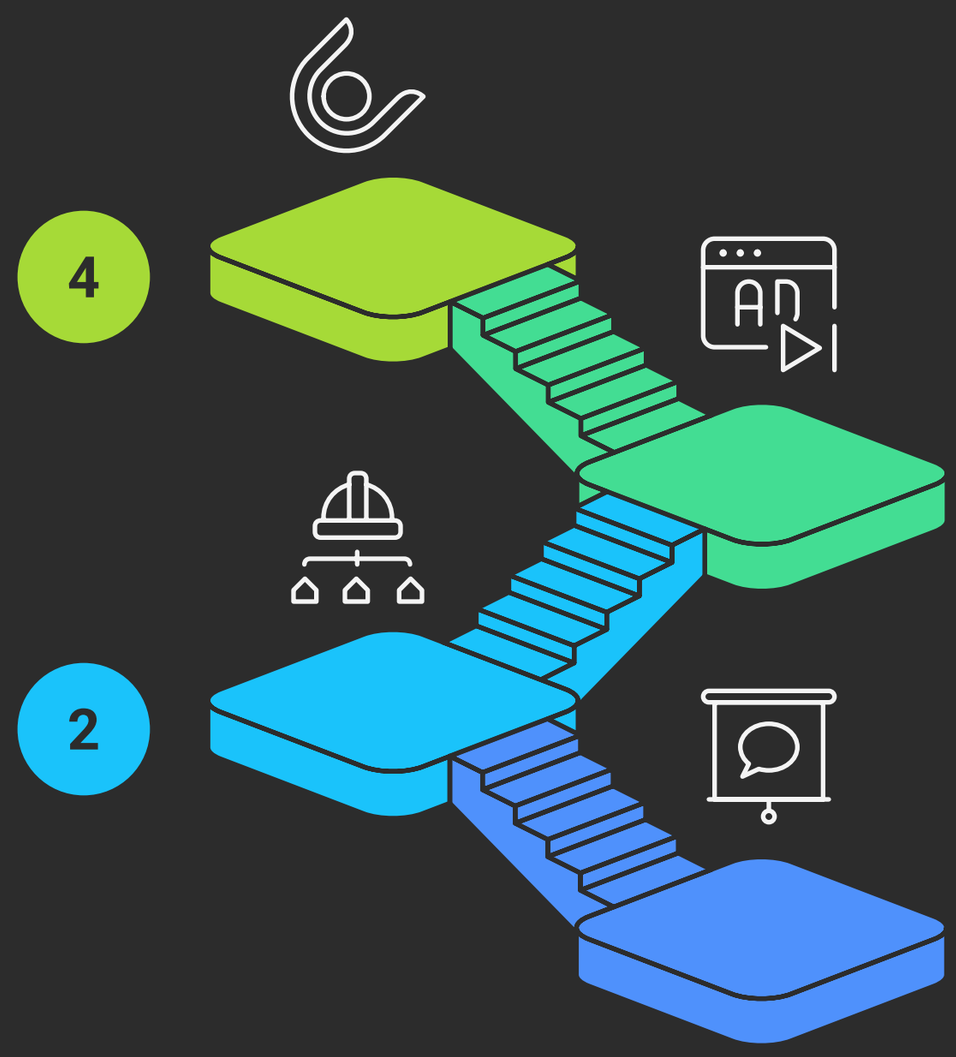


PPT Submission (Hackathon)

This phase prepares the project for presentation and evaluation.

- **Prepare One-Slide Summary:** Create a concise summary slide highlighting the problem, solution, key endpoints, and tech stack.
- **Add Architecture Diagram Screenshot:** Include a screenshot of the workflow chart to visually represent the project's architecture.
- **Include Demo Screenshots of /docs:** Showcase the API's documentation by including screenshots of the /docs interface.
- **Keep README and requirements.txt updated:** Ensure the README and requirements.txt files are up-to-date, as they are crucial for reviewers.

Preparing for Hackathon Submission



4 Update README & Requirements

Ensure the README and `requirements.txt` files are up-to-date, as they are crucial for reviewers.

3 Demo Screenshots

Showcase the API's documentation by including screenshots of the `/docs` interface.

2 Architecture Diagram

Include a screenshot of the workflow chart to visually represent the project's architecture.

1 One-Slide Summary

Create a concise summary slide highlighting the problem, solution, key endpoints, and tech stack.

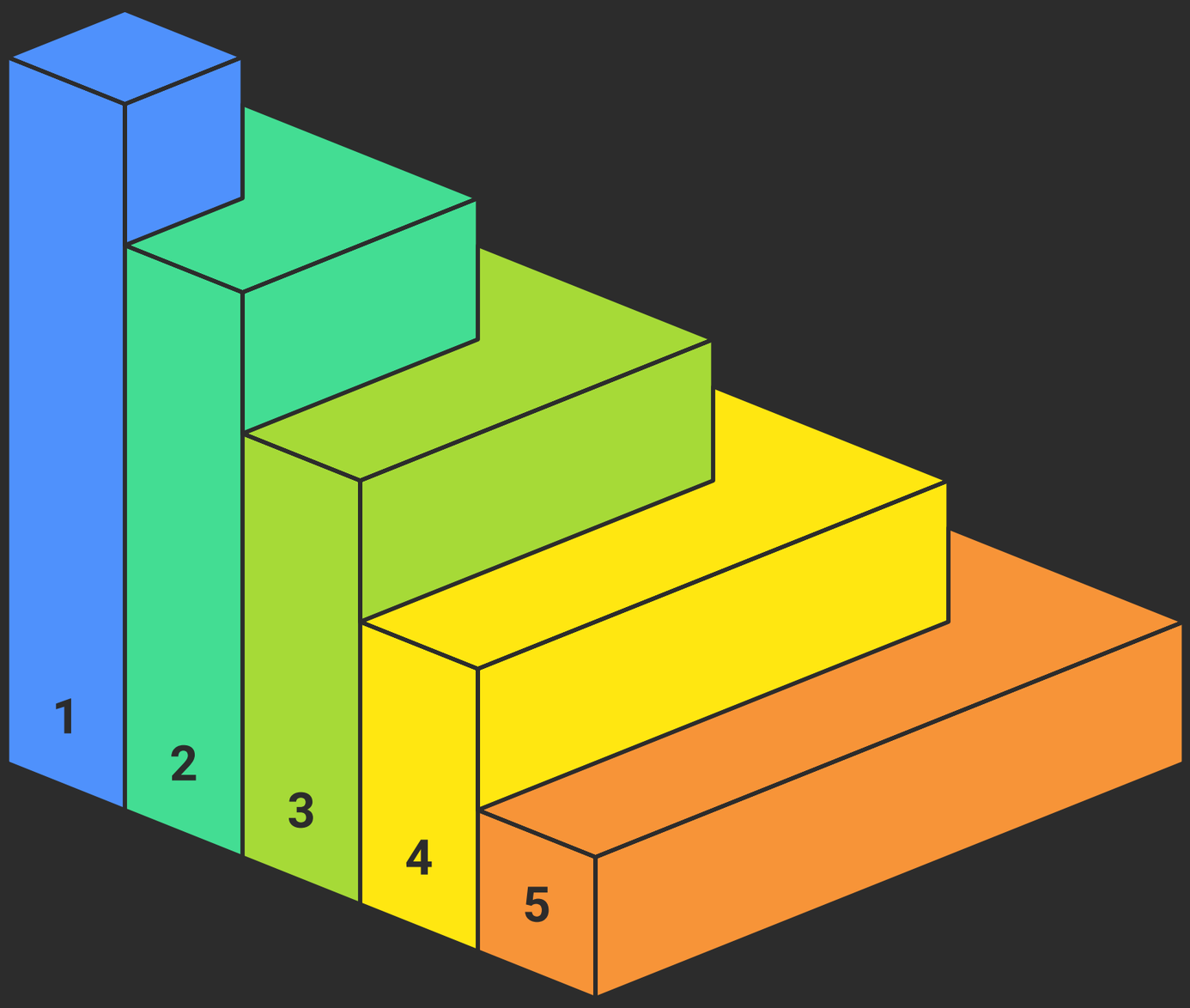
Deployment & Monitoring

The final phase involves deploying the API and monitoring its performance.

- **Deploy to Lightweight Host:** Deploy the API to a platform like Render, Railway, Heroku, or Vercel, utilizing one-click deployment or a simple Procfile.
- **Add env vars on host:** Configure environment variables on the hosting platform to securely store API keys.
- **Health-check endpoint:** Implement a /health endpoint to monitor the API's availability.

- **Basic logging + error alerts:** Set up basic logging and error alerts using stdout logs and the platform's logging capabilities.
- **Optionally set up simple uptime monitoring:** Consider using a service to monitor the API's uptime and receive alerts if it becomes unavailable.

API Deployment and Monitoring



Deploy to Host

Deploy the API to a lightweight hosting platform.

Configure Env Vars

Set up environment variables on the hosting platform.

Implement Health Check

Add a health-check endpoint to monitor API availability.

Set Up Logging

Establish basic logging and error alerts.

Uptime Monitoring

Optionally set up uptime monitoring for continuous availability.