Technical Documentation

Historical Weather Forecasts API

The Historical Weather Forecasts API is a RESTful API built using LoopBack 4. The API provides endpoints for retrieving historical 14-day weather forecast data, stored in a PostgreSQL database.

The API is deployed on DigitalOcean and has a domain on Namecheap.

The API is private and usable through a custom VPN connection only.

OpenAPI documentation can be found at the /explorer endpoint.
https://historical-forecasts-api.xyz/explorer

Github link: https://github.com/robocode2/historical-forecasts-api

1.1 Architecture

This project is built using **LoopBack 4**, adhering to a **layered architecture** with a focus on maintainability, scalability, and clean separation of concerns. The key components include:

1. **Controller Layer**

Purpose: Receives incoming HTTP requests and routes them to appropriate services for processing.

Responsibilities:

- **Receives Requests**: Accepts incoming API calls for weather forecast data.
- Validates Parameters: Checks the validity of query parameters (e.g., source, location, date range).
- **Retrieves Data**: Interacts with the repositories to fetch data (e.g., weather data for specific locations).
- **Passes Data to Service**: Sends the retrieved data to the DataService for formatting (CSV generation).
- **Returns Response**: Returns the formatted data to the client in text/csv

2. **Service Layer**

Purpose: Contains the business logic for data formatting and processing.

Responsibilities:

- **Formats Data**: Takes raw data retrieved by the controller and formats it into the desired output (CSV).

3. **Repository Layer**

Purpose: Manages communication with the database through LoopBack 4 repositories.

Responsibilities:

- Provides CRUD operations for database entities (e.g., locations, forecasts).
- Retrieves and filters weather forecast data using repository methods.
- Abstracts SQL queries and ensures safe, parameterized database interactions.
- Utilizes LoopBack's data sources to connect to PostgreSQL.

4. **Database (PostgreSQL on DigitalOcean)**

Purpose: Stores normalized weather forecast data.

Responsibilities:

- **Weather Data Storage**: Stores historical weather forecast data in normalized tables (e.g., city, country, source, forecast).
- **Efficient Querying:** Supports complex queries for historical weather data based on location and date range.

Deployment

- **Dockerized**: The application is containerized using Docker for consistent deployments across different environments.
- **Hosting**: The application is deployed on DigitalOcean
- **Domain Configuration:** The domain is managed through Namecheap, with DNS configuration pointing to the DigitalOcean server.
- **SSL/TLS:** SSL certificates are managed using **Certbot** for HTTPS encryption.

Logging

Pino is used for logging useful information and errors.

1.2 VPN Server Setup on DigitalOcean with Firewall

To secure the API application and prevent unauthorized access to the server, a **VPN server** is deployed on a DigitalOcean droplet. A firewall/ IP Whitelist ensures that only clients with VPN access can communicate with the backend API and database.

VPN Setup

1. **VPN Server Deployment**: A **WireGuard** VPN server is set up on the DigitalOcean droplet.
2. **Client Configuration**: Only clients with the correct configuration can access the server via the VPN.
3. **Encryption**: Traffic between the client and the server is encrypted, enhancing security.

Firewall Configuration

1. **Blocking Public Access**: A custom firewall on DigitalOcean is configured to block all incoming traffic except for VPN connections.
2. **Internal Networking**: Once connected to the VPN, clients can securely access the API and database.
3. **IP Whitelist:** the API’s IP is whitelisted for the DB, so it can retrieve data

1.3 Nginx Reverse Proxy Setup

Purpose

Nginx acts as a reverse proxy to handle incoming client requests efficiently and securely. It provides a layer of abstraction and control over the backend application while enabling features like SSL/TLS encryption, load balancing, and access control.

SSL/TLS Setup

1. **SSL Certificates**
    - **Certificate Authority**: SSL/TLS certificates are obtained from **Let’s Encrypt** using the **Certbot** utility.
    - **Automation**: Certbot is configured to renew certificates automatically, ensuring continuous HTTPS availability.
    - **Integration with Nginx**: The Nginx configuration automatically picks up renewed certificates without manual intervention.
2. **Redirection**
    - HTTP requests are redirected to HTTPS using Nginx’s `return 301 https://$host$request_uri` directive. This enforces secure communication by default.

Domain Configuration

1. **Domain Management**
    - The application domain is registered through **Namecheap**, which manages DNS records.
    - **DNS Configuration**:
        - An **A record** is set up to point the domain to the public IP address of the DigitalOcean droplet hosting the application.
2. **SSL Validation**
    - **Let's Encrypt** uses DNS-based or HTTP-based challenges to verify domain ownership before issuing SSL certificates.