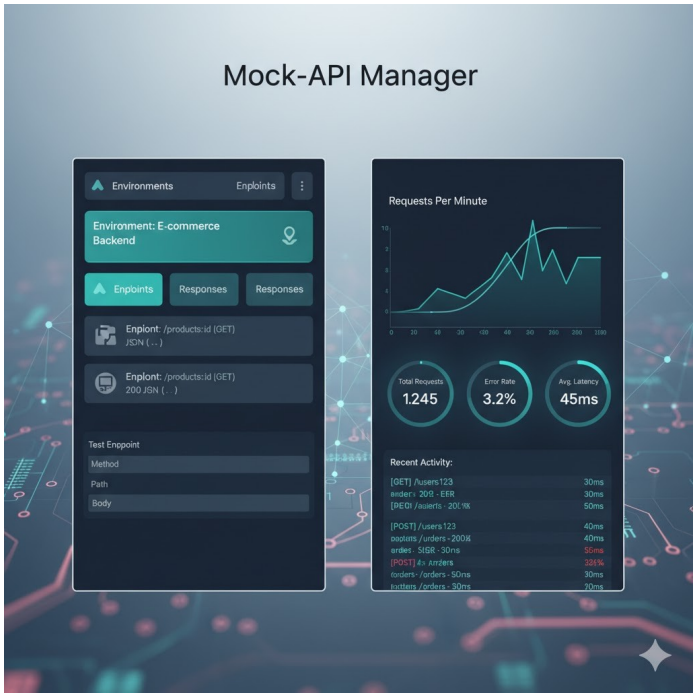

📚 Mock-API Manager: Complete Documentation

The Mock-API Manager is a powerful yet simple tool designed to create, manage, and monitor mock API endpoints for testing and development environments.

1. Core Architecture & Technology Stack

Component	Technology/Tool	Role
Backend	**Node.js** with **Express**	Handles all API logic, data persistence, and dynamic routing for mock endpoints.
Database	**PostgreSQL** (Inferred)	Stores all configuration data (Environments, Endpoints, Responses) and request logs.
Frontend	**PHP** & **JavaScript**	Provides the user interface for managing configurations and viewing monitoring data.
Security	**Helmet**	Secures the Express app by setting various HTTP headers.
Logging	**Morgan**	Logs HTTP requests to the console.
Utilities	**CORS**, **Body-Parser**, **Dotenv**	Standard middleware for security, cross-origin requests, parsing bodies, and environment variables.

---

2. Key Features

The manager provides a comprehensive set of features across API configuration and live monitoring.

A. API Configuration (api.js, index.php, script.js)

The manager offers full **CRUD (Create, Read, Update, Delete)** functionality for the core entities: **Environments**, **Endpoints**, and **Responses**.

Feature	**Description**	**API Routes (/api/...)**
Environments	Grouping for mock APIs. Can be configured with a **port**, **prefix**, **latency**, and **CORS/HTTPS** settings.	GET, POST, PUT, DELETE /environments
Endpoints	Defines a specific mock route using a **path** (e.g., /users/:id) and **HTTP method** (GET, POST, etc.). Can have an optional default **status code** and **headers**.	GET, POST /environments/:id/endpoints GET, PUT, DELETE /endpoints/:id
Responses	Defines the actual data returned by an Endpoint. Includes **status code**, **body** (JSON), **headers**, optional **latency**, and can be configured with **rules** (for dynamic mocking) and a **probability** (for testing randomness/failures).	GET, POST /endpoints/:id/responses GET, PUT, DELETE /responses/:id
Dynamic Mocking	A special route (/api/mock/:environmentId/*) is dedicated to serving the mock responses based on the configuration of the corresponding environment and endpoint, enabling dynamic behavior.	ALL /mock/:environmentId/*

B. Data & Utility Management (api.js, monitoring.php)

Feature	Description	API Routes (/api/...)
Request Logging	Every request to the management API routes and the dynamic mock routes is logged to the database, capturing **method**, **path**, **headers**, **body**, **status**, **latency**, and more.	GET /environments/:id/logs GET /logs/:id
Settings	Provides persistent storage for key/value application settings.	GET /settings, GET /settings/:key, POST /settings
Health Check	Verifies the status of the server and the **PostgreSQL database connection**.	GET /health (in server.js and api.js)
System Info	Gathers counts of environments, endpoints, responses, and request logs for dashboard display. Also provides server details like Node version and uptime.	GET /info

---

3. Ease of Use & Simplicity (Dashboard Frontend)

The application provides a two-page web interface designed for simplicity: the **Dashboard** (index.php) for management and the **Monitoring page** (monitoring.php) for live insights.

A. Management Dashboard (index.php, script.js)

The dashboard is structured around **Environments**, **Endpoints**, and **Responses**, with a clear sidebar for quick actions.

- **Scannable Stats:** The top of the page immediately displays **total counts** for environments, endpoints, responses, and request logs.
- **Environment-First Design:** The workflow guides the user to **select an environment** first, then view and manage its specific endpoints.
- **Simplified Forms:** Dedicated, straightforward forms for creating **Endpoints** and **Responses** streamline the process.
    - **Endpoint Form:** Requires Path, Method, Name, and Status Code.

    - **Response Form:** Requires Name, Status Code, and the Response Body (JSON).
- **Integrated Testing:** A dedicated "Test Endpoint" form in the sidebar allows users to immediately test a mock API call with a specified **path**, **method**, and **request body**, with the results displayed directly on the page.
- **One-Click Actions:** Buttons are provided for **refreshing** data and **deleting** items directly within the environment and endpoint lists.

B. Live Monitoring Page (monitoring.php)

The monitoring page focuses on providing a **"100% Real Data"** view of the mock API traffic, leveraging the request logs.

- **Key Performance Indicators (KPIs):** Displays crucial metrics based on **total request logs**:
    - **Configured Endpoints** vs. **Active Endpoints**
    - **Total Requests** and **Total Errors**
    - Overall **Error Rate**
    - **Requests Per Minute** (calculated from the last hour of logs)
- **Endpoint Health Status:** This section is the core of the monitoring. For every *configured* endpoint, it calculates:
    - **Total Requests**
    - **Error Rate**
    - **Average Response Time (Latency)**
    - **Health Status** badge: HEALTHY, SOME ERRORS (Error Rate > 5%), HIGH ERRORS (Error Rate > 10%), or NO DATA.
- **Recent Activity Feed:** A live-style feed that scrolls through the **most recent 50 requests**, displaying the **Method**, **Path**, **Status**, **Latency**, and **Log Type** (API route vs. Mock route). This offers an immediate look at traffic.
- **System Overview:** Provides a summary card of the overall API and data status.

---

4. Technical Simplicity & Extensibility

- **Modular Routing:** The server.js (main application) and api.js (API routes) clearly separate the concerns of application setup, middleware, and route handling.
- **Clear Error Handling:** The application includes explicit **404 handlers** for both the main app and API routes, as well as a general **500 error handler** in server.js, improving robustness and debugging.
- **Non-Blocking Logging:** The logApiRequest function in api.js is designed to be efficient by intentionally **not awaiting** the database insert for logging, ensuring the actual API response is not delayed by disk I/O.
- **PostgreSQL Support:** The use of PostgreSQL provides a **robust and scalable** database for handling configuration and potentially millions of request logs.