This is a **Cloud Native Observability Platform API**, which collects **metrics, logs, and alerts** from services, allowing organizations to monitor application performance, detect issues, and authenticate access using JWT login.

Your API currently supports 4 main domains:

Module	Purpose
Auth	Login and issue JWT tokens (no sign-up — assumes zero trust secured environment)
Metrics	Store, retrieve, and analyse system performance metrics
Logs	Store and read application logs
Alerts	Manage alert rules and trigger incidents

Auth Endpoints

Endpoint	Method	What it does
/v1/auth/login	**POST**	Takes login credentials (email + password), validates user, returns JWT token. Required to access secured resources.

Used for authentication.

No registration endpoint — meaning users exist beforehand (admin-created).

Metrics Module (App/Server performance data)

Metrics represent **numeric measurements** like CPU%, memory, requests per second, latency, etc.

Endpoint	Method	Explanation
/api/Metrics	**POST**	Services send metric data here to be stored — ingest endpoint.
/api/Metrics	**GET**	Retrieve list of metrics. Supports filtering by service.
/api/Metrics/{id}	**GET**	Fetch a single metric entry using its ID.
/api/Metrics/aggregate	**GET**	Returns summary stats — avg/min/max metrics over a period. Used for dashboards & trend analysis.

How to explain:
"Metrics allow us to track performance over time. We can push metrics, view them, or retrieve aggregated analytics for dashboards like Grafana/Datana style graphs."

Logs Module (Application logs and events)

Logs store **text-based runtime information**: warnings, errors, exceptions, system events.

Endpoint	Method	Explanation
/api/Logs	**POST**	Apps push log entries to observability backend.
/api/Logs	**GET**	Fetch all logs. Can later support filtering by level/service/date.
/api/Logs/{id}	**GET**	Retrieve details of a single log item — useful for investigation.

How to explain:
"Logs help engineers trace issues. If something breaks, logs tell us what happened, where, and when."

Alerts Module

Alerts notify when something abnormal happens (high CPU, too many errors, memory leak, downtime, etc).

Endpoint	Method	Explanation
/api/Alert	**GET**	Get list of all configured alerts.
/api/Alert	**POST**	Create a new alert rule or incident.
/api/Alert/{id}	**GET**	Read a specific alert rule/incident.
/api/Alert/{id}	**PUT**	Update alert rule — thresholds, severity changes, etc.
/api/Alert/{id}	**DELETE**	Remove alert rule.

How to explain:
"Alerts notify teams when metrics or logs show danger signals. This prevents downtime and speeds up incident response."

Example — Explaining to a Recruiter

"My API functions as an observability backend similar to Datadog or Prometheus.
Services send **metrics** and **logs**, which are stored and retrievable.
There is an **alerts system** that triggers when values exceed thresholds.
Authentication is handled using **JWT login** for secure access."

Next Steps You Can Add Later (for stronger portfolio)

Feature	**Why it's valuable**
Pagination & filtering in logs	Logs can be huge — filtering matters
Role-based admin controls	Enterprise-grade auth
WebSocket/SignalR real-time dashboards	Live metric charts
Notification channels (Email/Slack/SMS)	Alerts become actionable
Service-to-service API keys	Allows microservices to push data securely

WHAT TO ADD LATER

The only APIs you *might* add later (optional, not required now)

Not needed today — but good future upgrades:

Optional Future API	Why
WebSocket/SignalR realtime metrics	Live dashboard with live charts
User management (admin CRUD)	If you want signup or role-based access
Search/filter endpoints for logs & metrics	Makes dashboards faster & usable at scale
API key issuing for microservices	Machine-to-machine authentication
Alerts notification dispatching	Slack/email/webhook alerts