ChatGPT

Gateway functions implemented by FountainAi (Codex Deployer)

FountainAi (also known as the **codex-deployer**) is built around a plugin-driven **GatewayApp**. The gateway exposes HTTP endpoints and composes numerous plugins to enforce security and resilience. Each plugin implements the `GatewayPlugin` protocol, whose two methods allow hooks into request and response processing. Below is a breakdown of the gateway's public functions and their responsibilities.

GatewayPlugin protocol

- **prepare(_:)** – The `GatewayPlugin` protocol defines a `prepare` method that allows plugins to mutate or inspect an incoming request before it is routed. The default implementation simply returns the request unchanged [1].
- **respond(_:for:)** – After a request is handled, the `respond` method allows a plugin to inspect or modify the outgoing response. The default implementation returns the response unchanged [2].

These methods form the extension points used by every plugin in the gateway.

AuthPlugin

`AuthPlugin` enforces bearer-token authorization on management endpoints:

- **init(validator:protected:)** – Constructs the plugin with a `TokenValidator` (defaults to `CredentialStoreValidator`) and a map of path prefixes to required roles or scopes. By default, the `/metrics`, `/certificates`, `/routes`, and `/zones` paths require an `admin` role [3].
- **prepare(_:)** – Validates the `Authorization: Bearer` header for requests whose path matches a protected prefix. It extracts the token, uses the configured validator to obtain the token's claims and checks that the token's scopes or role satisfy the required role. If validation succeeds, it adds an `X-User-Role` header and forwards the request; otherwise it throws `UnauthorizedError` or `ForbiddenError` [4].

BudgetBreakerPlugin

The **BudgetBreakerPlugin** is an asynchronous actor that implements rate-limiting and circuit-breaking per user and route:

- **init(defaultBudget:failureThreshold:baseBackoff:)** – Sets a default request budget (requests per minute), a failure threshold before the circuit opens and a base back-off period [5].
- **updateHealth(isHealthy:)** – Updates an internal `healthy` flag, allowing external health signals to trigger load-shedding [6].
- **stats()** – Returns counters for the number of allowed and throttled requests [7].

- **prepare(_:)** – Implements a token-bucket algorithm. If the system is unhealthy or the circuit is open, it increments the throttled counter and throws `ServiceUnavailableError`. Otherwise it checks whether the caller's bucket contains at least one token and decrements it; if not it throttles the request and throws `TooManyRequestsError` [8].
- **respond(_:for:)** – Updates the circuit breaker after each response. Server-error responses (≥ 500) increment a failure count; once the failure threshold is reached, the circuit opens and schedules a retry using exponential back-off. Successful responses reset the breaker [9].

CertificateManager

`CertificateManager` orchestrates periodic TLS certificate renewal via an external script:

- **init(scriptPath:interval:)** – Specifies the path of the renewal script and the repeat interval (default is 86 400 seconds, i.e., one day) [10].
- **start()** – Starts a timer that invokes the renewal script at the configured interval using a background queue [11].
- **stop()** – Cancels the timer to stop future renewals [12].
- **triggerNow()** – Immediately runs the renewal script once outside of the regular schedule [13].

CoTLogger

The **CoTLogger** plugin records chain-of-thought (CoT) reasoning when clients request it and optionally consults the security sentinel:

- **init(logURL:sentinel:patterns:)** – Configures a log file and optionally a `SecuritySentinelPlugin` for risk checking. The `patterns` parameter lists keywords that trigger a sentinel consult when found in a CoT [14].
- **respond(_:for:)** – Activated for `/chat` responses when the request contains `include_cot: true`. It extracts the `cot` and `id` fields from the JSON response, sanitizes sensitive strings (e.g., "secret," "password"), and writes the sanitized CoT to a log file. If configured, it calls `sentinel.consult` when the CoT contains risky keywords and records the sentinel's decision [15]. The method returns the original response unchanged.

CredentialStore

`CredentialStore` manages client credentials and JWT signing:

- **init(environment:)** – Loads client credentials (via `GATEWAY_CRED_*` environment variables), roles (`GATEWAY_ROLE_*`) and a JWT signing secret (`GATEWAY_JWT_SECRET`) [16].
- **validate(clientId:clientSecret:)** – Returns `true` when the provided credentials match the stored values [17].
- **role(forClientId:)** – Retrieves the configured role for a client identifier [18].
- **signJWT(subject:expiresAt:role:)** – Creates a signed JWT using HMAC-SHA256. It encodes header and payload (subject, expiry and optional role), signs the concatenated string and returns the token [19].
- **verify(jwt:)** – Validates the JWT signature using the secret and checks that the expiry time has not passed [20].
- **role(for:)** – Extracts the role claim from a verified JWT [21].
- **subject(for:)** – Extracts the subject claim from a verified JWT [22].

DestructiveGuardianPlugin

The **DestructiveGuardianPlugin** prevents accidental destructive operations:

- **init(sensitivePaths:privilegedTokens:auditURL:)** – Defines the request paths considered sensitive (default is the root path so all deletes, puts and patches are protected), a list of privileged tokens that can bypass manual approval and an audit log location [23].
- **prepare(_:)** – For DELETE, PUT or PATCH requests to sensitive paths, it checks for an `X-Manual-Approval` header or a valid `X-Service-Token`. The decision (allow or deny) and reason are appended to an audit log. If neither approval nor a privileged token is present, the plugin throws `GuardianDeniedError` [24].

GatewayConfig

`GatewayConfig` holds run-time configuration for the gateway:

- **init(rateLimitPerMinute:)** – Stores the default per-client request limit (defaults to 60 requests per minute) [25].
- **loadGatewayConfig()** – Loads the configuration from `Configuration/gateway.yml`. It removes copyright lines to keep the YAML parseable, merges the loaded values with defaults, and decodes the result into a `GatewayConfig` instance [26].

GatewayServer

`GatewayServer` assembles the HTTP server, registers plugins and exposes numerous management endpoints. Each public method corresponds to a gateway function:

- **init(manager:plugins:zoneManager:routeStoreURL:certificatePath:rateLimiter:)** – Builds a server with a certificate manager, a list of plugins, an optional DNS zone manager, optional path to persist routes, optional certificate path for TLS inspection and an optional `RateLimiterPlugin`. It also preloads persisted routes via `reloadRoutes` [27].
- **gatewayHealth()** – Returns a JSON object `{ "status": "ok" }` for health monitoring [28].
- **gatewayMetrics()** – Asynchronously collects metrics from `DNSMetrics.shared`, parses the Prometheus-style exposition into a dictionary and returns it as JSON [29].
- **issueAuthToken(_ request)** – Accepts a JSON body containing `clientId` and `clientSecret`, validates the credentials and signs a JWT that expires in one hour. It returns a token and expiry time or an error status if the credentials are invalid or if signing fails [30].
- **certificateInfo()** – Reads the certificate file (in PEM or DER format) referenced by the server's `certificatePath`, extracts the not-after expiry date and issuer, and returns them as JSON. Returns `404` if the file is missing or `500` on errors [31].
- **renewCertificate()** – Invokes `CertificateManager.triggerNow()` to run the renewal script immediately and returns a `202` response with `{ "status": "triggered" }` [32].
- **listRoutes()** – Returns a JSON array describing all configured proxy routes [33].
- **createRoute(_ request)** – Accepts a JSON body describing a `RouteInfo` (id, path, target URL, HTTP methods, optional rate limit and proxy flag). It validates that the methods belong to the allowed set and that the id is not already in use, adds the route and persists it. Returns `201` on success, `409` if the id exists, or `400` on malformed input [34].
- **updateRoute(_ routeId, request)** – Updates an existing route with new path, target, methods, rate limit and proxy flag. It returns the updated route or appropriate error responses [35].

- **deleteRoute(_ routeId)** – Removes a route identified by its id. Returns `204` when deleted or `404` with an error message if not found [36].
- **reloadRoutes()** – Reloads the persisted route definitions from disk (if configured). Errors are logged to standard error but do not throw [37].
- **createZone(_ request)** – When a DNS zone manager is supplied, this method decodes a JSON payload with a zone name and calls `ZoneManager.createZone`; it returns the created zone or an error message [38].
- **deleteZone(_ zoneId)** – Converts the zoneId string to a UUID and calls `ZoneManager.deleteZone`. Returns `204` on success or an error response on failure [39].
- **listRecords(_ zoneId)** – Lists DNS records in the specified zone. Returns JSON with the records or error messages when the zone is missing or encoding fails [40].
- **createRecord(_ zoneId, request)** – Creates a DNS record from the JSON body (name, type and value). Returns the created record or appropriate error messages [41].
- **updateRecord(_ zoneId, recordId, request)** – Updates an existing DNS record using the supplied JSON definition. Returns the updated record or error messages [42].
- **deleteRecord(_ zoneId, recordId)** – Deletes the specified record from a zone and returns `204` or error messages [43].
- **start(port:)** – Starts the HTTP server on the given port and begins certificate renewal scheduling [44].
- **stop()** – Gracefully stops the server and cancels certificate renewal [45].

Request routing logic

`GatewayServer` contains a private `tryProxy` method that attempts to match incoming requests against configured routes. It performs a prefix match on path and method, enforces per-route rate limits using `RateLimiterPlugin.allow`, builds the upstream URL, forwards the request using `URLSession` and returns the upstream response or a `502` error [46]. The server also includes an `error(status:message:)` helper for creating JSON error responses [47].

LoggingPlugin

- **init()** – Creates an instance of the plugin [48].
- **prepare(_:)** – Prints the method and path of each incoming request to standard output and forwards the request unchanged [49].
- **respond(_:for:)** – Logs the HTTP status and path of each response [50].

PayloadInspectionPlugin

This plugin performs deep inspection of request and response bodies to enforce content policies:

- **init(allowList:denyList:transformations:logURL:)** – Defines regex patterns that allow or deny content and optional transformations; also accepts a log file path [51].
- **prepare(_:)** – Inspects the request body: if the body is non-empty, it converts it to text and scans for deny patterns. If a pattern matches and is not allowed, the plugin either replaces it using the `transformations` map or logs the violation and throws `PayloadRejectedError` [52].
- **respond(_:for:)** – Applies the same inspection to the response body and returns the sanitized or unmodified response [53].

PublishingFrontendPlugin

This plugin acts as a static file server for unhandled GET requests:

- **init(rootPath:)** – Points the plugin at the directory containing static files (defaults to `./Public`) [54].
- **respond(_:for:)** – For GET requests that resulted in a 404, it attempts to read a file corresponding to the request path (mapping `/` to `index.html`), determines the MIME type and returns a `200` with the file contents. If no file exists it returns the original response [55].

RateLimiterPlugin

`RateLimiterPlugin` implements a per-route, per-client token-bucket rate limiter:

- **init(defaultLimit:)** – Sets the fallback limit per minute when a route lacks an explicit limit [56].
- **stats()** – Returns counters for allowed and throttled requests [57].
- **allow(routeId:clientId:limitPerMinute:)** – Checks whether the caller may make a request. It refills the bucket based on elapsed time, consumes a token if available and records metrics via `DNSMetrics.shared`. Returns `true` when allowed and `false` when throttled [58].

SecuritySentinelPlugin

This plugin integrates with an external sentinel service to vet potentially destructive operations:

- **init(client:logURL:patterns:)** – Configures an API client, a log file and a list of path substrings considered destructive [59].
- **consult(summary:user:resources:)** – Sends a POST request to `/sentinel/consult` with a summary, user and list of resources. It logs the sentinel's decision and returns it [60].
- **prepare(_:)** – Intercepts requests with method DELETE or whose path contains one of the configured destructive patterns. It constructs a summary string, determines the user from the `X-User` header (or uses "anonymous"), consults the sentinel and either allows the request or throws `DeniedError` or `EscalateError` based on the sentinel's decision [61].

Token validation

The gateway defines a `TokenValidator` protocol and two implementations:

- **TokenValidator.validate(token:)** – Interface for asynchronous token validation returning optional `TokenClaims` [62].
- **CredentialStoreValidator** – Uses `CredentialStore` to verify a JWT, extract the role claim and return the claims as scopes. Returns `nil` for invalid tokens [63].
- **OAuth2Validator** – Validates an OAuth2 token by POSTing to an introspection endpoint. It supports HTTP basic authentication using the configured client ID and secret and decodes the `active`, `scope` and `role` fields. It returns `TokenClaims` when the token is active, otherwise `nil` [64].

Summary

The **codex-deployer** gateway exposes health and metrics endpoints, supports issuing JWTs for clients, manages certificate renewal, maintains a dynamic routing table with persistence, proxies requests to upstream targets, and offers DNS management endpoints. Plugins supply middleware functionality: authorization, rate limiting, circuit breaking, payload inspection, chain-of-thought logging, static file serving and integration with a security sentinel. These functions collectively form the gateway capabilities of FountainAi.

---

[1] [2] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/GatewayPlugin.swift

[3] [4] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/AuthPlugin.swift

[5] [6] [7] [8] [9] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/BudgetBreakerPlugin.swift

[10] [11] [12] [13] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/CertificateManager.swift

[14] [15] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/CoTLogger.swift

[16] [17] [18] [19] [20] [21] [22] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/CredentialStore.swift

[23] [24] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/DestructiveGuardianPlugin.swift

[25] [26] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/GatewayConfig.swift

[27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/GatewayServer.swift

[48] [49] [50] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/LoggingPlugin.swift

[51] [52] [53] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/PayloadInspectionPlugin.swift

[54] [55] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/PublishingFrontendPlugin.swift

[56] [57] [58] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/RateLimiterPlugin.swift

[59] [60] [61] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/SecuritySentinelPlugin.swift

[62] [63] [64] raw.githubusercontent.com
https://raw.githubusercontent.com/Fountain-Coach/codex-deployer/main/Sources/GatewayApp/TokenValidation.swift