zapier.georgelambert.org · Markdown indexes
VERAE TIME × ZAPIER
function reference
site/docs/developer/modules/function-reference.md

Function Reference — All Modules

Canonical I/O contracts for the Verae Zapier middleware and Zapier app.
Source of truth for reviewers; keep in sync with JSDoc in code.

Namespaces for debug: `auth`, `billing`, `http`, `nats`, `jobs`, `webhooks`, `trace`, `app`.

---

config.js

`loadEnvFile()`

For	Load `.env` into `process.env` if keys are unset
Input	none (reads file next to package root)
Output	`void`

`config` (exported object)

Field	Type	Description
`port`	number	HTTP port
`host`	string	Bind address
`veraeApiBaseUrl`	string	Upstream Verae base URL (no trailing slash)
`mockVerae`	boolean	Use mock client
`natsEnabled`	boolean	Use JetStream workers
`natsUrl`	string	NATS connection URL
`tokenSecret`	string	HMAC secret for `zmt_` tokens
`jobPollIntervalMs`	number	Poll delay
`jobPollMaxAttempts`	number	Max poll attempts
`storePath`	string	Absolute path to store file
`upgradeUrl`	string	Billing upgrade link for 402 bodies
`adminSecret`	string	Admin route secret

`PLAN_LIMITS`

Map of plan name → `{ timestamps, verifications, batch, batchMaxItems, requestsPerMinute }`.
`null` means unlimited.

---

errors.js

`class AppError extends Error`

For	Structured operational errors returned as JSON
Constructor	`(message: string, { status?, code?, details? })`
Properties	`status: number`, `code: string`, `details: unknown`

`asyncHandler(fn)`

For	Wrap async Express handlers so rejections hit error middleware
Input	`(req, res, next) => Promise<any>`
Output	Express middleware function

`sendError(res, err)`

For	Write JSON error response; log 5xx with debug
Input	Express `res`, `Error|AppError`
Output	`void`

app.js / index.js

`createApp()`

For	Build Express application (no listen)
Input	none
Output	`express.Application`
Mounts	`GET /health`, `/zapier/*`, error handler, `traceMiddleware`

`main()` (index.js)

For	Boot store, optional NATS, workers, listen
Input	none
Output	`Promise<void>`

---

store/db.js

`loadStore()`

For	Load JSON store into memory
Output	store object

`getStore()`

For	Access in-memory store
Output	`{ tenants, apiKeys, usage, webhooks, jobWatchers }`

`persist()`

For	Flush store to disk
Output	`void`

---

store/tenants.js

`getTenant(tenantId)`

Input | `tenantId: string` |
Output | `Tenant\|null` |

`getTenantByApiKey(apiKey)`

Input | `apiKey: string` |
Output | `Tenant\|null` |

`listTenants()`

Output | `Tenant[]` |

`createTenant({ id, name, plan, veraeUsername, veraePassword, contract?, apiKey?, metadata? })`

Output | `{ tenant: Tenant, apiKey: string }` |

`resolveLimits(tenant)`

Output | Effective limits including enterprise contract overrides |

---

store/usage.js

`getUsage(tenantId)`

Output | counters for current period |

`getUsageSummary(tenantId)`

Output | public-safe usage + limits snapshot |

`incrementUsage(tenantId, metric, amount?)`

For | Atomically increment a counter and persist |

---

store/webhooks.js

`createWebhook({ tenantId, targetUrl, event })`

Output | `{ id, tenantId, targetUrl, event, createdAt }` |

`deleteWebhook({ tenantId, hookId?, targetUrl? })`

Output | `boolean` removed |

`getActiveWebhooks(tenantId, event)`

Output | `Webhook[]` |

`listWebhooksForTenant(tenantId)`

Output | `Webhook[]` |

---

lib/tokens.js

`issueSessionToken({ tenantId, veraeToken, expiresAt })`

Output | `string` (`zmt_…`) |

`parseSessionToken(token)`

Output | `{ tenantId, veraeToken, expiresAt, nonce }\|null` |

`generateApiKey()`

Output | `string` (`zmw_…`) |

`isApiKey(value)`

Output | `boolean` |

`extractBearerToken(header)`

Input | `Authorization` header string |
Output | token string or `null` |

---

clients/veraeClient.js

All methods that hit the network log under `DEBUG_VERAE=http`.

`veraeClient.login(credentials)`

Input | `{ username, password }` |
Output | `{ token, expiresAt, user }` |

`veraeClient.validate(token)`

Input | Verae JWT |
Output | validation object |

`veraeClient.createTimestamp(token, body)`

Input | JWT, `{ data, hashAlg? }` |
Output | `{ jobId }` |

`veraeClient.createBatchTimestamp(token, body)`

Input | JWT, `{ items: [{ data, hashAlg? }] }` |
Output | `{ jobIds: string[] }` |

`veraeClient.getStatus(token, jobId)`

Output | StatusResponse |

`veraeClient.getBatchStatus(token, body)`

Input | `{ jobIds: string[] }` |
Output | `{ results }` |

`veraeClient.verify(token, body)`

Input | `{ certificate }` |
Output | `{ valid, timestamp?, blockIndex? }` |

`veraeClient.verifyBatch(token, body)`

Input | `{ certificates: string[] }` |
Output | `{ results }` |

`veraeClient.waitForJob(token, jobId, { maxAttempts, intervalMs })`

Output | terminal StatusResponse or throws `GATEWAY_TIMEOUT` |

---

services/authService.js

Debug namespace: `auth`.

`loginWithCredentials({ username, password, tenant? })`

Output | `{ accessToken, expiresAt, tenant, user }` |

`loginWithApiKey(apiKey)`

Output | same as loginWithCredentials after tenant lookup |

`resolveAuthContext(rawToken)`

Input | API key or session token |
Output | `{ tenantId, tenant, veraeToken, authMethod }` |

`validateSession(rawToken)`

Output | `{ valid, tenantId, authMethod, user }` |

---

services/entitlementService.js

Debug namespace: `billing`.

`checkEntitlement(tenantId, action, { amount? })`

Actions | `timestamp`, `verify`, `batch_timestamp` |
Throws | `402 QUOTA_EXCEEDED`, `403 PLAN_UPGRADE_REQUIRED` |
Output | `{ tenant, limits, usage }` |

`recordUsage(tenantId, action, { amount? })`

Output | `void` |

---

middleware/authenticate.js

`authenticate(req, res, next)`

For | Populate `req.auth` via `resolveAuthContext` |
Reads | `Authorization: Bearer` or `x-api-key` |

middleware/rateLimit.js

`rateLimit(req, res, next)`

For | Enforce plan `requestsPerMinute` |
Throws | `429 RATE_LIMITED` |

---

services/timestampService.js

Debug: `jobs`, `billing`, `http`.

`createTimestamp(ctx, body)`

Input | auth context, `{ data, hashAlg? }` |
Output | `{ jobId }` |
Side effects | usage++, enqueue watch (NATS or in-process) |

`createTimestampAndWait(ctx, body)`

Output | StatusResponse (or pending on timeout when NATS wait enabled) |

`createBatchTimestamp(ctx, body)`

Output | `{ jobIds }` |

`getJobStatus(ctx, jobId)` / `getBatchJobStatus` / `getJobVerification`

Output | status payloads from Verae |

---

services/verifyService.js

`verifyTimestamp(ctx, body)` / `verifyBatch(ctx, body)`

Output | verify results; records usage |

---

services/webhookService.js

Debug: `webhooks`.

`subscribe(ctx, { targetUrl, event })`

Output | webhook record |

`unsubscribe(ctx, { hookId?, targetUrl? })`

Output | `{ removed: true }` |

`deliverWebhook(targetUrl, payload)`

Output | `{ ok: boolean, status: number }` |

---

services/tenantService.js

`selfServeSignup({ email, name, veraeUsername, veraePassword })`

Output | `{ tenant, apiKey, zapierSetup }` |

`provisionTenant({ id?, name, plan, veraeUsername, veraePassword, contract?, metadata?, audience? })`

Output | `{ tenant, apiKey }` |

`listProvisionedTenants()`

Output | public tenant summaries (no passwords) |

---

nats/subjects.js

Constants

`SUBJECTS.JOBS_WATCH`, `JOBS_EVENTS`, `WEBHOOKS_DELIVER`, `USAGE`
`STREAMS.ZAPIER_JOBS`, `ZAPIER_EVENTS`, `ZAPIER_WEBHOOKS`

---

nats/connection.js

`connectNats(url?)`

Output | `{ nc, js, jsm }` NATS connection handles |

`ensureStreams(jsm)`

For | Idempotent stream create |
Output | `Promise<void>` |

`closeNats()`

Output | `Promise<void>` |

---

nats/publishers.js

`enqueueWatch(jobWatchPayload)`

Input | see architecture/nats-subjects.md |
Output | `Promise<{ seq }>` |

`publishJobEvent(eventPayload)`

Output | `Promise<void>` |

`enqueueWebhook(deliverPayload)`

Output | `Promise<void>` |

---

workers/jobPollerWorker.js

`startJobPollerWorker()`

For | Pull-consume watch queue; poll Verae; emit events |
Output | stop handle `{ stop() }` |

workers/webhookWorker.js

`startWebhookWorker()`

For | Deliver POSTs to Zapier with retry via JetStream |
Output | `{ stop() }` |

workers/inProcessJobPoller.js

`startInProcessJobPoller()` / `stopInProcessJobPoller()`

For | Phase 6 fallback when `NATS_ENABLED=false` |

Routes (all under `/zapier`)

Method	Path	Handler purpose
POST	`/v1/auth/login`	Credentials or api_key → session
GET	`/v1/auth/me`	Validate connection for Zapier
POST	`/v1/timestamp`	Async create
POST	`/v1/timestamp/wait`	Create and wait
POST	`/v1/timestamp/batch`	Batch create
POST	`/v1/verify`	Verify certificate
GET	`/v1/status/:jobId`	Job status
POST	`/v1/webhooks/subscribe`	REST Hook subscribe
DELETE	`/v1/webhooks/unsubscribe`	REST Hook unsubscribe
POST	`/v1/signup`	Self-serve tenant
POST	`/v1/admin/tenants`	Admin provision

Zapier app modules

`authentication.test` / fields

Input from user | `api_key` |
Test URL | `GET {MIDDLEWARE_BASE_URL}/zapier/v1/auth/me` |

`beforeRequest: addApiKey`

Sets | `Authorization: Bearer ${bundle.authData.api_key}` |

Creates

Key	Middleware path
`timestamp_and_wait`	POST `/zapier/v1/timestamp/wait`
`create_timestamp`	POST `/zapier/v1/timestamp`
`verify_timestamp`	POST `/zapier/v1/verify`
`batch_timestamp`	POST `/zapier/v1/timestamp/batch`

Search `job_status`

Path | GET `/zapier/v1/status/{jobId}` |

Trigger `timestamp_completed`

Subscribe | POST `/zapier/v1/webhooks/subscribe` |
Unsubscribe | DELETE `/zapier/v1/webhooks/unsubscribe` |
Perform | return `[bundle.cleanedRequest]` |