

Intacct API Usage - Technical Documentation

Table of Contents

1. [Overview](#)
2. [Architecture](#)
3. [Platform Compatibility](#)
4. [Project Structure](#)
5. [Backend Components](#)
 - [Server & Routes](#)
 - [Services](#)
 - [Database Layer](#)
6. [Frontend Components](#)
 - [Tab System](#)
 - [JavaScript Modules](#)
 - [HTML Partial](#)s
7. [Theming System](#)
8. [Data Storage & Encryption](#)
9. [Key Workflows](#)
 - [Manual Data Fetch](#)
 - [Scheduled Jobs](#)
 - [Report Generation](#)
 - [Notification System](#)
10. [Configuration](#)
11. [API Reference](#)
12. [Security Considerations](#)
13. [Troubleshooting](#)
14. [Dependencies](#)

Overview

Intacct API Usage is a Node.js web application that monitors and reports on Sage Intacct API usage across multiple company instances. The application queries the Intacct APIUSAGEDETAIL object to collect usage metrics, stores them in a local SQLite database, and provides reporting through a web interface with optional email notifications.

Key Features

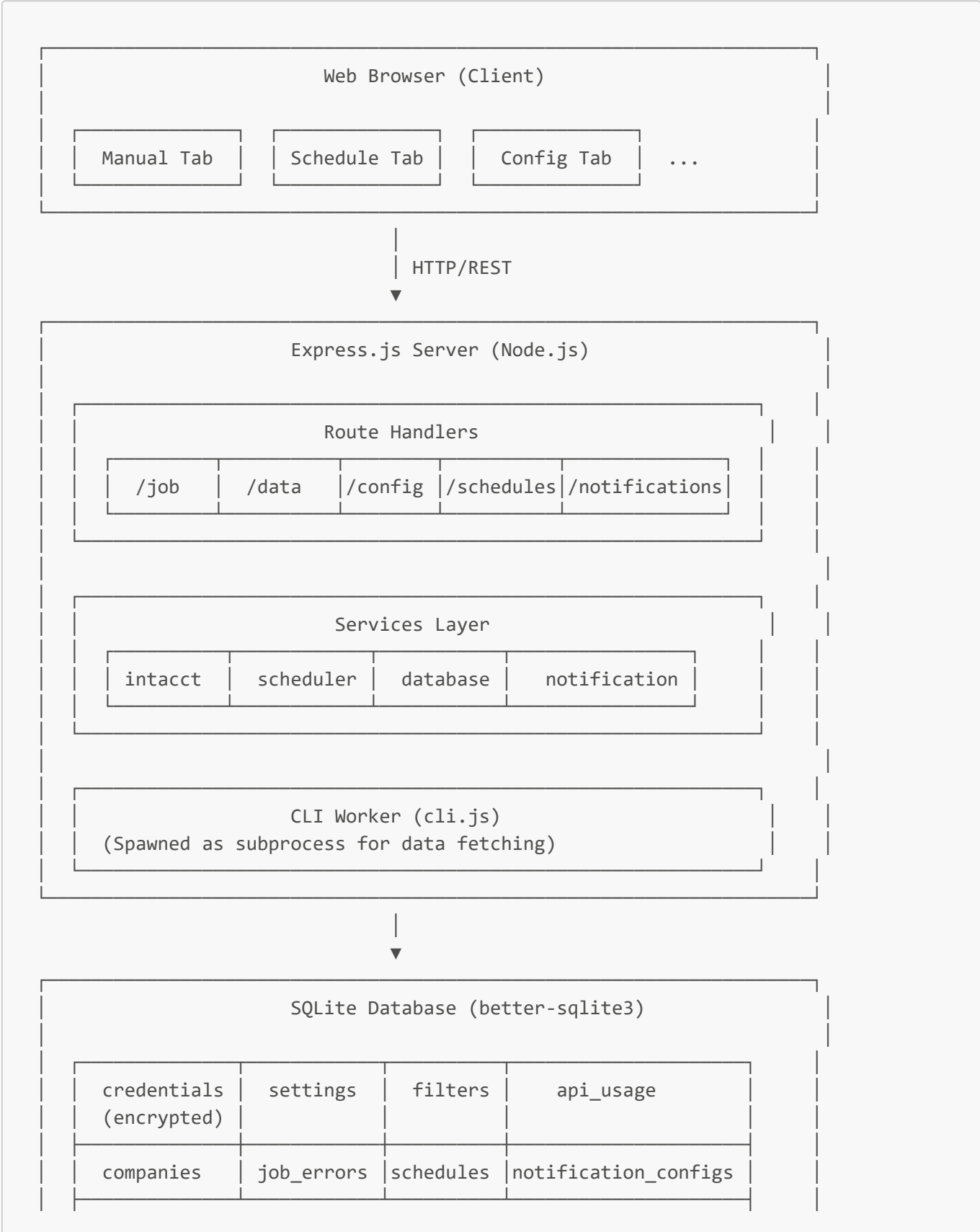
- **Multi-Company Support:** Query API usage data across multiple Intacct company instances
- **Filter-Based Queries:** Configure custom filters based on DOCCONTROLID prefixes (e.g., EC_ for ExcelConnect, FUSAPI_ for Fusion API)
- **Scheduled Reports:** Automated data fetching and email report delivery on configurable schedules
- **Multiple Notification Providers:** Support for SMTP and Microsoft Graph email delivery
- **Excel/CSV Export:** Download detailed, summary, and exception reports in multiple formats
- **Company Management:** Upload company lists via CSV or load dynamically from external APIs

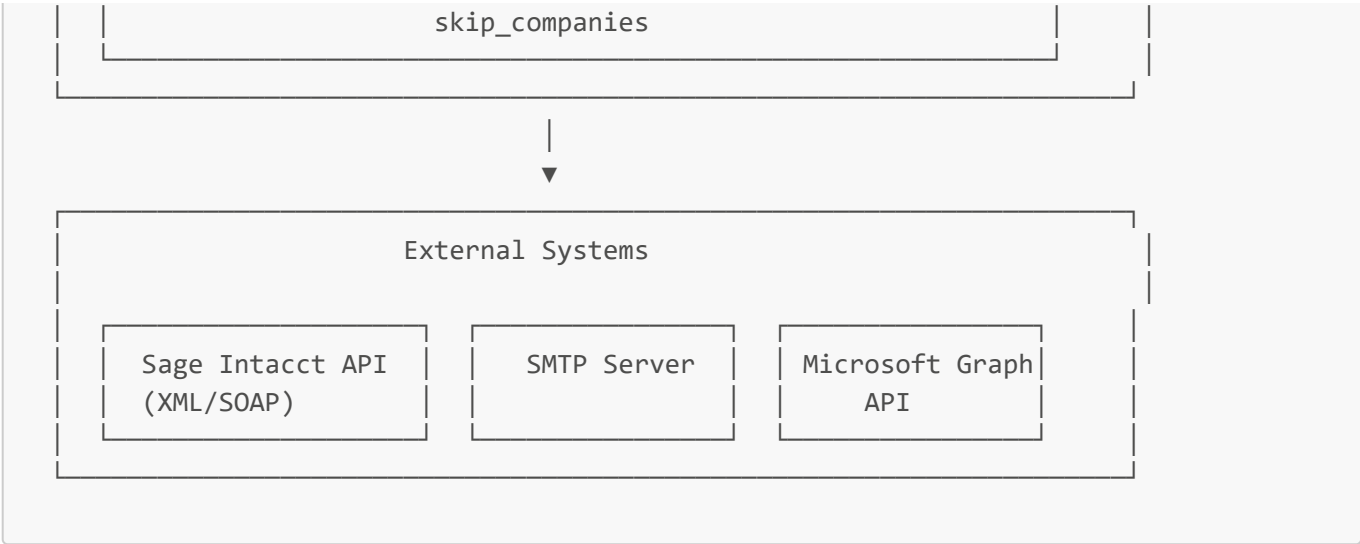
- **Exception Tracking:** Identify and manage companies that return errors or no data

Version

Current version: 3.0.2

Architecture





Request Flow

- 1. **User Interaction:** User interacts with the web UI (tabs, forms, buttons)
- 2. **API Calls:** Frontend JavaScript makes REST API calls to Express server
- 3. **Route Handling:** Express routes validate requests and call appropriate services
- 4. **Data Processing:** Services handle business logic, database operations, and external API calls
- 5. **CLI Subprocess:** Data fetching spawns cli.js as a subprocess for isolation and process control
- 6. **Database Storage:** All data persisted to SQLite using better-sqlite3 (synchronous operations)
- 7. **Response:** Results returned to frontend for display

Platform Compatibility

Platform	Support
Windows	Full support (development and production)
Linux	Full support (Docker recommended)
macOS	Full support
Docker	Fully containerized with docker-compose

Runtime Requirements

- **Node.js:** v18 or higher (v24 recommended for Docker)
- **npm:** v8 or higher
- **Python 3:** Required for better-sqlite3 native addon compilation
- **C++ Build Tools:** Required for better-sqlite3 compilation

Project Structure

```
IntacctAPIUsage/  
├─ source/           # Source code directory  
│  └─ js/           # Backend JavaScript (Node.js)
```

```

├─ server.js          # Express server entry point
├─ cli.js             # CLI worker for data fetching
├─ routes/            # Express route handlers
│   ├── about.js      # About/version endpoint
│   ├── config.js      # Configuration endpoints
│   ├── credentials.js # Credential management
│   ├── data.js        # Data/reporting endpoints
│   ├── job.js         # Job execution (run/stop/status)
│   ├── logs.js        # Log file access
│   ├── notifications.js # Notification config CRUD
│   ├── schedules.js   # Schedule CRUD and execution
│   └─ upload.js       # CSV upload handling
├─ services/          # Business logic services
│   ├── config.js      # Configuration management
│   ├── csv.js         # CSV parsing utilities
│   ├── database.js    # Database barrel export
│   ├── export.js      # Report generation utilities
│   ├── intacct.js     # Intacct API client
│   ├── job.js         # Job state management
│   ├── logger.js      # Winston logging with rotation
│   ├── notification.js # Email sending (SMTP/Graph)
│   ├── scheduler.js   # Scheduled job execution
│   └─ db/             # Database modules
│       ├── core.js    # Schema, encryption, lifecycle
│       ├── api-usage.js # API usage records
│       ├── companies.js # Company list management
│       ├── credentials.js # Encrypted credential storage
│       ├── job-errors.js # Job error tracking
│       ├── notification-configs.js # Email configs
│       ├── queries.js  # Filter definitions
│       ├── schedules.js # Schedule definitions
│       ├── settings.js # Key/value settings
│       └─ skip-companies.js # Exception list
├─ public/            # Frontend static files
│   ├── index.html     # Main SPA entry point
│   ├── css/
│   │   └─ styles.css  # Application styles
│   ├── js/            # Frontend JavaScript (ES modules)
│   │   ├── main.js    # Entry point, tab registration
│   │   ├── tabs.js    # Tab system logic
│   │   ├── config.js  # Configuration tab
│   │   ├── manual.js  # Manual run tab
│   │   ├── schedule.js # Schedules tab
│   │   ├── notifications.js # Notifications tab
│   │   ├── logs.js    # Logs viewer tab
│   │   ├── guide.js   # User guide tab
│   │   └─ about.js    # About tab
│   └─ partials/       # HTML partial templates
│       ├── config.html # Configuration form
│       ├── manual.html # Manual run interface
│       ├── schedule.html # Schedules management
│       ├── notifications.html # Notification configs
│       ├── logs.html   # Log viewer
│       └─ guide.html   # User guide content

```

```
| | | | about.html      # About page
| | | | images/        # Static images
| | | | package.json   # Node.js dependencies
| | | | TechnicalDocumentation/ # This documentation
| | | | Dockerfile      # Docker build instructions
| | | | docker-compose.yml # Docker compose configuration
| | | | env-sample.txt  # Environment variable template
| | | | example-companies.json # Sample companies file
```

Backend Components

Server & Routes

server.js

The main Express.js server entry point that:

- Loads environment variables from `.env`
- Initializes the SQLite database
- Mounts route handlers
- Starts the scheduler service
- Handles graceful shutdown (SIGTERM, SIGINT)

```
// Key initialization sequence:
1. Load .env configuration
2. Initialize database (initDatabase())
3. Connect database module to config service
4. Start scheduler service (startScheduler())
5. Listen on PORT (default 5050)
```

Route Handlers

Route File	Mount Point	Purpose
job.js	/	Run, stop, status, download endpoints
data.js	/	Reporting data endpoints
config.js	/config	Configuration management
credentials.js	/	Credential CRUD
upload.js	/	CSV file upload
schedules.js	/	Schedule CRUD and execution
notifications.js	/	Notification config CRUD
logs.js	/	Log file access

Route File	Mount Point	Purpose
<code>about.js</code>	<code>/</code>	Version endpoint

Services

`intacct.js` - Intacct API Client

Handles all communication with the Sage Intacct XML API:

- **XML Building:** Constructs `readByQuery` and `readMore` XML requests
- **API Communication:** Posts XML to Intacct endpoint via `axios`
- **Response Parsing:** Uses `fast-xml-parser` to parse XML responses
- **Pagination:** Handles automatic pagination with `readMore` requests
- **Error Handling:** Extracts control and operation errors from responses

```
// IntacctClient class options:
{
  senderId: string,      // Partner/Sender ID
  senderPassword: string, // Partner/Sender password
  userId: string,        // Intacct user login
  userPassword: string,  // Intacct user password
  url: string,           // API endpoint URL
  pageSize: number,      // Records per page (default 1000)
  fields: string[],       // Fields to retrieve
  queryName: string,     // Query identifier
  queryPrefix: string,   // DOCCONTROLID prefix filter
  onData: function       // Callback for streaming data to database
}
```

`scheduler.js` - Scheduled Job Execution

Manages automated data fetching and report delivery:

- **Interval Checking:** Runs every minute to check for due schedules
- **Data Fetching:** Spawns `cli.js` subprocess for each query
- **Report Generation:** Generates Excel reports after data fetch
- **Email Delivery:** Sends reports via configured notification provider
- **Status Tracking:** Updates schedule status after each run

Key functions:

- `startScheduler()` - Begins the scheduler interval
- `stopScheduler()` - Stops the scheduler
- `executeSchedule(schedule)` - Runs a single scheduled job
- `runDataFetch(queryFilter, scheduleName, useJobService, fullRefresh)` - Core data fetch logic
- `generateReportAttachments(queryFilter, selectedAttachments, startDate, endDate)` - Creates Excel reports

notification.js - Email Sending

Supports two email delivery methods:

1. **SMTP:** Direct SMTP connection using nodemailer
- SSL, STARTTLS, or no encryption
 - Connection pooling for performance
 - Custom from name and email
2. **Microsoft Graph:** OAuth2-based Azure email
- Uses @azure/identity for authentication
 - Client credential flow
 - Send-as capability for mailbox users

logger.js - Winston Logging

Provides structured logging with:

- **Console Output:** Colorized, timestamped messages
- **File Output:** JSON format, monthly rotation
- **Log Files:** Named `app-YYYY-MM.log` for easy archival
- **API Access:** Endpoints to list and read log files from UI

Database Layer

Core Module (db/core.js)

Central database management with:

- **Schema Definitions:** Single source of truth for all table structures
- **Migration System:** Automatic schema updates on startup
- **Encryption:** AES-256-GCM encryption for credentials
- **WAL Mode:** Write-Ahead Logging for concurrent access

Tables managed:

Table	Purpose
credentials	Encrypted Intacct credentials
settings	Key/value configuration pairs
filters	Query filter definitions
api_usage	Collected API usage records
companies	Company ID list
job_errors	Error tracking per job run
notification_configs	Email configuration

Table	Purpose
<code>schedules</code>	Scheduled job definitions
<code>skip_companies</code>	Exception list (companies to skip)

Database Barrel Export (database.js)

Re-exports all database functions for convenient importing:

```
import {
  initDatabase,
  getApiUsageRecords,
  getAllCompanies,
  getSchedule,
  // ... all other database functions
} from './services/database.js';
```

Frontend Components

Tab System

The application uses a single-page architecture with lazy-loaded tabs.

tabs.js

Core tab management functions:

- `registerTabInit(tabName, initFunction)` - Register initialization function
- `registerTabCleanup(tabName, cleanupFunction)` - Register cleanup function
- `switchTab(tabName)` - Switch to a tab, triggering init/cleanup
- `loadInitialTab(defaultTab)` - Load initial tab from URL hash

Tab Lifecycle

1. **Cleanup:** Previous tab's cleanup function called (stops polling, removes listeners)
2. **Load HTML:** Fetch partial HTML from `/partials/{tab}.html`
3. **Insert DOM:** Replace tab container content
4. **Initialize:** Call tab's init function

JavaScript Modules

Each tab has a corresponding JavaScript module:

Module	Tab	Key Functions
<code>manual.js</code>	Manual	Run/stop jobs, live log, reports display

Module	Tab	Key Functions
schedule.js	Scheduled	CRUD schedules, run now, status display
config.js	Configuration	Settings, credentials, company management
notifications.js	Notifications	Email config CRUD, test send
logs.js	Logs	Log file viewer
guide.js	User Guide	Static help content
about.js	About	Version and system info

HTML Partial

Located in `/public/partial/`, each partial contains:

- HTML structure for the tab
- Form elements
- Modal dialogs
- Inline event handlers pointing to JavaScript functions

Theming System

CSS Custom Properties

The application uses CSS custom properties for consistent theming:

```
:root {
  /* Colors */
  --bg-light: #ffffff;
  --bg-dark: #1a1a1a;
  --primary: #00AFAA;      /* Sage/Intacct teal */
  --primary-dark: #008F8A;
  --text-dark: #333333;
  --text-light: #f0f0f0;
  --border-color: #e0e0e0;
  --success: #28a745;
  --error: #dc3545;
  --warning: #ffc107;

  /* Spacing */
  --spacing-sm: 8px;
  --spacing-md: 16px;
  --spacing-lg: 24px;

  /* Typography */
  --font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  --font-size-base: 14px;
}
```

Component Classes

Key component styles:

- `.card` - Content containers with shadow
- `.btn`, `.btn-primary`, `.btn-secondary` - Button styles
- `.form-group`, `.form-row` - Form layout
- `.toggle-row`, `.toggle-option` - Checkbox/radio groups
- `.modal`, `.modal-content` - Modal dialogs
- `.status-*` - Status indicator colors

Mobile Responsiveness

- Bottom navigation bar for mobile devices
- Collapsible admin menu
- Responsive grid layouts
- Touch-friendly button sizes

Data Storage & Encryption

SQLite Database

File Location: `{DATA_DIR}/intacct.db`

Features:

- WAL mode for concurrent access
- Automatic schema migration on startup
- better-sqlite3 for synchronous, fast operations

Credential Encryption

All sensitive credentials are encrypted using:

- **Algorithm:** AES-256-GCM
- **Key Derivation:** PBKDF2 with 100,000 iterations
- **Salt:** 32 bytes, randomly generated per credential
- **IV:** 16 bytes, randomly generated per encryption
- **Auth Tag:** 16 bytes for authentication

Encryption Key: Must be provided via `ENCRYPTION_KEY` environment variable

```
// Encryption process:  
1. Generate random salt (32 bytes)  
2. Derive key using PBKDF2(ENCRYPTION_KEY, salt)  
3. Generate random IV (16 bytes)  
4. Encrypt with AES-256-GCM  
5. Store: salt, iv, auth_tag, encrypted_data
```

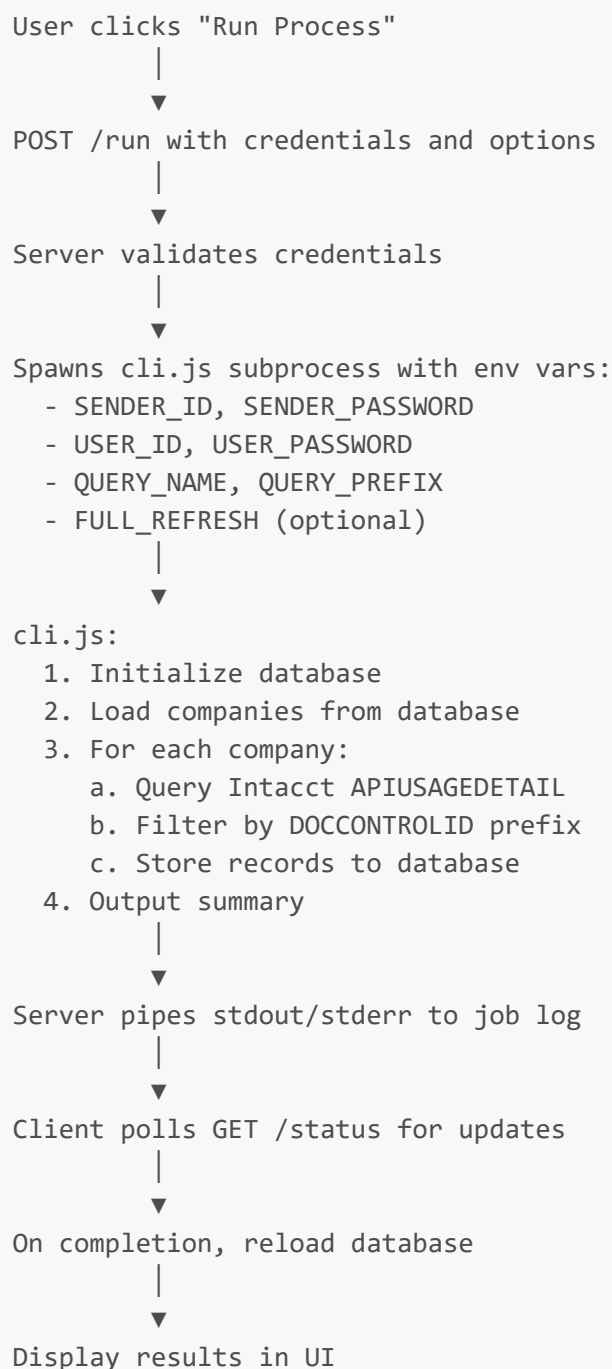
Protected Tables

The `credentials` table is marked as `preserveData: true` in schema definitions. This ensures:

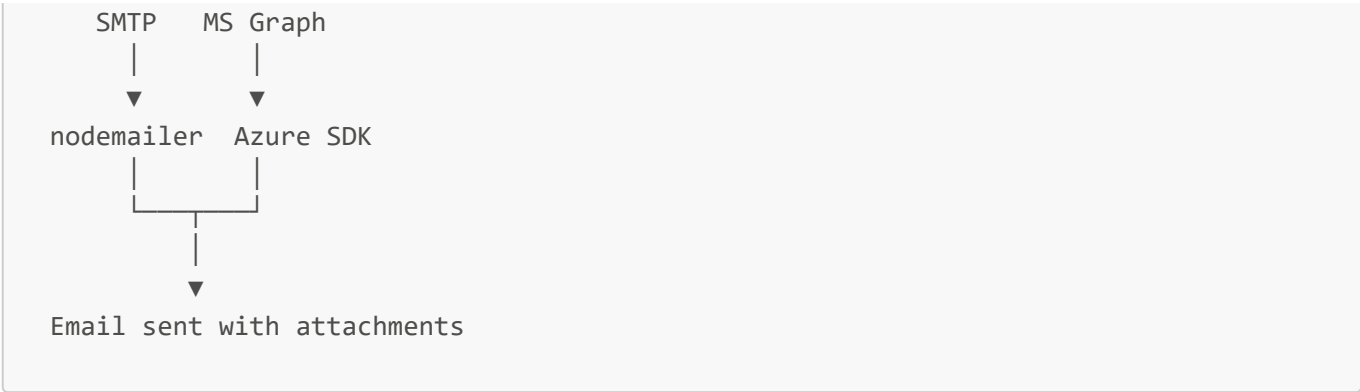
- Table is never dropped during migrations
 - Only missing columns are added
 - Existing encrypted data is preserved
-

Key Workflows

Manual Data Fetch



Scheduled Jobs



Configuration

Environment Variables

Variable	Required	Default	Description
ENCRYPTION_KEY	Yes	-	32-byte hex key for credential encryption
DATA_DIR	No	./data	Directory for SQLite database
LOG_DIR	No	./logs	Directory for log files
PORT	No	5050	Server listening port
NODE_ENV	No	development	Environment mode
LOG_LEVEL	No	info	Logging verbosity (error, warn, info, debug)
LIVE_LOG_LEVEL	No	info	Live log verbosity during job runs
LOCAL_COMPANIES_API_ENABLED	No	false	Enable local companies API endpoint
LOCAL_COMPANIES_API_TOKEN	No	-	Bearer token for local companies API

Database Settings

Key/value pairs stored in `settings` table:

Key	Description
url	Intacct API endpoint URL
pageSize	Records per API request page
fields	Comma-separated field list
baseStartDate	Earliest date to fetch data from
companyIdColumn	CSV column name for company IDs
includeCurrentMonth	Include current month in reports

Key	Description
companiesSource	"csv" or "api"
companiesApiUrl	External API URL for company list

Filter Definitions

Stored in filters table:

Field	Description
sort_order	Display order
name	Filter display name (e.g., "ExcelConnect")
prefix	DOCCONTROLID prefix (e.g., "EC_")

API Reference

Job Endpoints

POST /run

Start a data fetch job.

Request Body:

```
{
  "senderId": "string",
  "senderPassword": "string",
  "userId": "string",
  "userPassword": "string",
  "queryName": "string (optional)",
  "runAllQueries": "boolean (optional)",
  "forceFullRefresh": "boolean (optional)",
  "notificationConfigId": "number (optional)",
  "attachments": ["detailed", "summary", "exceptions"],
  "reportStartDate": "YYYY-MM-DD (optional)",
  "reportEndDate": "YYYY-MM-DD (optional)"
}
```

POST /stop

Stop the running job.

GET /status

Get current job status.

Response:

```
{
  "running": true,
  "scheduledJobRunning": false,
  "startedAt": "ISO timestamp",
  "exitCode": 0,
  "stopped": false,
  "log": "job output...",
  "queryName": "All Filters",
  "dataAvailable": true,
  "recordCount": 12345,
  "exceptionCount": 5,
  "errorCount": 2,
  "errorSummary": [...]
}
```

GET /download/:which

Download report (detailed, summary, exceptions).

Query Parameters:

- **format**: "csv" or "xlsx"

Data Endpoints

GET /detailed.json

Get record count and aggregates.

GET /reporting.json

Get monthly aggregated data for charts.

Query Parameters:

- **startDate**: YYYY-MM-DD
- **endDate**: YYYY-MM-DD

GET /api-usage/stats.json

Get overall statistics.

GET /api-usage/summary.json

Get per-company summary.

Config Endpoints

GET /config

Get all settings and queries.

POST /config

Save settings and queries.

POST /config/settings

Save individual setting.

Schedule Endpoints**GET /schedules**

List all schedules.

POST /schedules

Create new schedule.

PUT /schedules/:id

Update schedule.

DELETE /schedules/:id

Delete schedule.

POST /schedules/:id/run

Execute schedule immediately.

POST /schedules/stop

Stop running scheduled job.

Notification Endpoints**GET /notifications/configs**

List all notification configs.

POST /notifications/configs

Create notification config.

PUT /notifications/configs/:id

Update notification config.

DELETE /notifications/configs/:id

Delete notification config.

POST /notifications/test

Send test email.

Security Considerations

Credential Protection

- All Intacct credentials stored with AES-256-GCM encryption
- Encryption key never stored in database
- PBKDF2 key derivation prevents rainbow table attacks
- Unique salt and IV per credential

API Security

- Local Companies API disabled by default
- Bearer token authentication when enabled
- No authentication on main API (designed for internal network use)

Input Validation

- CSV file size limits
- Filename sanitization for log file access
- SQL injection prevention via parameterized queries
- XSS prevention (no user content rendered as HTML)

Process Isolation

- Data fetch runs in subprocess
- SIGINT/SIGKILL handling for job termination
- Graceful shutdown on server termination

Recommendations

1. Run behind reverse proxy with authentication for external access
 2. Use strong, unique ENCRYPTION_KEY
 3. Enable HTTPS in production
 4. Restrict network access to Intacct API IPs
 5. Regular backup of SQLite database
 6. Monitor log files for errors
-

Troubleshooting

Common Issues

"Database not ready"

- Ensure DATA_DIR is writable
- Check for disk space
- Verify ENCRYPTION_KEY is set

"Missing credentials"

- Upload credentials via Configuration tab
- Verify ENCRYPTION_KEY matches original key used to save credentials

"No companies found"

- Upload CSV file or configure external API
- Check companies source setting

Job hangs or doesn't complete

- Check job log for errors
- Verify Intacct credentials are valid
- Check network connectivity to api.intacct.com

Email not sending

- Verify notification config settings
- Check SMTP server connectivity
- For MS Graph: verify Azure app registration and permissions

Debug Mode

Set `LIVE_LOG_LEVEL=debug` for verbose output:

- Query details
- Prefix filtering results
- Page-by-page progress
- Sample DOCCONTROLID values

Log File Location

Logs are written to `{LOG_DIR}/app-YYYY-MM.log`

View via:

1. Logs tab in UI
2. Direct file access
3. Docker: `docker logs intacct-api-usage`

Dependencies

Backend (Node.js)

Package	Version	Purpose
express	^4.19.2	Web framework
cors	^2.8.5	CORS middleware
dotenv	^16.4.5	Environment variables
better-sqlite3	^11.7.0	SQLite database
axios	^1.7.2	HTTP client
fast-xml-parser	^4.4.0	XML parsing
exceljs	^4.4.0	Excel generation
json2csv	^5.0.7	CSV generation
csv-parse	^5.5.6	CSV parsing
multer	^1.4.5-lts.1	File upload handling
nodemailer	^6.9.0	SMTP email
@azure/identity	^4.0.0	Azure authentication
@microsoft/microsoft-graph-client	^3.0.0	MS Graph API
winston	^3.17.0	Logging

Frontend

Library	Version	Purpose
Chart.js	CDN	Chart rendering

Build Dependencies

Tool	Purpose
Python 3	better-sqlite3 compilation
node-gyp	Native addon compilation
make, g++	C++ compilation

Licensing

This is an internal tool developed by Fusion/Datel for monitoring Sage Intacct API usage. All rights reserved.

