Table of Contents

Features

- Full CRUD operations: create, read, update, delete
- Batch insert, update, and delete
- Advanced filtering with type-aware comparisons
- Schema support with data types: string, number, integer, double, boolean, date
- Automatic type casting and validation
- Intelligent caching with cache statistics and management
- Auto sheet/header creation and unique ID generation
- Existence checks: `sheetExists()`, `recordExists()`
- Generic field search: `getByField()`, `getIdByField()`
- Row counting and bulk deletion operations
- Rate limiting protection with exponential backoff retry
- Event-driven readiness (`sheetapi:ready` or custom event)
- Debug mode for development and troubleshooting
- Standardized `{status, data, error}` responses
- CDN-ready: use via `<script>` tag (no modules)

Step-by-Step Google Sheet Setup

Follow these steps to set up your Google Sheet and get the credentials needed to use SheetAPI.js:

1. **Create a Google Sheet**
    - Go to Google Sheets and create a new spreadsheet.
    - Name your sheet (e.g., `Users`).
2. **Get Your Spreadsheet ID**
    - Open your sheet in the browser. The URL will look like: `https://docs.google.com/spreadsheets/d/your-spreadsheet-id/edit#gid=0`
    - Copy the long string after `/d/` and before `/edit` — this is your **spreadsheetId**.
3. **Create a Google Cloud Project**
    - Go to the Google Cloud Console.
    - Click the project dropdown (top left) and select "New Project". Give it a name and create it.
4. **Enable the Google Sheets API**

- In your project, go to "APIs & Services > Library".
- Search for "Google Sheets API" and click "Enable".

5. **Create OAuth2 Credentials**
   - Go to "APIs & Services > Credentials".
   - Click "Create Credentials" > "OAuth client ID".
   - If prompted, configure the consent screen (just fill required fields).
   - Choose "Desktop app" as the application type.
   - Download the credentials JSON file. It contains your **client_id** and **client_secret**.
6. **Get a Refresh Token and Access Token**
   - Use a tool like [OAuth 2.0 Playground] or a sample script to:
     - Authorize with your client ID/secret.
     - Select the scope: `https://www.googleapis.com/auth/spreadsheets`
     - Exchange the code for a refresh token and access token.
   - Save your **accessToken** and **refreshToken**.
7. **Share Your Sheet for API Access**
   - In your Google Sheet, click "Share" and add the email address shown in your OAuth credentials (often ends with `@developer.gserviceaccount.com` or your Google account email).
   - Give it "Editor" access.
8. **Use These in SheetAPI.js**
   - Use the `spreadsheetId`, `accessToken`, and (optionally) `refreshToken`, `clientId`, `clientSecret` in your SheetAPI config as shown in the usage examples below.

If you get stuck, search for "Google Sheets API quickstart" for more detailed guides and screenshots.

Installation

Browser (CDN)

```
<script src="SheetAPI.js"></script>
```

Node.js

```
const SheetAPI = require('./SheetAPI');
```

Usage Examples

Initialization

You can initialize SheetAPI in two mutually exclusive ways:

1. Using an Access Token

```
const api = new SheetAPI({
  spreadsheetId: 'your_sheet_id',
  accessToken: 'ya29.your_token',
  sheetName: 'Users',
  headers: ['id', 'name', 'age', 'status'], // optional
  createIfMissing: true, // optional, default true
  debug: false, // optional, enables debug logging
  schema: { // optional, defines data types for columns
    id: 'string',
    name: 'string',
    age: 'integer',
    status: 'boolean'
  }
});
await api.ready;
```

2. Using a Refresh Token (OAuth2)

```
const api = new SheetAPI({
  spreadsheetId: 'your_sheet_id',
  refreshToken: 'your_refresh_token',
  clientId: 'your_client_id',
  clientSecret: 'your_client_secret',
  sheetName: 'Users',
  headers: ['id', 'name', 'age', 'status'], // optional
  createIfMissing: true, // optional, default true
  debug: false, // optional, enables debug logging
  schema: { // optional, defines data types for columns
    id: 'string',
    name: 'string',
    age: 'integer',
    status: 'boolean'
  }
});
await api.ready;
```

Usage

All methods return a Promise resolving to `{status, data, error}`.

Wait for Ready

```
api.ready.then(() => {
  // Safe to use API methods
});
// Or listen for event
window.addEventListener('sheetapi:ready', e => {
  if (e.detail.status) {
    // Ready
  } else {
    // Error: e.detail.error
```

```
  }
});
```

CRUD Examples

```
// Create
await api.create({ name: 'John', age: 30 });
// Read all (with filtering, sorting, pagination)
await api.getAll({
  filter: { age: { op: 'ge', value: 18 } },
  sort: [{ field: 'name', direction: 'asc' }],
  fields: ['id', 'name'],
  page: 1,
  perPage: 10
});
// result = { status: 'success', data: [ ...records ], error: null }
```

Get by ID

```
const result = await api.getById('abc123');
// result = { status: 'success', data: { id: 'abc123', ... }, error: null } // or
status: 'error' if not found
```

Update by ID

```
const result = await api.updateById('abc123', { age: 25 });
// result = { status: 'success', data: { ...updatedRecord }, error: null }
```

Update by Object

```
// Update a record by passing an object with id and fields to update
const result = await api.update({ id: 'abc123', age: 30 });
// result = { status: 'success', data: { ...updatedRecord }, error: null }
```

Generate Unique ID

```
// Generate a unique string ID (default length 15)
const id = api.generateUniqueId();
// id = 'randomString...'
```

Delete by ID

```
const result = await api.deleteById('abc123');
// result = { status: 'success', data: 'abc123', error: null }
```

Check if Sheet Exists

```
const result = await api.sheetExists('Users');
// result = { status: 'success', data: true, error: null }
```

Check if Record Exists

```
const result = await api.recordExists('abc123');
// result = { status: 'success', data: true, error: null }
```

Get by Field

```
const result = await api.getByField('email', 'john@example.com');
// result = { status: 'success', data: { ...record }, error: null }
```

Get ID by Field

```
const result = await api.getIdByField('email', 'john@example.com');
// result = { status: 'success', data: 'abc123', error: null }
```

Batch Insert Multiple Records

```
// Insert multiple user records at once (IDs auto-generated if missing)
const records = [
  { name: 'Alice', email: 'alice@example.com', status: 'active' },
  { name: 'Bob', email: 'bob@example.com', status: 'inactive' }
];
const insertResult = await api.batchInsert(records);
// insertResult = { status: 'success', data: [ ...insertedRecordsWithIds ] }
```

Batch Update Multiple Records by ID

```
// Update the status of multiple users by their IDs
const updates = [
  { id: 'abc123', status: 'inactive' },
  { id: 'def456', status: 'active' }
];
const updateResult = await api.batchUpdate(updates);
// updateResult = { status: 'success', data: [ ...updatedRecords ] }
```

Batch Delete Multiple Records by ID

```
// Delete multiple users by their IDs
const ids = ['abc123', 'def456'];
const deleteResult = await api.batchDelete(ids);
// deleteResult = { status: 'success', data: ['abc123', 'def456'] }
```

Filtering, Sorting, and Pagination

- `cs` : contains string
- `sw` : starts with
- `ew` : ends with
- `eq` : equals
- `neq` : not equals

- `lt`, `le`, `ge`, `gt`: numeric comparisons
- `bt`: between (comma-separated)
- `in`: in list (comma-separated)
- `is`: is null/empty

Example: Filtering, Sorting, and Pagination

```
// Get users whose name contains 'A', age >= 18, sorted by age descending, page 2, 5 per page
const result = await api.getAll({
  filter: {
    name: { op: 'cs', value: 'A' },
    age: { op: 'ge', value: 18 }
  },
  sort: [ { field: 'age', direction: 'desc' } ],
  page: 2,
  perPage: 5
});
// result = { status: 'success', data: [ ...records ], error: null }
```

Schema and Data Types

SheetAPI supports automatic type casting and validation through schema definitions. Supported data types:

- `string`: Text values
- `number`: Numeric values (includes integers and floats)
- `integer`: Whole numbers only
- `double`: Floating-point numbers
- `boolean`: true/false values
- `date`: Date values (ISO format or common date formats)

Schema Example

```
const api = new SheetAPI({
  // ... other config
  schema: {
    id: 'string',
    name: 'string',
    age: 'integer',
    salary: 'double',
    active: 'boolean',
    createdAt: 'date'
  }
});

// Data is automatically cast to the correct type
await api.create({
  name: 'John',      // string
  age: '25',         // cast to integer 25
  salary: '50000',   // cast to double 50000.0
  active: 'true',    // cast to boolean true
```

```
  createdAt: '2023-01-01' // cast to Date object
});
```

Caching

SheetAPI includes intelligent caching to improve performance and reduce API calls:

Intelligent Cache Configuration

SheetAPI automatically optimizes cache durations based on your application's user load:

```
// Default: Single-user optimization (recommended for most apps)
const api = new SheetAPI({
  spreadsheetId: 'your-sheet-id',
  accessToken: 'your-token',
  sheetName: 'Users'
});
// Auto-sets: ~15-30 minute cache durations

// Multi-user application
const api = new SheetAPI({
  spreadsheetId: 'your-sheet-id',
  accessToken: 'your-token',
  sheetName: 'Users',
  cache: {
    userCount: 50  // Optimize for 50 concurrent users
  }
});
// Auto-sets: ~6-12 minute cache durations

// Real-time experience (disable caching)
const api = new SheetAPI({
  spreadsheetId: 'your-sheet-id',
  accessToken: 'your-token',
  sheetName: 'Users',
  cache: {
    userCount: 0  // Disable all caching
  }
});
// Every request hits Google Sheets directly
```

Cache Duration Algorithm

- **userCount = 1**: Longest caches (15-30 minutes) - optimal for single user
- **userCount = 5**: Moderate caches (13-26 minutes) - good for small teams
- **userCount = 50**: Short caches (6-12 minutes) - suitable for larger applications
- **userCount = 0**: No caching - real-time data for high-frequency updates

Cache Methods

```
// Get cache statistics
const stats = await api.getCacheStats();
```

```
// stats = { status: 'success', data: { metaCache: {...}, dataCache: {...},
recordCache: {...} } }

// Clear all caches
const result = await api.clearCaches();
// result = { status: 'success', data: true }
```

Cache Types

- **Meta Cache**: Sheet metadata (headers, structure)
- **Data Cache**: Query results and filtered data
- **Record Cache**: Individual record lookups

Additional Utility Methods

Row Count

```
// Get total number of rows in the sheet
const result = await api.rowCount();
// result = { status: 'success', data: 150 }
```

Delete All Records

```
// Delete all records (keeps headers)
const result = await api.deleteAll();
// result = { status: 'success', data: true }
```

Error Handling

All methods return a standardized object:

```
{ status: 'success' | 'error', data: ..., error: ... }
```

Rate Limiting and Retry Logic

SheetAPI includes automatic rate limiting protection with exponential backoff retry:

- Automatically retries failed requests due to rate limits
- Implements exponential backoff with jitter
- Handles 401 errors by automatically refreshing OAuth2 tokens
- Configurable retry attempts and delays

Debug Mode

Enable debug mode to see detailed logging:

```
const api = new SheetAPI({
  // ... config
  debug: true
});
// Logs will show API calls, token refreshes, caching, and errors
```

Requirements

- Google Sheets API enabled
- OAuth2 access token with Sheets API scope

License

MIT

Generated by GitHub Copilot based on SheetAPI.js source code.