readme.md for @apiclient.xyz/gitlab

A powerful, fully-typed TypeScript client for the GitLab API □□

npm version license

`@apiclient.xyz/gitlab` gives you a clean, object-oriented interface to the GitLab REST API. It wraps projects, groups, pipelines, jobs, CI/CD variables, branches, tags, protected branches, and test reports into rich domain classes with full auto-pagination support. Works with any GitLab instance -- cloud or self-hosted.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

```
npm install @apiclient.xyz/gitlab
# or
pnpm install @apiclient.xyz/gitlab
```

Quick Start ⚡

```
import { GitLabClient } from '@apiclient.xyz/gitlab';

const client = new GitLabClient('https://gitlab.example.com', 'your-private-token');
```

```
// Verify connectivity
const { ok, error } = await client.testConnection();
console.log(ok ? '□ Connected!' : `□ ${error}`);

// Grab all your projects
const projects = await client.getProjects();
for (const project of projects) {
  console.log(`${project.name} — ${project.webUrl}`);
}
```

That's it. No boilerplate, no callback soup, just straightforward async/await.

Core Concepts □□

Rich Domain Objects

Every API response is wrapped in a **domain class** that exposes typed properties and chainable methods. You never deal with raw JSON unless you want to -- every class has a `.toJSON()` escape hatch.

```
GitLabClient
  ├── GitLabGroup          → .getProjects(), .getVariables(), .update(), .delete(), ...
  ├── GitLabProject        → .getPipelines(), .getBranches(), .getVariables(),
.triggerPipeline(), ...
  │     ├── GitLabBranch
  │     ├── GitLabTag
  │     ├── GitLabProtectedBranch
  │     └── GitLabVariable
  ├── GitLabPipeline       → .getJobs(), .getTestReport(), .retry(), .cancel(), .delete()
  │     ├── GitLabJob      → .getLog(), .retry(), .cancel(), .play(), .erase()
  │     ├── GitLabPipelineVariable
  │     └── GitLabTestReport
  │           └── GitLabTestSuite
  │                 └── GitLabTestCase
  └── GitLabVariable
```

Auto-Pagination □□

List endpoints automatically page through **all** results by default. Need just one page? Pass `{ page: N }` explicitly.

```
// Fetches ALL groups (auto-paginates transparently)
const allGroups = await client.getGroups();

// Fetches only page 2, 10 per page
const page2 = await client.getGroups({ page: 2, perPage: 10 });
```

The `autoPaginate` helper is also exported if you want to use it in custom integrations.

Usage

Creating a Client

```
import { GitLabClient } from '@apiclient.xyz/gitlab';

const client = new GitLabClient('https://gitlab.example.com', 'your-private-token');
```

The constructor takes the base URL of your GitLab instance and a personal access token (or project/group token) for `PRIVATE-TOKEN` header authentication.

Testing the Connection

```
const result = await client.testConnection();

if (result.ok) {
  console.log('Connected successfully.');
} else {
  console.error('Connection failed:', result.error);
}
```

Groups □□

List All Groups

```
const groups = await client.getGroups();

// Search by name
const devGroups = await client.getGroups({ search: 'platform' });

for (const group of devGroups) {
  console.log(`${group.id} — ${group.fullPath} (${group.visibility})`);
}
```

Get a Single Group

```
const group = await client.getGroup('my-org/my-team');
console.log(group.name, group.webUrl);
```

Create a Group

```
const newGroup = await client.createGroup('backend-team', 'backend-team');

// Create a sub-group under an existing parent
const subGroup = await client.createGroup('api-squad', 'api-squad', parentGroup.id);
```

Group Operations

```
// Update properties
await group.update({ description: 'The backend crew □□', visibility: 'internal' });

// Upload / remove avatar
await group.setAvatar(imageBytes, 'logo.png');
await group.deleteAvatar();

// Transfer to another parent group
await group.transfer(targetGroupId);

// List descendant sub-groups
const children = await group.getDescendantGroups();

// List projects inside the group
const projects = await group.getProjects({ search: 'api' });

// Delete
```

```
await group.delete();
```

Projects

List All Projects

```
const projects = await client.getProjects();

// Search with pagination
const matches = await client.getProjects({ search: 'my-service', perPage: 20 });

for (const p of matches) {
  console.log(`${p.id} — ${p.fullPath} [${p.defaultBranch}]`);
}
```

Get a Single Project

```
// By numeric ID
const project = await client.getProject(42);

// By full path
const project = await client.getProject('my-org/my-repo');
```

Create a Project

```
const project = await client.createProject('new-service', {
  namespaceId: group.id,
  visibility: 'internal',
  description: 'Our shiny new microservice',
});
```

Project Operations

```
// Update properties
await project.update({
  description: 'Updated description',
  defaultBranch: 'develop',
  topics: ['typescript', 'api'],
```

```
});

// Avatar management
await project.setAvatar(imageBytes, 'project-logo.png');
await project.deleteAvatar();

// Transfer to a different namespace
await project.transfer(otherNamespaceId);

// Delete
await project.delete();
```

Branches & Tags □□

```
// List all branches (auto-paginated)
const branches = await project.getBranches();
for (const b of branches) {
  console.log(`${b.name} @ ${b.commitSha}`);
}

// List all tags
const tags = await project.getTags();
for (const t of tags) {
  console.log(`${t.name} @ ${t.commitSha}`);
}
```

Protected Branches □□

```
const protectedBranches = await project.getProtectedBranches();
for (const pb of protectedBranches) {
  console.log(`${pb.name} — force push: ${pb.allowForcePush}`);
}

// Remove branch protection
await project.unprotectBranch('develop');
```

CI/CD Variables □□

Variables work identically on both projects and groups.

Project Variables

```
// List all
const vars = await project.getVariables();

// Create
const dbVar = await project.createVariable('DATABASE_URL', 'postgres://localhost/db', {
  protected: true,
  masked: true,
  environment_scope: 'production',
});

// Update
await project.updateVariable('DATABASE_URL', 'postgres://newhost/db', {
  protected: true,
});

// Delete
await project.deleteVariable('DATABASE_URL');
```

Group Variables

```
const groupVars = await group.getVariables();

await group.createVariable('NPM_TOKEN', 'tok-xxx', {
  protected: false,
  masked: true,
  environment_scope: '*',
});

await group.updateVariable('NPM_TOKEN', 'tok-yyy');
await group.deleteVariable('NPM_TOKEN');
```

Each `GitLabVariable` instance exposes typed properties:

```
console.log(dbVar.key);              // 'DATABASE_URL'
console.log(dbVar.value);            // 'postgres://...'
console.log(dbVar.variableType);     // 'env_var'
console.log(dbVar.protected);        // true
console.log(dbVar.masked);           // true
console.log(dbVar.environmentScope); // 'production'
```

Pipelines □□

List & Filter Pipelines

```
// Recent pipelines (auto-paginated)
const pipelines = await project.getPipelines();

// Advanced filtering
const failed = await project.getPipelines({
  status: 'failed',
  ref: 'main',
  source: 'push',
  orderBy: 'updated_at',
  sort: 'desc',
  updatedAfter: '2025-01-01T00:00:00Z',
});

for (const p of failed) {
  console.log(`Pipeline #${p.id} [${p.status}] on ${p.ref} — ${p.webUrl}`);
}
```

Trigger a Pipeline

```
const pipeline = await project.triggerPipeline('main', [
  { key: 'DEPLOY_ENV', value: 'staging' },
]);
console.log(`Triggered pipeline #${pipeline.id}`);
```

Pipeline Actions

```
// Retry all failed jobs
const retried = await pipeline.retry();

// Cancel a running pipeline
const cancelled = await pipeline.cancel();

// Delete a pipeline
await pipeline.delete();
```

Pipeline Variables & Test Reports

```
// Get variables used in a pipeline run
const pipelineVars = await pipeline.getVariables();
for (const v of pipelineVars) {
  console.log(`${v.key} = ${v.value} (${v.variableType})`);
}

// Get the test report
const report = await pipeline.getTestReport();
console.log(`Tests: ${report.totalCount} total, ${report.successCount} passed,
${report.failedCount} failed`);

for (const suite of report.testSuites) {
  console.log(`  Suite "${suite.name}": ${suite.totalCount} tests (${suite.failedCount}
failures)`);
  for (const tc of suite.testCases) {
    if (tc.status === 'failed') {
      console.log(`    □ ${tc.name}: ${tc.systemOutput}`);
    }
  }
}
```

Jobs ⚙

```
// Get all jobs for a pipeline
const jobs = await pipeline.getJobs();

// Filter by scope
```

```
const failedJobs = await pipeline.getJobs({ scope: ['failed'] });

for (const job of jobs) {
  console.log(`Job "${job.name}" (${job.stage}) — ${job.status} [${job.duration}s]`);
}
```

Job Actions

```
// Read the raw log output
const log = await job.getLog();
console.log(log);

// Retry a failed job
const retriedJob = await job.retry();

// Cancel a running job
const cancelledJob = await job.cancel();

// Trigger a manual job
const played = await job.play();

// Erase job artifacts and trace
await job.erase();
```

Job Properties

Each `GitLabJob` exposes rich metadata:

```
job.id              // number
job.name            // 'build'
job.stage           // 'test'
job.status          // 'success' | 'failed' | 'running' | ...
job.ref             // 'main'
job.tag             // false
job.webUrl          // full URL to the job
job.duration        // seconds
job.queuedDuration  // seconds waiting in queue
job.coverage        // code coverage percentage
job.allowFailure    // whether failure is acceptable
job.failureReason   // reason for failure, if any
```

API Reference □□

`GitLabClient`

Method	Returns	Description
`new GitLabClient(baseUrl, token)`	`GitLabClient`	Create a client instance
`.testConnection()`	`Promise<ITestConnectionResult>`	Verify token and connectivity
`.getGroups(opts?)`	`Promise<GitLabGroup[]>`	List all accessible groups (auto-paginated)
`.getGroup(fullPath)`	`Promise<GitLabGroup>`	Get a single group by full path
`.createGroup(name, path, parentId?)`	`Promise<GitLabGroup>`	Create a new group
`.getProjects(opts?)`	`Promise<GitLabProject[]>`	List your projects (auto-paginated)
`.getProject(idOrPath)`	`Promise<GitLabProject>`	Get a single project by ID or path
`.createProject(name, opts?)`	`Promise<GitLabProject>`	Create a new project

`GitLabGroup`

Method	Returns	Description
`.getProjects(opts?)`	`Promise<GitLabProject[]>`	List projects in the group
`.getDescendantGroups(opts?)`	`Promise<GitLabGroup[]>`	List descendant sub-groups
`.getVariables()`	`Promise<GitLabVariable[]>`	List CI/CD variables
`.createVariable(key, value, opts?)`	`Promise<GitLabVariable>`	Create a CI/CD variable
`.updateVariable(key, value, opts?)`	`Promise<GitLabVariable>`	Update a CI/CD variable
`.deleteVariable(key)`	`Promise<void>`	Delete a CI/CD variable
`.update(data)`	`Promise<void>`	Update group properties
`.setAvatar(imageData, filename)`	`Promise<void>`	Upload avatar image
`.deleteAvatar()`	`Promise<void>`	Remove avatar
`.transfer(parentGroupId)`	`Promise<void>`	Transfer to another parent group
`.delete()`	`Promise<void>`	Delete the group
`.toJSON()`	`IGitLabGroup`	Serialize to raw interface

GitLabProject

Method	Returns	Description
`.getBranches(opts?)`	`Promise<GitLabBranch[]>`	List repository branches
`.getTags(opts?)`	`Promise<GitLabTag[]>`	List repository tags
`.getProtectedBranches()`	`Promise<GitLabProtectedBranch[]>`	List protected branches
`.unprotectBranch(name)`	`Promise<void>`	Remove branch protection
`.getVariables()`	`Promise<GitLabVariable[]>`	List CI/CD variables
`.createVariable(key, value, opts?)`	`Promise<GitLabVariable>`	Create a CI/CD variable
`.updateVariable(key, value, opts?)`	`Promise<GitLabVariable>`	Update a CI/CD variable
`.deleteVariable(key)`	`Promise<void>`	Delete a CI/CD variable
`.getPipelines(opts?)`	`Promise<GitLabPipeline[]>`	List pipelines (with rich filtering)
`.triggerPipeline(ref, variables?)`	`Promise<GitLabPipeline>`	Trigger a new pipeline
`.update(data)`	`Promise<void>`	Update project properties
`.setAvatar(imageData, filename)`	`Promise<void>`	Upload avatar image
`.deleteAvatar()`	`Promise<void>`	Remove avatar
`.transfer(namespaceId)`	`Promise<void>`	Transfer to another namespace
`.delete()`	`Promise<void>`	Delete the project
`.toJSON()`	`IGitLabProject`	Serialize to raw interface

GitLabPipeline

Method	Returns	Description
`.getJobs(opts?)`	`Promise<GitLabJob[]>`	List jobs (with scope filtering)
`.getVariables()`	`Promise<GitLabPipelineVariable[]>`	Get pipeline variables
`.getTestReport()`	`Promise<GitLabTestReport>`	Get the test report
`.retry()`	`Promise<GitLabPipeline>`	Retry failed jobs
`.cancel()`	`Promise<GitLabPipeline>`	Cancel the pipeline
`.delete()`	`Promise<void>`	Delete the pipeline
`.toJSON()`	`IGitLabPipeline`	Serialize to raw interface

GitLabJob

Method	Returns	Description
`.getLog()`	`Promise<string>`	Get raw job trace/log
`.retry()`	`Promise<GitLabJob>`	Retry the job
`.cancel()`	`Promise<GitLabJob>`	Cancel the job
`.play()`	`Promise<GitLabJob>`	Trigger a manual job
`.erase()`	`Promise<void>`	Erase artifacts and trace
`.toJSON()`	`IGitLabJob`	Serialize to raw interface

Value Classes

Class	Key Properties
`GitLabBranch`	`name`, `commitSha`
`GitLabTag`	`name`, `commitSha`
`GitLabProtectedBranch`	`id`, `name`, `allowForcePush`
`GitLabVariable`	`key`, `value`, `variableType`, `protected`, `masked`, `environmentScope`
`GitLabPipelineVariable`	`key`, `value`, `variableType`
`GitLabTestReport`	`totalTime`, `totalCount`, `successCount`, `failedCount`, `skippedCount`, `errorCount`, `testSuites`
`GitLabTestSuite`	`name`, `totalTime`, `totalCount`, `successCount`, `failedCount`, `skippedCount`, `errorCount`, `testCases`
`GitLabTestCase`	`status`, `name`, `classname`, `executionTime`, `systemOutput`, `stackTrace`

TypeScript Interfaces □□

All raw GitLab API shapes are exported as TypeScript interfaces so you can use them in your own type definitions.

`IListOptions`

```
interface IListOptions {
  search?: string;   // Filter results by keyword
  page?: number;     // Page number (default: 1)
```

```
  perPage?: number;  // Items per page (default: 50)
}
```

IPipelineListOptions

Extends `IListOptions` with pipeline-specific filters:

```
interface IPipelineListOptions extends IListOptions {
  status?: string;          // Filter by pipeline status
  ref?: string;             // Filter by branch/tag ref
  source?: string;          // Filter by trigger source (push, web, schedule, api, ...)
  scope?: string;           // Filter by scope (running, pending, finished, branches, tags)
  username?: string;        // Filter by the triggering user
  updatedAfter?: string;    // ISO 8601 date — only pipelines updated after this
  updatedBefore?: string;   // ISO 8601 date — only pipelines updated before this
  orderBy?: string;         // Order by field (id, status, ref, updated_at, user_id)
  sort?: string;            // Sort direction (asc, desc)
}
```

IJobListOptions

```
interface IJobListOptions extends IListOptions {
  scope?: string[];  // Filter by job scope(s): created, pending, running, failed, success,
...
}
```

IVariableOptions

```
interface IVariableOptions {
  protected?: boolean;         // Only expose to protected branches/tags
  masked?: boolean;            // Mask the value in job logs
  environment_scope?: string;  // Environment scope (default: '*')
}
```

ITestConnectionResult

```
interface ITestConnectionResult {
  ok: boolean;
  error?: string;  // Present when ok is false
}
```

All raw data interfaces (`IGitLabProject`, `IGitLabGroup`, `IGitLabPipeline`, `IGitLabJob`, `IGitLabVariable`, `IGitLabBranch`, `IGitLabTag`, `IGitLabProtectedBranch`, `IGitLabPipelineVariable`, `IGitLabTestReport`, `IGitLabTestSuite`, `IGitLabTestCase`, `IGitLabUser`) are also exported for advanced use cases.

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

Revision #7
Created 2026-03-28 10:48:15 UTC by foss.global Team
Updated 2026-03-28 12:13:28 UTC by foss.global Team