readme.md for @apiclient.xyz/gitea

A fully typed TypeScript client for the Gitea API. Manage repositories, organizations, secrets, branches, tags, and CI/CD action runs with a clean, object-oriented interface and built-in auto-pagination.

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/gitea
# or
pnpm install @apiclient.xyz/gitea
```

Quick Start

```
import { GiteaClient } from '@apiclient.xyz/gitea';

// 1. Connect to your Gitea instance
const client = new GiteaClient('https://gitea.example.com', 'your-api-token');

// 2. Verify the connection
const { ok } = await client.testConnection();
console.log(ok ? '□ Connected!' : '□ Connection failed');
```

```
// 3. List all your repos
const repos = await client.getRepos();
repos.forEach(repo => console.log(`□□${repo.fullName}`));
```

That's it — you're up and running. □□

Usage

All examples use ESM imports and top-level `await`. The package ships as a **pure ESM module** with full TypeScript type declarations.

□□ Creating a Client

```
import { GiteaClient } from '@apiclient.xyz/gitea';

const client = new GiteaClient('https://gitea.example.com', 'your-api-token');
```

- `baseUrl` — Your Gitea instance root URL (trailing slashes are stripped automatically)
- `token` — An API token generated in Gitea under **Settings > Applications**

□□ Testing the Connection

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

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

□□ Repositories

The client returns rich `GiteaRepository` objects with built-in methods for managing branches, tags, secrets, action runs, and more.

List / Search Repositories

```
// Get all repositories (auto-paginated — fetches every page automatically)
const allRepos = await client.getRepos();

// Search with a query
const results = await client.getRepos({ search: 'my-project' });

// Grab a specific page
const page2 = await client.getRepos({ page: 2, perPage: 20 });

for (const repo of results) {
  console.log(`${repo.fullName} — ${repo.description}`);
  console.log(`  Default branch: ${repo.defaultBranch}`);
  console.log(`  Private: ${repo.isPrivate}`);
  console.log(`  Topics: ${repo.topics.join(', ')}`);
}
```

Get a Single Repository

```
const repo = await client.getRepo('my-org/my-repo');
console.log(repo.fullName, repo.htmlUrl);
```

Create a Repository in an Organization

```
const newRepo = await client.createOrgRepo('my-org', 'new-service', {
  description: 'A brand new microservice',
  private: true,
});
console.log(`Created: ${newRepo.htmlUrl}`);
```

Update Repository Properties

```
const repo = await client.getRepo('my-org/my-repo');

await repo.update({
  description: 'Updated description',
  defaultBranch: 'develop',
  private: false,
  archived: false,
```

```
});
```

Manage Topics

```
await repo.setTopics(['typescript', 'api', 'gitea']);
```

Avatars

```
// Upload an avatar (base64-encoded image)
await repo.setAvatar(base64ImageString);

// Remove the avatar
await repo.deleteAvatar();
```

Transfer Ownership

```
await repo.transfer('new-owner-org');
```

Delete a Repository

```
await repo.delete();
```

□□ Organizations

Organizations are returned as rich `GiteaOrganization` objects.

List All Organizations

```
// Auto-paginated — fetches every org across all pages
const orgs = await client.getOrgs();

for (const org of orgs) {
  console.log(`${org.name} (${org.repoCount} repos) — ${org.visibility}`);
}
```

Get a Single Organization

```
const org = await client.getOrg('my-org');
console.log(org.fullName, org.description);
```

Create an Organization

```
const newOrg = await client.createOrg('new-team', {
  fullName: 'New Team',
  description: 'Our shiny new team',
  visibility: 'public',
});
```

List Repos in an Organization

```
const org = await client.getOrg('my-org');
const repos = await org.getRepos({ search: 'api' });
```

Update Organization Properties

```
await org.update({
  description: 'Updated org description',
  visibility: 'private',
  fullName: 'My Organization (Renamed)',
});
```

Organization Avatars

```
await org.setAvatar(base64ImageString);
await org.deleteAvatar();
```

Delete an Organization

```
await org.delete();
```

□□ Secrets Management

Manage Gitea Actions secrets at both the **repository** and **organization** level. Perfect for CI/CD automation.

Repository Secrets

```
const repo = await client.getRepo('my-org/my-repo');

// List all secrets
const secrets = await repo.getSecrets();
for (const secret of secrets) {
  console.log(`□□${secret.name} (created ${secret.createdAt})`);
}

// Create or update a secret
await repo.setSecret('DEPLOY_TOKEN', 's3cret-value');

// Delete a secret
await repo.deleteSecret('DEPLOY_TOKEN');
```

Organization Secrets

```
const org = await client.getOrg('my-org');

// List all org-level secrets
const orgSecrets = await org.getSecrets();

// Create or update
await org.setSecret('NPM_TOKEN', 'npm_abc123');

// Delete
await org.deleteSecret('NPM_TOKEN');
```

□□ Branches & Tags

Fetch branches and tags for any repository — auto-paginated by default.

```
const repo = await client.getRepo('my-org/my-repo');

// Get all branches
const branches = await repo.getBranches();
for (const branch of branches) {
```

```
  console.log(`Branch: ${branch.name} @ ${branch.commitSha}`);
}

// Get all tags
const tags = await repo.getTags();
for (const tag of tags) {
  console.log(`Tag: ${tag.name} @ ${tag.commitSha}`);
}
```

⚡ Gitea Actions (CI/CD Runs)

Full support for listing, filtering, inspecting, and managing Gitea Actions workflow runs and jobs.

List Action Runs

```
const repo = await client.getRepo('my-org/my-repo');

// Get all recent runs (auto-paginated)
const runs = await repo.getActionRuns();

for (const run of runs) {
  console.log(`#${run.runNumber} ${run.displayTitle}`);
  console.log(`  Status: ${run.resolvedStatus}`);
  console.log(`  Branch: ${run.headBranch}`);
  console.log(`  Event: ${run.event}`);
  console.log(`  Duration: ${run.duration}s`);
  console.log(`  Triggered by: ${run.actorLogin}`);
}
```

Filter Runs

```
// Filter by status, branch, event, or actor
const failedRuns = await repo.getActionRuns({ status: 'failed' });
const mainRuns = await repo.getActionRuns({ branch: 'main' });
const pushRuns = await repo.getActionRuns({ event: 'push' });
const myRuns = await repo.getActionRuns({ actor: 'phil' });

// Combine filters
```

```
const filtered = await repo.getActionRuns({
  status: 'running',
  branch: 'develop',
  event: 'pull_request',
});
```

Supported status values: `running`, `failed`, `pending`, `success`, `skipped`, `waiting`, `canceled` — automatically translated to Gitea's native API values.

Inspect Jobs & Steps

```
const runs = await repo.getActionRuns();
const latestRun = runs[0];

// Get all jobs in a run
const jobs = await latestRun.getJobs();

for (const job of jobs) {
  console.log(`Job: ${job.name} [${job.resolvedStatus}] (${job.duration}s)`);
  console.log(`  Runner: ${job.runnerName}`);

  // Inspect individual steps
  for (const step of job.steps) {
    console.log(`    Step ${step.number}: ${step.name} — ${step.resolvedStatus}
(${step.duration}s)`);
  }
}
```

Fetch Job Logs

```
const jobs = await latestRun.getJobs();
const log = await jobs[0].getLog();
console.log(log); // Raw log output
```

Re-run a Workflow

```
// Re-dispatches the workflow on the same ref
await latestRun.rerun();

// With custom inputs
```

```
await latestRun.rerun({ environment: 'staging' });
```

Delete an Action Run

```
await latestRun.delete();
```

□□ Auto-Pagination

By default, all list methods (`getRepos`, `getOrgs`, `getBranches`, `getTags`, `getActionRuns`, etc.) **auto-paginate** and return every item across all pages.

To disable auto-pagination and fetch a specific page, pass the `page` option:

```
// Only fetch page 3
const page3 = await client.getRepos({ page: 3, perPage: 25 });
```

The `autoPaginate` helper is also exported if you need it for custom endpoints:

```
import { autoPaginate } from '@apiclient.xyz/gitea';
```

□□ Exported Classes

Class	Description
`GiteaClient`	Main entry point — connects to a Gitea instance
`GiteaOrganization`	Rich object for an organization with repos, secrets, avatars
`GiteaRepository`	Rich object for a repo with branches, tags, secrets, runs
`GiteaBranch`	Branch with name and commit SHA
`GiteaTag`	Tag with name and commit SHA
`GiteaSecret`	Secret metadata (name + creation timestamp)
`GiteaActionRun`	CI/CD workflow run with jobs, rerun, and delete
`GiteaActionRunJob`	Individual job within a run, with steps and log access
`GiteaActionRunJobStep`	Individual step within a job

□□ Exported Interfaces

All interfaces are exported for type annotations:

```
import type {
  IGiteaUser,
  IGiteaRepository,
  IGiteaOrganization,
  IGiteaSecret,
  IGiteaBranch,
  IGiteaTag,
  IGiteaActionRun,
  IGiteaActionRunJob,
  IGiteaActionRunJobStep,
  ITestConnectionResult,
  IListOptions,
  IActionRunListOptions,
} from '@apiclient.xyz/gitea';
```

□□ Exported Helpers

Utility functions for working with Gitea API data:

Helper	Description
`autoPaginate(fetchPage, opts?)`	Auto-paginate any list endpoint
`computeDuration(startedAt, completedAt)`	Compute duration in seconds from ISO timestamps
`resolveGiteaStatus(status, conclusion)`	Resolve Gitea's split status/conclusion into a single string
`extractRefFromPath(path)`	Extract a human-readable ref from Gitea's `path` field
`extractWorkflowIdFromPath(path)`	Extract the workflow filename from Gitea's `path` field
`toGiteaApiStatus(status)`	Translate friendly status names to Gitea API values

Full API Reference

GiteaClient

Method	Signature	Description
`constructor`	`(baseUrl: string, token: string)`	Create a new client
`testConnection`	`() => Promise<ITestConnectionResult>`	Verify credentials and connectivity
`getRepos`	`(opts?: IListOptions) => Promise<GiteaRepository[]>`	List/search repositories (auto-paginated)
`getRepo`	`(ownerRepo: string) => Promise<GiteaRepository>`	Get a single repository
`createOrgRepo`	`(orgName, name, opts?) => Promise<GiteaRepository>`	Create a repo in an organization
`getOrgs`	`(opts?: IListOptions) => Promise<GiteaOrganization[]>`	List organizations (auto-paginated)
`getOrg`	`(orgName: string) => Promise<GiteaOrganization>`	Get a single organization
`createOrg`	`(name, opts?) => Promise<GiteaOrganization>`	Create a new organization

GiteaRepository

Method	Description
`getBranches(opts?)`	List branches (auto-paginated)
`getTags(opts?)`	List tags (auto-paginated)
`getSecrets()`	List repository secrets
`setSecret(key, value)`	Create or update a secret
`deleteSecret(key)`	Delete a secret
`getActionRuns(opts?)`	List CI/CD runs with optional filters
`update(data)`	Update repo properties (name, description, default branch, etc.)
`setTopics(topics)`	Replace all repository topics
`setAvatar(base64)`	Upload an avatar image
`deleteAvatar()`	Remove the avatar
`transfer(newOwner)`	Transfer ownership
`delete()`	Delete the repository

GiteaOrganization

Method	Description
`getRepos(opts?)`	List org repositories (auto-paginated)
`getSecrets()`	List organization secrets
`setSecret(key, value)`	Create or update an org secret
`deleteSecret(key)`	Delete an org secret
`update(data)`	Update org properties (description, visibility, fullName)
`setAvatar(base64)`	Upload an avatar image
`deleteAvatar()`	Remove the avatar
`delete()`	Delete the organization

`GiteaActionRun`

Property	Type	Description
`id`	`number`	Run ID
`runNumber`	`number`	Sequential run number
`name`	`string`	Workflow name
`displayTitle`	`string`	Display title of the run
`status`	`string`	Raw status (running, waiting, completed)
`conclusion`	`string`	Raw conclusion (success, failure, cancelled)
`resolvedStatus`	`string`	**Computed** — single human-readable status
`duration`	`number`	**Computed** — duration in seconds
`headBranch`	`string`	Branch that triggered the run
`headSha`	`string`	Commit SHA
`event`	`string`	Trigger event (push, pull_request, etc.)
`ref`	`string`	**Computed** — human-readable ref
`workflowId`	`string`	**Computed** — workflow filename
`actorLogin`	`string`	User who triggered the run

Method	Description
`getJobs()`	List all jobs in this run
`rerun(inputs?)`	Re-dispatch the workflow on the same ref

Method	Description
`delete()`	Delete this action run

`GiteaActionRunJob`

Property	Type	Description
`id`	`number`	Job ID
`name`	`string`	Job name
`resolvedStatus`	`string`	**Computed** — resolved status
`duration`	`number`	**Computed** — duration in seconds
`steps`	`GiteaActionRunJobStep[]`	Individual steps
`runnerName`	`string`	Runner that executed the job

Method	Description
`getLog()`	Fetch raw log output for this job

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

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.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #7
Created 2026-03-28 10:48:13 UTC by foss.global Team
Updated 2026-03-28 12:13:25 UTC by foss.global Team