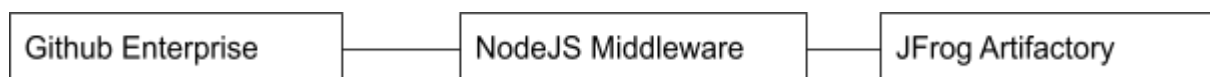


Alternative B

Alternative B	1
High Level Solution	1
Tools Selection	2
Assumptions	2
Tools Initial Setup	3
Github Enterprise	3
NodeJS Middleware	3
JFrog Artifactory	3
Design	3
Implementation	4
NodeJS Middleware	4
Deep into implementation	5
Endpoint POST sync	5
Security aspects	7
Design aspects	8
Github Enterprise	8
Creating a organization webhook	8
JFrog Artifactory	11
Authentication of REST API Calls	11
REST API Endpoints Usage	12
Integration Tests	12
Case 1	12
Case 2	12
Case 3	12

High Level Solution

According the description of the exercise “Alternative B”, at high level, three main services can be identified:



- Github Enterprise: where users/ groups and repositories are created. Also the event on a new issue is generated here.
- NodeJS Middleware: handle the event on a new issue and perform actions over the Artifactory instance adjusting the repositories, users , groups and permissions.

- JFrog Cloud Artifactory: package repository where the NodeJS Middleware will perform actions based on the event info received from Github Enterprise instance.

So, the idea of this implementation is create a middleware which handle the Artifactory's entities when Github event is detected (taking advantage of Github's Webhooks)

Notice: in my opinion the idea of performing actions over an Artifactory based on new issue events (over a Github's repository) is not a common scenario but for testing purposes is useful because to demonstrate the integration capabilities between Github and JFrog Artifactory. Probably a more realistic scenario (and maybe out of scope of this test) could be to create a custom action event which is part of a CICD pipeline.

Tools Selection

Assumptions

- Github Enterprise Tool used is trial version
- JFrog Artifactory is trial version
- NodeJS Middleware deployed locally in my computer and exposed using [ngrok](#) tool

In order to speed up the integration of the three main parts described in the previous section, the tools selected are going to be deployed in a cloud service (instead on premise) getting rid of potential issues related with certificates, custom installation steps and so on.

- Github Enterprise Tool: Github is de facto versioning/hosting tool, cloud based and is ready to be used instead of on premise alternatives like "Git Server".
- NodeJS Middleware: Node.js is an open-source and cross-platform JavaScript runtime environment. The reason why I choose it instead of other alternatives are:
 - Fast-processing (V8 Javascript Engine) and event-based model (asynchronous, non-blocking, single-threaded nature,)
 - Scalability technology for microservices (lightweight technology)
 - Multiple open source libraries npm (Marketplace)
 - Native support for JSON Format in comparison with other technologies (like Ruby or PHP) NodeJS uses JSON instead of converting binary models.

So, the idea is to create a NodeJS REST API using Express framework. Besides that in order to make it accessible through a public URL , [ngrok](#) tool will be used just for a PoC purpose.

- A real option could be to create a Github Bot using [probot](#) but for this test's objective is out of scope due to learning curve and time consumption. But definitely is the best solution for a real github organization

- JFrog Artifactory Cloud: given the cloud free version meets the minimum requirements needed for this exercise (handle repositories, users, permission and groups through REST API) and also many others feature out of the box like reachable through a public, secure URL.

Tools Initial Setup

Github Enterprise

- A Github Organization was created <https://github.com/norbertorodas-cariad-techtask-org>
- Github Organization teams created: **writers, readers, maintainers**
- Github Organization repositories created: **techtask-audi-repo, techtask-vw-repo, techtask-seat-repo** (*)
- 4 Users invited manually (**) . **The role by default will be member**
- For Rest API usage a personal token will be created. Then this be used by NodeJS Middleware

(*) could be a repetitive task. There are native options like [template-repository](#) or do not repeat repository settings <https://probot.github.io/apps/settings/>

(**) In Alternative A this users should be synchronized for a custom tool from Keycloak instance

NodeJS Middleware

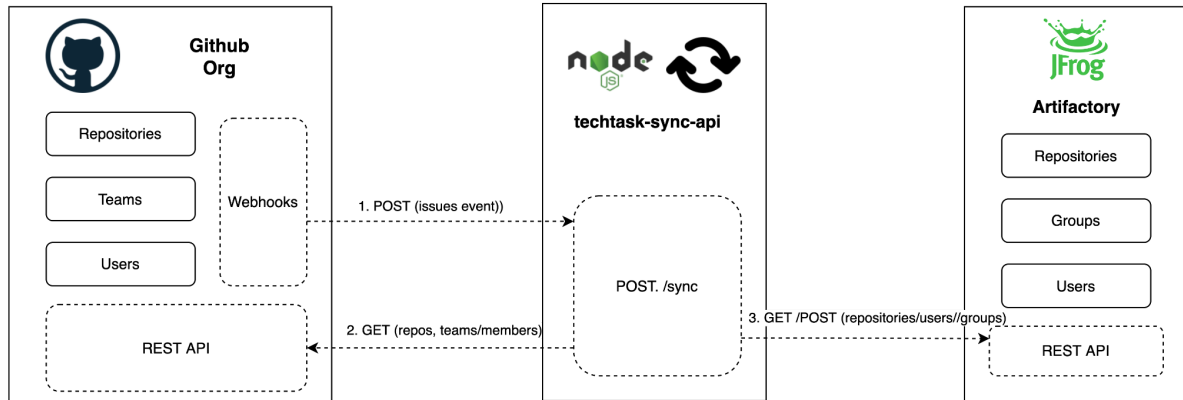
- Repository created <https://github.com/norbertorodas-cariad-techtask-org/sync-github-jfrog-artifactory>
- NodeJS Express REST API deployed and running locally.
 - Exposed through a public URL through [ngrok](#) tool.
 - Github and JFrog credentials will be stored as environment variables (testing purpose)

JFrog Artifactory

- A cloud instance is deployed with domain <https://nrodas.jfrog.io>
- A service user will be created with the name **techttest-sincronizator** and proper permissions. Then username and API Key will be used by NodeJS Middleware

Design

The following diagram describes the integration between the services



Implementation

NodeJS Middleware

Given testing purposes we will expose the NodeJS REST API using [ngrok](#) tool which enables us to expose a local development server to the Internet with minimal effort (*)

(*) This alternative was chosen for the sake of simplicity (<https://dashboard.ngrok.com/get-started/setup>). More robust, scalable and secure approaches could be:

- Create a user plugin for JFrog Artifactory. Then the plugin is in charge to handle the whole synchronization logic between Github and Artifactory. More details [here](#)
- Expose the NodeJS REST API in a reachable server using SSL.
- Execute or sync logic code as Lambda function provided by a well known cloud provider.

- After ngrok is installed on local, localhost:8082 is exposed . Then we can run our NodeJS REST API in our local environment on port 8082

```

ngrok by @inconshreveable

Session Status      online
Account             betens85 (Plan: Free)
Version             2.3.40
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://01d4-188-192-137-111.ngrok.io -> http://localhost:8082
Forwarding           https://01d4-188-192-137-111.ngrok.io -> http://localhost:8082

Connections          ttl      opn      rt1      rt5      p50      p90
                    5        0        0.00     0.00     10.05    10.10

```

The URL is `http://<random>.ngrok.io` where **<name>** is a random name

(*) More details [here](#)

Deep into implementation

As is mentioned in the previous section the language selected is NodeJS, the web framework “Express” and the lib `axions` (all under MIT License). Besides that for JSON text parsing (github new issue event and REST api calls to Github and Artifactory) `JSON.parse` will be used because is part of the JavaScript standard since ECMAScript 5 and is provided by V8 (Javascript Engine of Node.js).

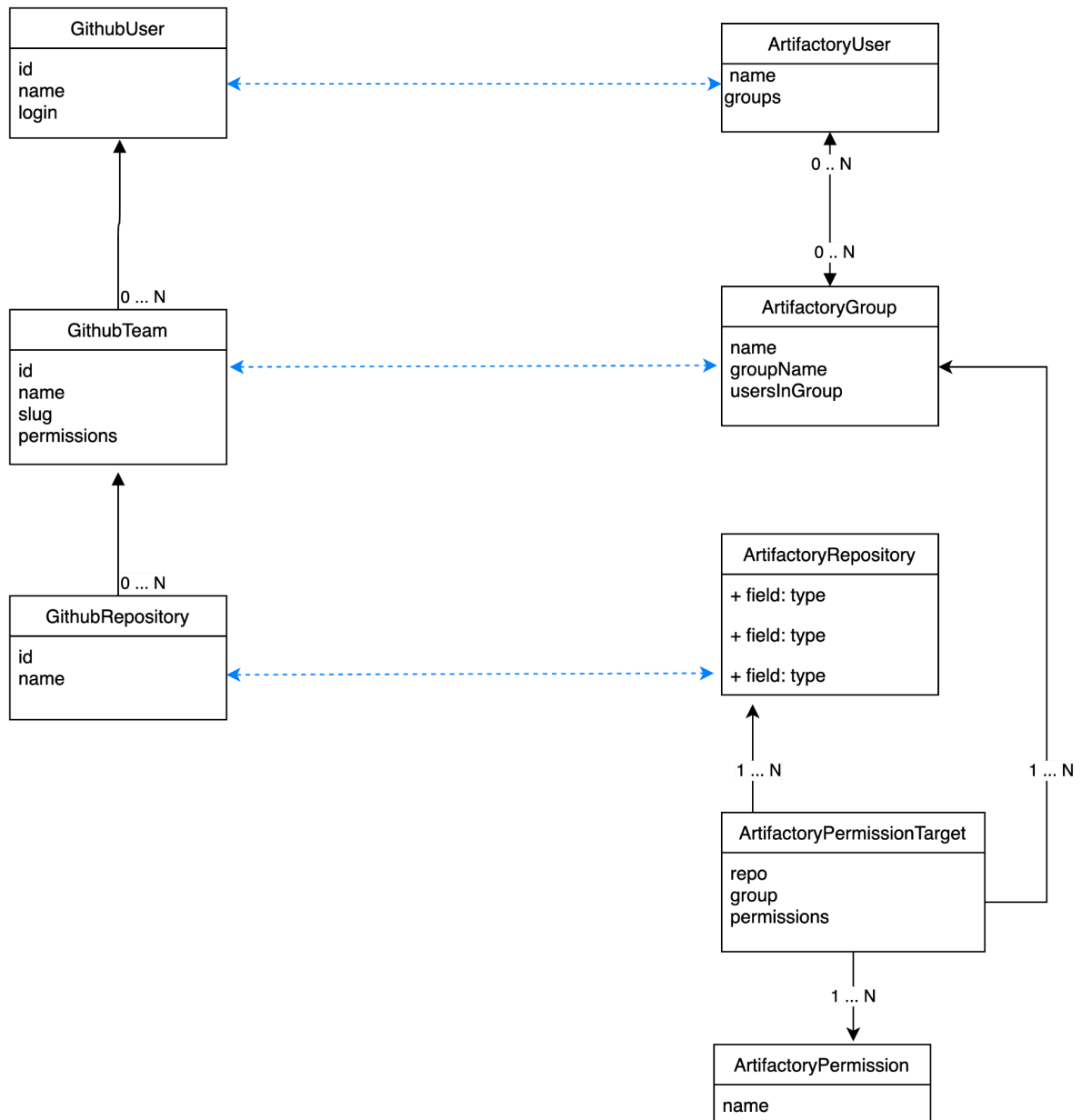
- Npm Axios lib is used because is a promise based http client and simplifies a lot the interaction with Github/Artifactory REST API.

Endpoint POST sync

This endpoint will be pushed with an issue event from Github (webhook configuration). The main purpose of this endpoint is act as mediator between the entities of Github and Artifactory. Internally it will handle multiple requests, obtaining from Github API and Artifactory and doing the synchronization based on a preconfigured mapping structure: this means our implementation must define in advance the mapping between the entities (Github - Artifactory):

- Repositories - Repositories
- Teams - Groups
- Users - Users
- Permissions - PermissionsTarget (*)

(*) PermissionTarget is an entity which could contain permissions for users/groups given on or more repositories. I decided to map into a permissionTarget repository multiple permissions of github team to 1 artifactory repository and 1 artifactory group (simplicity)



Github Permission	Artfactory Permission	Description
pull	read, annotate	Configured per repository and group
push	read,annotate, write, delete	Configured per repository and group

Security aspects

- Github REST API and Artifactory Credentials will be injected as environment variables into the VM where our NodeJS Middleware (**techtest-sync-api**) is deployed.
- As an extra layer of security we will create a SECRET variable into our Github Organization then we should verify specific headers received and make a validation. More details [here](#)

We can create an organization's secret **SYNCSECRET** which could be used into the NodeJS REST API side in order to validate when a request effectively is coming from Github (**very important**). This will be excluded for the scope of this PoC in order to not increase the complexity of the source code (lib candidate : npm-js)



norbertorodas-cariad-techtask-org

Organization account [Switch to another account](#) ▼

Account settings

Profile

Billing & plans

Member privileges

Organization security

Security & analysis

Verified & approved domains

Audit log

Sponsorship log

Webhooks

Third-party access

Installed GitHub Apps

Actions secrets / New secret

Name

SYNCSECRET

Value

myvalue

Repository access

Private repositories ▼

Add secret

Design aspects

- Rate limit: in our case is not specified but probably is influenced of rate limits of external services like Github (5000 per hour) and Artifactory
- The endpoint is async given Github does not care about the response of the middleware.
- Logging: logs will be presented in console, but a better option is use a library and create persistent logs files given the nature of our middleware app (server side). Then logs are useful for troubleshooting issues.

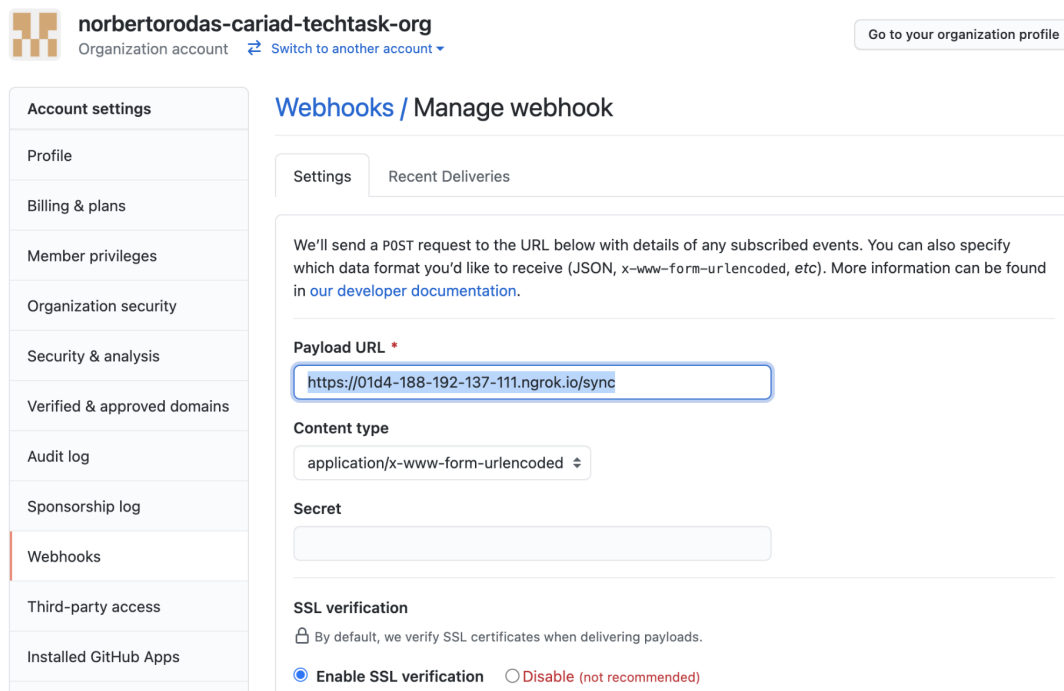
Github Enterprise

Webhooks, on the other hand, are automated calls from server1 to server2. Those calls are triggered when a specific event happens on server1. In our case, server1 is Github Enterprise and server2 is Artifactory. More details about webhooks in github [here](#)

So, for simplicity, I decided to proceed with an organization webhook configuration, so using this approach we can configure a unique webhook which will be shared for all the repositories that are part of my organization. This means we have a single point of configuration which simplifies the whole integration and avoids repetition of configurations (creating multiple webhooks, one per repository).

Creating a organization webhook

1. Go to the organization settings section and choose “Webhooks”. Then the url of our endpoint is added and the content type is selected.



The screenshot shows the GitHub organization settings page for 'norbertorodas-cariad-techtask-org'. The left sidebar contains a list of settings: Account settings, Profile, Billing & plans, Member privileges, Organization security, Security & analysis, Verified & approved domains, Audit log, Sponsorship log, Webhooks (highlighted), Third-party access, and Installed GitHub Apps. The main content area is titled 'Webhooks / Manage webhook' and has two tabs: 'Settings' and 'Recent Deliveries'. The 'Settings' tab is active, showing a configuration form. The form includes a 'Payload URL' field with the value 'https://01d4-188-192-137-111.ngrok.io/sync', a 'Content type' dropdown menu set to 'application/x-www-form-urlencoded', a 'Secret' field, and an 'SSL verification' section with the option 'Enable SSL verification' selected. A note at the bottom states: 'By default, we verify SSL certificates when delivering payloads.'

2. Finally in events sections we choose “Issues” and the option “Active”

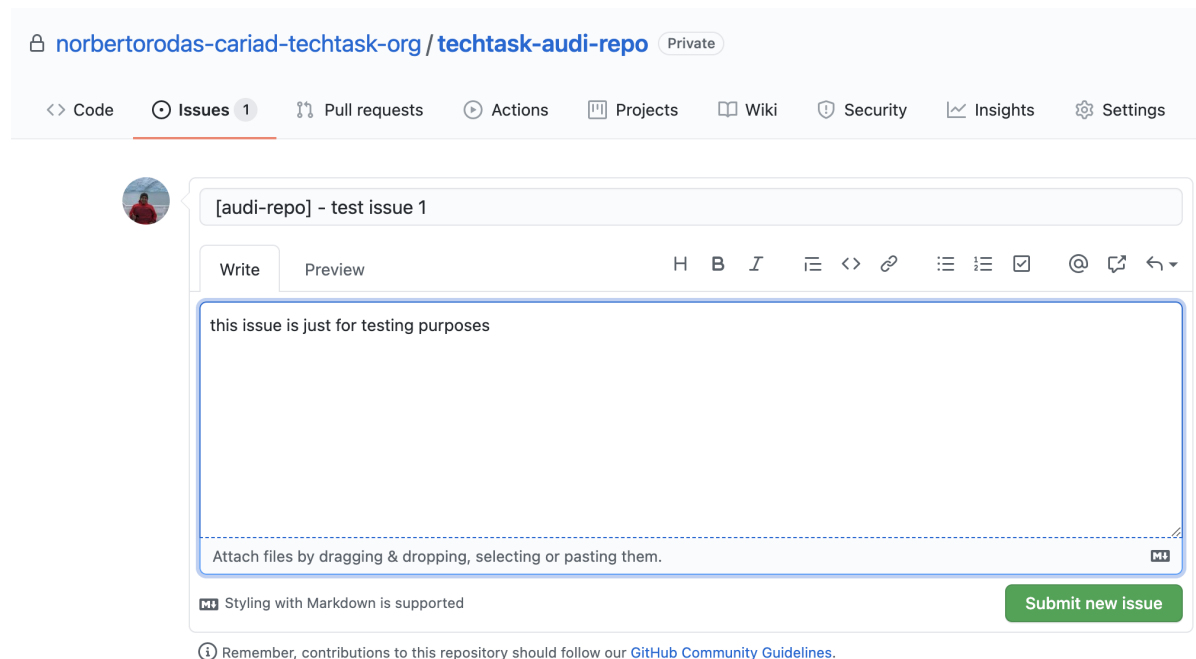
Which events would you like to trigger this webhook?

- ☐ Just the push event.
- ☐ Send me **everything**.
- ☒ Let me select individual events.

3. Once we configured this our organization webhook is ready to push the payload of our events directly to our NodeJS Middleware which is the main core of our test solution.

4. Fast integration test (from github to NodeJS API without JFrog integration):

- a. Create an issue on repository **techtask-audi-repo**
<https://github.com/norbertorodas-cariad-techtask-org/techtask-audi-repo/issues/4>



The screenshot shows the GitHub interface for creating a new issue in the repository 'norbertorodas-cariad-techtask-org / techtask-audi-repo'. The repository is marked as 'Private'. The navigation bar includes links for Code, Issues (1), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Issues' tab is active, showing a new issue titled '[audi-repo] - test issue 1'. The issue body contains the text 'this issue is just for testing purposes'. Below the text area, there is a note about attaching files and a 'Submit new issue' button. At the bottom, a reminder states: 'Remember, contributions to this repository should follow our GitHub Community Guidelines.'

- b. Then we can see the payload received by our NodeJS REST API **techtest-sync-api** through the POST endpoint /sync

```
Norbertos-MacBook-Pro-2:techtask-sync-api beto$ node server.js
Server started at http://localhost:8082
```

```
payload: '{"action": "opened", "issue": {"url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-audi-repo/issues/4", "repository_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-audi-repo", "labels_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-audi-repo/issues/4/labels/{name}", "comments_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-audi-repo/issues/4/comments", "events_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-audi-repo/issues/4/events", "html_url": "https://github.com/norbertorodas-carlad-techtask-org/techtask-audi-repo/issues/4", "id": "978531821", "node_id": "MDU6SXN2dWUSNzg1MzE4MjE=", "number": "4", "title": "[audi-repo] - test issue 1", "user": {"login": "norbertorodas", "id": "71797591", "node_id": "MDQ6VXNlcnZkZkZkXk", "avatar_url": "https://avatars.githubusercontent.com/u/71797591?v=4", "gravatar_id": "", "url": "https://api.github.com/users/norbertorodas", "html_url": "https://github.com/norbertorodas", "followers_url": "https://api.github.com/users/norbertorodas/followers", "following_url": "https://api.github.com/users/norbertorodas/following/{other_user}", "gists_url": "https://api.github.com/users/norbertorodas/gists/{gist_id}", "starred_url": "https://api.github.com/users/norbertorodas/starred/{owner}/{repo}", "subscriptions_url": "https://api.github.com/users/norbertorodas/subscriptions", "organizations_url": "https://api.github.com/users/norbertorodas/orgs", "repos_url": "https://api.github.com/users/norbertorodas/repos", "events_url": "https://api.github.com/users/norbertorodas/events/{privacy}"}, "received_events_url": "https://api.github.com/users/norbertorodas/received_events", "type": "User", "site_admin": false}, "labels": [], "state": "open", "locked": false, "assignee": null, "assignees": [], "milestone": null, "comments": 0, "created_at": "2021-08-24T22:23:56Z", "updated_at": "2021-08-24T22:23:56Z", "closed_at": null, "author_association": "CONTRIBUTOR", "active_lock_reason": null, "body": "this issue is just for testing pur
```

c. Create an issue on another repository of organization **techtask-vw-repo**

d. Finally we can see the payload received by our NodeJS REST API **techtest-sync-api** through the POST endpoint /sync

```

    "payload": {
      "action": "opened",
      "issue": {
        "url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-vw-repo/issues/2",
        "repository_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-vw-repo",
        "labels_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-vw-repo/issues/2/labels{name}",
        "comments_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-vw-repo/issues/2/comments",
        "events_url": "https://api.github.com/repos/norbertorodas-carlad-techtask-org/techtask-vw-repo/issues/2/events",
        "html_url": "https://github.com/norbertorodas-carlad-techtask-org/techtask-vw-repo/issues/2",
        "id": 978533173,
        "node_id": "MDU6SXN2dWU5Ng1MzNxMzE=",
        "number": 2,
        "title": "[Vw-repo] - test issue",
        "user": {
          "login": "norbertorodas",
          "id": 71797591,
          "node_id": "MDQ6VXN1cjc4NzNkNTk=",
          "avatar_url": "https://avatars.githubusercontent.com/u/71797591?v=4",
          "gravatar_id": "",
          "url": "https://api.github.com/users/norbertorodas",
          "followers_url": "https://api.github.com/users/norbertorodas/followers",
          "following_url": "https://api.github.com/users/norbertorodas/following/other_user",
          "gists_url": "https://api.github.com/users/norbertorodas/gists/gist_id",
          "starred_url": "https://api.github.com/users/norbertorodas/starred/{owner}/{repo}",
          "subscriptions_url": "https://api.github.com/users/norbertorodas/subscriptions",
          "organizations_url": "https://api.github.com/users/norbertorodas/orgs",
          "repos_url": "https://api.github.com/users/norbertorodas/repos",
          "events_url": "https://api.github.com/users/norbertorodas/events/privacy",
          "received_events_url": "https://api.github.com/users/norbertorodas/received_events",
          "type": "User",
          "site_admin": false,
          "labels": [],
          "state": "open",
          "locked": false,
          "assignee": null,
          "assignees": [],
          "milestone": null,
          "comments": 0,
          "created_at": "2021-08-24T22:26:50Z",
          "updated_at": "2021-08-24T22:26:50Z",
          "closed_at": null,
          "author_association": "CONTRIBUTOR",
          "active_lock_reason": null,
          "body": "testing purposes",
          "performed_via_github_app": null,
          "repository": {
            "id": 399239898,
            "node_id": "MDw0LjEjC9za0RvcnkzOTkyMzQ0Tg=",
            "name": "techtask-vw-repo",
            "full_name": "norbertorodas-carlad-techtask-org/techtask-vw-repo"
          }
        }
      }
    }
  }
}

```

JFrog Artifactory

Authentication of REST API Calls

There are 4 alternatives to authenticate all the requests performed against the REST API:

1. Basic authentication using your username and password
2. Basic authentication using your username and **API Key**.
3. Using a dedicated header (X-JFrog-Art-Api) with your API Key.
4. Using an **access token** instead of a password for basic authentication.
5. Using an **access token** as a bearer token in an authorization header (Authorization: Bearer) with your access token.

I decided to proceed with option 2 because it is simplest and is enough for the scope of this test. A more secure and reliable option is 5.

The API KEY will be generated manually as is described in this [guide](#) but the process can be automated using REST API calls.

REST API Endpoints Usage

A brief overview of the endpoints used by NodeJS Middleware to handle Artifactory resources:

- List repositories
- Create repository
- List groups
- Create group
- Get Users
- Create or Replace User
- Get User Details
- Get Repository
- Create/Replace PermissionTarget

Integration Tests

Case 1

Precondition: **techtest-audi-repo** is configured with 2 teams (readers, writers) and one user on each team.

Scenario: Repository ,users and groups does not exists in JFrog

Expected: a new repository is created in JFrog with the same name as the issue's github repository name. Also the users and groups (teams in github) are created into Artifactory.

Middleware logs

```
[INFO] The issue event is opened
[INFO] - getArtifactoryRepository
[INFO] - getArtifactoryRepository - repository.name= techtask-audi-repo
[INFO] - internalArtifactoryRepositoryHandling
[INFO] - internalArtifactoryGroupsHandling
[INFO] - getGithubTeamsPerRepository
[INFO] - getGithubTeamsPerRepository - repository.name= techtask-audi-repo
[INFO] - getArtifactoryGroupByName
[INFO] - createUpdateArtifactoryUsers
[INFO] - getGithubTeamMembers
[INFO] - getGithubTeamMembers - team= readers
[INFO] - getArtifactoryUserByName
[INFO] - getArtifactoryUserByName
[INFO] - createArtifactoryGroup
[INFO] - handlePermissionTargetInArtifactoryPerRepoAndGroup
permissionsInArtifactory read,annotate
[INFO] - getArtifactoryGroupByName
[INFO] - createUpdateArtifactoryUsers
[INFO] - getGithubTeamMembers
[INFO] - getGithubTeamMembers - team= writers
[INFO] - getArtifactoryUserByName
[INFO] - getArtifactoryUserByName
[INFO] - createArtifactoryGroup
[INFO] - handlePermissionTargetInArtifactoryPerRepoAndGroup
permissionsInArtifactory read,annotate,write,delete
```

Artifactory repository

The screenshot shows the JFrog Platform Admin UI. The browser address bar displays `nrondas.jfrog.io/ui/admin/repositories/local/techtask-audi-repo/edit`. The left sidebar contains the 'Administration' menu with options like 'Repositories', 'Layouts', 'Identity and Access', 'Security', 'General', and 'Monitoring'. The main content area is titled 'Edit techtask-audi-repo Repository' and features two tabs: 'Basic' (selected) and 'Advanced'. Under the 'Basic' tab, there are four fields: 'Package Type' (set to 'Generic'), 'Repository Key' (set to 'techtask-audi-repo'), 'Repository Layout' (set to 'maven-2-default'), and 'Public Description' (empty). The bottom of the sidebar has a 'Getting Started' button.

Artifactory groups

← → ↺

nrodas.jfrog.io/ui/admin/management/groups

JFrog Platform

Search admin resources

Administration

Repositories

Identity and Access

Users

Groups

Permissions

Access Tokens

Security

General

Groups

Filter

Group Name ^	Permissions	Extern.
example-group	1 testpermission	✓
readers	1 techtask-audi-repo-readers-pt	
writers	1 techtask-audi-repo-writers-pt	

Artifactory permissionTargets

← → ↺

nrodas.jfrog.io/ui/admin/management/permissions

JFrog Platform

Search admin resources

Administration

Repositories

Identity and Access

Users

Groups

Permissions

Access Tokens

Security

General

Monitoring

SERVICES

Artifactory

Permissions

Search by permission name

Q

Permission Target Name ^	Groups
Any Remote	-
Anything	-
techtask-audi-repo-readers-pt	1 readers
techtask-audi-repo-writers-pt	1 writers
testpermission	1 example-group

Case 2

Precondition: **techtest-audi-repo** is configured with 2 teams (readers, writers) and one user on each team.

Scenario: Repository, users and groups already exists in JFrog

Expected: JFrog entities (repositories, users, groups) must not change

Case 3

Precondition: **techtest-audi-repo** is configured with 1 teams (readers) with 2 users

Scenario: Repository, users and groups already exists in JFrog

Expected: the JFrog repository 'techtest-audi-repo' is not recreated. Group 'readers' is updated (2 users instead 1) and group 'writers' is removed.