Seed Auth API Documentation

Release 0.0.0

Praekelt Foundation

Jan 06, 2017

Contents

Contents:

CHAPTER 1

Seed Auth HTTP API

1.1 Authentication

Authentication is done via token authentication.The tokens endpoint can be used to create a token. The token can then be placed in the header for requests to the other endpoints.

Example request:

```
POST /endpoint/ HTTP/1.1
Content-Type: application/json
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

1.2 Permissions

While you can store any string as a permission, there are a few permissions that have specific meaning within the Seed Auth API.

Create permissions are not linked to specific instances, whereas admin, read, and write permissions are.

A user automatically has read/write permissions for themselves.

All users have read permissions for all users and organizations.

All members of an organization have read access to that organization's teams.

Permissions can be assigned either to a user or a team. All users can add and remove permissions for all teams and users, except for the following permissions, which can only be set be set by a person with org:admin permission.

org:admin Can create/read/write/delete users, and teams that are part of the organization, and can add users to the organization that they are an admin for. Can read/write the organization that they are admin for.

team:admin Can modify the team they have permission for, and add and remove existing users to that team.

GET /user/

Get the user details, and list of permissions that a user currently has access to, for the user authorized by the given token.

Response Headers

- Authorization – "Token " followed by the token to verify

Status Codes

- 200 OK – The token is valid.
- 401 Unauthorized – The token is invalid/missing.

Example request:

```
GET /user/ HTTP/1.1
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "1",
    "url": "https://example.org/users/1/",
    "first_name": "Jon",
    "last_name": "Snow",
    "email": "jonsnow@castleblack.net",
    "admin": false,
    "active": true,
    "permissions": [
        {
            "id": "2",
            "type": "org:admins",
            "object_id": "3",
            "namespace": "__auth__"
        },
        {
            "id": "5",
            "type": "foo:bar:baz",
            "object_id": "7",
            "namespace": "foo_app"
        }
    ]
}
```

Permission structure:

Each permission is an object containing the following fields.

id The unique ID for the permission

type The string representing the type of permission.

object_id A string that uniquely identifies the object that this permission acts upon. null if this permission does not act on a specific object.

namespace A string used to namespace a set of permissions for a specific app, to avoid "type" collisions.

1.3 Pagination

When the results set is larger than a configured amount, the data is broken up into pages.

You can navigate to specific pages using the 'page' parameter. Links to the next and previous page (if available) will be provided in the 'Link' header.

Example:

```
GET /endpoint/ HTTP/1.1
Authorization: token .....


HTTP/1.1 200 OK
Content-Type: application/json
Link: <https://example.com/endpoint/?page=2>; rel="next"

[....]
```

1.4 Tokens

For the token endpoints, no authentication is required.

POST /user/tokens/

Create a new token for the provided user. This will invalidate all other tokens for that user.

Request JSON Object

- **email** (*str*) – The username of the user to create the token for.
- **password** (*str*) – The password of the user to create the token for.

Response JSON Object

- **token** (*str*) – The generated token.

Status Codes

- 201 Created – When the token is successfully generated.
- 401 Unauthorized – When the user credentials are incorrect.
- 403 Forbidden – When the user is inactive.

Example request:

```
POST /user/tokens/ HTTP/1.1
Content-Type: application/json

{
  "email": "testuser@example.org",
  "password": "testpassword"
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "token": "9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b"
}
```

1.5 Password resets

Attention: This endpoint has not yet been implemented.

For the password reset endpoints, no authentication is required.

To reset a user's password, the following steps should be followed:

1. Make a request to the reset endpoint. This will make an HTTP request to the preconfigured endpoint with the user's details, and a token.
2. Make a request to the confirm endpoint, with the provided token and the new password.

POST /passwords/resets/

Start the process for resetting a user's password.

Request JSON Object

- **email** (*str*) – The email of the user to reset the password for.
- **app** (*str*) – The application that the token should go to, configured in settings. This value is optional, defaults to the default configured application.

Status Codes

- 202 Accepted – The password reset process was started, or the username doesn't exist. The same code is returned for both as to not leak user information

Example request:

```
POST /passwords/resets/ HTTP/1.1
Content-Type: application/json

{"email":"jonsnow@castleblack.org","app":"numi"}
```

Example response:

```
HTTP/1.1 202 Accepted
```

POST /passwords/confirmations/

Reset the users password using the provided token.

Request JSON Object

- **token** (*str*) – The provided token.
- **password** (*str*) – The new password.

Status Codes

- 204 No Content – The password was successfully reset.
- 401 Unauthorized – The token was incorrect.

Example request:

```
POST /password/confirmations/ HTTP/1.1
Content-Type: application/json

{"password":"gh0st","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvbiBTbm93In0.H7huFJ_ioqf1-_
↪qzZQ6VLHOJpnqhdDiZFV2VdkIt7LY"}
```

Example response:

```
HTTP/1.1 204 No Content
```

1.6 Organizations

Organizations provide a grouping of users, although users do not have to belong to an organization, and they can also belong to many organizations. Teams have to belong to exactly one organization, but an organization can have many teams.

`POST /organizations/`

Creates a new organization.

Requires admin user.

Request JSON Object

- **`title`** (*str*) – The title of the organization.

Response JSON Object

- **`title`** (*str*) – The title of the created organization.
- **`id`** (*str*) – The id of the created organization.
- **`teams`** (*list*) – The list of teams that the organization has.
- **`users`** (*list*) – The list of users that are part of the organization.
- **`url`** (*str*) – The URL for this organization.
- **`archived`** (*bool*) – True if the organization has been archived.

Status Codes

- 201 Created – When the organization is successfully generated.
- 400 Bad Request – When there is invalid information to create the organization.

Example request:

```
POST /organizations/ HTTP/1.1
Content-Type: application/json

{
 "title": "Nights Watch"
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json
```

```
{
    "title": "Nights Watch",
    "id": "4",
    "teams": [],
    "url": "https://example.org/organizations/4/",
    "users": [],
    "archived": false
}
```

GET /organizations/

Get a list of existing organizations

Requires any user.

Query Parameters

- **archived** – (optional) If true, shows archived organizations. If false, shows organizations that are not archived. If both, shows all organizations. Defaults to false.

Example request:

```
GET /organizations/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
 {
     "title":"Nights Watch",
     "id":4,
     "teams":[],
     "url":"https://example.org/organizations/4/",
     "users":[],
     "archived": false
 },
 {
     "title":"Brotherhood Without Banners",
     "id":5,
     "teams":[],
     "url":"https://example.org/organizations/5/",
     "users":[],
     "archived": false
 }
]
```

GET /organizations/(**int:** *organization_id*)**/**

Get the details of an organization.

Requires any user.

Response JSON Object

- **title** (*str*) – The title of the created organization.
- **id** (*str*) – The id of the created organization.
- **teams** (*list*) – The list of teams that the organization has.
- **users** (*list*) – The list of users that are a part of the organization.

- **`url`** (*`str`*) – The URL for this organization.
- **`archived`** (*`bool`*) – True if the organization has been archived.

Example request:

```
GET /organizations/4/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "title":"Night's Watch",
 "id":4,
 "teams":[],
 "url":"https://example.org/organizations/4/",
 "users":[],
 "archived": false
}
```

`PUT /organizations/`(int: *organization_id*)`/`

Update an existing organization.

Requires admin user, or any user that has 'org:admin' permission for the specific organization.

Request JSON Object

- **`title`** (*`str`*) – The title of the organization.
- **`archived`** (*`str`*) – True if the organization is to be archived.

Response JSON Object

- **`id`** (*`str`*) – The id of the created organization.
- **`teams`** (*`list`*) – The list of teams that the organization has.
- **`users`** (*`list`*) – The list of users that are part of the organization.
- **`url`** (*`str`*) – The URL for this organization.
- **`archived`** (*`bool`*) – True if the organization has been archived.

Status Codes

- 200 OK – When the organization is successfully generated.
- 400 Bad Request – When there is invalid information to update the organization.

Example request:

```
PUT /organizations/4/ HTTP/1.1
Content-Type: application/json

{
 "title": "Brotherhood Without Banners",
 "archived": false
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "title":"Brotherhood Without Banners",
 "id":4,
 "teams":[],
 "url":"https://example.org/organizations/4/",
 "users":[],
 "archived": false
}
```

DELETE /organizations/ (**int:** *organization_id*) **/**

Archive an organization. The organization will by default no longer be shown when *listing organizations*, the organization's teams will by default no longer be shown when *listing teams*, and any permissions associated with the organization's teams will no longer take effect when checking whether a user has permission to perform an action.

Archiving can be reversed by setting `archived` to `true` when *updating* an organization.

Requires admin user, or any user that has 'org:admin' for this organization.

status 204 Organization successfully archived

Example request:

```
DELETE /organizations/4/ HTTP/1.1
```

Example response:

```
HTTP/1.1 204 No Content
```

PUT /organizations/ (**int:** *organization_id*) **/users/**
int: *user_id* **/** Add a user to an existing organization.

Requires admin user, or any user that has 'org:admin' permissions for that organization.

Status Codes

- 204 No Content – User was successfully added.

Example request:

```
PUT /organizations/4/users/2/ HTTP/1.1
Content-Type: application/json
```

Example response:

```
HTTP/1.1 204 No Content
```

DELETE /organizations/ (**int:** *organization_id*) **/users/**
int: *user_id* **/** Remove a user from an organization.

Requires admin user, or any user that has 'org:admin' permission for that organization.

Status Codes

- 204 No Content – User was successfully removed from an organization

Example request:

```
DELETE /organizations/4/users/2/ HTTP/1.1
```

Example response:

```
HTTP/1.1 204 No Content
```

POST /organizations/(int: *organization_id*)/teams/

Create a new team for an organization

Only admin users, and users with org:admin permissions for the org may create teams.

Request JSON Object

- **title** (*str*) – The title of the team.
- **archived** (*bool*) – True if the team is archived.

Response JSON Object

- **id** (*str*) – The ID of the created team.
- **url** (*str*) – The URL of the created team.
- **title** (*str*) – the title of the team.
- **users** (*list*) – The list of users that belong to this team.
- **organization** (*obj*) – The summary of the organization that the team belongs to.
- **permissions** (*list*) – The permission list for the team.
- **archived** (*bool*) – True if the team is archived.

Status Codes

- 201 Created – Successfully created team.
- 400 Bad Request – Missing required information to create team.

Example request:

```
POST /organizations/7/teams/ HTTP/1.1
Content-Type: application/json

{
    "title": "Lord Commanders",
    "archived": false
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": "2",
    "title": "Lord Commanders",
    "users": [],
    "permissions": [],
    "url": "https://example.org/teams/2/",
    "organization": {
        "url": "https://example.com/organizations/7/",
        "id": "7"
    },
    "archived": false
}
```

GET /organizations/(**int:** *organization_id*)**/teams/**
See *Get list of teams*. Limited to teams that belong to the organization.

GET /organizations/(**int:** *organization_id*)**/teams/(int:team:id)/**
See *Get team details*. Limited to teams that belong to the organization.

PUT /organizations/(**int:** *organization_id*)**/teams/(int:team:id)/**
See *Update team details*. Limited to teams that belong to the organization.

DELETE /organizations/(**int:** *organization_id*)**/teams/(int:team:id)/**
See *Archive team*. Limited to teams that belong to the organization.

POST /organizations/(**int:** *organization_id*)**/teams/(int:team:id)/permissions/**
See *Add permission to team*. Limited to teams that belong to the organization.

DELETE /organizations/(**int:** *organization_id*)**/teams/(int:team:id)/permissions/**
int: *permission_id***/** See *Remove permission from team*. Limited to teams that belong to the organization.

PUT /organizations/(**int:** *organization_id*)**/teams/(int:team:id)/users/**
int: *user_id***/** See *Add user to team*. Limited to teams that belong to the organization.

DELETE /organizations/(**int:** *organization_id*)**/teams/(int:team:id)/user/**
int: *user_id***/** See *Remove user from team*. Limited to teams that belong to the organization.

1.7 Teams

GET /teams/
Get a list of all the teams you have read access to.

admin users have read access to all teams. users with org:admin for an organization have read access to that organization's teams. users with team:admin permissions for a team have read access to that team. users that are part of the team, or part of the team's organization, have read access to that team.

Example request:

```
GET /teams/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK

[
    {
        "id": "4",
        "title": "admins",
        "permissions": [],
        "users": [],
        "url": "https://example.org/teams/4/",
        "organization": {
            "url": "https://example.org/organizations/7/",
            "id": "7"
        },
        "archived": false
    }
]
```

GET /teams/

Allows filtering of teams to retreive a subset.

Query Parameters

- **type_contains** (*string*) – The type field on one of the resulting team's permissions must contain this string. (optional)
- **object_id** (*string*) – All the object_id fields on one of the resulting team's permissions must equal this string. (optional)

Quert string namespace All the namespace fields on one of the resulting team's permissions must equal this string. (optional)

Example request:

```
GET /teams/?permission_contains=org&object_id=3&namespace=__auth__ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": "4",
        "title": "organization admins",
        "users": [],
        "permissions":
            [
                {
                    "id": "2",
                    "type": "org:admin",
                    "object_id": "3",
                    "namespace": "__auth__"
                }
            ],
        "url": "https://example.org/teams/4/",
        "organization": {
            "url": "https://example.org/organizations/3/",
            "id": "3"
        },
        "archived": false
    },
    {
        "id": "7",
        "title": "other organization admins",
        "users": [],
        "permissions":
            [
                {
                    "id": "3",
                    "type": "org:admin",
                    "object_id": "3",
                    "namespace": "__auth__"
                },
                {
                    "id": "4",
                    "type": "foo:bar",
                    "object_id": "",
                    "namespace": "foo_app"
                }
            ],
        "url": "https://exmple.org/teams/6/",
        "organization": {
            "url": "https://example.org/organizations/3/",
            "id": "3"
        },
        "archived": false
    }
]
```

GET /teams/ (**int:** *team_id*) **/**

Get the details of a team.

admin users have read access to all teams. users with org:admin for an organization have read access to that organization's teams. users with team:admin permissions for a team have read access to that team. users that are part of the team, or part of the team's organization, have read access to that team.

Response JSON Object

- **id** (*str*) – the ID of the team.
- **url** (*str*) – the URL of the team.
- **title** (*str*) – the title of the team.
- **users** (*list*) – The list of users that belong to this team.
- **organization** (*obj*) – An object representing the organization that the team belongs to.
- **permissions** (*list*) – The permission list for the team.
- **archived** (*bool*) – True if team is archived.

Status Codes

- 200 OK – Successfully retrieved team.

Example request:

```
GET /teams/2/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "2",
    "title": "Lord Commanders",
    "permissions": [],
    "users": [],
    "url": "https://example.org/teams/2/",
    "organization": {
        "url": "https://example.org/organizations/7/",
        "id": "7"
    },
    "archived": false
}
```

PUT /teams/(int: *team_id*)/

Update the details of a team.

Admin users can update teams. Users with org:admin permissions for a team's organization can update teams. Users with team:admin can modify the team that they are admin for.

Request JSON Object

- **title** (*str*) – The title of the team.
- **archived** (*bool*) – True if the team is archived.

Response JSON Object

- **id** (*str*) – the id of the updated team.
- **url** (*str*) – The URL of the updated team.
- **title** (*str*) – the title of the team.
- **users** (*list*) – The list of users that belong to this team.
- **organization** (*obj*) – The summary of the organization that the team belongs to.
- **permissions** (*list*) – The permission list for the team.
- **archived** (*bool*) – True if the team is archived.

Status Codes

- 200 OK – successfully updated team.

Example request:

```
PUT /teams/2/ HTTP/1.1
Content-Type: application/json

{
    "title": "Brotherhood without banners",
    "archived": false
}
```

Example reponse:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "2",
    "title": "Brotherhood without banners",
    "permissions": [],
    "users": [],
    "url": "https://example.org/teams/2/",
    "organization": {
        "url": "https://example.org/organizations/7/",
        "id": "7"
    },
    "archived": false
}
```

DELETE /teams/(int: *team_id*)/

Archive a team. The team will by default no longer be shown when *listing teams*, and any permissions associated with the team will no longer take effect when checking whether a user has permission to perform an action.

Archiving can be reversed by setting `archived` to `true` when *updating* a team.

Admin users can archive teams. Users with org:admin permissions for a team's organization can archive teams. Users with team:admin can archive the team that they are admin for.

Status Codes

- 204 No Content – Team successfully archived.

Example request:

```
DELETE /teams/2/ HTTP/1.1
```

Example response:

```
HTTP/1.1 204 No Content
```

POST /teams/ (**int:** *team_id*) **/permissions/**

Add a permission to a team.

Any user that can see the team, can add permissions to that team.

If the permission to be added is of type team:admin, then the user must have team:admin for the team specified by object_id, or org:admin for the team's organization, with the team specified by the object_id.

If the permission to be added is of type org:admin, then the user must have org:admin for the organization specified by object_id.

admin users can add any permissions.

Request JSON Object

- **type** (*str*) – The string representing the permission.
- **object_id** (*str*) – The id of the object that the permission acts on. null if it doesn't act on any object.
- **namespace** (*str*) – The namespace for the permission, to avoid "type" collisions between apps.

Response JSON Object

- **id** (*str*) – the id of the permission.
- **type** (*str*) – the type of the permission.
- **object_id** (*str*) – the object id that the permission acts on.

Status Codes

- 200 OK – successfully added permission to the team.

Example request:

```
POST /teams/2/permissions/ HTTP/1.1
Content-Type: application/json

{
    "type": "org:admin",
    "object_id": "2",
    "namespace": "__auth__"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "17",
    "type": "org:admin",
    "object_id": "2",
    "namespace": "__auth__"
}
```

DELETE /teams/(**int:** *team_id*)**/permissions/**
int: *permission_id***/** Remove a permission from a team.

Any user that can see the team, can remove permissions from that team.

If the permission to be removed is of type team:admin, then the user must have team:admin for the team specified by object_id, or org:admin for the team's organization, with the team specified by the object_id.

If the permission to be removed is of type org:admin, then the user must have org:admin for the organization specified by object_id.

admin users can remove any permission.

Status Codes

- 204 No Content – successfully removed permission from the team.

Example request:

```
DELETE /teams/2/permissions/17/ HTTP/1.1
```

Example response:

```
HTTP/1.1 204 No Content
```

PUT /teams/(**int:** *team_id*)**/users/**
int: *user_id***/** Add an existing user to an existing team.

Status Codes

- 204 No Content – successfully added the user to the team.

Example request:

```
PUT /teams/2/users/1/ HTTP/1.1
Content-Type: application/json
```

Example response:

```
HTTP/1.1 204 No Content
```

DELETE /teams/(**int:** *team_id*)**/users/1/**
Remove a user from a team.

Status Codes

- 204 No Content – successfully removed the user from the team.

Example request:

```
DELETE /teams/2/users/1/ HTTP/1.1
```

Example response:

```
HTTP/1.1 204 OK
```

1.8 Users

GET /users/

Get a list of all users.

Requires any authenticated user.

Example request:

```
GET /users/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": "1",
        "url": "https://example.org/users/1/",
        "first_name": "Jon",
        "last_name": "Snow",
        "email": "jonsnow@castleblack.net",
        "admin": false,
        "active": true,
        "teams": [
            {
                "id": "2",
                "url": "https://example.org/teams/2/"
            }
        ],
        "organizations": [
            {
                "id": "4",
                "url": "https://example.org/organizations/4/"
            }
        ]
    }
]
```

POST /users/

Create a new user.

Only admin users, and users with org:admin permissions can create new users. Only admin users are allowed to create other admin users.

Request JSON Object

- **first_name** (*str*) – The (optional) first name of the user.
- **last_name** (*str*) – The (optional) last name of the user.
- **email** (*str*) – The email address of the user.
- **password** (*str*) – The password for the user.

- **admin** (*bool*) – (optional) True if the user is an admin user. Defaults to False.
- **active** (*bool*) – (optional) False if the user is inactive. Inactive users cannot have tokens created, and permissions are also inactive. They do not show up in any users listing. Defaults to True.

Response JSON Object

- **id** (*str*) – The ID for the user.
- **url** (*str*) – The URL for the user.
- **first_name** (*str*) – The (optional) first name of the user.
- **last_name** (*str*) – The (optional) last name of the user.
- **email** (*str*) – The email address of the user.
- **admin** (*bool*) – True if the user is an admin user.
- **active** (*bool*) – True if the user is active.
- **teams** (*list*) – A list of all the teams a user is a member of.
- **organizations** (*list*) – A list of all the organizations the user is a member of.

Status Codes

- 201 Created – Successfully created user.

Example request:

```
POST /users/ HTTP/1.1
Content-Type: application/json

{
    "first_name": "Jon",
    "last_name": "Snow",
    "email": "jonsnow@castleblack.net",
    "password": "gh0st",
    "admin": false
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": "1",
    "url": "https://example.org/users/1/",
    "first_name": "Jon",
    "last_name": "Snow",
    "email": "jonsnow@castleblack.net",
    "admin": false,
    "active": true,
    "teams": [],
    "organizations": []
}
```

GET /users/(**int:** *user_id*)**/**

Get details on a specific user.

Requires an authenticated user.

Response JSON Object

- **id** (*str*) – The ID for the user.
- **url** (*str*) – The URL for the user.
- **first_name** (*str*) – The (optional) first name of the user.
- **last_name** (*str*) – The (optional) last name of the user.
- **email** (*str*) – The email address of the user.
- **admin** (*bool*) – True if the user is an admin user.
- **active** (*bool*) – True if the user is active.
- **teams** (*list*) – A list of all the teams a user is a member of.
- **organizations** (*list*) – A list of all the organizations the user is a member of.

Example request:

```
GET /users/1/ HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "1",
    "url": "https://example.org/users/1/",
    "first_name": "Jon",
    "last_name": "Snow",
    "email": "jonsnow@castleblack.net",
    "admin": false,
    "active": true,
    "teams": [
        {
            "id": "2",
            "url": "https://example.org/teams/2/"
        }
    ],
    "organizations": [
        {
            "id": "4",
            "url": "https://example.org/organizations/4/"
        }
    ]
}
```

PUT /users/(**int:** *user_id*)**/**

Update the information of an existing user.

Only the user themself, or a user with org:admin, or an admin user can update the user information. Only admin users can change a user to be an admin user.

Request JSON Object

- **first_name** (*str*) – The (optional) first name of the user.
- **last_name** (*str*) – The (optional) last name of the user.
- **email** (*str*) – The email address of the user.

- **password** (*str*) – The password for the user.
- **admin** (*bool*) – (optional) True if the user is an admin user.
- **active** (*bool*) – (optional) True if the user is active.

Response JSON Object

- **id** (*str*) – The ID for the user.
- **url** (*str*) – The URL for the user.
- **first_name** (*str*) – The (optional) first name of the user.
- **last_name** (*str*) – The (optional) last name of the user.
- **email** (*str*) – The email address of the user.
- **admin** (*bool*) – True if the user is an admin user.
- **active** (*bool*) – True if the user is active.
- **teams** (*list*) – A list of all the teams a user is a member of.
- **organizations** (*list*) – A list of all the organizations the user is a member of.

Status Codes

- 200 OK – Successfully updated user.

Example request:

```
PUT /users/1/ HTTP/1.1
Content-Type: application/json

{
    "first_name": "Jon",
    "last_name": "Snow",
    "email": "jonsnow@castleblack.org",
    "password": "gh0st",
    "admin": true
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": "1",
    "url": "https://example.org/users/1/",
    "first_name": "Jon",
    "last_name": "Snow",
    "email": "jonsnow@castleblack.org",
    "admin": true,
    "active": true,
    "teams": [],
    "organizations": []
}
```

DELETE /users/(**int:** *user_id*)**/**

Remove an existing user. Sets the user to inactive instead of deleting the user.

Only the user themself, or a user with org:admin, or an admin user can deactivate a user.

User can be reactivated by setting active to true in updating users.

Status Codes

- 204 No Content – Successfully deleted the user.

Example request:

```
DELETE /users/1/ HTTP/1.1
```

Example response:

```
HTTP/1.1 204 No Content
```

CHAPTER 2

Overview

Seed Auth was designed to allow one or more applications to be able to delegate authentication and user and permission management to a single, internal service. For the case of multiple applications, this avoids the need for users to authenticate separately for each application.

An application would use Seed Auth as follows:

1. Permissions are added to teams of users in Seed Auth, where each permission could be an application-specific permission, or a *permission relevant to Seed Auth*.
2. An authentication token is acquired for a user in Seed Auth.
3. When a client sends a request to an application, the application makes a request to Seed Auth using the authentication token given in the client's request's `Authorization` header, and is given back the permissions granted to the authenticated user. The user's permissions are used by the application to determine whether the user has access to the requested resource.

```
client ------------------> application ----------------> seed-auth-api

        GET /things/23                  Get /user
        Authorization: Token 1234       Authorization: Token 1234

       <-------------------            <----------------

          {id: '23'}                   {
                                         ...
                                         permissions: [{
                                           namespace: 'app:foo',
                                           type: 'thing:read',
                                           object_id: '23'
                                         }]
                                       }
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

HTTP Routing Table