TimeSync Documentation

Release 0.1

OSU Open Source Lab

November 10, 2017

Contents

TimeSync is an OSU Open Source Lab developed Time Tracking system.

CHAPTER 1

API Changelog

Version	Path	Release date
v0.1.0	/v0/	Jun 16, 2016

Jun 16, 2016 - v0.1.0

New Features in v0.1.0

- Everything!

CHAPTER 2

TimeSync Usage Docs

If you are a TimeSync administrator or TimeSync user, these documents contain relevant information that you can use to submit new time entries and manage projects and activities.

Administration

A TimeSync administrator is a user whose responsibilities include creating projects, creating activities, managing users, and reporting. In general, TimeSync is designed to be as straightforward as possible; however, there are still a few things to keep in mind as a TimeSync admin.

Creating new projects and activities

Projects

A project is something that a TimeSync user works on – a website, product, etc. In TimeSync, a project contains a few pieces of information to make users' lives easier:

- a `uri`: a uri linking to the project's homepage or repo.
- a `name`: this is the full name of the project that should be displayed to the user.
- a `default_activity`: if a time does not reference any activities, but *does* reference a project which has a default activity, that default will be used for the time's activities field. The default activity may be null, in which case times *must* provide their own activities.
- a list of `slugs`: these are generally short things that people can refer to the project with. If your project is named Protein Geometry Database, it makes users' lives easier to only need to type in `pgd` when submitting new entries.

Slugs can't refer to multiple projects at the same time – so if you have Fine Software Project and the Fancy Scalable Project, you can't refer to both of them as `fsp`.

Activities

An activity is something that a TimeSync user spends their time doing – software development, documentation, code reviewing, etc. An activity contains only two pieces of information:

- a `name`: this is a human-readable name of the activity, like Documentation.

- a `slug`: this is a human-typeable version of the activity. It should be memorable and quick to type, like `dev` for Development and `docs` for Documentation.

Activities can only have one slug, so choose wisely.

Getting a subset of time entries

TimeSync lets you choose a subset of time entries based on things like the user that did the work, the project worked on, the date range, etc. All parameters can be used in combination, for example to limit a query to only a certain user's time entries on a certain project in a certain week.

Projects

The list of times can be set to the exact projects you want reports on. For instance, if you're only interested in two projects, `gwm` and `pgd`, out of a large number of projects , you can easily choose to only view the time entries associated with one, the other, or both.

Activities

If you're only interested in the time users spent on one particular activity, that's easy as well. Just as with projects, you can elect to view an individual activity, or some subset of all activities.

Users

Just as with projects and activities, the time entries can be selected by the users that worked on them.

Date worked

If you are only interested in a certain date range worked, you can set the start, end, or both to limit the timespan of the time entries selected.

Example

An administrator can opt to view only entries by `alice` and `bob` on the `awesome-proj` and `boring-proj` from `2015-02-27` to `2015-03-14` by setting the parameters on their client to the above values.

Getting a subset of projects

Users

You may also filter projects to only show those on which a given user is a member, using a query parameter similar to that used with times.

CHAPTER 3

API Specification

The current release of the API is v0.1.0.

Click here if you would like to skip the pre-amble and go straight to the endpoints examples

Contents

Note: Variables are indicated with the `:variable_name` syntax ([colon][variable name]). If you see something like `:time` or `:slug` being referenced it is not the literal string ':slug' and ':time' but a variable.

Connection

All requests are made via HTTPS. Available methods include:

- GET to request an object
- POST to create and/or edit a new object
- DELETE to remove an object.

Format

- Responses are returned in standard JSON.
- Multiple results are sent as a list of JSON objects.
  - Order of results is not guaranteed.
- Single results are returned as a single JSON object.

Note: Throughout this API, any form of dates will use a simplified ISO-8601 format, as defined by ECMA International. This format appears as "YYYY-MM-DD", where Y is the year, M is the month (01-12), and DD is the day (01-31).

Versions

The API is versioned with the letter 'v' followed by increasing integers.

For example: `https://timesync.osuosl.org/v0/projects`

Versions are updated each time there is a significant change to the public API (not an implementation). A 'significant change' include updates which force a client to change how it interacts with the API (backward-compatibility breaks, endpoint renaming, etc).

Slugs

Slugs are used to get objects from the back-end, reference objects from within other objects, etc. A valid slug follows a very specific format:

1. May only contain numbers and lowercase letters
2. Sets of lowercase letters and numbers can be separated with a single hyphen
3. Must contain at least one letter

For instance:

Not a Slug	A Slug
–2cool–	e
!ir0ck~	my-username
@username	bossperson

Revisions

When an object is first created, it is assigned a unique tracking ID (UUID). This UUID will refer to all versions of the same object. For example:

```
de305d54-75b4-431b-adb2-eb6b9e546014
```

When an object is updated, a new revision is created. This allows one to easily keep track of changes to an object over time (the object's *audit trail*). An implementation specific backend database key, like an auto-assigned ID (*1*, *9*, *2001*), would only be used to point to a revision of a given object.

A specific revision of an object can be referred to by its unique compound key (UUID, revision) where revision is a number which refers to the position of that version of the object in the audit trail (where 1 is the original version from object creation, 2 is created after the first update, etc.). This revision number is re-used between objects.

Auditing History

There are three variables in all objects that assist in an audit process (viewing revisions of an object through its history).

- `created_at`: the date at which a given object (specified by a UUID) was created.
- `updated_at`: The date at which an object was modified (the day this revision of the object was created).
- `deleted_at`: When the DELETE operation is performed on an object its `deleted_at` field is set to the date it was deleted. Historical (`parents`) copies of an object do not have `deleted_at` set unless the object was deleted for a given historical copy (and later un-deleted).

To view the audit trail of an object pass the `?include_revisions=true` parameter to an endpoint and inspect the `parents` variable (a list of object revisions).

Note: The `include_revisions` parameter does not work on all endpoints.

Check out the *GET Parameters* for more details.

Pagination

All multiple-item GET endpoints support a pair of query parameters to be used in pagination: `?skip` and `?limit`.

GET results will always be returned in chronological order by updated_at value or by created_at where updated_at is NULL.

?limit=n causes the endpoint to return at most *n* objects from its query. If *n = 0*, the full list of items is returned. The default is 25.

?skip=s causes the first *s* items to be dropped from the result list; they are not counted in the limit.

To count by an integer, zero-indexed page count *p*, you can request `?limit=n&skip=(p*n)`; for a one-indexed page count, `skip=((p+1)*n)`.

GET Endpoints

GET /projects

```
[
  {
    "uri": "https://code.osuosl.org/projects/ganeti-webmgr",
    "name": "Ganeti Web Manager",
    "slugs": ["gwm", "ganeti"],
    "uuid": "a034806c-00db-4fe1-8de8-514575f31bfb",
    "created_at": "2014-04-17",
    "deleted_at": null,
    "updated_at": "2014-04-19",
    "revision": 2,
    "users": {
      "user1": {
        "member": true,
        "spectator": false,
        "manager": false
      },
      "user2": {
        "member": true,
        "spectator": true,
        "manager": true
      },
      // ...
    }
  },
  {
    // ...
  }
]
```

GET /projects/:slug

```
{
  "uri": "https://code.osuosl.org/projects/ganeti-webmgr",
  "name": "Ganeti Web Manager",
```

```
  "slugs": ["ganeti", "gwm"],
  "uuid": "a034806c-00db-4fe1-8de8-514575f31bfb",
  "revision": 4,
  "created_at": "2014-07-17",
  "deleted_at": null,
  "updated_at": "2014-07-20",
  "users": {
    "user1": {
      "member": true,
      "spectator": false,
      "manager": false
    },
    "user2": {
      "member": true,
      "spectator": true,
      "manager": true
    },
    // ...
  }
}
```

GET /activities

```
[
  {
    "name": "Documentation",
    "slugs": ["docs", "doc"],
    "uuid": "adf036f5-3d49-4a84-bef9-062b46380bbf",
    "revision": 1,
    "created_at": "2014-04-17",
    "deleted_at": null,
    "updated_at": null
  },
  {
    // ...
  }
]
```

GET /activities/:slug

```
{
  "name": "Documentation",
  "slugs": ["doc", "docs"],
  "uuid": "adf036f5-3d49-4a84-bef9-062b46380bbf",
  "revision": 5,
  "created_at": "2014-04-17",
  "deleted_at": null,
  "updated_at": "2014-04-26"
}
```

GET /times

```
[
  {
    "duration": 12000,
    "user": "example-user",
    "project": ["ganeti", "gwm"],
    "activities": ["docs", "planning"],
    "notes": "Worked on documentation toward settings configuration.",
    "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/40",
    "date_worked": "2014-04-17",
    "revision": 1,
    "created_at": "2014-04-17",
    "updated_at": null,
    "deleted_at": null,
    "uuid": "c3706e79-1c9a-4765-8d7f-89b4544cad56"
  },
  {
    //...
  }
]
```

Caution: Be aware that this endpoint will return different values depending on the permissions of the caller. For more information, see *Authorization and Permissions*, below.

GET /times/:time-entry-uuid

```
{
  "duration": 12000,
  "user": "example-user",
  "project": ["gwm", "ganeti"],
  "activities": ["doc", "research"],
  "notes": "Worked on documentation toward settings configuration.",
  "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/40",
  "date_worked": "2014-04-17",
  "created_at": "2014-04-17",
  "updated_at": "2014-04-21",
  "deleted_at": null,
  "uuid": "c3706e79-1c9a-4765-8d7f-89b4544cad56",
  "revision": 3
}
```

GET Request Query Parameters

TimeSync's response data can be narrowed even further than the /:endpoints return statements by adding parameters:

- user
- project
- activity
- date range

- object revisions
- deleted objects

Reference Table

Parameter	Value(s)	Endpoint(s)
?user=	:username	• /times • /projects
?project=	:project-slug	/times
?activity=	:activity-slug	/times
?start=	:date (ISO format)	/times
?end=	:date (ISO format)	/times
?include_revisions=	:bool	• /activities/ • /activities/:slug • /projects/ • /projects/:slug • /times • /times/:uuid
?include_deleted=	:bool	• /activities • /projects • /times • /times/:uuid • /users • /users/:username

Note: A query parameter may only be used once in a given query. Duplicate instance of the same query parameter will be discarded.

?user=:username

`/times?user=:username` Filters results to a set of time submitted entries by a specified user.

`/projects?user=:username` Filters results to a set of projects on which a specified user is a member.

?project=:projectslug

`/times?project=:projectslug` Filters results to a set of time entries of a specified project slug.

?activity=:activityslug

`/times?activity=:activityslug` Filters results to a set of time entries with a specified activity slug.

?start=:date

`/times?start=:date` Filters results to a set of time entries on or after a specified date.

`/times?end=:date&start=:date` Can be combined with ?end to create a date range.

?end=:date

`/times?end=:date` Filters results to a set of time entries on or before a specified date.

`/times?start=:date&end=:date` Can be combined with ?start to create a date range.

?include_revisions=:bool

The 'parents' field is added to the specified object when this parameter is included and not set to `false`.

This field is a list of all previous revisions of the object in descending order by revision number (i.e. `time.parents[0]` will be the previous revision, and `time.parents[n-1]` will be the first revision).

?include_deleted=:bool

Deleted entries are included in the returned results when this parameter is included and not set to `false`.

These are objects which have the 'deleted_at' parameter set to an ISO date (i.e., a non-null value).

Multiple Parameters Per Request

When multiple parameters are used, they narrow down the result set

```
$ GET /times?user=example-user&activity=dev&token=...
# This will return all time entries which were entered by example-user AND
# which were spent doing development.
```

Date ranges are inclusive on both ends.

Malformed or Exceptional Parameter Usage

If a query parameter is provided with a bad value (e.g. invalid slug, or date not in ISO-8601 format), a Bad Query Value error is returned.

Any query parameter other than those specified in this document will be ignored.

For more information about errors, check the *errors* docs.

If multiple `start`, `end`, `include_deleted`, or `include_revisions` parameters are provided, the first one sent is used. If a query parameter is not provided, it defaults to 'all values'.

Including Revisions of Objects (include_revisions)

GET /projects/:slug?include_revisions=true

```
{
  "uri": "https://code.osuosl.org/projects/ganeti-webmgr",
  "name": "Ganeti Web Manager",
  "slugs": ["ganeti", "gwm"],
  "uuid": "a034806c-00db-4fe1-8de8-514575f31bfb",
  "revision": 4,
  "created_at": "2015-04-16",
  "deleted_at": null,
  "updated_at": "2015-04-23",
  "parents": [
    {
    "uri": "https://code.osuosl.org/projects/old-ganeti-webmgr",
    "name": "Old Ganeti Web Manager",
    "slugs": ["ganeti", "gwm"],
    "uuid": "a034806c-00db-4fe1-8de8-514575f31bfb",
    "revision": 3,
    "created_at": "2015-04-16",
    "deleted_at": null,
    "updated_at": "2015-04-21",
    },
    {
      // ...
    },
    // ...
  ],
  "users": {
    "user1": {
      "member": true,
      "spectator": false,
      "manager": false
    },
    "user2": {
      "member": true,
      "spectator": true,
      "manager": true
    },
    // ...
  }
}
```

Note: Member lists are not stored for old revisions, so when requesting projects with ?include_revisions, the parents will not have "users" fields.

GET /projects?include_revisions=true

```
[
  {
    "uri": "https://code.osuosl.org/projects/ganeti-webmgr",
    "name": "Ganeti Web Manager",
    "slugs": ["ganeti", "gwm"],
    "uuid": "a034806c-00db-4fe1-8de8-514575f31bfb",
    "revision": 4,
    "created_at": "2015-04-16",
    "deleted_at": null,
    "updated_at": "2015-04-23",
    "parents": [
```

```
      {
      "uri": "https://code.osuosl.org/projects/old-ganeti-webmgr",
      "name": "Old Ganeti Web Manager",
      "slugs": ["ganeti", "gwm"],
      "uuid": "a034806c-00db-4fe1-8de8-514575f31bfb",
      "revision": 3,
      "created_at": "2015-04-16",
      "deleted_at": null,
      "updated_at": "2015-04-21",
      },
      {
        // ...
      },
      // ...
    ],
    "users": {
      "user1": {
        "member": true,
        "spectator": false,
        "manager": false
      },
      "user2": {
        "member": true,
        "spectator": true,
        "manager": true
      },
      // ...
    }
  },
  {
    // ...
  },
  // ...
]
```

GET /times/:uuid?include_revisions=true

```
{
  "duration": 2000,
  "user": "example-user",
  "project": ["ganeti", "gwm"],
  "activities": ["doc", "research"],
  "notes": "Worked on documentation toward settings configuration.",
  "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/40",
  "date_worked": "2015-04-12",
  "created_at": "2015-04-12",
  "updated_at": "2015-04-18",
  "uuid": "aa800862-e852-4a40-8882-9b4a79aa3015",
  "deleted_at": null,
  "revision": 2,
  "parents": [
    {
      "duration": 20,
      "user": "example-user",
      "project": ["ganeti", "gwm"],
      "activities": ["doc", "research"],
      "notes": "Worked on documentation toward settings configuration.",
```

```
      "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/40",
      "date_worked": "2015-04-12",
      "created_at": "2015-04-12",
      "updated_at": null,
      "uuid": "aa800862-e852-4a40-8882-9b4a79aa3015",
      "deleted_at": null,
      "revision": 1
    }
  ]
}
```

GET /times?include_revisions=true

```
[
  {
    "duration": 2000,
    "user": "example-user",
    "project": ["ganeti", "gwm"],
    "activities": ["doc", "research"],
    "notes": "Worked on documentation toward settings configuration.",
    "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/40",
    "date_worked": "2015-04-12",
    "created_at": "2015-04-12",
    "updated_at": "2015-04-18",
    "uuid": "aa800862-e852-4a40-8882-9b4a79aa3015",
    "deleted_at": null,
    "revision": 2,
    "parents": [
      {
        "duration": 20,
        "user": "example-user",
        "project": ["ganeti", "gwm"],
        "activities": ["doc", "research"],
        "notes": "Worked on documentation toward settings configuration.",
        "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/40",
        "date_worked": "2015-04-12",
        "created_at": "2015-04-12",
        "updated_at": null,
        "uuid": "aa800862-e852-4a40-8882-9b4a79aa3015",
        "deleted_at": null,
        "revision": 1
      }
    ]
  },
  {
    "duration": 12000,
    "user": "example-user",
    "project": ["timesync", "ts"],
    "activities": ["doc"],
    "notes": "Improved readability of API documentation.",
    "issue_uri": "https://github.com/osuosl/timesync/issues/66",
    "date_worked": "2016-03-23",
    "created_at": "2016-03-23",
    "updated_at": "2016-03-25",
    "uuid": "941a39b1-2507-48a6-8530-a83419661300",
    "deleted_at": null,
    "revision": 1
```

```
  }
]
```

GET /activities/:slug?include_revisions=true


```
{
  "name": "Testing Infra",
  "slug": "test",
  "uuid": "3cf78d25-411c-4d1f-80c8-a09e5e12cae3",
  "created_at": "2014-04-17",
  "deleted_at": null,
  "updated_at": "2014-04-18",
  "revision": 2,
  "parents": [
    {
      "name": "Testing Infrastructure",
      "created_at": "2014-04-17",
      "deleted_at": null,
      "updated_at": null,
      "uuid": "3cf78d25-411c-4d1f-80c8-a09e5e12cae3",
      "revision": 1
    }
  ]
}
```


GET /activities?include_revisions=true


```
[
  {
    "name": "Testing Infra",
    "slug": "test",
    "uuid": "3cf78d25-411c-4d1f-80c8-a09e5e12cae3",
    "created_at": "2014-04-17",
    "deleted_at": null,
    "updated_at": "2014-04-18",
    "revision": 2,
    "parents": [
      {
        "name": "Testing Infrastructure",
        "slug": "test",
        "created_at": "2014-04-17",
        "deleted_at": null,
        "updated_at": null,
        "uuid": "3cf78d25-411c-4d1f-80c8-a09e5e12cae3",
        "revision": 1
      }
    ]
  },
  {
    "name": "Build Infra",
    "slug": "build",
    "uuid": "e81e45ef-e7a7-4da2-88cd-9ede610c5896",
    "created_at": "2014-04-17",
    "deleted_at": null,
    "updated_at": "2014-04-23",
```

```
    "revision": 2,
    "parents": [
      {
        "name": "Testing Infrastructure",
        "slug": "build",
        "created_at": "2014-04-17",
        "deleted_at": null,
        "updated_at": null,
        "uuid": "e81e45ef-e7a7-4da2-88cd-9ede610c5896",
        "revision": 1
      }
    ]
  }
]
```

Retrieving Deleted Objects (include_deleted)

Alongside revision history, you can also view objects that have been soft-deleted. To view an object that has been soft deleted, send a GET request with the `?include_deleted` parameter set to true. Doing so will return all objects matching the query, both current and deleted.

Note: When passing the `include_deleted` parameter to your request, note that you cannot specify a project/activity by their slug. This is because slugs are permanently deleted from activities and projects when they are deleted, in order to allow slug re-use.

GET /projects?include_deleted=true

```
[
  {
    "uri": "https://code.osuosl.org/projects/ganeti-webmgr",
    "name": "Ganeti Web Manager",
    "slugs": ["ganeti", "gwm"],
    "uuid": "a034806c-00db-4fe1-8de8-514575f31bfb",
    "revision": 4,
    "created_at": "2014-04-17",
    "deleted_at": null,
    "updated_at": null
  },
  {
    // ...
  },
  {
    // ...
  },
  {
    "uri": "https://github.com/osuosl/timesync",
    "name": "Timesync",
    "slugs": ["ganeti", "gwm"],
    "uuid": "1f8788bd-0909-4397-be2c-79047f90c575",
    "revision": 1,
    "created_at": "2014-04-17",
    "deleted_at": "2015-10-01",
    "updated_at": null
```

```
  }
]
```

Note: Note that this now includes the Timesync project, which had previously been deleted.

GET /activities?include_deleted=true

```
[
  {
    "name": "Documentation",
    "slug": "doc",
    "uuid": "adf036f5-3d49-4a84-bef9-062b46380bbf",
    "revision": 5,
    "created_at": "2014-04-17",
    "deleted_at": null,
    "updated_at": "2014-05-23"
  },
  {
    // ...
  },
  {
    // ...
  },
  {
    "name": "Meetings"
    "slug": "doc",
    "uuid": "6552d14e-12eb-4f1f-83d5-147f8452614c",
    "revision": 1,
    "created_at": "2014-04-17",
    "deleted_at": "2015-05-01",
    "updated_at": null
  }
]
```

Note: Note that this now includes the Meetings activity, which had previously been deleted.

GET /times?include_deleted=true

```
[
  {
    "duration": 2000,
    "user": "example-user",
    "project": ["ganeti", "gwm"],
    "activities": ["doc", "research"],
    "notes": "Worked on documentation toward settings configuration.",
    "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/40",
    "date_worked": "2015-04-12",
    "created_at": "2015-04-12",
    "updated_at": "2015-04-18",
    "uuid": "aa800862-e852-4a40-8882-9b4a79aa3015",
    "deleted_at": null,
    "revision": 2
  },
```

```
  {
    "duration": 3000,
    "user": "example-user",
    "project": ["timesync", "ts"],
    "activities": ["doc"],
    "notes": "Worked on documentation toward include_deleted parameter.",
    "issue_uri": "https://github.com/osuosl/timesync/issues/52",
    "date_worked": "2015-08-18",
    "created_at": "2015-08-18",
    "updated_at": "2015-09-10",
    "deleted_at": "2015-10-12",
    "uuid": "e283a2cd-39c6-4133-95ec-5bc10dd9a9ef",
    "revision": 2
  }
]
```


Note: Note that this now includes the second time, which had previously been deleted.

GET /times/:uuid?include_deleted=true


```
{
  "duration": 30,
  "user": "example-user",
  "project": ["timesync", "ts"],
  "activities": ["doc"],
  "notes": "Worked on documentation toward include_deleted parameter.",
  "issue_uri": "https://github.com/osuosl/timesync/issues/52",
  "date_worked": "2015-08-18",
  "created_at": "2015-08-18",
  "updated_at": "2015-09-10",
  "deleted_at": "2015-10-12",
  "uuid": "e283a2cd-39c6-4133-95ec-5bc10dd9a9ef",
  "revision": 2
}
```


Note: As above, this time is deleted (note the deleted_at field), but instead of a 404, it returns the object.

POST Endpoints

To add a new object, POST to */:object-name/* with a JSON body. The response body will contain the object in a similar manner as the GET endpoints above.

POST /projects/

Request body:


```
{
  "uri": "https://code.osuosl.org/projects/timesync",
  "name": "TimeSync API",
  "slugs": ["timesync", "time"],
```

```
  "users": {
    "user1": {
      "member": true,
      "spectator": false,
      "manager": false
    },
    "user2": {
      "member": true,
      "spectator": true,
      "manager": true
    },
    // ...
  }
}
```

Response body:

```
{
  "uri": "https://code.osuosl.org/projects/timesync",
  "name": "TimeSync API",
  "slugs": ["timesync", "time"],
  "uuid": "b35f9531-517f-47bd-aab4-14298bb19555",
  "created_at": "2014-04-17",
  "updated_at": null,
  "deleted_at": null,
  "revision": 1,
  "users": {
    "user1": {
      "member": true,
      "spectator": false,
      "manager": false
    },
    "user2": {
      "member": true,
      "spectator": true,
      "manager": true
    },
    // ...
  }
}
```

Note: Because of sitewide manager and admin permissions, no users are automatically added to a project, unless a `users` field is passed to add them.

POST /activities/

Request body:

```
{
 "name": "Quality Assurance/Testing",
 "slug": "qa"
}
```

Response body:

```
{
  "name": "Quality Assurance/Testing",
  "slug": "qa",
  "uuid": "cfa07a4f-d446-4078-8d73-2f77560c35c0",
  "created_at": "2014-04-17",
  "updated_at": null,
  "deleted_at": null,
  "revision": 1
}
```


POST /times/

Request body:


```
{
  "duration": 12,
  "user": "example-2",
  "project": "ganeti_web_manager",
  "activities": ["docs"],
  "notes": "Worked on documentation toward settings configuration.",
  "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/56",
  "date_worked": "2014-04-17"
}
```


Response body:


```
{
  "duration": 12,
  "user": "example-2",
  "project": "ganeti_web_manager",
  "activities": ["docs"],
  "notes": "Worked on documentation toward settings configuration.",
  "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/56",
  "date_worked": "2014-04-17",
  "created_at": "2014-04-17",
  "updated_at": null,
  "deleted_at": null,
  "uuid": "838853e3-3635-4076-a26f-7efe4e60981f",
  "revision": 1
}
```


POST /users/

User documentation can be found in the *User Documentation*

Likewise, if you'd like to edit an existing object, POST to `/projects/:slug`, `/activities/:slug`, or `/times/:uuid` with a JSON body. The object only needs to contain the part that is being updated. The response body will contain the saved object, as shown above.

Note: If a deleted time or user is updated using these endpoints, the new revision is no longer deleted; the old revision still has its deleted_at set, but the new revision does not, allowing it to appear in GET responses, etc. Note that this does not apply to activities or projects; because their slugs are deleted, they cannot be referenced by these endpoints, and thus must be recreated.

POST /projects/:slug

Request body:

```
{
  "uri": "https://code.osuosl.org/projects/timesync",
  "name": "TimeSync API",
  "slugs": ["timesync", "ts"]
}
```

Response body:

```
{
  "uri": "https://code.osuosl.org/projects/timesync",
  "name": "TimeSync API",
  "slugs": ["timesync", "ts"],
  "created_at": "2014-04-16",
  "updated_at": "2014-04-18",
  "deleted_at": null,
  "uuid": "309eae69-21dc-4538-9fdc-e6892a9c4dd4",
  "revision": 2,
  "users": {
    "user1": {
      "member": true,
      "spectator": false,
      "manager": false
    },
    "user2": {
      "member": true,
      "spectator": true,
      "manager": true
    },
    // ...
  }
}
```

Note: If a slugs field is passed to `/projects/:slug`, it is assumed to overwrite the existing slugs for the object. Any slugs which already exist on the object but are not in the request are dropped, and the slugs field on the request becomes canonical.

If any of the slugs provided belong to any other projects, a Slug Already Exists error is returned listing all slugs already associated with other projects, and no changes are made.

POST /activities/:slug

Request body:

```
{
  "slug": "testing"
}
```

Response body:

```
{
  "name": "Testing Infra",
  "slug": "testing",
  "uuid": "3cf78d25-411c-4d1f-80c8-a09e5e12cae3",
```

```
  "created_at": "2014-04-16",
  "updated_at": "2014-04-17",
  "deleted_at": null,
  "revision": 2
}
```

POST /times/:uuid

Original object:

```
{
  "duration": 12000,
  "user": "example-2",
  "activities": ["qa"],
  "project": ["gwm", "ganeti"],
  "notes": "",
  "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/56",
  "date_worked": "2014-06-10",
  "created_at": "2014-06-12",
  "updated_at": null,
  "deleted_at": null,
  "uuid": "aa800862-e852-4a40-8882-9b4a79aa3015",
  "revision": 1
}
```

Request body:

```
{
  "duration": 18000,
  "notes": "Initial duration was inaccurate. Date worked also updated.",
  "date_worked": "2014-06-07"
}
```

The response body will be:

```
{
  "duration": 18000,
  "user": "example-2",
  "activities": ["qa"],
  "project": ["gwm", "ganeti"],
  "notes": "Initial duration was inaccurate. Date worked also updated.",
  "issue_uri": "https://github.com/osuosl/ganeti_webmgr/issues/56",
  "date_worked": "2014-06-07",
  "created_at": "2014-06-12",
  "updated_at": "2014-07-02",
  "deleted_at": null,
  "uuid": "aa800862-e852-4a40-8882-9b4a79aa3015",
  "revision": 2
}
```

POST /users/:username

User documentation can be found in the *User Documentation*

Note: If a value of `""` (an empty string) or `[]` (an empty array) are passed as values for a string or array optional field (see the *model docs*), the value will be set to the empty string/array. If a value of undefined is provided (i.e. the value is not provided), the current value of the object will be used.

Note: In the case of a malformed object sent in the request, or a foreign key (such as project on a time) that does not point to a valid object, a Malformed Object, Object Not Found or error (respectively) will be returned, validation will return immediately, and the object will not be saved.

The following content is checked by the API for validity:

- Time/Date must be a valid ISO 8601 Date/Time.
- URI must be a valid URI.
- Activities must exist in the database.
- The Project must exist in the database.
- Project and activity slugs must not already belong to another project/activity.

DELETE Endpoints

The single object endpoints (e.g. `/times/:uuid`, `/projects/:slug`) support DELETE requests; these remove an object from the records.

If the object is successfully deleted, an empty response body is sent, with a 200 OK status. If the deletion fails for any reason, an error object is returned.

These objects will always be soft-deleted; that is, the object will still exist, but will not be returned for a normal GET request. Requests for lists of objects (e.g. `GET /projects`) will exclude the object from the results, and requests for single objects (e.g. `GET /times/:uuid`) will return a 404. The parameter `?include_deleted` circumvents this requirement and allows deleted objects to be included in the returned set of objects.

An object's deleted status is indicated by setting its `deleted_at` field to the date of deletion; if the value is null, the object is not deleted. Only the most recent revision is set. In addition, activities and projects have their `slugs` removed in order to allow these slugs to be reused by future objects.

This means that it is impossible to request or update a project or activity after it is deleted, even when using the `?include_deleted` parameter. Instead, a new project or activity must be made; because the original slugs were deleted, the new object can share any or all of the original project's user-defined metadata.

When deleting a project or activity it must not be referenced by a current time entry (i.e. one which is neither deleted nor updated). If it is referenced by a current time, then a Request Failure error is returned.

Authorization and Permissions

There are two classes of permissions in TimeSync: project roles and site roles. Each user may be any combination of the following:

- site_spectator

- site_manager
- site_admin.

In addition, each user may be any combination of the following:

- project_member
- project_spectator
- project_manager

for an individual project.

These project permissions exist independently. A user may only be a site_spectator, or may be a project_member and project_manager but not project_spectator; sitewide permissions override those of projects. Permissions are defined here:

Permission	Allowed to
Project member	Create time entries
Project spectator	View time entries for that project (see `GET Endpoints`, below)
Project manager	Update projects and members
Sitewide spectator	View all time entries
Sitewide manager	Create projects and activities, create users
Sitewide admin	Any action, including promote users to managers and admins

A user may be a member, spectator, and/or manager of multiple projects, and a project may have multiple members, spectators, and managers.

If a user attempts to access an endpoint which they are not authorized for, the server will return an Authorization Failure.

Note: It is recommended that the site have one admin user which belongs to no one in particular, similarly to the Linux `root` user, which may add other users/admins.

GET Endpoints

GET /activities, GET /activities/:slug, GET /projects, and GET /projects/:slug are accessible to anyone who has successfully authenticated.

GET /times will return:

- The authenticated user's times
- All times in projects for which a user is a spectator or manager
- All times if the user is a sitewide spectator or manager

GET /times/:uuid follows the same rules (i.e. it will return the time if that time would be in the results of /times, or Authentication Failure otherwise).

User documentation can be found in the *User Documentation*

POST Endpoints

POST /activities and POST /activities/:slug can be accessed by sitewide managers.

POST /projects is accessible to sitewide managers.

POST /projects/:slug is accessible to the project's manager(s) and sitewide managers. In addition, note that both project managers and sitewide managers may promote another user to manager and demote other managers. As well, note that a project manager may in this way demote themselves or remove themselves from the project.

POST /times is accessible to members of the project for which they intend to create a time.

POST /times/:slug is accessible to the user who created the time originally.

User documentation can be found in the *User Documentation*

DELETE Endpoints

DELETE /activities/:slug is accessible to sitewide managers.

DELETE /projects/:slug is accessible to the project's manager(s) and sitewide managers.

DELETE /times/:uuid is accessible to the user who created the time and sitewide managers.

User documentation can be found in the *User Documentation*

CHAPTER 4

Authentication and Authorization

TimeSync allows for various forms of authentication determined by the implementation.

Contents

The general scheme of an authentication module looks like this:

```
{
  "auth": {
    "type": // type
    // ...
  },
  "object": {
    "name": "Ganeti Web Manager",
    // ...
  }
}
```

There are two top-level blocks to an authenticated POST request:

- `auth`, containing any authentication information necessary
- `object`, containing the data of the POST request (see the *API docs*)

An auth block typically looks like the above example, with a `type` field specifying what kind of authentication the client is connecting with, and other fields as necessary for that kind of connection.

These other fields might be things like `password`, `token`, `key`, etc.

Token-based authentication

Token-based is the primary and default authentication method for TimeSync. TimeSync provides an endpoint at `/login`, which should be sent a POST request, with the body fitting one of the other available authentication schemes (e.g. Password, LDAP). The endpoint returns a JWT token string as its response body. This response body is used to access the other endpoints:

POST endpoints:

```
{
  "auth": {
    "type": "token",
    "token": // ...
  },
  // ...
}
```

GET and DELETE endpoints: Use the query key `token`, as in `GET /times?token=:token` or `DELETE /activities/example?token=:token`

For example, the workflow may occur as follows:

`POST /login`

```
{
  "auth": {
    "type": "password",
    "username": "example-user",
    "password": "pass"
  }
}
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0aW1lc3luYyIsInN1YiI6ImV4Y
  W1wbGUtdXNlciIsImV4cCI6MTQ0NjQ5NzQyNzEzMywiaWF0IjoxNDQ2NDk1NTc5OTY3fQ.k8ij2cXBRs5tUe
  _cq2RDePCYMpFjVkKqnpU11Q1XEnk"
}
```

`POST /projects`

```
{
  "auth": {
    "type": "token",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0aW1lc3luYyIsInN1YiI6ImV
        4YW1wbGUtdXNlciIsImV4cCI6MTQ0NjQ5NzQyNzEzMywiaWF0IjoxNDQ2NDk1NTc5OTY3fQ.k8ij2c
        XBRs5tUe_cq2RDePCYMpFjVkKqnpU11Q1XEnk"
  },
  "object": {
    "name": "Example Project",
    "owner": "example-user",
    "uri": "http://example.com/",
    "slugs": ["example", "example-project"]
  }
}
```

Response:

```
{
  "name": "Example Project",
  "slugs": ["example", "example-project"],
  "uri": "http://example.com/",
  "owner": "example-user",
  "uuid": "9ac95604-28dd-44e0-9ba5-ff9c5e2b2212",
  "revision": 1,
  "created_at": "2015-11-02",
  "updated_at": null,
```

```
  "deleted_at": null
}
```

To later get this object back:

```
GET /projects/example?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0aW1lc3luYyI
sInN1YiI6ImV4YW1wbGUtdXNlciIsImV4cCI6MTQ0NjQ5NzQyNzEzMywiaWF0IjoxNDQ2NDk1NTc5OTY3fQ.k8ij2c
XBRs5tUe_cq2RDePCYMpFjVkKqnpU11Q1XEnk
```

Response:

```
{
  "name": "Example Project",
  "slugs": ["example", "example-project"],
  "uri": "http://example.com/",
  "owner": "example-user",
  "uuid": "9ac95604-28dd-44e0-9ba5-ff9c5e2b2212",
  "revision": 1,
  "created_at": "2015-11-02",
  "updated_at": null,
  "deleted_at": null
}
```

API tokens have a life of 30 minutes, and must be used on the same TimeSync instance as they are created.

Password authentication

When used with password-based authentication, TimeSync requires a username field and a password field:

```
{
  "auth": {
    "type": "password",
    "username": "tschuy",
    "password": "password"
  }
}
```

This username/password combination is compared to values stored in the local database for authentication.

LDAP Authentication

This form is nearly identical to password-based authentication, using a username and password:

```
{
  "auth": {
    "type": "ldap",
    "username": "tschuy",
    "password": "password"
  }
}
```

Instead of comparing the username/password combination to values in a local database, however, the combination is provided to a configured LDAP provider for authentication.

CHAPTER 5

Errors

Error types from the TimeSync API.

Contents

Errors consist of:

1. a descriptive HTTP status code
2. a standardized error name
3. informational text
4. a `values` array containing variables relevant to the error, if any

The existence of an `error` key indicates an error.

In the following docs string literals are indicated by double quotes (as in the JSON standard), but use the ECMAScript 2015 string interpolation specification to represent variables like slugs and object types.

TimeSync uses the following error codes:

1. Object Not Found

To be returned if the server receives a valid key (e.g. Time UUID or Activity Slug) which does not match an object in the database.

```
{
  "status": 404,
  "error": "Object not found",
  "text": "Nonexistent ${object}"
}
```

---

2. Server Error

A generic catch-all for when there is a server error outside of the client's control. This may be the result of an uncaught exception, a database error, or any other condition which renders the server unable to process a valid request. Note that in a production environment, `text` may be empty to avoid disclosing sensitive information.

```
{
  "status": 500,
  "error": "Server error",
  "text": "${server_error}" // (e.g. exception text or sql error)
}
```

---

3. Invalid Foreign Key

If a client attempts to make a POST request to create or update an object, but the new object sent by the client contains a foreign key parameter which does not point to a valid object in the database. (E.g. the client sends a new time which does not have a valid project.)

```
{
  "status": 409,
  "error": "Invalid foreign key",
  "text": "The ${object_type} does not contain a valid ${foreign_key} reference"
}
```

---

4. Bad Object

A client attempts to make a POST request to create or update an object, but the new object sent by the client contains a non-existent key, lacks a necessary key, or contains an invalid value for a key (e.g. a time with a string in the duration field, or a project without a name.)

```
unknown_field_error = {
  "status": 400,
  "error": "Bad object",
  "text": "${object_type} does not have a ${field_name} field"
}
missing_field_error = {
  "status": 400,
  "error": "Bad object",
  "text": "The ${object_type} is missing a ${field_name}"
}
```

```
invalid_field_error = {
  "status": 400,
  "error": "Bad object",
  "text": "Field ${field_name} of ${object_type} should be ${expected_type} but was sent as ${receive
}
```

---

5. Invalid Identifier

This error would be returned when an identifier field (e.g. time UUID or activity slug) is malformed or otherwise not valid for use; an identifier is considered invalid if it does not match the expected format (e.g. a slug with special characters or a non-numeric ID field).

This is to be distinguished from Object Not Found: Object Not Found occurs when a perfectly valid, well-formed identifier is supplied, but no object matching the identifier could be found.

Object Not Found is therefore considered to be a temporary error (making an identical request later may return an object instead of an error), while Invalid Identifier is considered a permanent error (the request will always return this error, pending changes to the specification).

```
{
  "status": 400,
  "error": "Invalid identifier",
  "text": "Expected ${slug/uuid} but received ${received_identifier}",
  "values": [${received_identifier}]
}
```

With multiple invalid identifiers, the error is formatted like so:

```
{
  "status": 400,
  "error": "Invalid identifier",
  "text": "Expected ${slug/uuid} but received: ${bad}, ${bad}, ${bad}",
  "values": [${bad}, ${bad}, ...]
}
```

---

6. Invalid Username

This error is returned when creating a new user and a username with invalid characters is provided.

```
{
  "status": 401,
  "error": "Invalid username",
  "text": "Invalid username ${username} is not a valid username"
}
```

---

7. Authentication Failure

This error is returned when authentication fails for any reason. The text of the error may change based on what kind of authentication backend the TimeSync server is running.

```
{
  "status": 401,
  "error": "Authentication failure",
  "text": "${authentication_error}" // (e.g. "Invalid username or password")
}
```

---

8. Slug Already Exists

This error is returned when a new object is being created but the slugs provided contain a slug that already exists.

```
{
  "status": 409,
  "error": "Slug already exists",
  "text": "Slug ${slug} already exists on another object",
  "values": [${slug}]
}
```

If multiple slugs are duplicated:

```
{
  "status": 409,
  "error": "Slugs already exist",
  "text": "Slugs ${slug}, ${slug} already exist on another object",
  "values": [${slug}, ${slug}, ...]
}
```

---

9. Authorization Failure

This error is returned when the user is successfully authenticated, but lacks the authorization to complete the task they are attempting to do. This is used when a non-administrator user attempts to create time or project entries for another user.

```
{
  "status": 401,
  "error": "Authorization failure",
  "text": "${user} is not authorized to ${action}"
}
```

---

10. Request Failure

This error is returned when a request is sent to an object and is rejected. This is used mainly in the instances when a user tries to delete something they are not supposed to. For example, a user may attempt to delete a project that has associated times.

Allowed methods must be returned along with the error object, which will be listed in the HTTP Allow header.

```
{
  "status": 405,
  "error": "Method not allowed",
  "text": "The method specified is not allowed for the ${objectType} identified"
}
```

---

11. Bad Query Value

This error is returned when a GET request is made with query parameters, but the value of a parameter is invalid in some way. This includes dates which are not sent in ISO 8601 format, and slugs and IDs which are not considered valid. This error is not returned, however, if a query parameter is missing (default values are assumed), or if an extra query parameter is used (nonexistent keys are ignored).

```
{
    "status": 400,
    "error": "Bad query value",
    "text": "Parameter ${key} contained invalid value ${value}"
}
```

---

12. Username Already Exists

This error is returned when creating a user and the username provided for the new user is already used by another user. Compare 8. Slug Already Exists.

```
{
  "status": 409,
  "error": "Username already exists",
  "text": "Username ${username} already exists",
  "values": [${username}]
}
```

CHAPTER 6

API Object Model

Below is a listing of the structure of objects in the TimeSync API.

Each object is listed with the fields it consists of. Type represents the type of data passed; it may represent data types (such as "string" or "array") or formats (such as "ISO date"). POST Required represents whether "empty" values are accepted: `true` means that values must be complete, while `false` means that values such as an empty string, empty array, or null are valid.

Note that this documentation does not reflect the way the values are stored in the database, only the way they are passed into and out of the API. For documentation on storing the objects, see the implementation.

Common

These fields are shared between all objects except users. Do not pass any of them in a POST request; they are provided automatically.

Name	Type	POST Required	Description	Notes
uuid	UUID	false	A tracking ID between object revisions	
revision	positive number	false	A version number of an object	
created_at	ISO date	false	The date the object (UUID) was created	
updated_at	ISO date or null	false	The date this revision was created	
deleted_at	ISO date or null	false	The date this revision was deleted	Always null unless ?include_deleted is used
parents	array	false	The previous revisions of this object	Only provided if ?revisions=true is used

Times

Name	Type	POST Required	Description	Notes
duration	positive number	true	Length of time user worked, in seconds	
user	username	true	Username of the user who worked	Must be an existing username
project	mixed	true	The project the user worked on	Array of slugs in GET requests, single slug in POST
activities	array of slugs	sometimes	The activities the user performed	Required on creation if the project has no default activity; not nullable on update
date_worked	ISO date	true	The date this time began on	May be in the future or the past
issue_uri	URI	false	The issue the user worked on	May link to any issue tracker; must be a valid URI
notes	string	false	Other notes the user wishes to provide	

Projects

Name	Type	POST Required	Description	Notes
name	string	true	The name of the project	
slugs	array of slugs	true	Slugs to identify the project	Must be unique to the project
uri	URI	false	The URI of the project	
default_activity	slug	false	The activity times will default to	

Activities

Name	Type	POST Required	Description	Notes
name	string	true	The human-readable name of the activity	
slug	string	true	A unique slug to identify the activity	

Users

Caution: User objects do not include the 'Common' fields listed at the top of this document.

Field	Type	Description	Notes
display_name	string	User's public display name.	While username cannot change, display_name can.
username	string	Permanent username.	This username will remain in use even if the user is deleted.
password	string	Password for user login.	Stored as hash; not returned in GET requests.
email	string	Email address of user.	
roles	array	List of the user's org roles.	Array of strings; contains only values from the organization's role list.
active	bool	Whether the user can login.	Used by admins to invalidate users.
site_spectator	bool	Site-wide spectator flag.	Can be set by a manager or admin.
site_manager	bool	Site-wide manager flag.	Can only be set by an admin.
site_admin	bool	Site-wide admin flag.	Can only be set by another admin.
created_at	date	Date account was created at.	Automatically set, unchangeable.
updated_at	date	Date account was last updated.	Automatically set when updated.
deleted_at	date	Date account was soft-deleted.	Automatically set by server upon DELETE.
meta	string	Miscellaneous user meta-data.	

Note: Users are updated "in-place" (i.e. without an audit trail or revision system), there is no `uuid` or `revision` field.

Organization Roles

Caution: Roles do not include any of the 'Common' fields listed at the top of this document. They are solely a name and a slug.

Field	Type	Description	Notes
name	string	A human-readable name for the role, such as "Software Developer"	Must be unique
slug	string	A machine-readable slug (see API docs) which identifies the role	Must be unique

CHAPTER 7

User Management

Users are managed primarily through the API, by admin users.

Contents

Organization Roles

The users model provides the concept of organization roles. These are string values which can be associated with users in order to help define what purpose the user serves within the organization (e.g. "Intern", "Project Manager", "Executive"). A user may belong to one or more organization roles, which are defined by the `/users/org-roles` endpoints.

An organization role consists of a machine-readable slug, which shall be passed to and from the API as references to roles, and a human-readable name, which shall be returned from the `GET /users/org-roles/:slug` endpoint.

Slugs follow the same rules as in the rest of the *API*.

The list of users may be filtered by roles, in order to get a list only of members of that role.

Roles should not be confused with the authorization levels (e.g. site_manager), which configure what data a user is allowed to access or modify within the TimeSync system. Roles exist largely as metadata for use within the organization itself.

Users Model

For reference of the user fields, see the *model* docs.

Here is an example user object:

```
{
  "display_name": "User One",
  "username": "user1",
  "email": "user1@example.org",
  "org-roles": ["intern"],
  "site_spectator": true,
  "site_manager": false,
  "site_admin": false,
  "created_at": "2016-02-15",
  "updated_at": "2016-02-15",
  "deleted_at": null,
  "active": true,
  "meta": "extra metadata about user"
}
```

Note: Usernames may consist of uppercase and lowercase letters, decimal digits, hyphen, period, underscore, and tilde characters.

Note: Usernames are considered case-insensitive: they will be displayed using the case they were created with, but both the `/users/:username` endpoints and the `/login` endpoint will match on any capitalization, i.e. if I have a user named 'User1', `GET /users/User1`, `GET /users/user1`, and `GET /users/uSeR1` will all return that user.

Admin Users

A site-wide admin user is defined by the boolean `site_admin` field. Admins are able to access any endpoint, and manage all users (including being the only users which can promote others to admin status).

/users Endpoint

User management involves a new endpoint, at `/users`. In most respects it is similar to the other endpoints, with certain key differences.

GET /users

Returns a list of all user objects.

```
[
  {
    "display_name": "User One",
    "username": "user1",
    "email": "user1@example.org",
    "org-roles": ["intern"],
    "site_spectator": true,
    "site_manager": false,
    "site_admin": false,
    "created_at": "2016-02-15",
    "updated_at": "2016-02-15",
    "deleted_at": null,
    "active": true,
    "meta": "extra metadata about user"
  },
  {
    // ...
  },
  // ...
]
```

Note: The /users endpoint also includes the ability to ?include_deleted objects.

Note: Usernames are permanent, even upon deletion.

GET /users?role=:role

Similar to the base `GET /users` endpoint, this endpoint filters only for users matching the role slug provided. If the query is repeated, it functions as an "OR" statement.

That is, `GET /users?role=intern` returns all interns in the system.

`GET /users?role=intern&role=mentor` returns everyone who is either an intern OR a mentor.

```
[
  {
    "display_name": "User One",
    "username": "user1",
    "email": "user1@example.org",
    "org-roles": ["intern"],
    "site_spectator": true,
    "site_manager": false,
    "site_admin": false,
    "created_at": "2016-02-15",
    "updated_at": "2016-02-15",
    "deleted_at": null,
    "active": true,
    "meta": "extra metadata about user"
  },
  {
    // ...
  },
```

```
  // ...
]
```

GET /users/:username

Returns a single user object.

```
{
  "display_name": "User One",
  "username": "user1",
  "email": "user1@example.org",
  "org-roles": ["intern"],
  "site_spectator": true,
  "site_manager": false,
  "site_admin": false,
  "created_at": "2016-02-15",
  "updated_at": "2016-02-15",
  "deleted_at": null,
  "active": true,
  "meta": "extra metadata about user"
}
```

GET /users?include_deleted=true

```
[
  {
    "display_name": "User One",
    "username": user1,
    "email": "user1@example.org",
    "org-roles": ["intern"],
    "site_spectator": true,
    "site_manager": false,
    "site_admin": false,
    "created_at": "2016-02-15",
    "updated_at": "2016-02-15",
    "deleted_at": "2017-06-21",
    "active": false,
    "meta": "extra metadata about user"
  },
  {
    // ...
  },
  // ...
]
```

GET /users/:username?include_deleted=true

```
{
  "display_name": "User One",
  "username": "user1",
  "email": "user1@example.org",
  "org-roles": ["intern"],
  "site_spectator": true,
```

```
  "site_manager": false,
  "site_admin": false,
  "created_at": "2016-02-15",
  "updated_at": "2016-02-15",
  "deleted_at": "2017-06-21",
  "active": false,
  "meta": "extra metadata about user"
}
```

POST /users

Create a new user.

Request:

```
{
  "displayname": "X. Ample User",
  "username": "example",
  "password": "password",
  "email": "example@example.com"
  "org-roles": ["intern"],
  "site_spectator": true,
  "site_manager": false,
  "site_admin": false,
  "active": true,
  "meta": "Some metadata about the user"
}
```

Response:

```
{
  "displayname": "X. Ample User",
  "username": "example",
  "email": "example@example.com"
  "org-roles": ["intern"],
  "site_spectator": true,
  "site_manager": false,
  "site_admin": false,
  "active": true,
  "created_at": "2016-02-15",
  "updated_at": "2016-02-15",
  "deleted_at": null,
  "active": true,
  "meta": "Some metadata about the user"
}
```

Caution: It is the client's responsibility to hash the password before sending it to this endpoint, unlike the /login endpoint (see *auth*). The password should be hashed with bcrypt, using 10 rounds. The bcrypt prefix should be "2a" (different implementations may use different prefixes, but the API requires consistency for authentication).

Note: This endpoint may only be accessed by admins and sitewide managers.

Note: It is recommended that admins provide the user with a temporary password and have the user change the password when they log in.

Note: If an organizational role which does not exist in the system is provided to this endpoint, a *Invalid Foreign Key error* will be returned.

POST /users/:username

Original object:

```
{
  "display_name": "User One",
  "username": "user1",
  "email": "user1@example.org",
  "org-roles": ["intern"],
  "site_spectator": true,
  "site_manager": false,
  "site_admin": false,
  "active": true,
  "created_at": "2016-02-15",
  "updated_at": "2016-02-15",
  "deleted_at": null,
  "active": false,
  "meta": "extra metadata about user"
}
```

Request body (made by a `site_admin` user):

```
{
  "display_name": "New Displayname",
  "password": "Battery Staple",
  "email": "user1+new@example.org",
  "org-roles": ["developer"],
  "meta": "Different metadata about user1",
  "site_spectator": true,
  "site_manager": true,
  "site_admin": false,
}
```

The response will be:

```
{
  "display_name": "New Displayname",
  "username": "user1",
  "email": "user1+new@example.org",
  "org-roles": ["developer"],
  "site_spectator": true,
  "site_manager": true,
  "site_admin": false,
  "created_at": "2016-02-15",
  "updated_at": "2016-02-18",
  "deleted_at": null,
  "meta": "Different metadata about user1"
}
```

Caution: It is the client's responsibility to hash the password before sending it to this endpoint, unlike the /login endpoint (see *auth*). The password should be hashed with bcrypt, using 10 rounds. The bcrypt prefix should be "2a" (different implementations may use different prefixes, but the API requires consistency for authentication).

Note: Site-wide admins can modify other users' site_spectator, site_manager, and site_admin fields.

Site-wide managers can modify other users' site_spectator fields.

Note: If an organizational role which does not exist in the system is provided to this endpoint, a *Invalid Foreign Key error* will be returned.

Note: The `org-roles` field, when passed to this endpoint, overwrites the existing value, including if `[]` (an empty array) or `null` (which is treated like an empty array) is passed as the value. To maintain the current role list, the existing list must be passed as-is, or else the field must be omitted entirely from the request.

This endpoint may be accessed by admins, sitewide managers, or the user who is being updated. However, users may not set their own permissions unless they are an admin, and managers may *only* set the `site_spectator` field; thus the `site_admin` and `site_manager` fields may only be set by an admin.

DELETE /users/:username

Soft-delete a user. Returns a 200 OK with empty response body on success, or an *error* on failure. Only accessible to admins.

For more information on deletion, see the DELETE section of the *API* docs.

Roles Endpoints

The following endpoints retrieve or modify the list of roles to which users may belong.

GET /users/org-roles

This endpoint returns the list of roles in the system to which users may belong.

```
[
  {
    "name": "Summer Intern",
    "slug": "intern"
  },
  {
    "name": "Software Developer",
    "slug": "developer"
  },
  ...
]
```

GET /usrs/org-roles/:slug

This endpoint allows one to request the name of an organization role by its slug.

```
{
  "name": "Summer Intern",
  "slug": "intern"
}
```

POST /users/org-roles

This endpoint creates a new organization role to which users may later be added.

Both role names and slugs must be unique; if they are not, an error will be returned.

Request body:

```
{
  "name": "C-Level Executive",
  "slug": "executive"
}
```

Response will be identical to the request in case of success.

POST /users/org-roles/:slug

This endpoint edits the name and/or slug of an existing role.

As with the creation endpoint, both the new role name and slug must not already exist.

All users who currently have this role will return the new slug after this request.

Original object:

```
{
  "name": "Summer Intern",
  "slug": "intern"
}
```

Request body:

```
{
  "slug": "summer"
}
```

Response body:

```
{
  "name": "Summer Intern",
  "slug": "summer"
}
```

DELETE /users/org-roles/:slug

This endpoint deletes a role from the organization, preventing any new users from being given the role.

Only roles to which no users belong may be deleted. If a role is passed to which users still belong, a *Request Failure error* will be returned. Edit users with this role to another role to delete the error.

Unlike other objects in TimeSync, roles are permanently deleted by this request. This means that there is no way to retrieve them after this (and that there is no `include_deleted` query for roles). This also means that the name

and slug previously taken by this role are freed, and a new role with the same name and/or slug may be created in the future.

Authorization Management

Authorization management is handled through the `projects` and `users` endpoints.

The user object contains the `site_spectator`, `site_manager`, and `site_admin` fields, which are booleans designating those permissions. As stated above, a sitewide manager may promote a user to sitewide spectator or demote sitewide spectators; a sitewide admin may also promote a user to sitewide manager or to admin, or demote sitewide managers or other admins (including themselves).

The project object contains a `users` object, which map users (by username) to their permissions on the project. An admin, sitewide manager, or project manager may set these at any time, adding to or removing from any of the lists. A project may have zero or more of members, spectators, and managers; if a project has no managers, sitewide managers and admins may still manage the project.