# Assumptions

To design the api of the services I took the following assumptions regarding its functionality:

1. Task creation and assignment are two separate actions.
2. Only user who created a task can view, change it or assign to a different user.
3. The completed task cannot be removed, modified or re-assigned
4. It is only possible to modify the unassigned task
5. Both assignee and owner can complete the task
6. There are two user’ roles:
   1. Manager who can create and assign tasks as well as execute assigned tasks
   2. Regular user who only can execute assign tasks

# Design details

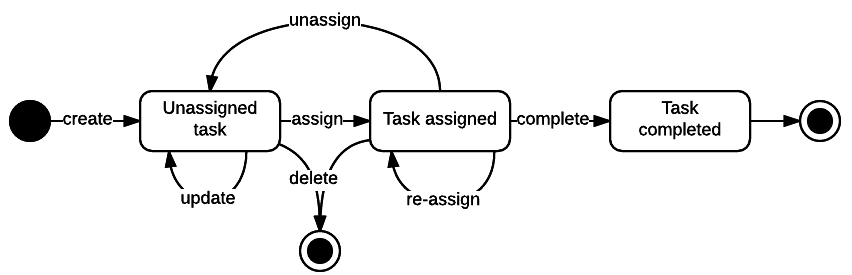


Figure 1 Task state diagram

## API

Since creating and managing of tasks semantically differ from executing of tasks I decided to separate access to the tasks on api level based on the intention of a user (either managing or taking for execution). All task state transitions are implemented via separate resources to emphasize the semantic of operation and easy control availability of a particular state transition. (Implementing this by changing properties on a task object seems much harder and not intuitive to me.)

### API home page

Any user has access to this api but only managers will see “tasks” relation for editing tasks.

|  |  |  |  |
| --- | --- | --- | --- |
| URI | Description | Verb | Details |
| /api | Returns Task API home page | GET |  |

### Task management api

Only users with “manager” role has access to this api.

|  |  |  |  |
| --- | --- | --- | --- |
| URI | Description | Verb | Details |
| /api/tasks{?status} | Represents a collection of tasks the currently logged in user has created (owner). | GET | By default, returns all unassigned tasks. Valid values for ‘status’ parameter: unassigned, assigned, completed |
| POST | Creates a new task object. |
| /api/tasks/<id> | Provides access to a particular task resource. | GET | Returns task representation |
| PUT | Updates unassigned task |
| DELETE | Deletes uncompleted tasks |
| /api/tasks/<id>/assign{?user} | Assigns task to the provided user | PUT |  |
| /api/tasks/<id>/complete | Completes the task | PUT |  |

### Task execution api

Any user (in manager or user roles) has access to this api

|  |  |  |  |
| --- | --- | --- | --- |
| URI | Description | Verb | Details |
| /api/inbox | Represents a collection of tasks assigned to the currently logged in user. | GET | By default, returns all uncompleted tasks assigned to the user. |
| /api/inbox/<id> | Provides access to a particular inbox task resource. | GET | Returns inbox task representation |
| /api/inbox/<id>/complete | Completes the task | PUT |  |

Relations available for task resource

|  |  |
| --- | --- |
| Relation | Description |
| edit | URL for editing the task |
| http://identifiers.dmitry.org/linkrel/remove | URL for deleting the task |
| http://identifiers.dmitry.org/linkrel/assign | URL for assigning the task to a user |
| http://identifiers.dmitry.org/linkrel/complete | URL for completing the task |
| http://identifiers.dmitry.org/linkrel/target | URL for external object associated with the task |

# Implementation details

REST API supports only application/hal+json media type.

Embedded H2 is used as a backend db.

The security naïve implementation is a basic authentication (look at WebSecurityConfiguration class):

1. Prepopulated user: bender (manager role), fry (manager role), leela (user role); all passwords – “password”
2. Users are prepopulated from sql script in the classpath automatically by Spring on startup.

# TODO

Some highly desirable things which are not implemented:

1. Input validation
2. Pagination of collection resources
3. Task id is an integer but if exposed outside, it is better to use different identification (e.g. to avoid forced browsing attacks and insecure direct object references)
4. API versioning
5. Code analyzes tools (PMD, FindBug, Fortify, Clover/Jacoco, etc.)