Task Management System

The purpose of this project is to design and implement a task management system using the Object Oriented Programming principles. This Task Management System (TMS) will make possible for its users to create, manage and organize tasks within different projects. Each user can create multiple projects and within each of these projects, multiple tasks can be added. Users also can set priorities for every task and mark them completed once the tasks are done.

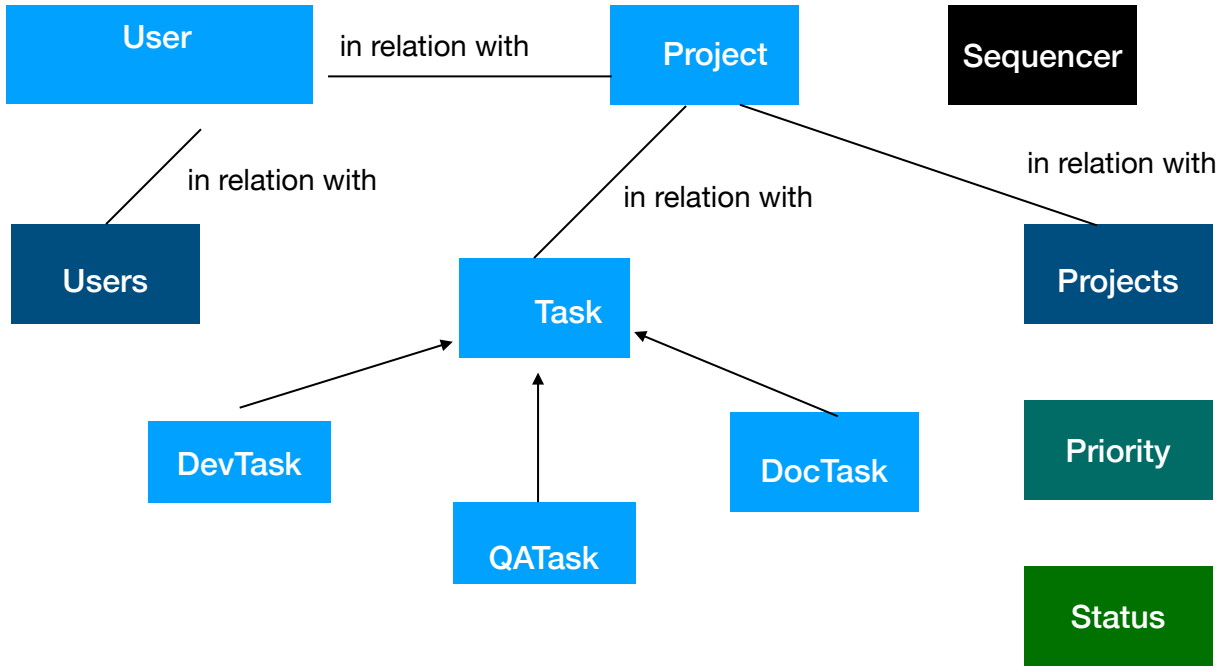
The system can differentiate between various types of tasks such development tasks, quality assurance tasks or documentation tasks (the list of possible tasks is far to be closed, one just offered a possible examples here). Inheritance could be used here to extend the functionality of the various types of tasks.

In order to persist the content of the TMS on the disk, JSON serialization must be used.

Features to be implemented:

- Users can create, delete, remove and modify multiple projects
- Each project contains multiple tasks
- Tasks can be assigned a priority level [low, medium, high]
- Users can mark tasks with different statuses [not started, in progress, completed]
- Users can view the task details

A possible class structure could use like below



Description of the classes

User class
- `id`
- `name`
- `surname`
- `smail`
- `projects`

Possible methods

- __str__
- __repr__
- __eq__
- __hash__
- create_project
- add_project
- remove_project
- get_projects

Project class
- `id`
- `name`
- `description`
- `tasks`
- `deadline`

Possible methods

- __str__
- __repr__
- __eq__
- __hash__
- add_task
- set_deadline

Task class (could be an abstract class)
- `id`
- `name`
- `description`
- `priority`
- `status`
- `user`

Possible methods

- __str__
- __repr__
- set_priority
- get_priority
- get_status
- get_task_info (abstract method)

DevTask(Task) class

- `id`
- `name`
- `description`
- `priority`
- `status`
- `user`
- `language`

Possible methods

- __str__
- __repr__
- get_task_info

QATask(Task) class

- `id`
- `name`
- `description`
- `priority`
- `status`
- `user`
- `test_type`

Possible methods

- __str__
- __repr__
- get_task_info

DocTask(Task) class

- `id`
- `name`
- `description`
- `priority`

- `status`
- `user`
- `document`

    Possible methods

    - __str__
    - __repr__
    - get_task_info

Users class (used to manage all the users in the TMS)
- `users`

    Possible methods

    - add_user
    - remove_user
    - get_users
    - save_data
    - load_data

Projects class (used to manage all the projects in the TMS)
- `projects`

    Possible methods

    - add_project
    - remove_project
    - get_projects

Sequencer class (used to generate unique ids over the TMS)
- `sequence`

    Possible methods

    - generate_sequence
    - set_sequence
    - get_sequence

Priority (better an Enum than a class
- `LOW`
- `MEDIUM`
- `HIGH`
- `CRITICAL`

Status (better an Enum than a class)
- `NOT_STARTED`
- `IN_PROGRESS`
- `COMPLETED`

Possible extensions for the TMS (optional)

- **Task Filtering**: add functionality for filter tasks, based on priority or status

Running the application

The TMS application can/will be run in different ways

1. You can choose to run it from the main module as a chain of operations like creating a user, creating a project, assign it to a user, create one or more tasks and assign them to a project, etc
2. You can choose to use a menu (text format) whether to be flat or hierarchical and run the comamnds from there.

Example of flat menu:

1. Create user
2. Delete user
3. Modify user
4. List users
5. Create project
6. Delete project
7. Modify project
8. List projects
9. Create development task
10. Create quality assurance task
11. Create documentation task
12. Delete task
13. Modify task
14. List tasks
15. Exit

Example of hierarchical menu:

1. Users
    1. Create user
    2. Delete user
    3. Modify user
    4. List users

2. Projects
    1. Create project
    2. Delete project
    3. Modify project
    4. List projects

3. Tasks
    1. Create development task
    2. Create quality assurance task
    3. Create documentation task
    4. Delete task
    5. Modify task
    6. List tasks
4. Exit

Suggestion for implementing the menus: one possible way is to use multiple if-s, but imagine that the code will become a mess with so many possible branches, so I would simulate a switch instruction using dictionaries.

```
def add_project(project):
// add project stuff

def remove_project(project):
// remove project stuff

def display_projects():
// display projects stuff
.
.
.
def error_handler():
print(“This option does not exist”)

menus = {1: add_project, 2: remove_project, 3: display_projects, …}
option = int(input(“Enter your option:”))
func = menus.get(option, error_handler)
func()
```

Sending the files

In order to correctly send your work to be checked, please prepare the following files in a zip archive and upload it to the classroom space:

1. All the source files (not the .pyc files which only increase the size of the archive)
2. The files containing the serialization of the objects you’ve used