Task Planner System

Design and implement a task planner system

A task has the following details :

1. Title
2. Creator
3. Assignee (Optional)
4. Status
5. Type
6. Due date

A task can be of following types with additional information apart from what is mentioned above

1. Feature
   1. Feature summary
   2. Impact (Low, Moderate, High)
2. Bug
   1. Severity (P0, P1 or P2)
3. Story
   1. Story Summary
   2. A story can have multiple SubTracks  
      A SubTrack has the following details :
      1. Title
      2. Status

A subtrack can be created and attached only to an existing story in non-completed status

It should be **easy** to add a new task type to your application

The status can change from a state to any state.

Status field takes one of the following states depending on the task type :

**Feature** => Open, In progress, Testing, Deployed

**Bug** => Open, In progress, Fixed

**Story** => Open, In Progress, Completed

**Subtrack** status field takes one of these values  
Open, In Progress, Completed

**A sprint** is defined as a collection of tasks used to track progress. You can add or remove a task from sprints.  
A task can be part of only one sprint at a time.

Your task planner should have the following functionalities:

1. Task
   1. Create a task of any type
   2. Create subtrack
   3. Change the status of the task/subtrack
   4. Change assignee of the task
   5. Display tasks assigned to a user categorised by task type
2. Sprint
   1. Create/Delete a Sprint
   2. Add/remove task to/from sprint
   3. Display sprint snapshot. This should display the tasks that are part of the sprint and also display if they are delayed or on track
3. **Bonus Question (Only if time permits) :-**The transition/change in the status should be based on allowed transitions  
   eg :-  
   For task type feature, we might want to have only following allowed transitions :

Open => In progress

In progress => Testing

Testing => Deployed

In progress  => Deployed

The examples below are just to understand the functionalities and **may not necessarily** be used in the same format as input to your driver program.

Let’s say we want to add the following tasks to a sprint :

Tasks:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Title** | **Creator** | **Assignee** | **Status** | **Due date** | **Type** | **Type attributes (comma separated)** | **Sprint** |
| Create Dashboard | Brad | Peter | Open | 2019-04-12 | Feature | Create console for debugging, Low |  |
| Fix mysql issue | Ryan | Ryan | In progress | 2019-04-14 | Bug | P0 | Sprint-1 |
| Create a microservice | Amy | Ryan | Completed | 2019-03-12 | Story | Add logging to the feature | Sprint-1 |
| Setup console | Ryan | Ryan | In progress | 2019-04-14 | Feature | Create console for debugging, High |  |
| Console api | Ryan | Ryan | In progress | 2019-04-14 | Feature | Create api for console , High |  |

Subtrack:

|  |  |  |
| --- | --- | --- |
| **Title** | **Status** | **Parent task** |
| Development | Open | Create microservice |
| Unit Test | Open | Create microservice |
| Integration Test | Open | Create microservice |

**Display tasks assigned to a user categorised by task type:**

Eg: For assignee Ryan

User => Ryan:

Task Type => Bug

Title => Fix mysql issue

Sprint =>  Sprint-1

Task Type => Feature

Title => Setup console

Sprint =>

Title => Console api

Sprint =>

Task Type => Story

Title => Create a microservice

Sprint => Sprint-1

SubTrack:

Development

Unit Test

Integration Test

Eg: For assignee Peter

User => Peter:

Task Type => Feature:

Title => Create Dashboard

Sprint => Sprint Id/Sprint name

**Display status of Sprint-1**

**Sprint title => Sprint-1**

On track Tasks:

Fix mysql issue

Setup console

Create Dashboard

Delayed Tasks:

Create a microservice

Note: Task will come under “delayed task” if  the task (with non-completed status) has crossed the due date.

**Expectations :**

Please make use of in memory data structures and do not use any external datastore for  
storage/querying.

Input/Output can be from terminal or file or taken from some data structure inside the driver program whichever is convenient for you. It should be easy for the evaluator to give new input or change existing input and test new cases.

You can look for API references online.

1. Demoable code. Functionalities mentioned above
2. Clean Interface design for the module.
3. Clean internal design and implementation of the library and the application.
4. **Functional completeness** is must. Bonus questions are good to have.
5. **Extensibility**
6. Take care of Exception and Corner case handling.
7. Test cases covering various cases are good to have.
8. You are free to use the language of your choice.