# Design Description

## Assumptions

* All timestamps are represented in UTC time.
* GetUsers and GetUser returns the same type.
* An issue may have several comments.
* An issue may only have one assigned user.
* Assigning a user to an issue does not mean that the issue is changing state to “in progress”. The issue will keep its current state. An issue only changes state if the set function is explicitly called.
* Transitioning from one state to the same state is a valid use case and will record the transition change. “The issue can transition to any state from any other state”.
* A user can be removed even if assigned to an issue. The issue will not be modified in any way.
* I was not sure how to interpret “Users can comment on issues” and the fact that the AddIssueComment signature did not have a user ID as parameter. It could be part of the user object to get the ID, but I simply added it as an operation on issues. The AssignUser signature does have a user ID in the signature which indicates that the ID is provided in the signatures.

## Design Decisions

I separated the code in two layers, the Service and the Repository layer. The Service layer contains the core/business logic and the models, while the Repository layer contains the data storage infrastructure. I kept the files in the same project for simplicity. The repository layer implements the repository pattern by providing interface abstractions for the data storage. This way the data storage implementation can be substituted if necessary. It also makes the core logic more testable since you can simply mock the interfaces.

The only classes that are public are those defined in the IUserService and IIssueService interfaces, which serves as the public interface of the engine. These interfaces contain the specified operations in the case description.

I was not too concerned with error handling and did not validate input data other than some null checks. When trying to query the data layer with an ID, the repository layer will throw an exception if the ID is not found in the data storage. This way I make sure only valid IDs are used.

## Performance Considerations

I do not see any obvious concerns about performance since all data is stored in memory and no heavy computing is done. Would I for example implement this as a web app, I would probably make the calls asynchronous. And if the data storage would have been substituted for an actual database, then unnecessary roundtrips to the database would have been of greater concern.