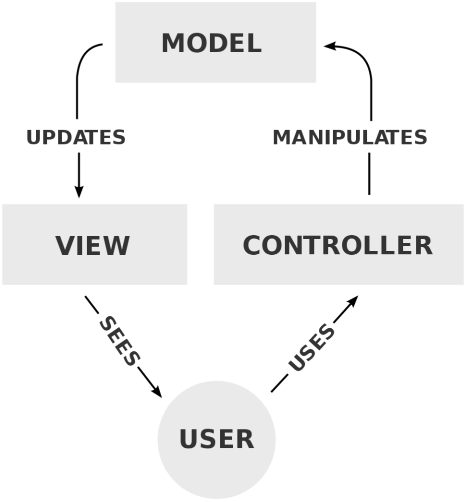
CMPE 476 ASSIGNMENT 2 REPORT

Chapter 1: MVC Pattern Research

Model-View-Controller is a software design pattern which utilizes the advantages of being more than one to perform divide and conquer strategy over the desired tasks. It has been mostly used in mobile applications, web applications, and other applications that have a graphical user interface to interact with the users.

It simply has 3 parts, such as models, views, and controllers. For each desired task, the requirements should be stated elaborately. After requirements elicitation, the specifications are set and other necessary configurations are regulated according to the elicitation. Hence, the task is ready to be applied with the Divide & Conquer strategy.

At first, the model is defined since it is both used by the viewer and the controller in different ways. In requirements, there is enough information to create all necessary fields, functions and constructors of the model. Model definition is quite important, since every action which calls either viewer or controller depends on the model and if any error occurs in model definition, it may lead the maintenance and development process of both the viewer and controller in further stages.



After model definition, the controller should be defined to apply any changes to the state of the model in the database. It includes the logic of the program, the necessary connectors to the database and several configurations to run the app in a stable state. There are more than one controllers for one model or more than one model for one controller since the desired task may include several distinct subtasks which can not be united as one. After the implementation of the logic in the controller, there has to be an access point for either a controller or a viewer to call this controller for a task execution. Endpoint is the term for such access points for a controller. Viewers and other controllers use the endpoint of a controller to utilize it in their environment. For a controller, a call of another controller can be simply illustrated as blackbox testing since only the input and output of the called controller can be observed inside the caller controller. No intermediate form of the input or any source code can be observed inside the caller controller.

Therefore, the only thing that has to be implemented is the viewer. Viewer is the part of the graphical user interface which interacts with the user by providing interaction with the controllers to perform tasks. However, viewers are composed of tiny basic graphical components which are encouraging users to perform more tasks in the system. Also, viewers are simplifying the usage of the controllers for the users, which is very much preferred in software engineering.

In conclusion, it is very convenient to say that for any task scaled in different complexities, it could be beneficial to use model-viewer-controller strategy to achieve fast results within tight intervals, since this structure provides a concurrent working pattern for developers in that task. However, any unexpected error or change in one component eventually leads to a chain of changes in all other components in MVC which increases the workload and slows the development process.

Chapter 2: Project Implementation

- Requirements:
    - a) Registration

        1. Users must register to the application with an email, password, name, and surname.
        2. Passwords must include 1 Uppercase Letter, 1 Lowercase Letter, and 1 punctuation mark.
        3. Punctuation mark can either be a semicolon, colon, hyphen, or exclamation mark.
        4. Users must verify their account after registration by clicking the link in the email sent by the application.

    - b) Login & Login Operations

        1. Users must login to the app with their email and password.
        2. Users may reset their password if the password is forgotten.

    - c) Workspace

        1. Users may create a workspace by providing a workspace name.
        2. Users may invite other users by sending invitation emails via the application.
        3. Users may join a workspace by accepting the invitation sent to their email.

    - d) Channels

        1. Workspaces must contain one or more channels in which users can communicate.
        2. Channels may be protected for users with a distinct role.
        3. Workspace creators can restrict the channel usage by applying role protection.
        4. Workspace creators can restrict the channel usage by applying user protection.

- e) Roles

1. Workspace creators can create roles for different purposes, which can be either restraining users to send messages in a channel or assigning tasks.
2. One user may have more than one role.
3. Assigning roles or unassigning roles can be performed only by workspace creators.

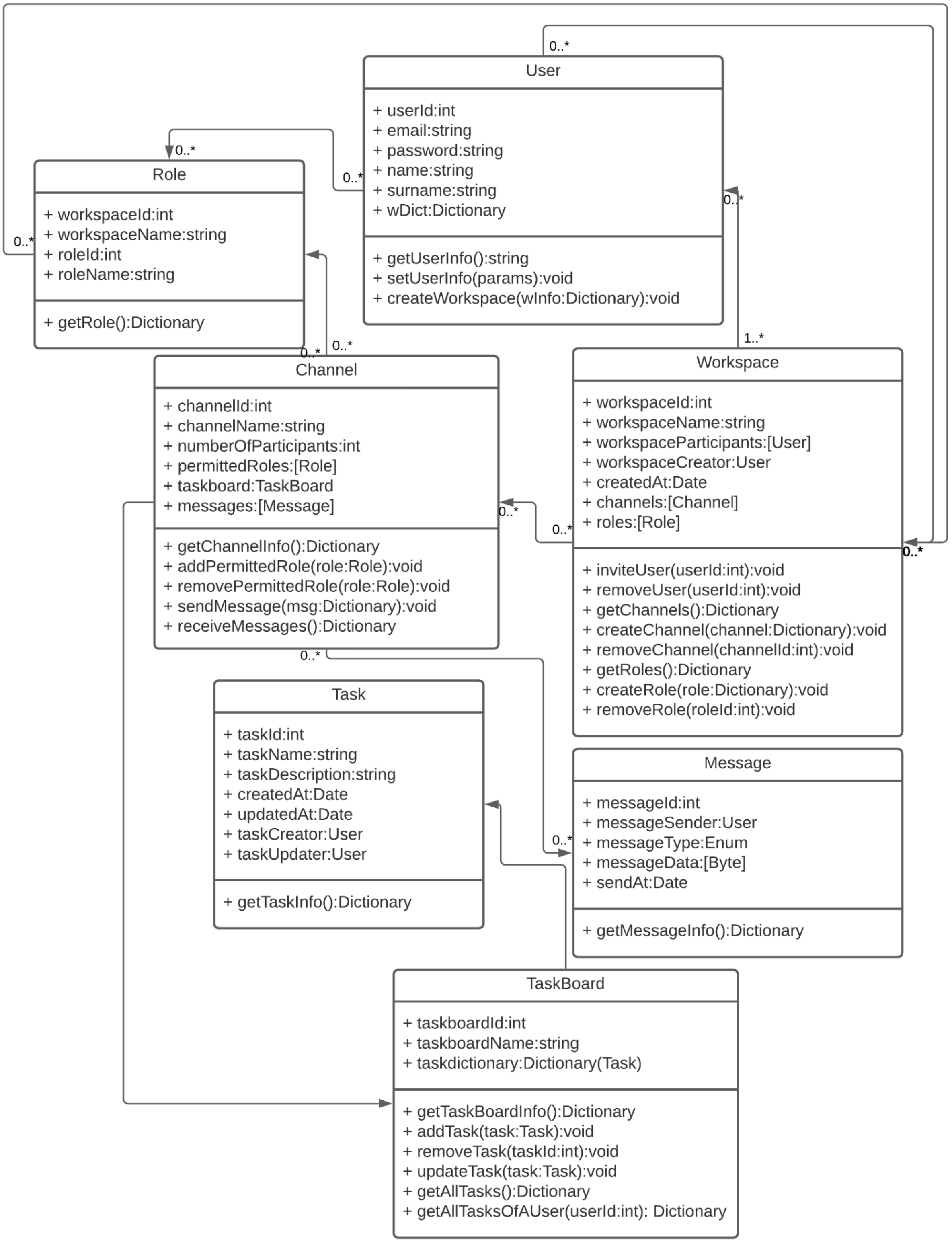
- f) Sending & Receiving Messages

1. Users can send messages in a channel if they are permitted.
2. Users can receive messages in a channel if they are permitted.
3. Sending messages may include text messages, hyperlinks or files.

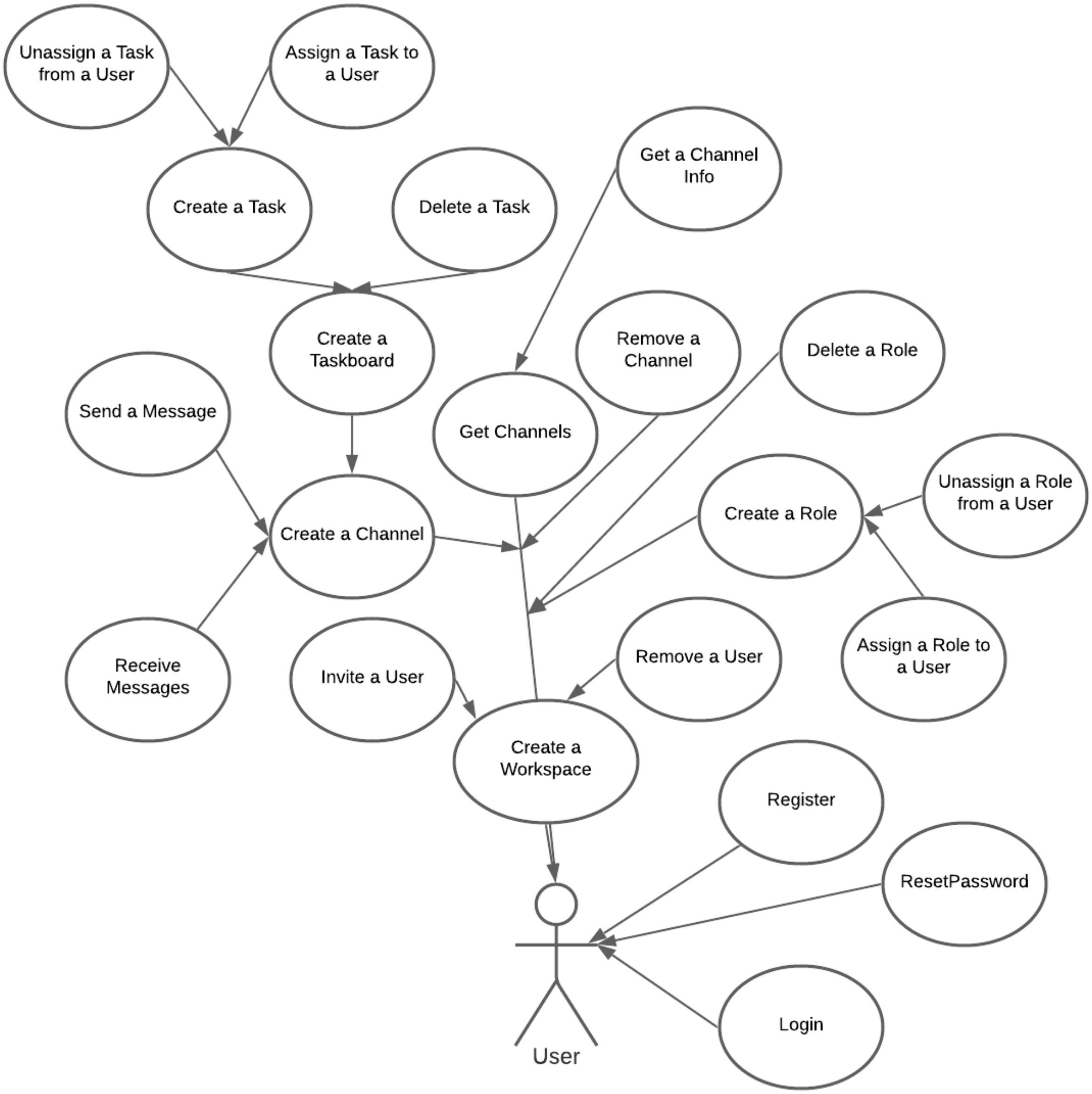
- g) Taskboard

1. Workspace creators must create a task board for a channel in which users with permitted roles may assign tasks to different users.
2. Tasks must have one of the given status: ToDo, InProgress, Completed, Verified.
3. Tasks must have a creator, an assignee, date-time-status of creation, date-time-status of updates.
4. Users in a channel may create a task with the features explained in g.3.
5. A task can be deleted only by the workspace creators.
6. A user can view all their tasks or another user’s tasks in the task board.

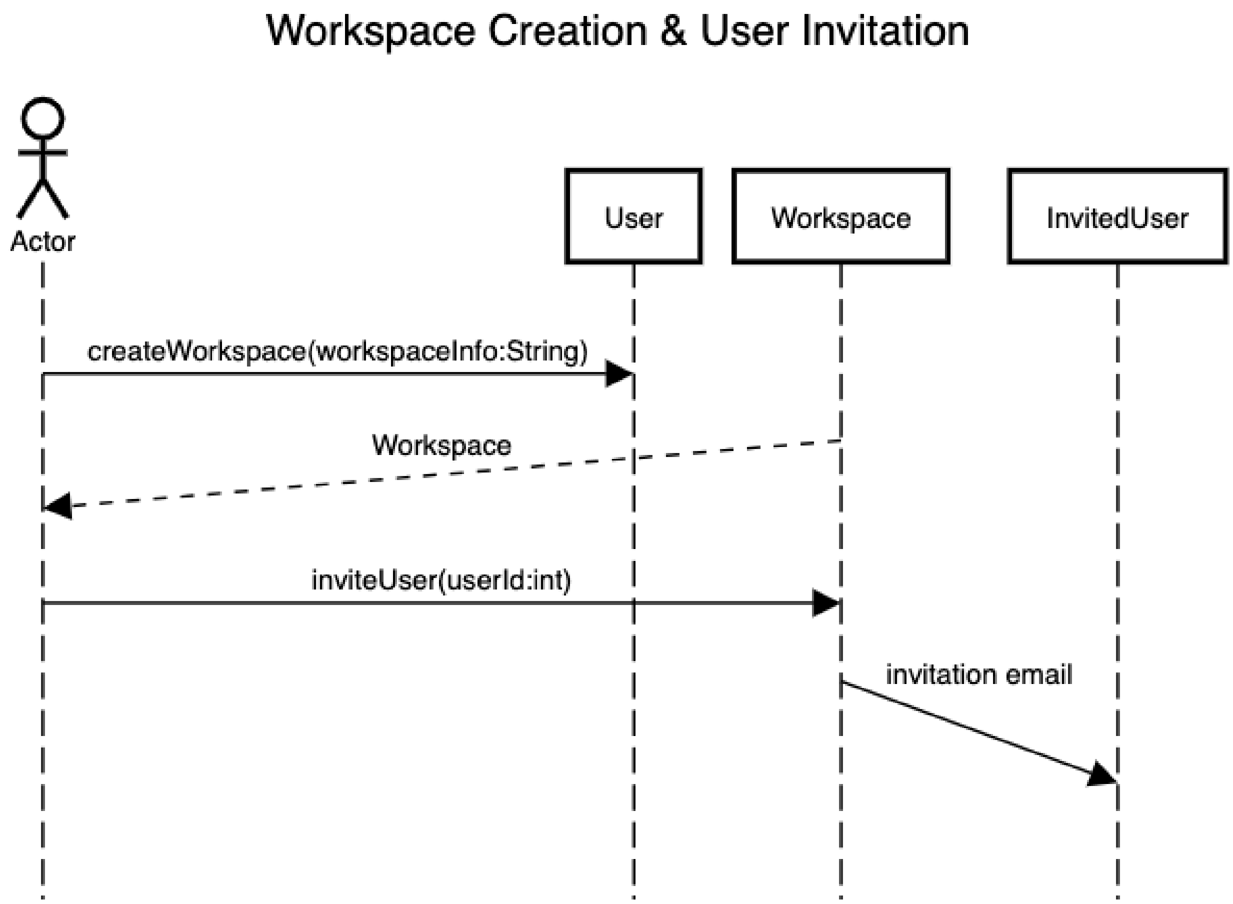
- Class Diagram


0..*
User
+ userId:int
+ email:string
+ password:string
+ name:string
+ surname:string
+ wDict:Dictionary
+ getUserInfo():string
+ setUserInfo(params):void
+ createWorkspace(wInfo:Dictionary):void
0..*
0..*
Role
+ workspaceId:int
+ workspaceName:string
+ roleId:int
+ roleName:string
+ getRole():Dictionary
0..*
0..*
0..*
0..*
1..*
Channel
+ channelId:int
+ channelName:string
+ numberOfParticipants:int
+ permittedRoles:[Role]
+ taskboard:TaskBoard
+ messages:[Message]
+ getChannelInfo():Dictionary
+ addPermittedRole(role:Role):void
+ removePermittedRole(role:Role):void
+ sendMessage(msg:Dictionary):void
+ receiveMessages():Dictionary
Workspace
+ workspaceId:int
+ workspaceName:string
+ workspaceParticipants:[User]
+ workspaceCreator:User
+ createdAt:Date
+ channels:[Channel]
+ roles:[Role]
+ inviteUser(userId:int):void
+ removeUser(userId:int):void
+ getChannels():Dictionary
+ createChannel(channel:Dictionary):void
+ removeChannel(channelId:int):void
+ getRoles():Dictionary
+ createRole(role:Dictionary):void
+ removeRole(roleId:int):void
0..*
0..*
0..*
0..*
Task
+ taskId:int
+ taskName:string
+ taskDescription:string
+ createdAt:Date
+ updatedAt:Date
+ taskCreator:User
+ taskUpdater:User
+ getTaskInfo():Dictionary
Message
+ messageId:int
+ messageSender:User
+ messageType:Enum
+ messageData:[Byte]
+ sendAt:Date
+ getMessageInfo():Dictionary
0..*
TaskBoard
+ taskboardId:int
+ taskboardName:string
+ taskdictionary:Dictionary(Task)
+ getTaskBoardInfo():Dictionary
+ addTask(task:Task):void
+ removeTask(taskId:int):void
+ updateTask(task:Task):void
+ getAllTasks():Dictionary
+ getAllTasksOfAUser(userId:int): Dictionary

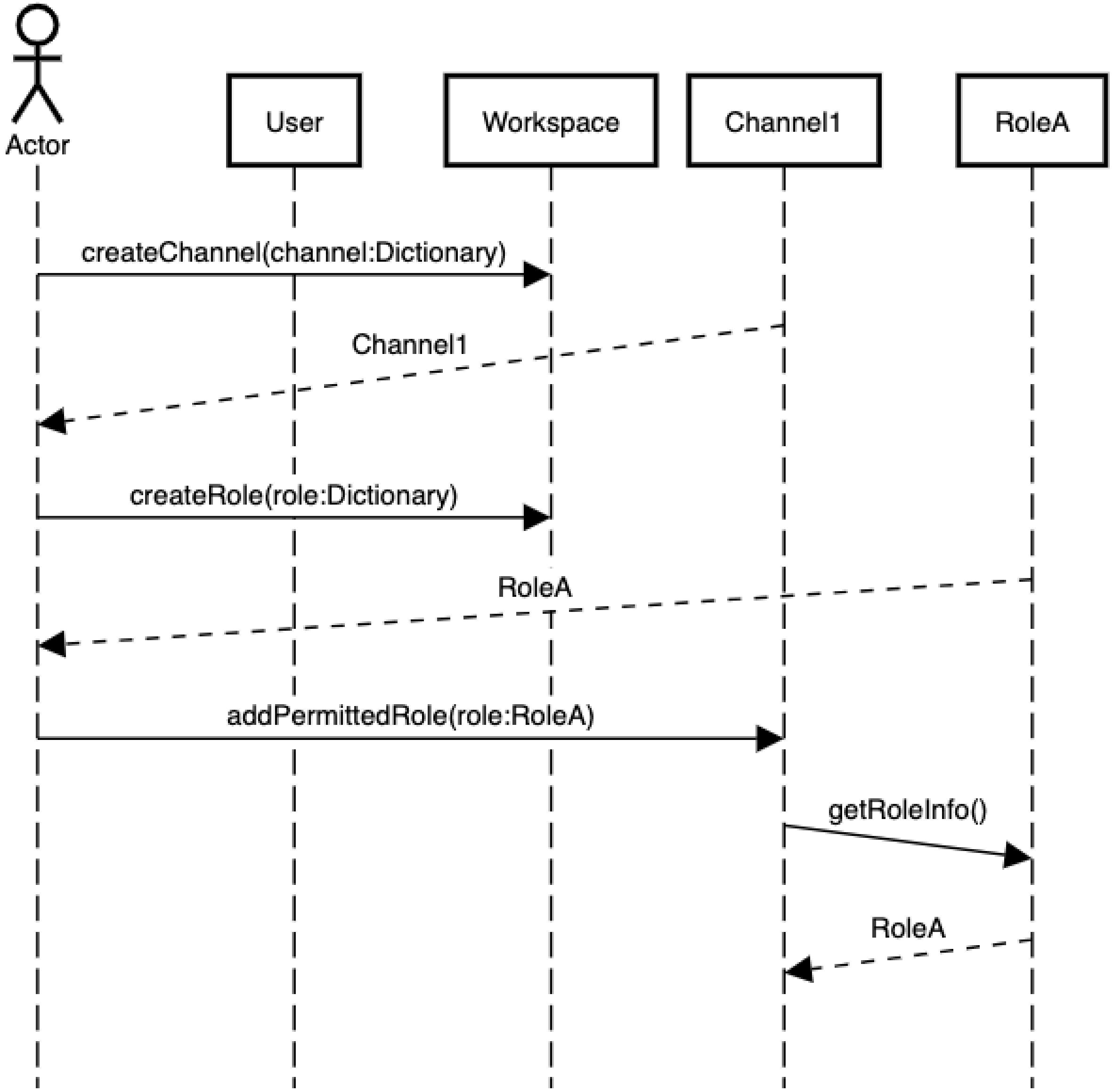
- Use Case Diagram



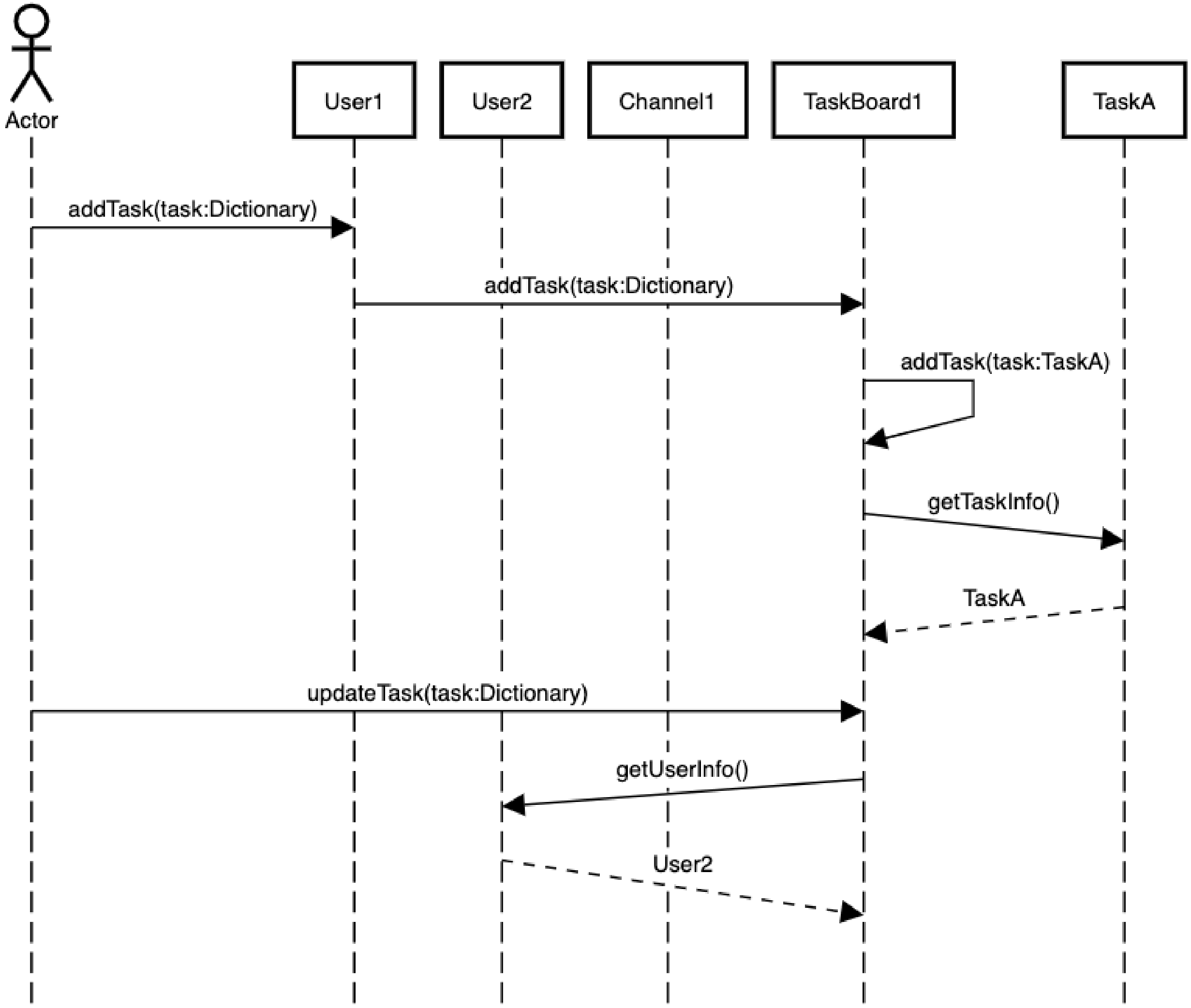
- Sequence Diagrams



Channel Creation & Role Creation & Apply Role Restriction


Actor
User
Workspace
Channel1
RoleA
createChannel(channel:Dictionary)
Channel1
createRole(role:Dictionary)
RoleA
addPermittedRole(role:RoleA)
getRoleInfo()
RoleA

Create Task & Assign Task to User


Actor
User1
User2
Channel1
TaskBoard1
TaskA
addTask(task:Dictionary)
addTask(task:Dictionary)
addTask(task:TaskA)
getTaskInfo()
TaskA
updateTask(task:Dictionary)
getUserInfo()
User2