Attachment

1. /createNewAttachment

Type: post

Returned Data: message of success or failure.

Body Parameters

* name (name of attachment).
* path (path of attachment  where that attachment is saved).
* member\_id (the user or members id, who have uploaded it).
* task\_id (id of that task to which this attachment is related).
* project\_id

First it will create new attachment and then it will add this attachment to ==> array of attachments in Task Scheme (Table or document).

1. /getAttachmentById

Type: post

Returned Data: data of attachment with the member’s data who uploaded it

Body Parameters

* \_id

It will return the attachment by using its \_id attribute.

Comment

1. /createNewComment

Type: post

Returned Data: message of success or failure.

Body Parameters

* message (comment content).
* member\_id (the user or members id, who have commented it).
* task\_id (id of that task to which this comment is related).
* project\_id

First it will create new comment and then it will add this comment to  array of comments in Task Scheme (Table or document).

1. /getCommentById

Type: post

Returned Data: data of comment with the member’s data who uploaded it

Body Parameters

* \_id

It will return the comment by using its \_id attribute.

Project

1. /createNewProject

Type: post

Returned Data: message of success and failure

Body Parameters

* name
* description
* start\_date (date format >> MM-DD-YYYY)
* end\_date (same as above)
* project\_type >> software/other
* leader\_id (User id)
* team\_id: >> (if of team then pass team\_id OR if there is no team then pass ‘no-team’)

First it will create new project with its details and sets projects status to ‘in-progress’ automatically, and if this project is created by any team then this project\_id will be added to teams schema  in their projects (field or document).

1. /getProjectById

Type: post

Returned Data: data of project with all timelines details and all of its members details.

Body Parameters

* \_id  id of project

1. / updateProjectLeader

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* leader\_id (id of leader  User)

First it will update the project leader and then add leader to members of the project.

1. / updateProjectStatus

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* status (status of projectin-progress OR done).

1. / updateProjectType

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* project\_type (type of projectother OR software).

1. / updateProjectCost

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* project\_cost (cost of project  if software project cost will be calculated using UCP Rule OR if project is related to other categories then project cost is automatically set to ‘Sorry No Information Provided’).

1. / addMemberToProject

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* member\_id (id of member(User) ).

1. / addMultipleMemberToProject

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* member\_id[ARRAY] (array of ids of member(User) ).

1. / updateTTAESOfMIProject

Elaboration: update Total Tasks And Efficiency Score Of Member In Project.

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* member\_id (id of member(User) of which efficiency score and total tasks are supposed to be updated ).
* efficiency\_score (the score we calculate through checking its tasks time , and how much time member take to do it).

First this router update the efficiency score in Project Schema  it simply add this efficiency score to previous one and increase total tasks by ‘1’ , after it will also update total tasks and efficiency score in User schema. In that way we can keep track of User’s individual performance in Projects and User’s overall performance.

Sub Task

1. /createNewSubTask

Type: post

Returned Data: success or failure message

Body Parameters :

* name
* description
* member\_id (the user id to which this sub task is assigned.)
* task\_id (the id of task  because sub task is always part of one task)
* project\_id

This router first create sub task and after this, it’s id will be added to Task Schema.

1. / getSubTaskById

Type: post

Returned Data: data of sub task with details of its member.

Body Parameters :

* \_id

1. / updateSubTaskStatus

Type : put

Returned Data: success or failure message.

Body Parameter:

* \_id (id of sub task)
* status  (“in-progress” OR “done”)

Task List

1. / createNewTaskList

Type: post

Returned Data: message of success or failure.

Body Parameters:

* name
* project\_id (id of that project of which this task list is part of).
* description

1. / getTaskListById

Type: post

Returned Data : data of task list with all information of its project and its containing tasks.

Body Parameters:

* \_id

Task

1. / createNewTask

Type: post

Returned Data: message of success or failure.

Body Parameters:

* name
* description
* pre\_req\_id (id of pre requisite , if no any pre requisite then pass just 0)
* due\_date (the deadline for that task)
* member\_id\_array (array of members id to which task is assigned , if only one member assigned then just pass array of single element)
* task\_list\_id
* project\_id

1. / getTaskById

Type: post

Returned Data : data of task with all information of its comments, attachments, members and sub\_tasks.

Body Parameters:

* \_id

1. /updateTaskStatusLeader

Type: put

Returned Data: message of success or failure.

Body Parameter:

* \_id
* project\_id
* status (status can be “in-progress” OR “done” just)

1. /updateTaskStatus

Type: put

Returned Data: message of success or failure.

Body Parameter:

* \_id
* member\_id
* project\_id
* status (status “pending” )

when status set to waiting the leader will decide it to be set as “done” or again “in-progress”. By the way member just set as task for “pending”. Then the work is of leader.

Team

1. /createNewTeam

Type: post

Returned Data: message of success or failure

Body Parameters:

* name
* description
* leader\_id (the id of leader)

1. /getTeamById

Type: post

Returned Data: data of team with all information of its projects, members and leader.

Body Parameters:

* \_id

1. /addMultipleMemberToTeam

Type: put

Returned Data: success or failure message

Body Parameters

* \_id (id of project)
* member\_id\_array[ARRAY] (array of ids of member(User) ).

It takes array of members id and insert one by one using for loop.

Timeline

1. /createNewTimeline

Type: post

Returned data: message of success or failure

Body Parameters:

* content
* project\_id (id of project to which this timeline is part of)

First it will create new timeline and after that it will add it to project schema which have field of timelines which is actually an array.

User

1. /registerUser

Type : post

Returned Data: message of success or failure.

Body Parameters:

* name
* email
* password
* phone\_number

First it will checks is same username is already exists or not. Then it will proceed to insert user data in database.

1. /loginUser

Type: post

Returned Data: data of user with all data of its teams and its projects.

Body Parameters:

* email
* Password

1. /registerUserGoogleFB

Type : post

Returned Data: message of success or failure.

Body Parameters:

* name
* email

First it will checks is same username is already exists or not. Then it will proceed to insert user data in database.

1. /loginUserGoogleFB

Type: post

Returned Data: data of user with all data of its teams and its projects.

Body Parameters:

* email

1. /updateUser

Type: put

Returned Data: message of success or failure

Body Parameters:

* name
* phone\_number

If you want to change just number , then pass new number and old name, so that it doesn’t effect the original name. And also same for name or any thing else.

1. /getUserById

Type: post

Returned Data: data of team with all information of its projects, teams etc.

Body Parameters:

* \_id