Project requirements

Phase 1

Read, because I actually spend my time writing this and I had to pause GOT S08E04 in the middle to complete this:

Motivation:

1) We are going to help an organization which helps hundreads of specially able kids and uneducated pople.

2) If we do this properly we might get a chance to make a success story somewhere July or August of 2019.

3) Imagine your first ever youtube video from Google's official account. And they are also going to promote the video to reach more people to inspire.

4) **No sucess story for shitty product. So, lets do this.**

Some Convenient:

1) Kindly use the same string tagname of the dictionary I use here when returns back from the server to app.

2) All TimeStamps should be UNIX TimeStamp. Refer the below link if require https://pypi.org/project/django-unixtimestampfield/

3) Don't neglect the ids while returning data back to the app. It is useful to remove duplicate items from the list while display to the user.

Articles: Method.GET

[

{

article_id:

username: :: username of the person who posted the article

team_name: [

...
] :: all the team names of the user
time_stamp: :: article created timestamp
profile_picture_url: :: of the creator
Image_url: :: **null** if the article doesn't hava picture
description:
}

]

1) http:// … /articles/
Return 20 / 30 recent articles.

2) http:// … /articles/before?time_stamp e.g. http:// … /articles/before?1556604826
Return 20 / 30 recent articles posted before the given timestamp.

Articles: Method.DELETE

1) http:// … /articles/delete/id e.g. http:// … /articles/delete/155
Delete the article which has given id.

Articles: Method.POST

{
user_id
image :: image is optional
description
}

1) http:// … /add_article/
Adding a new article.

Events: Method.GET

[
{
event_id:
event_title:
date:
team_name: [
...

```
                    ]
        assign_by:
        investment_amount:
        investment_in_return:
        description:
        organizers:    [
                        {
                            user_id:
                            username:
                            profile_picture_url:
                        }
                        ...
                    ]
    }
    ...
]
```

1) http:// … /events/

Return 20 / 30 events from all the events according to the event date. Use for initial load and refreshing for higher authorities which has access to all the event.

2) http:// … /events/before?time_stamp e.g. http:// … /events/before?1556604826

Return 20 / 30 events from all the events which event date is before the given timestamp. Use for load more events for higher authorities which has access to all the event.

3) http:// … /user_events/

Return 20 / 30 events according to the event date which the sign in user is one of the organizers of the event. Use for initial load and refreshing for individual user.

4) http:// … /user_events/before?time_stamp e.g. http:// … /events/before?1556604826

Return 20 / 30 events which event date is before the given timestamp which the sign in user is one of the organizers of the event. Use for load more event for individual user.

5) http:// … /events/event_id e.g. http:// … /events/event_id/155

Return a list which contains a single and same type of dictionary object of the above event list which has the given event id. Used for single event details.

Events: Method.DELETE

1) http:// … /events/delete/event_id e.g. http:// … /events/delete/155

Delete the event which has event id “event_id” and its corresponding investment and organizers data from all the three tables; events, organizers and investment.

Events: Method.POST

{

user_id: :: this user id will be the assign_by of the event

date: :: date of the event

title:

description:

selected_team: [

...

] :: contains list of team name

organizers: [

...

] :: list of user id which will be the organizers of the event

}

1) http:// … /add_new_event/

Adding a new event

Events: Method.POST / UPDATE

{

date: :: date of the event

description:

organizers: [

...

] :: list of user id which will be the organizers of the event

}

1) http:// … /update_event/event_id/

Update details of the event which has event id “event_id”.

TIP: Delete all the existing organizers of the event from the organizers table and add the new organizers. Because when someone update new organizers of the event it might be entirely different from the existing organizers.

Investment: Method.POST / UPDATE

{

investment_on: amount

…

investment_amount:

investment_in_return:

}

1) http:// … /events/update_investment/event_id

Submitting investment details of a spicific event.

“investment _on” is the reason of the investment, it may have different name such as water, food, etc. so cannot use request.POST[‘tag’] with a specific tag string.

“amount” is the how much amount invested on its corresponding “investment_on”.

“investment_amount” is the total amount invested (sum of investment_on), so no need to calculate to store and return for an event.

“investment_in_return” is the how much money they get back from their investment amount of the event

Everytime this request occurs, delete all the existing information of investment of the event from investment table and store the new data “investment_amount” and “investment_in_return” of the event which has event id of “event_id”.

Users: Method.GET

{

[

user_id:

user_name:

profile_picture_url:

team_name: [

...

]

]

...

}

1) http:// … /users/

Return all the user.