# WEB FRAMEWORK PROJECT

CLUB MANAGEMENT SYSTEM

BY -

ADITYA OASIS - PES2201800007

VARUN SESHU - PES2201800074

PRITHVI RAJ - PES2201800307

DHRUV SHARMA - PES2201800115

# 1. INTRODUCTION

## 1.1 Purpose of this document

This document is used to give definition and structure to the eventual implementation of our project ‘College Club Management’. It provides direction to the implementation of the project, with updates on the changing facets of the project. It’s a description on what the project can and can’t do, along with stretch goals for the features in the project.

## 1.2 Scope of this document

This document discusses the various aspects and features of the project we wish to deliver and with a high level description of the UX. It’s a snapshot of the implementation process as the project development continues. It also has the stretch goals we wish to deliver if possible by the submission date. It has various use cases and target clients for the project.

## 1.3 Overview

The project is a College Club Management Site, it’s a one place stop for college clubs to broadcast information about their events, handle recruitment and run their club’s internal communications and projects without the hassle of email, chat, drive and so on. It’s use cases are for both college clubs and college students who want information on clubs and events.

## 1.4 Business Context

The business potential for the project cannot be understated. The project, if adopted by clubs for ease of management could streamline and improve the productivity of clubs, while simultaneously improving the outreach about their events to the students in the college. A reasonable fee could be charged to the clubs for availing these services to offset our hosting and storage costs and potentially turn a profit.

# 2. GENERAL DESCRIPTION

## 2.1 Product Functions

The product is mainly used by college clubs to help broadcast information about an event that they are holding, also providing a platform for the clubs to work and be productive about their projects by having all information in one place without the hassle of multiple email, chats and google drive requirements.

## 2.2 User Characteristics

We have 2 classes of target users -

1. College students - They are visitors to the site, they get information about the events that the clubs are holding along with requirements on how to join the events. They can also explore the club's pages and participate in their recruitment process.
2. College club members - They also have visitor access along with access to the club's private page, where the club decides the projects, and club's chat, drive and to-do list live.

## 2.3 User Problem Statement

For College Student User -
Make the information about clubs and events hosted in college transparent and easy to access and improve ability to contact clubs for recruitment

For College Club Members -
Make it easier for clubs to market themselves and make sure no event/project they are holding slips through the cracks. Also provide a platform for increasing productivity of the club by melding multiple branches of communication.(real time registration count)

## 2.4 User Objectives

For College Students -
To get better information about the clubs in college, the events they are hosting to increase participation and streamline the recruitment process in clubs

For club members -
To help organize and improve the functioning of a club by increasing productivity through a common platform. To help them market the events they are organizing to a larger group of college students.

## 2.5 General Constraints

The College student user doesn’t have access to a club’s private member’s account.
Only a club’s leader can create, delete, delegate projects.
Club members can participate in club activities and projects.
Users can be part of multiple clubs and hold different positions in the clubs.

## 3. Functional Requirements

For College Student User -

In the Main page of the site, where the College Student users first arrive, consists of the following features -

- Calendar of events for the semester
- Notifications for events happening in today, tomorrow, and the next 7 days
- Advanced registrations to events happening in college
- Links to the pages for all the clubs hosted by the college

Each club's page registered on the site has its own page which consists of the following –

- The logo, images and other insignia of club
- Short description of the club, and their vision
- The events being held by the club in the current semester, their description, along with links to register for them
- Links to the club's website, application to join club and other important links
- Sign In option for members of the club
-

For College Club Users -

- Once signed In, a member is greeted by a club member page
- A global Chat for all members of the club
- A global Drive, to store important documents for the club
- A global To-Do list for tasks of the club
- An Announcements bar, to communicate with all members of club

Club's project page

- There are 3 types of users for projects
- Club Admin – Can create new projects, assign/remove Sub-Admins and Members to them, assign work across projects
- Club Sub-Admin – Heads a project, can add/remove members from a project, controls the discourse of the project
- Club Member – Can only be assigned tasks in a project
- The current projects/events of the club will be listed in this page

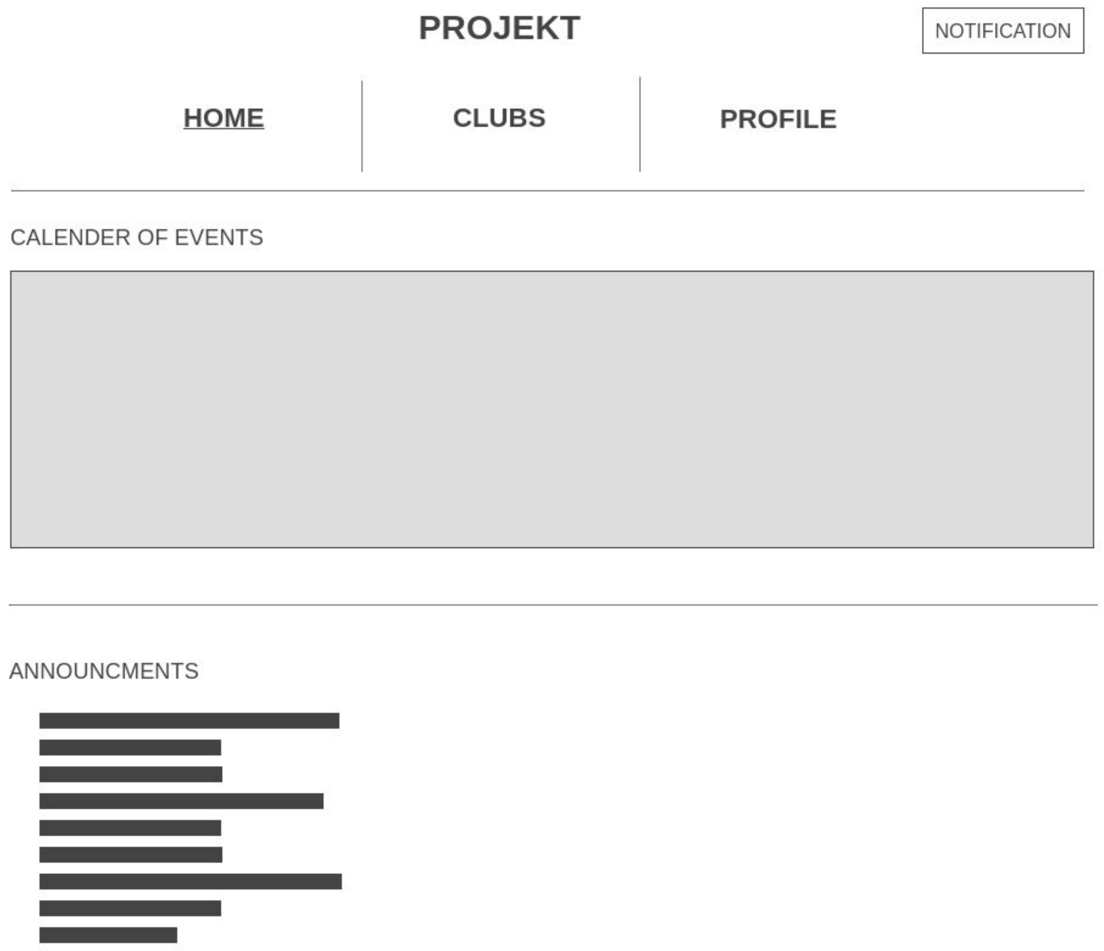
Projects can be dynamically created in the club projects page, each project in turn has it own -

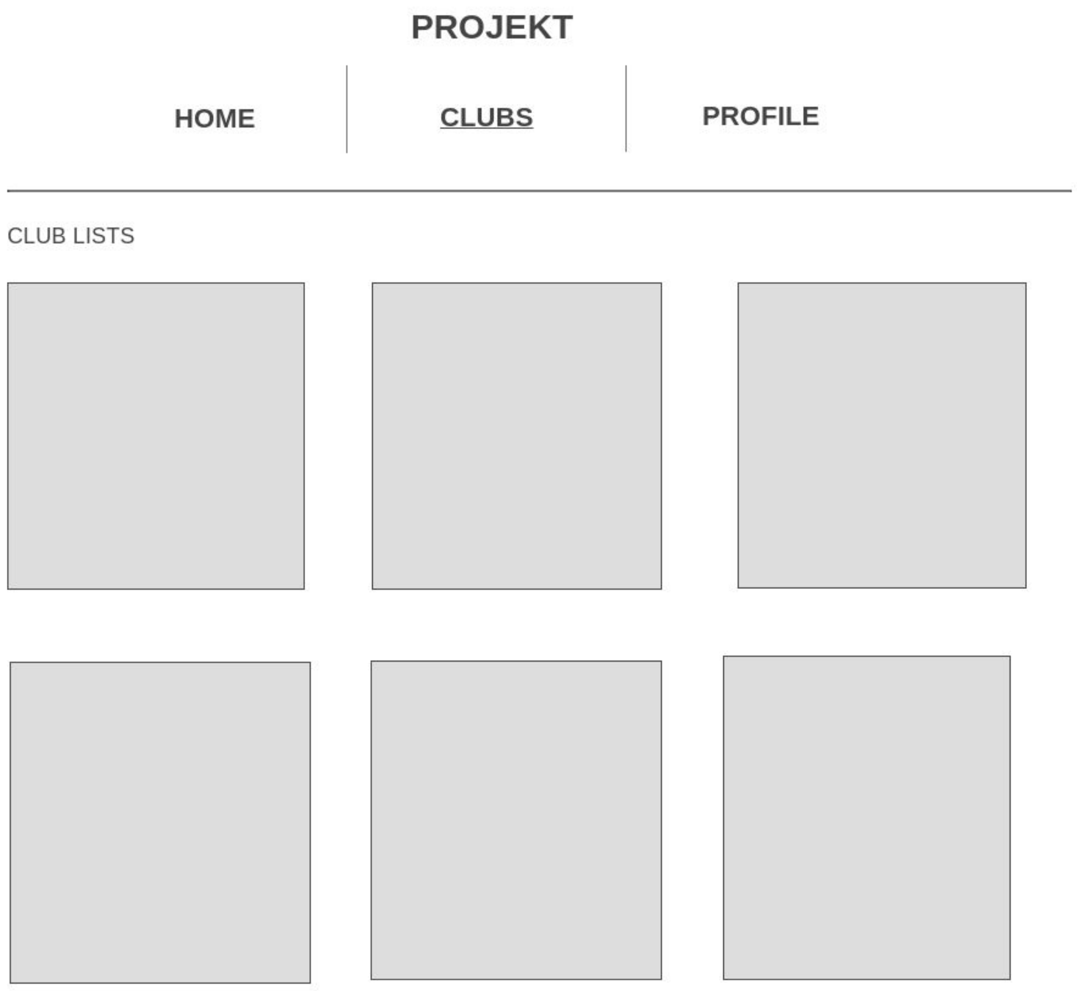
- Project Chat
- Project Drive
- Project To-Do list
- Announcements bar

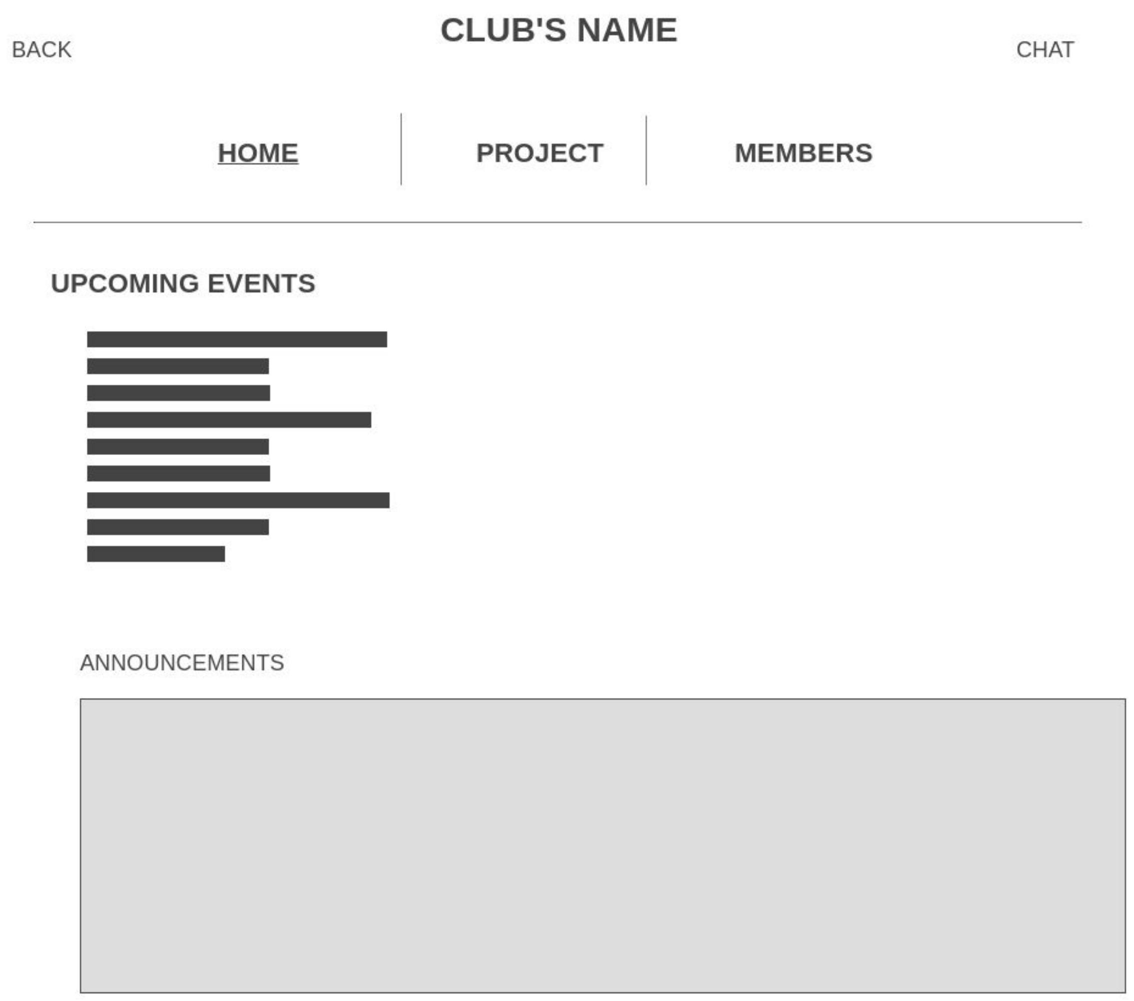
# 4. INTERFACE REQUIREMENTS AND USER INTERFACES

The project uses the Django framework in python for the backend, with React.js for frontend development and Postgresql for the database requirements. The UI must be simple and

USER INTERFACE SKETCHES :


PROJEKT
NOTIFICATION
HOME
CLUBS
PROFILE
CALENDER OF EVENTS
ANNOUNCMENTS


PROJEKT
HOME
CLUBS
PROFILE
CLUB LISTS


BACK
CLUB'S NAME
CHAT
HOME
PROJECT
MEMBERS
UPCOMING EVENTS
ANNOUNCEMENTS

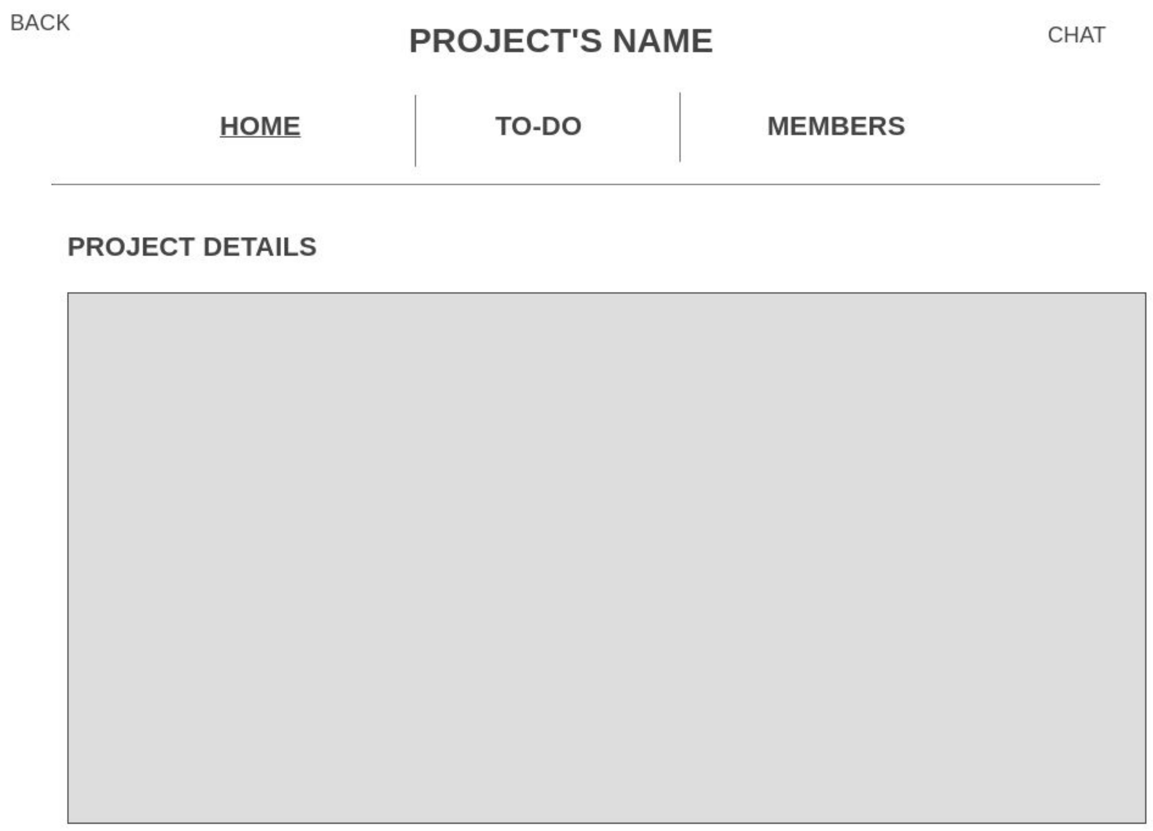
BACK

## CLUB'S NAME

CHAT

**HOME** | **PROJECT** | **MEMBERS**

*ONGOING PROJECTS*


BACK
PROJECT'S NAME
CHAT
HOME
TO-DO
MEMBERS
PROJECT DETAILS

# 5.PERFORMACE REQUIREMENTS

Performance:

The product must be interactive and the delays involved must be less . In case of opening a chat,  forms, links or popping error messages and saving the settings or sessions, the front end can have vanilla Javascript to improve performance and allow dynamic updates in the browser itself to reduce load on  servers. In case of opening databases, there must be minimum delays and the operation is performed.Hosting must be on a service such that when connecting to the server the delay the distance of the 2 systems is minimized, thus minimizing the delay.

Using the requisite backend and frontend provides fast and reliable performance, with a  good load time (depending on distance to hosting server). Since we are using the ginger library of python in Django, the pages are created and updated dynamically, helping with the performance rather than fetching the entire page again from the server machine. This helps in making the site faster along with reducing the load on the server.

Reliability:

As the product provides the right tools for project management, such as chats, to-dos and so on, it must be made sure that the system is reliable in its operations and for securing the sensitive details.

Availability:

If the internet service gets disrupted while sending information to the server, the information can be sent again for verification.

# 6. OTHER NON-FUNCTIONAL ATTRIBUTES

Security:

The main security concern is for users' accounts, hence proper login mechanism should be used to avoid hacking. Security is also provided against unauthorized use by a particular account, such as a club member cannot have the same privileges as a club Admin. Information transmission should be securely transmitted to server without any changes in information.Using the Robust Django framework, which provides security for CORS requests with stability in the website, coupled with postgresql, the backend framework and database provide a safe, secure and reliable user interface.

Scalability:

Scaling is essential for the product as due to the dynamic nature of college clubs and college students, the database, backend and frontend must scale to match the increase in data. The Tech stack used is adequate for this requirement

Usability:

As the system is easy to handle, intuitive and navigates in the most expected way with no delays. The product seamlessly traverses between the different states of the site

# 7. Operational scenarios

There will be different types of user access.

1.) App admin.
2.) User:-
    a.) Club head
    b.) Club rep
    c.) Project head
    d.) Member

New clubs can only be created by the Admin who can appoint users as the club heads.

1) Club heads are the only ones who can modify the major aspects of the club and accept the new recruits and accept/create new projects. They also create the rules of the club and update the to-do lists

2) Club reps can screen the new requests and forward them to the head, request new projects and head projects.

3) Project head is appointed to the one who created the project and is the one who can accept the members who want to join in. They also maintain the to-do lists.

-> Upon entering the website the user will be prompted to login, according to the credentials they will be given their access specific options.

-> Home page would be the same for everyone.Home page would be the same for everyone.

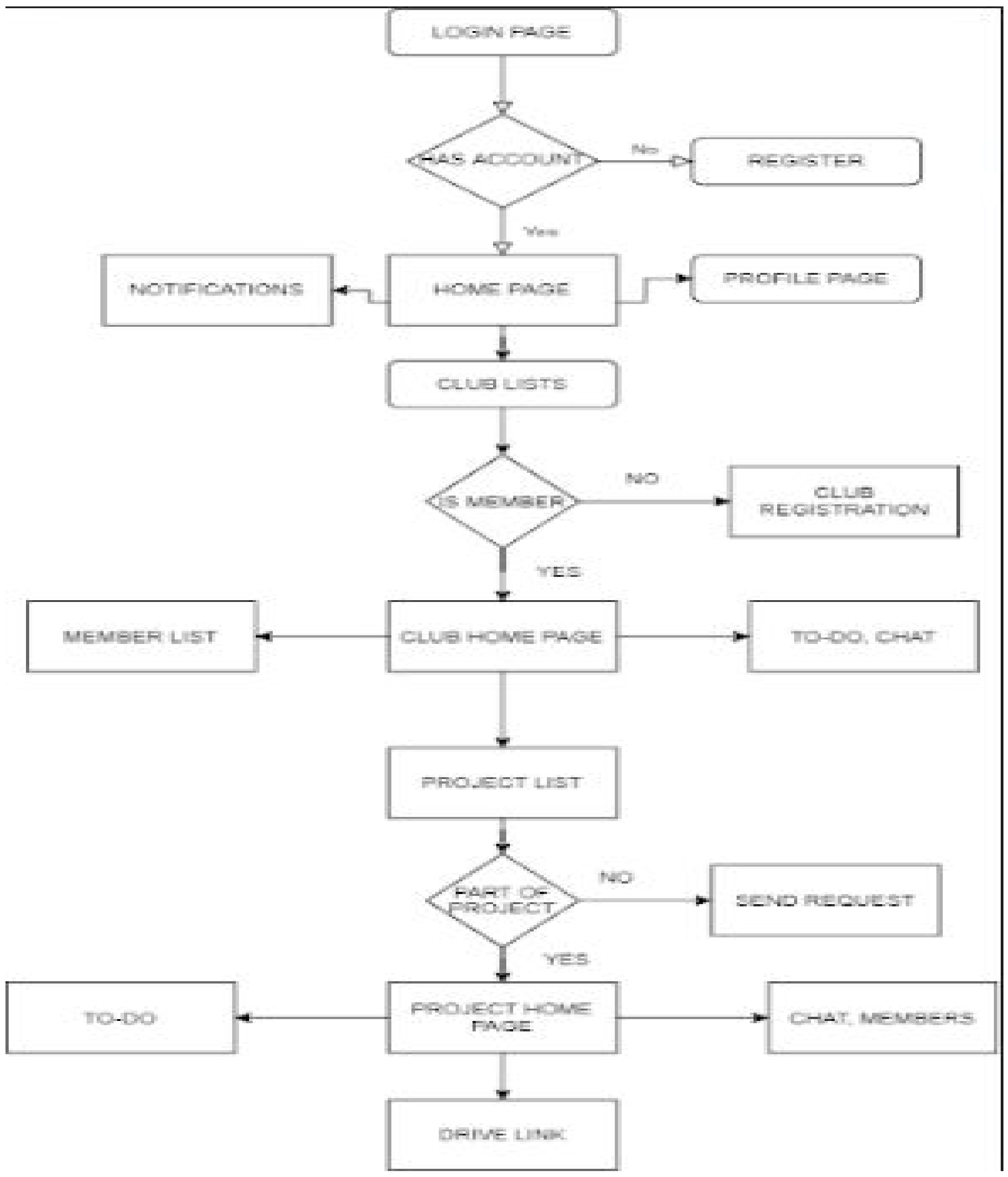
->.In the club list option- admins would be able to add clubs, everyone else would have the same view. Clicking on a club tile, would expand the tile showing a brief description of the club. Clicking it again will take you to the club page if the user is already a part of the club, else it gives the option of sending a request to join the club.

-> Upon entering the club page, the users will again be given different rights with the club head being able to create new projects.and update the to-do list,make announcements and accept new requests.

->Changing to the project list, the club heads have the ability to create new projects/accept new projects and the club reps can request for new projects.A club member can request to take part in the project.Clicking on a project should take you to the project home page if already a part of it.

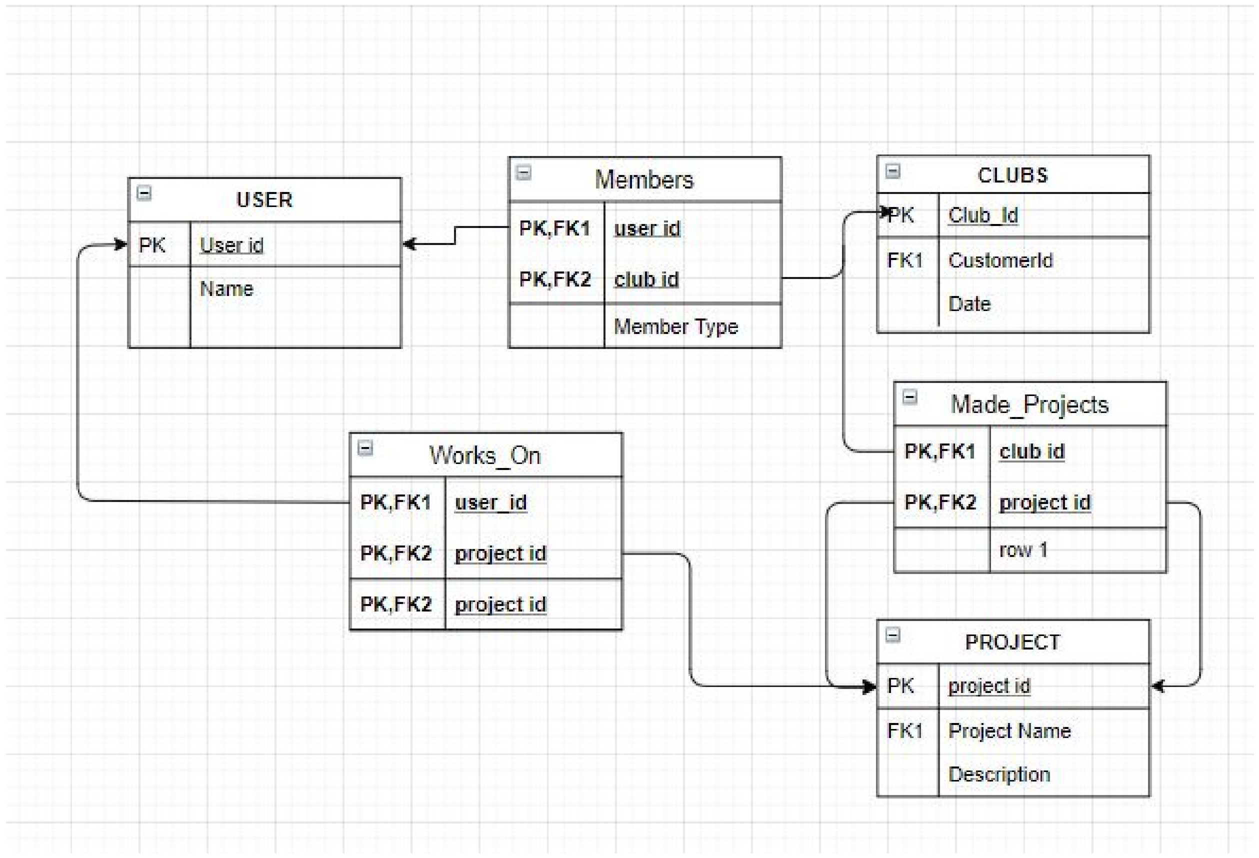
->In the project home page, the project reps that head the projects can assign work to the members and accept the changes while controlling the chat. Members can report back if they have completed their part.

## 8. SEQUENCE DIAGRAM



## 9. IMPLEMENTATION CODE

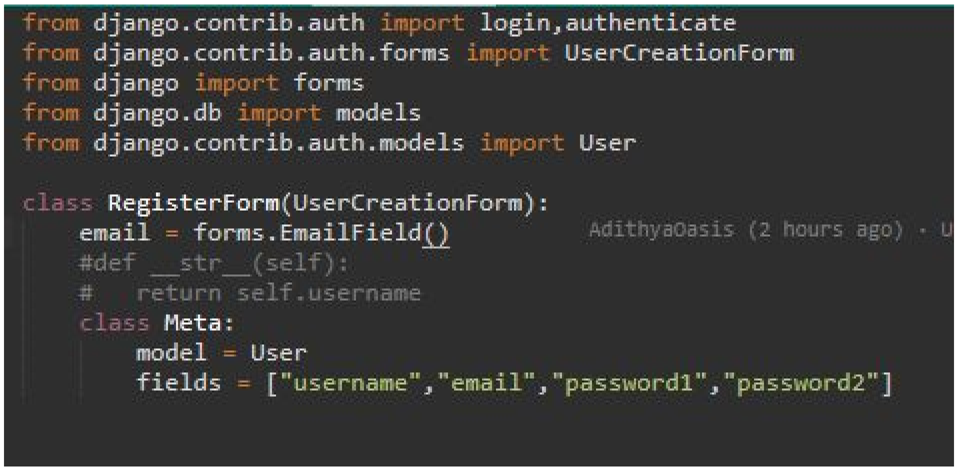
## BASIC SCHEMA:-



## APPS:-

### Home/Register

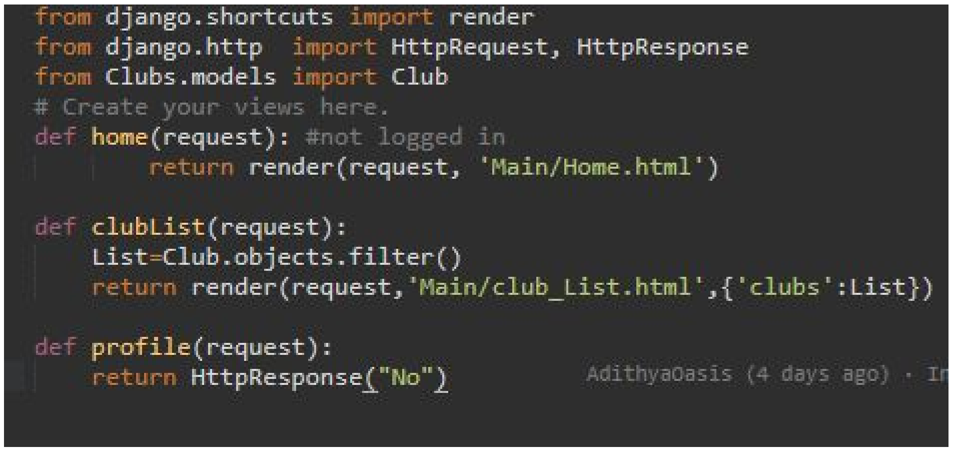
### MODELS:-



```python
from django.contrib.auth import login,authenticate
from django.contrib.auth.forms import UserCreationForm
from django import forms
from django.db import models
from django.contrib.auth.models import User

class RegisterForm(UserCreationForm):
    email = forms.EmailField()        AdithyaOasis (2 hours ago) • U
    #def __str__(self):
    #   return self.username
    class Meta:
        model = User
        fields = ["username","email","password1","password2"]
```

We use the django authentication system to register and authenticate users. Here we create RegisterForm which is a subclass of UserCreationForm that comes preinstalled with django. We do this so that we can add extra attributes to the “User” relation that isn't present with the default one.

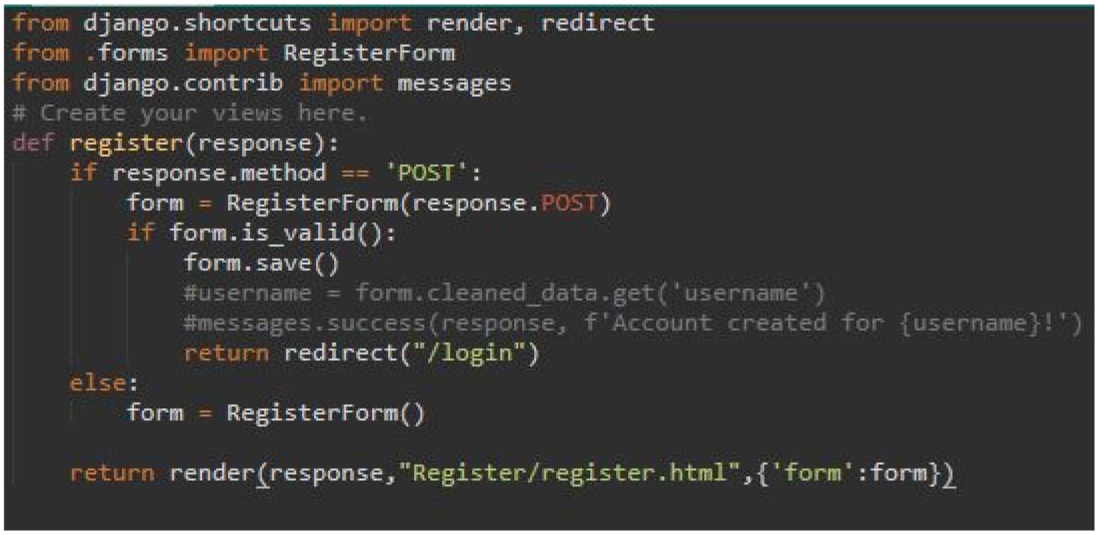
VIEWS:-



```
from django.shortcuts import render
from django.http  import HttpRequest, HttpResponse
from Clubs.models import Club
# Create your views here.
def home(request): #not logged in
        return render(request, 'Main/Home.html')

def clubList(request):
    List=Club.objects.filter()
    return render(request,'Main/club_List.html',{'clubs':List})

def profile(request):
    return HttpResponse("No")         AdithyaOasis (4 days ago) · In
```

We open with the Home page that shows the information about upcoming events,etc.

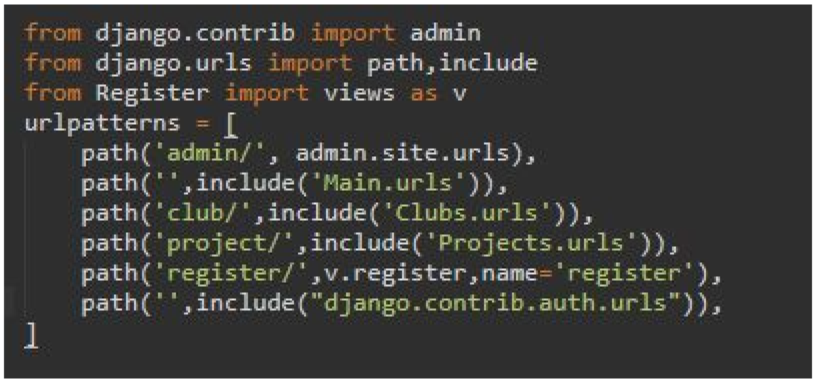
Using the nav bar we can shift to a clublist view that gives the entire list of Clubs from the same college.In Club List view, we info about the club that we hover on. Once clicked it will redirect to the club home page if a member.



```
from django.shortcuts import render, redirect
from .forms import RegisterForm
from django.contrib import messages
# Create your views here.
def register(response):
    if response.method == 'POST':
        form = RegisterForm(response.POST)
        if form.is_valid():
            form.save()
            #username = form.cleaned_data.get('username')
            #messages.success(response, f'Account created for {username}!')
            return redirect("/login")
    else:
        form = RegisterForm()

    return render(response,"Register/register.html",{'form':form})
```

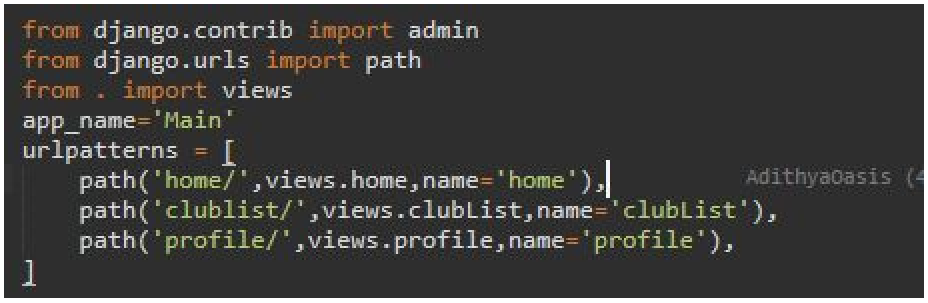
For the Register view we use the help of the django preinstalled middlewares to allow registration of new accounts., and the same for logging in the user.

## URLS:-



```python
from django.contrib import admin
from django.urls import path,include
from Register import views as v
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('Main.urls')),
    path('club/',include('Clubs.urls')),
    path('project/',include('Projects.urls')),
    path('register/',v.register,name='register'),
    path('',include("django.contrib.auth.urls")),
]
```

The Main urls.py file of the project reroutes to the Main,Clubs,Projects and Register apps accordingly. Main app contains the home page. With the url, django.contrib.auth.urls is used to check whether the User is logged in or not by using sessions. This is again a preinstalled django utility that we are using.



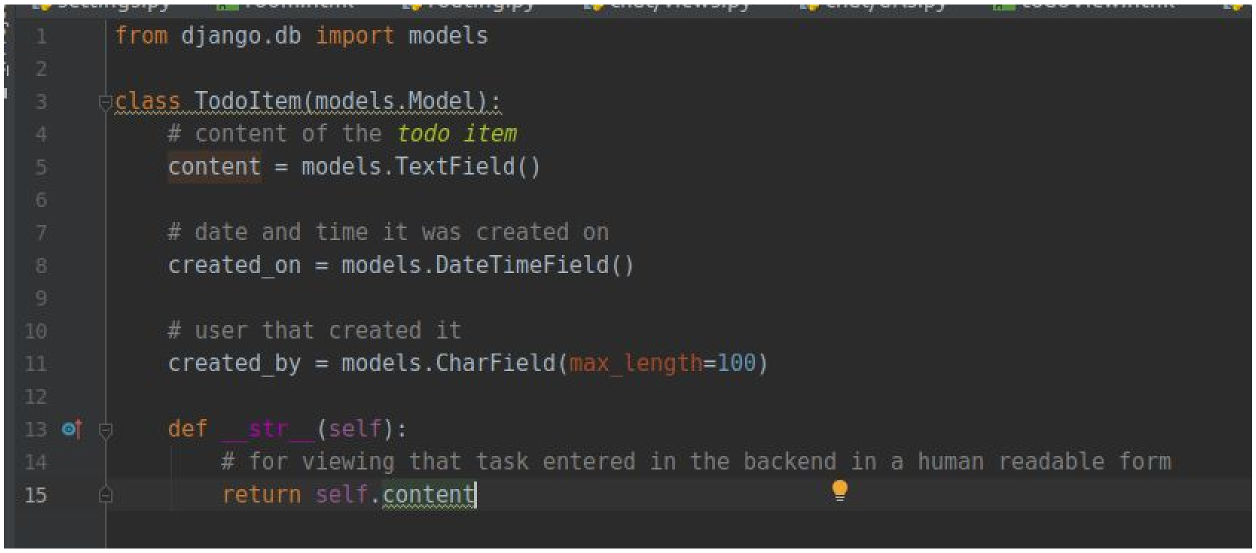
```python
from django.contrib import admin
from django.urls import path
from . import views
app_name='Main'
urlpatterns = [
    path('home/',views.home,name='home'),        AdithyaOasis (
    path('clublist/',views.clubList,name='clubList'),
    path('profile/',views.profile,name='profile'),
]
```

This is the snippet of Main/urls.py which routes to the home , The club list and the profile pages accordingly.

# To-do/Announcements

The todo app is a convenient way for club members to keep track of the tasks that need to be completed along with the tasks already completed.

## MODELS: -

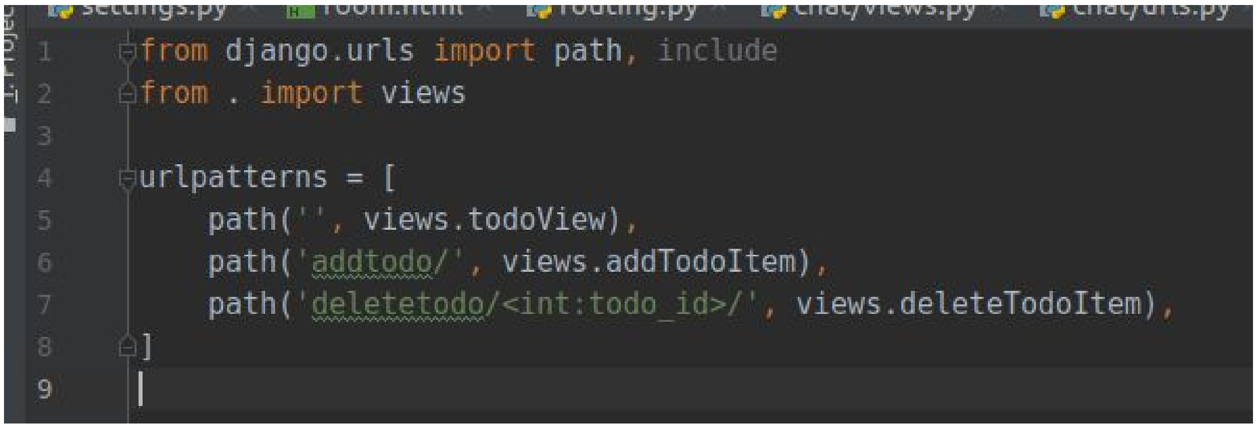


```python
from django.db import models

class TodoItem(models.Model):
    # content of the todo item
    content = models.TextField()

    # date and time it was created on
    created_on = models.DateTimeField()

    # user that created it
    created_by = models.CharField(max_length=100)

    def __str__(self):
        # for viewing that task entered in the backend in a human readable form
        return self.content
```

The implementation of the app in a database is simple, each task entry stores the task (in content), along with the time it was created on and details of the user that created it, the method __str__ is used simply for identifying a task at the backend and aids future debugging and is not used in the frontend.

## URLS: -

CODE FOR todo/urls.py



```python
from django.urls import path, include
from . import views

urlpatterns = [
    path('', views.todoView),
    path('addtodo/', views.addTodoItem),
    path('deletetodo/<int:todo_id>/', views.deleteTodoItem),
]
```

In urls, we have the paths corresponding to the urls of the todo app, all urls in this app are prefixed with todo/

Thus there are 3 routed urls :-

1. todo/

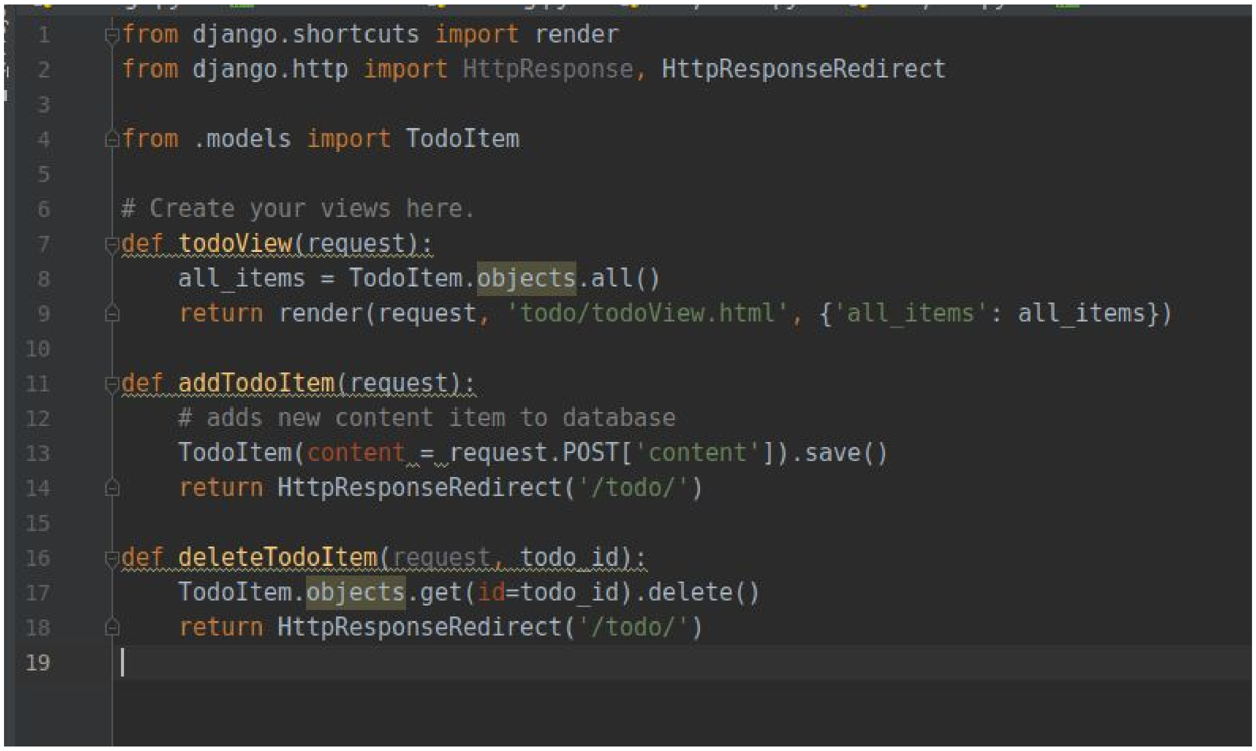
2. todo/addtodo/

3. todo/deletetodo/{{ todo_item_id }}

Where todo item id is a variable, it changes depending on the item to be deleted

## VIEWS: -

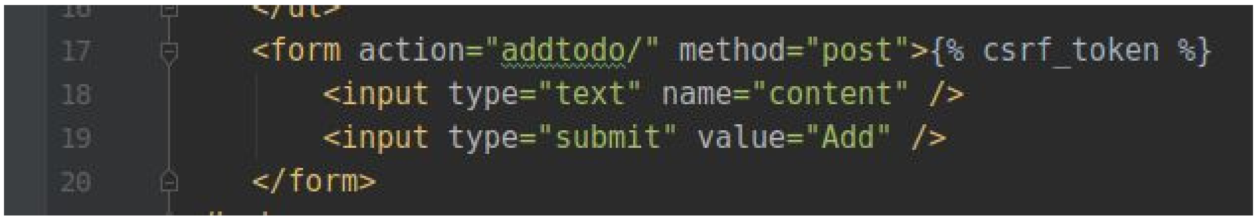
CODE FOR todo/views.py



```python
from django.shortcuts import render
from django.http import HttpResponse, HttpResponseRedirect

from .models import TodoItem

# Create your views here.
def todoView(request):
    all_items = TodoItem.objects.all()
    return render(request, 'todo/todoView.html', {'all_items': all_items})

def addTodoItem(request):
    # adds new content item to database
    TodoItem(content = request.POST['content']).save()
    return HttpResponseRedirect('/todo/')

def deleteTodoItem(request, todo_id):
    TodoItem.objects.get(id=todo_id).delete()
    return HttpResponseRedirect('/todo/')
```

The views of the todo app are used to display, add an item and delete an item in the todo list.

Adding and deleting an item happens with a post request made to the backend, to add an item the content of the item to be added is returned as given in the code snippet below



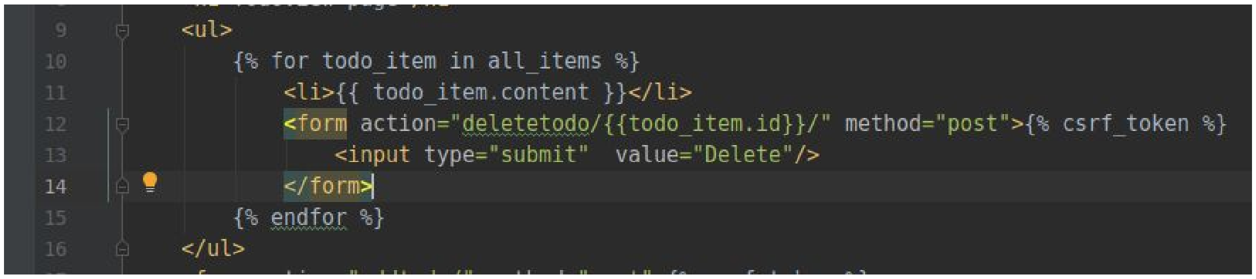
```html
    <form action="addtodo/" method="post">{% csrf_token %}
        <input type="text" name="content" />
        <input type="submit" value="Add" />
    </form>
```

As you can see in the 'action' attribute, the url the form is posting to todo/addtodo/, which in urls.py is routed to the addTodoItem view. addtodo view returns the content to the database and then redirects to todo/ url, the corresponding view - todoView, then lists all the items in chronological order of entry to the database as given in the code snippet below:

Thus it gives the effect of an item being added to the TodoList.

Deletion action happens in a similar manner, a POST request is made to the deletetodo/<todoItem id> url, where todoItem id represents the unique id of the todoItem in the datbase, the view that it url routs to is the

deleteTodoItem view, this view then deletes the task from the database and again redirects to todo/, where the are listed with the modified database as is given in the final code snippet below

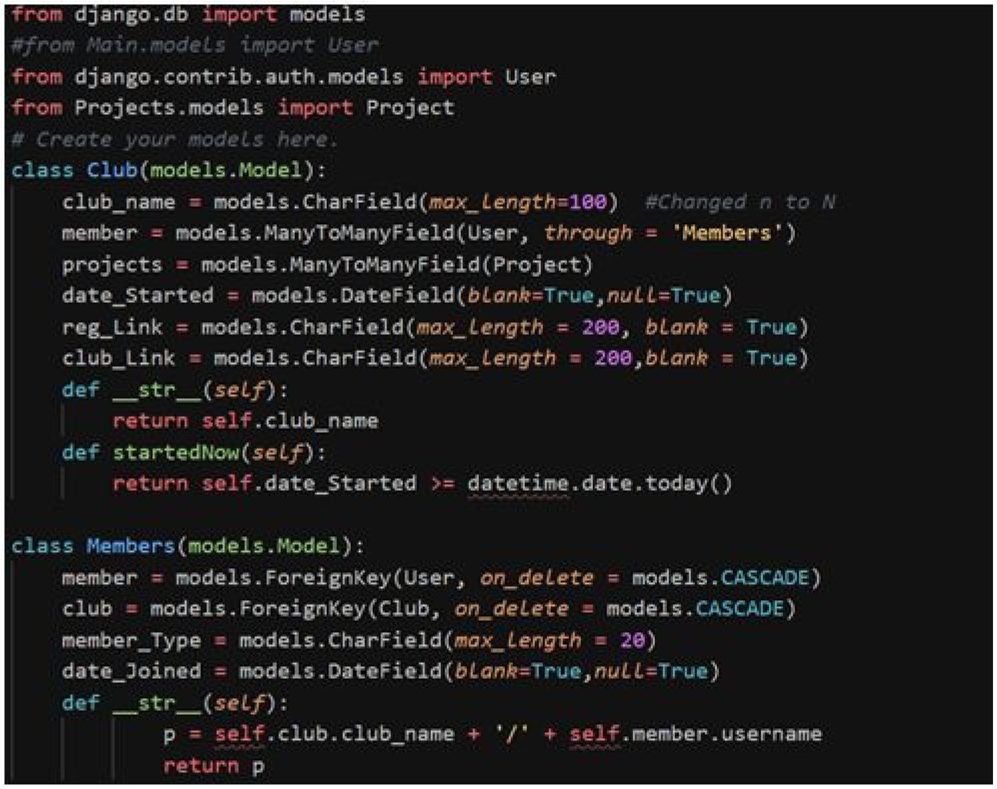


```
<ul>
    {% for todo_item in all_items %}
        <li>{{ todo_item.content }}</li>
        <form action="deletetodo/{{todo_item.id}}/" method="post">{% csrf_token %}
            <input type="submit"  value="Delete"/>
        </form>
    {% endfor %}
</ul>
```

Thus all these together tie up the functionality of the todo app and give a smooth, minimalist design to keep all tasks of the club in one place to stay organise

## CLUBS

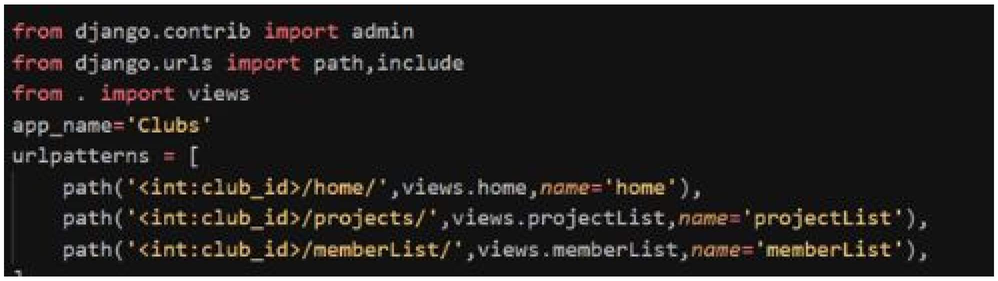
MODELS:-



```python
from django.db import models
#from Main.models import User
from django.contrib.auth.models import User
from Projects.models import Project
# Create your models here.
class Club(models.Model):
    club_name = models.CharField(max_length=100)  #Changed n to N
    member = models.ManyToManyField(User, through = 'Members')
    projects = models.ManyToManyField(Project)
    date_Started = models.DateField(blank=True,null=True)
    reg_Link = models.CharField(max_length = 200, blank = True)
    club_Link = models.CharField(max_length = 200,blank = True)
    def __str__(self):
        return self.club_name
    def startedNow(self):
        return self.date_Started >= datetime.date.today()

class Members(models.Model):
    member = models.ForeignKey(User, on_delete = models.CASCADE)
    club = models.ForeignKey(Club, on_delete = models.CASCADE)
    member_Type = models.CharField(max_length = 20)
    date_Joined = models.DateField(blank=True,null=True)
    def __str__(self):
            p = self.club.club_name + '/' + self.member.username
            return p
```

Club Model has many to many relation with User and Projects.
We use Members model as a proxy model for information on each instance of the Club-User relations..

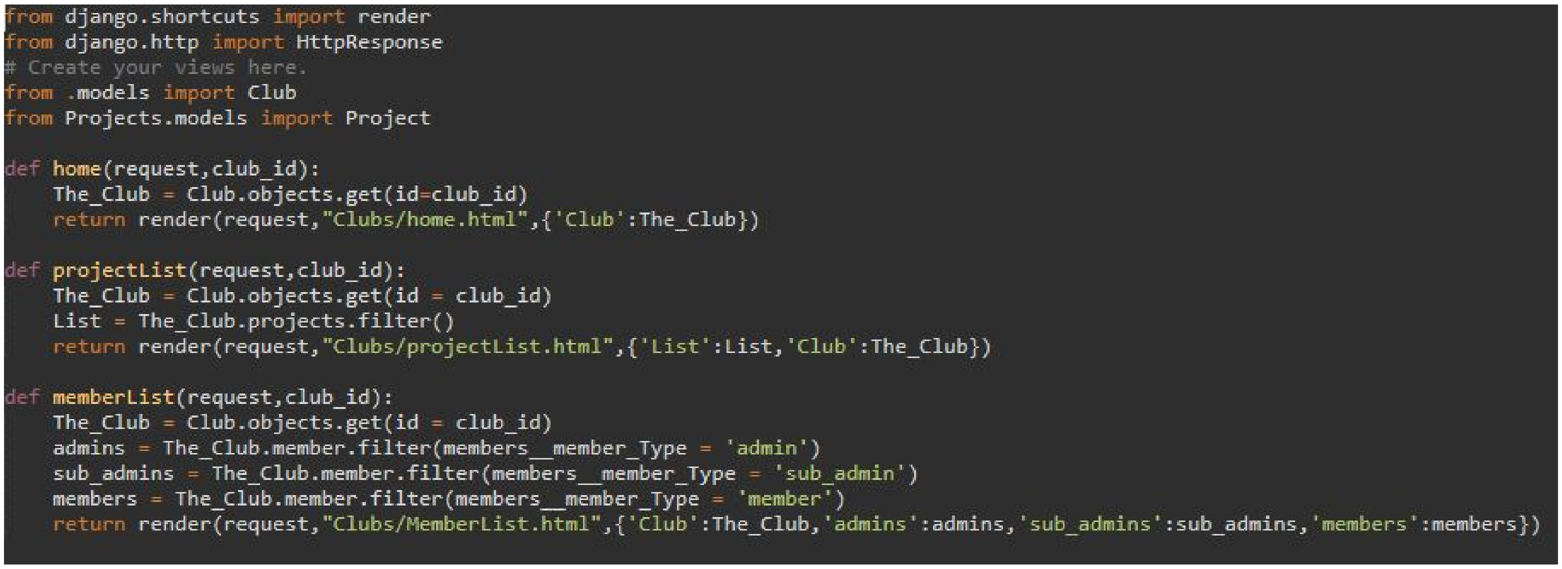
## URLS:-



```python
from django.contrib import admin
from django.urls import path,include
from . import views
app_name='Clubs'
urlpatterns = [
    path('<int:club_id>/home/',views.home,name='home'),
    path('<int:club_id>/projects/',views.projectList,name='projectList'),
    path('<int:club_id>/memberList/',views.memberList,name='memberList'),
```

1) urls.py
Urls.py works as:-
1) Django loads python module & looks for the variable urlpattern.
2) Django runs through URL pattern & stops at the first one that matches the requested url & path info
3) once one of the patterns matches django imports & calls the view which is a python function or a class based view
So ,a request to /1234(any club ID)/home/ would match the first pattern in the list
A request to /1234/projects/ would match the second pattern in the list
/1234/ wouldn’t match & give an error.

## VIEWS-



```python
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
from .models import Club
from Projects.models import Project

def home(request,club_id):
    The_Club = Club.objects.get(id=club_id)
    return render(request,"Clubs/home.html",{'Club':The_Club})

def projectList(request,club_id):
    The_Club = Club.objects.get(id = club_id)
    List = The_Club.projects.filter()
    return render(request,"Clubs/projectList.html",{'List':List,'Club':The_Club})

def memberList(request,club_id):
    The_Club = Club.objects.get(id = club_id)
    admins = The_Club.member.filter(members__member_Type = 'admin')
    sub_admins = The_Club.member.filter(members__member_Type = 'sub_admin')
    members = The_Club.member.filter(members__member_Type = 'member')
    return render(request,"Clubs/MemberList.html",{'Club':The_Club,'admins':admins,'sub_admins':sub_admins,'members':members})
```

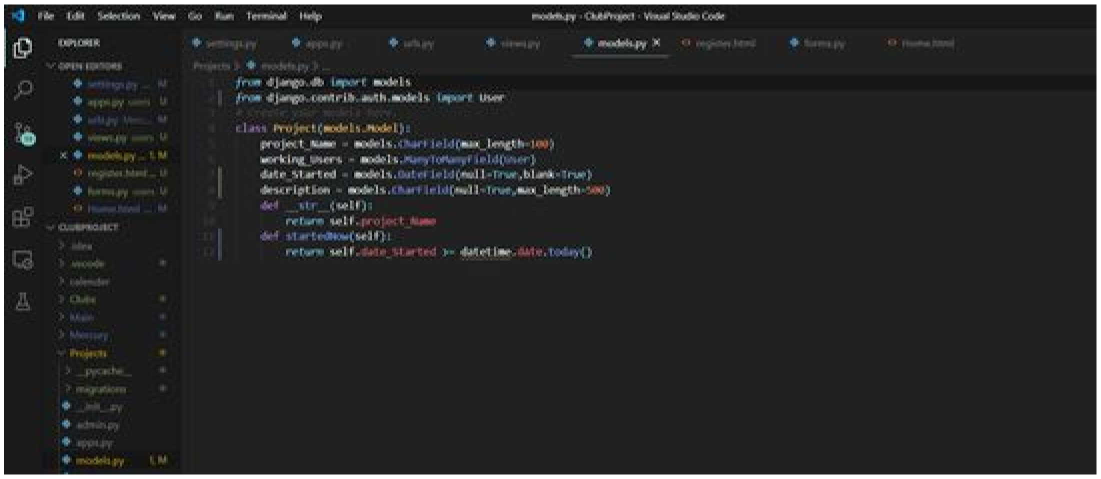
Entering the Clubs app we first start with the Home view that shows the club specific announcement and to-do lists. There will also be group chat feature for the particular clubs.

We then move to the project list where normal members can view the clubs current projects while the admins can create new projects appointing a subadmin as its leader.

Finally we have the Member list giving the information about all the club members and their roles.

## PROJECTS

Models.py:-

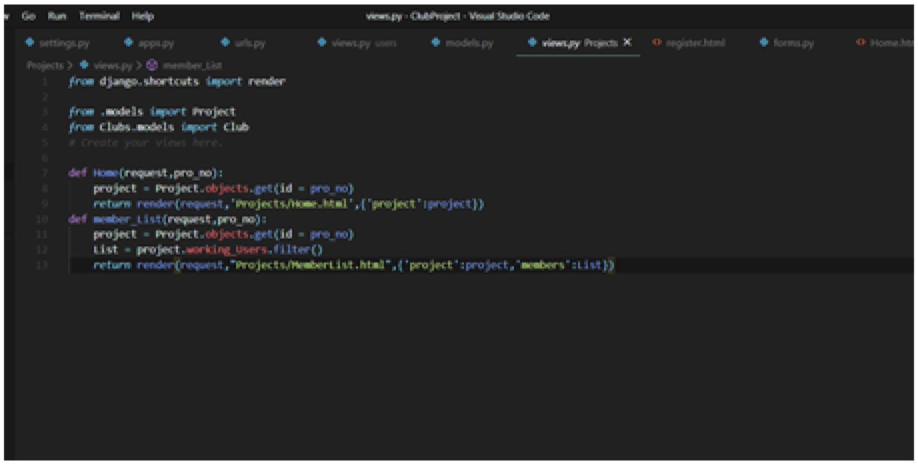


```python
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class Project(models.Model):
    project_Name = models.CharField(max_length=100)
    working_Users = models.ManyToManyField(User)
    date_Started = models.DateField(null=True,blank=True)
    description = models.CharField(null=True,max_length=500)
    def __str__(self):
        return self.project_Name
    def startedNow(self):
        return self.date_Started >= datetime.date.today()
```

In many to many representing the relationship between these two lists of entities is as simple as making two tables and creating linked record fields. Here we have a many to many relationship between Users and Projects relations.

We also have two functions, one that returns the project name, is used specifically for debugging in the backend and another which lets us know when a project has started.

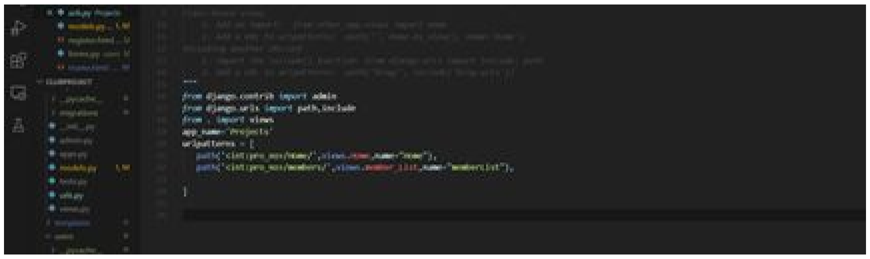
## Views.py :-



```python
from django.shortcuts import render

from .models import Project
from Clubs.models import Club
# Create your views here.

def Home(request,pro_no):
    project = Project.objects.get(id = pro_no)
    return render(request,'Projects/Home.html',{'project':project})
def member_List(request,pro_no):
    project = Project.objects.get(id = pro_no)
    List = project.working_Users.filter()
    return render(request,"Projects/MemberList.html",{'project':project,'members':List})
```

Views.py returns the content or moderates the content such as project no. /Name to the render website/html page. We also show the members list along with the project name and number.

## Urls.py:-



```python
'''
from django.contrib import admin
from django.urls import path,include
from . import views
app_name='Projects'
urlpatterns = [
    path('<int:pro_no>/Home/',views.Home,name="Home"),
    path('<int:pro_no>/members/',views.member_List,name="memberList"),
]
```

Urls.py gives the path to which an app opens in, here in projects urls.py we are giving the url patterns according to the project no.

## 11. UPDATED SCHEDULE

We will do our project in 5 stages,& in a time limit of 5 weeks.

Stage 1:-

We make the login/registration page complete with backend using postgresql database & django

Stage 2:-

We make the backend for our site using django & postgre sql

Stage 3:-

We make the front end part of our site using javascript ,react js,node js & html

Stage 4:-

We integrate the front end & back end & debug it.

Stage 5:-

We optimize our site & see if we can modify anything to improve the overall project

We would like to take approximately 1 week per stage

# 12. APPENDICES

## REFERENCES

DJANGO DOCUMENTATION -

https://docs.djangoproject.com/en/3.0/

POSTGRESQL DOCUMENTATION -

https://www.postgresql.org/docs/

REACT.js DOCUMENTATION -

https://reactjs.org/docs/getting-started.html