2CSDE86
Application Development Framework

Innovative Assignment
DevTalks

Submitted By,

18BCE164 - Meet Patel

18BCE166 - Poojan Patel

18BCE171 - Shubh Patel

Report Content

Features

DevTalks is a Question-Answer website similar to stackoverflow for professional and enthusiast programmers. Students/developers can post their questions with appropriate tags and answer others' questions. This helps to build a community of experienced developers and students where everybody can learn from each other.

Major features of DevTalks are as follows:

SignUp/SignIn with email verification

Users can create an account. On SignUp, the system will send an email to the user's email id. When a user clicks on the verification link in the email, he/she will be redirected to the profile page.

Update user profile details

User model contains various fields such as first name, last name, middle name, profile picture, bio, profession, birthdate, age, gender, contact no., pincode, city, country and skills. Any of these fields can be updated by the user.

Forgot password

An email containing a random password of length 8 will be sent to the user. A user can use that password to login.

Performance based badges for the users

Users will be ranked according to their number of accepted answers. Top 10% users will get a GOLD badge, Top 11-25% will get a SILVER badge and Top 26-50% will get a BRONZE badge.

Ask your questions with tags, images, code etc.

Users can ask a question with the appropriate tags. Question body can contain code, images, lists, text and many more for better explanation.

Answer to others' questions

A user can add answers to the questions asked by others.

Verify the answers posted on your question - accepted answers

A user can verify/accept the answer to his/her question. These accepted answers will be highlighted separately.

Upvote answers and Like questions

A user can upvote the answers and like the questions asked by other users.

Tag based filtering of questions

Each question will have some tags. Users can filter the questions by searching for tags in the search bar.

Update your questions and answers

Permission for Edition Question / Answer

A user can update the answer or question added by him/her.

JQuery Based validation during SignUp

Every user must have a unique username and email. To enforce this constraint, we're using jQuery in the SignUp page. It will call functions from views in the user module to check whether the username or email entered by the user are available or not.

Question Ordering based on Latest Updated time, Answer Ordering based on Upvotes

In the feed, questions will be ordered by their latest update time. In the questions' details page, answers will be sorted by their upvotes.

Django Concept Used

Abstract User:

We have extended the Abstract User model to customize and use it in our model

Custom Template Tags:

We have created some template tags, like is_authorized, to check in the frontend, if logged one is the same user who have asked the question/ or answered the question

Default Auth Model:

We have used Default Auth model, for authentication purpose

Email Based Authentication:

We have used Email based Authentication when user registers/ forgets the password, a mail will be sent to registered account

Custom Models:

We have created custom models, for implementing business logic

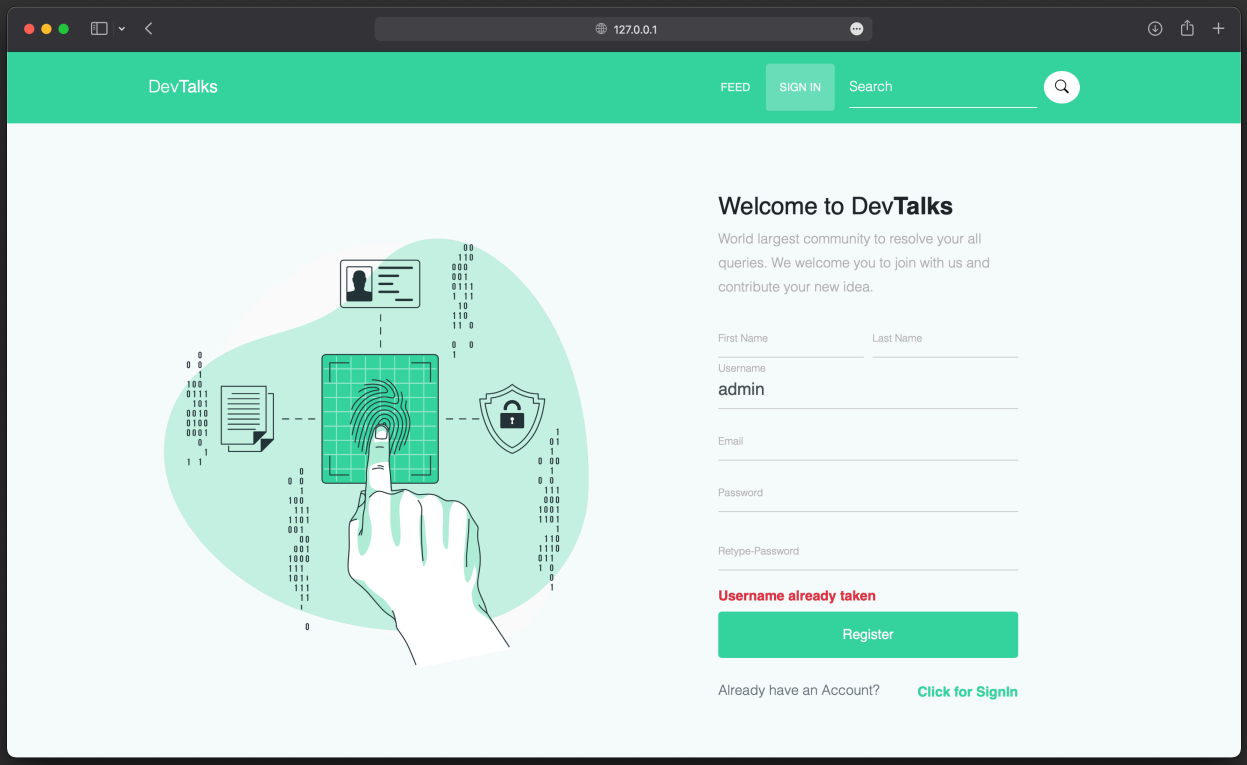
Uploading Image:

We have used media folder to store the uploaded image and show it to user whenever queried

Messages:

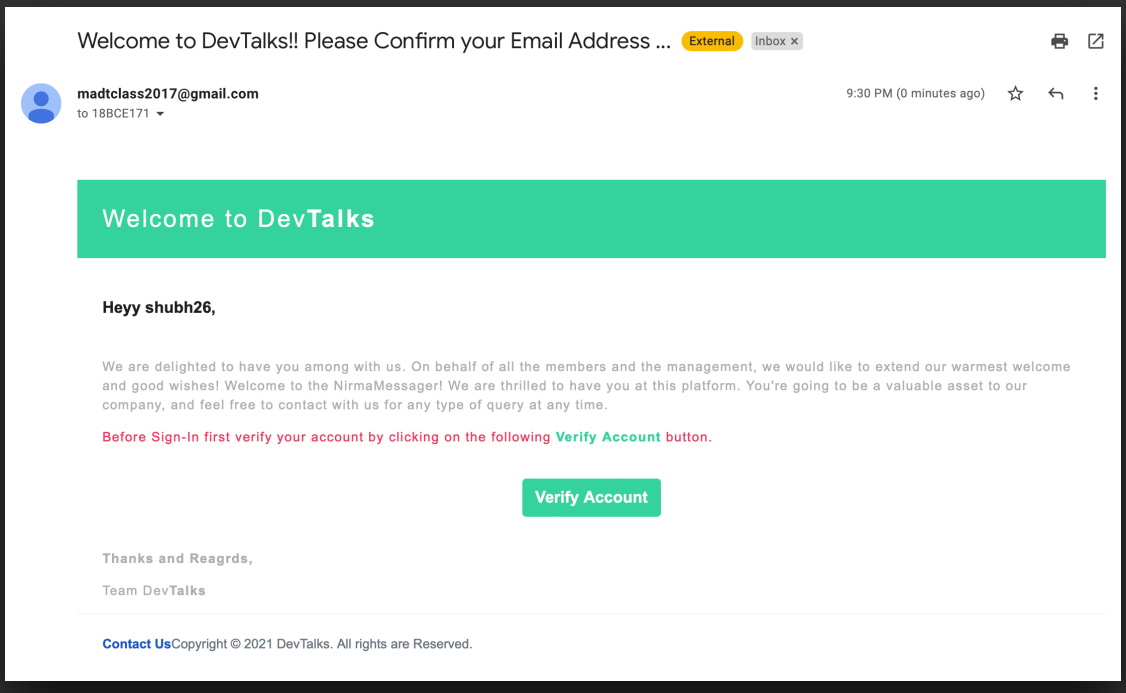
Messages are used to give a quick alert regarding error, warning, info to user from backend

User Interface Snapshots

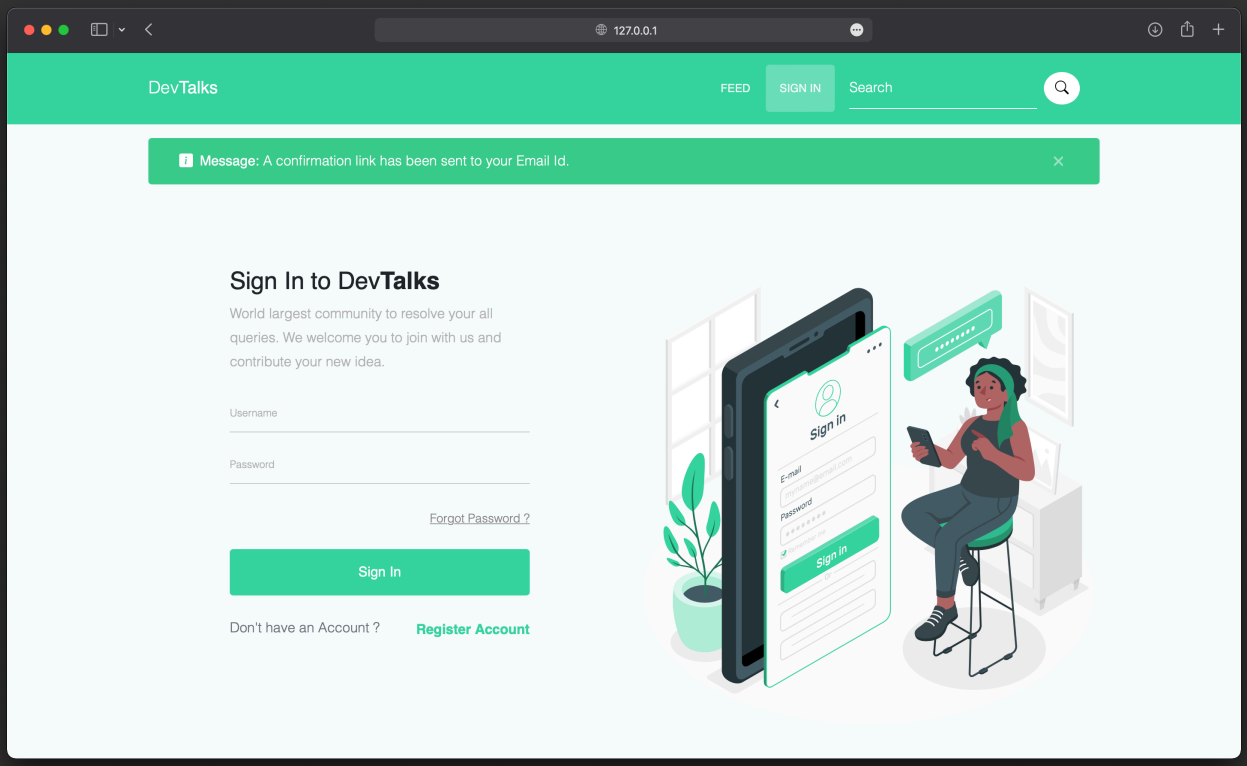


Registration Page

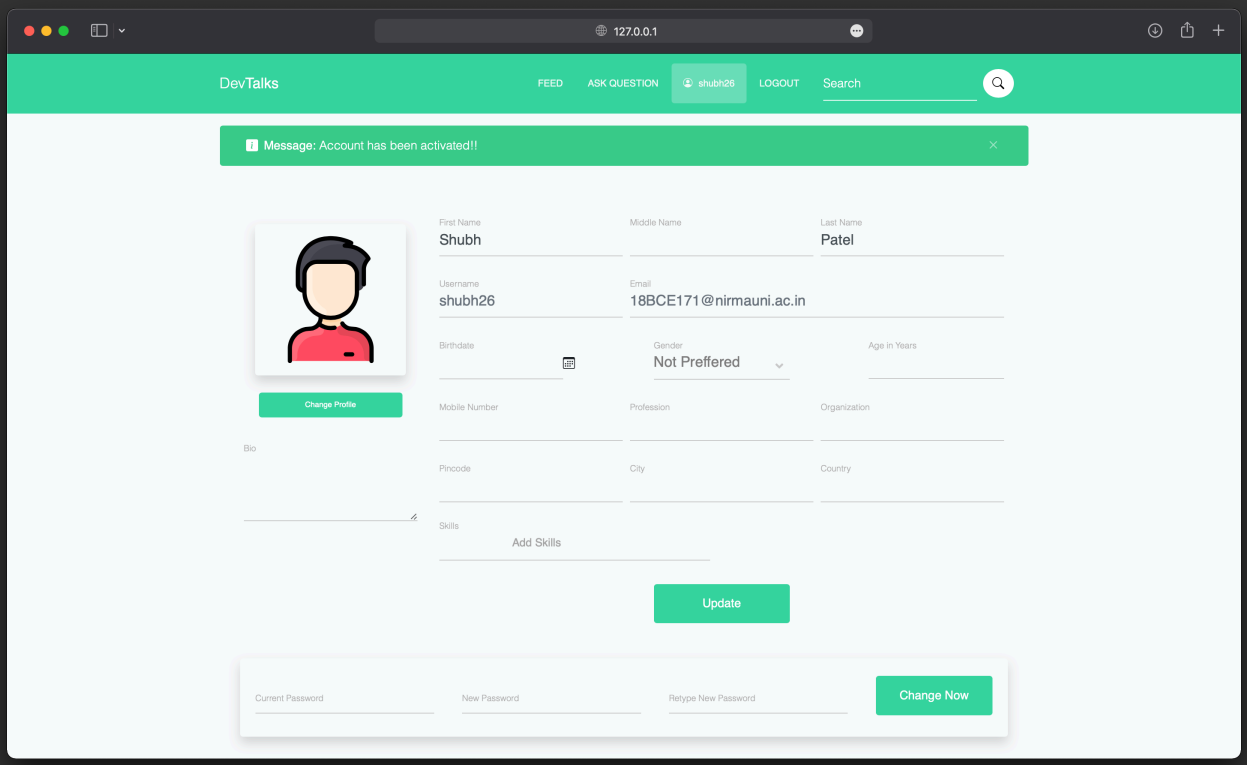
User Verification Email



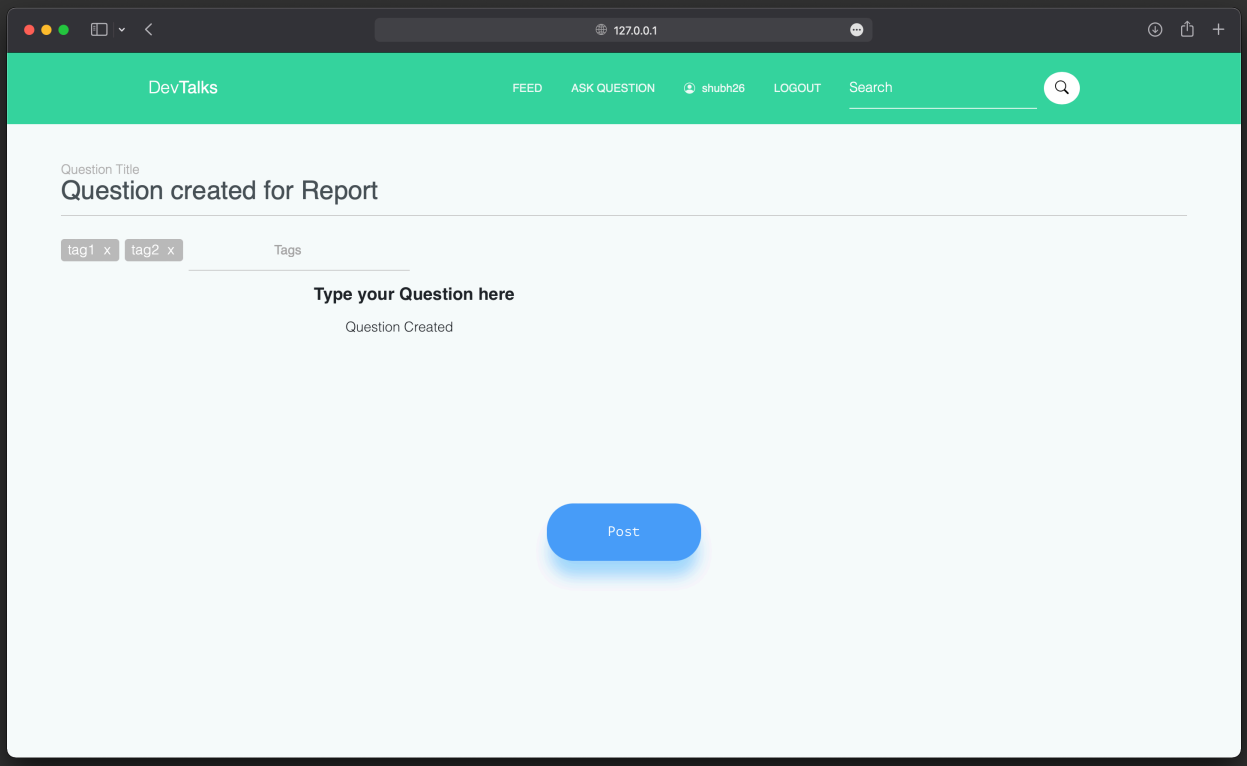
Sign In Page



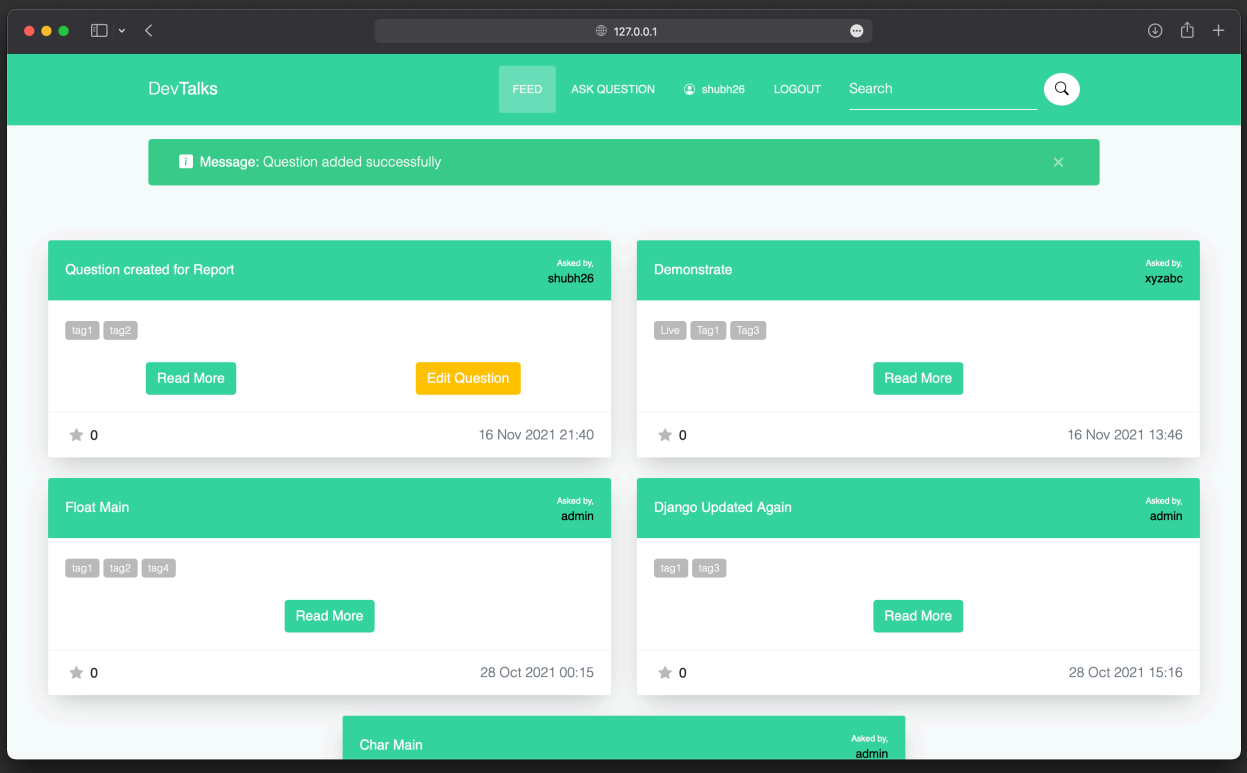
Profile Page



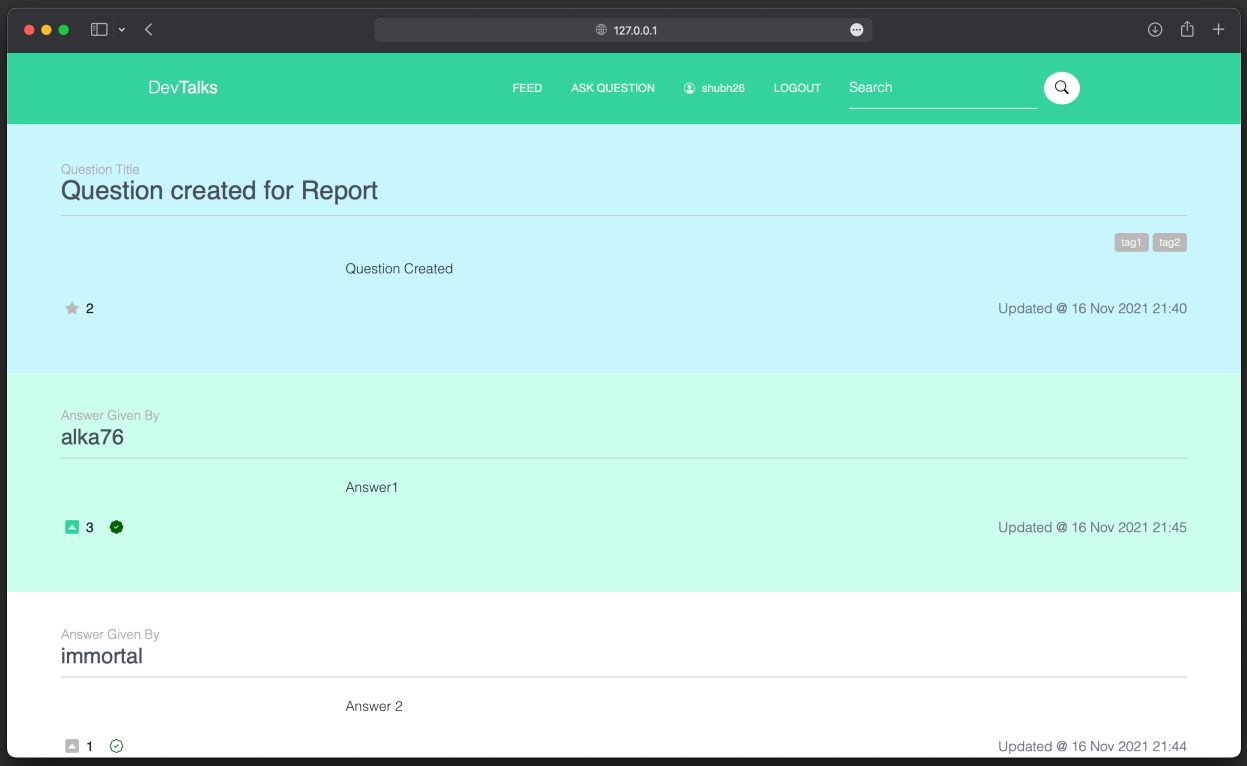
Ask Question Page



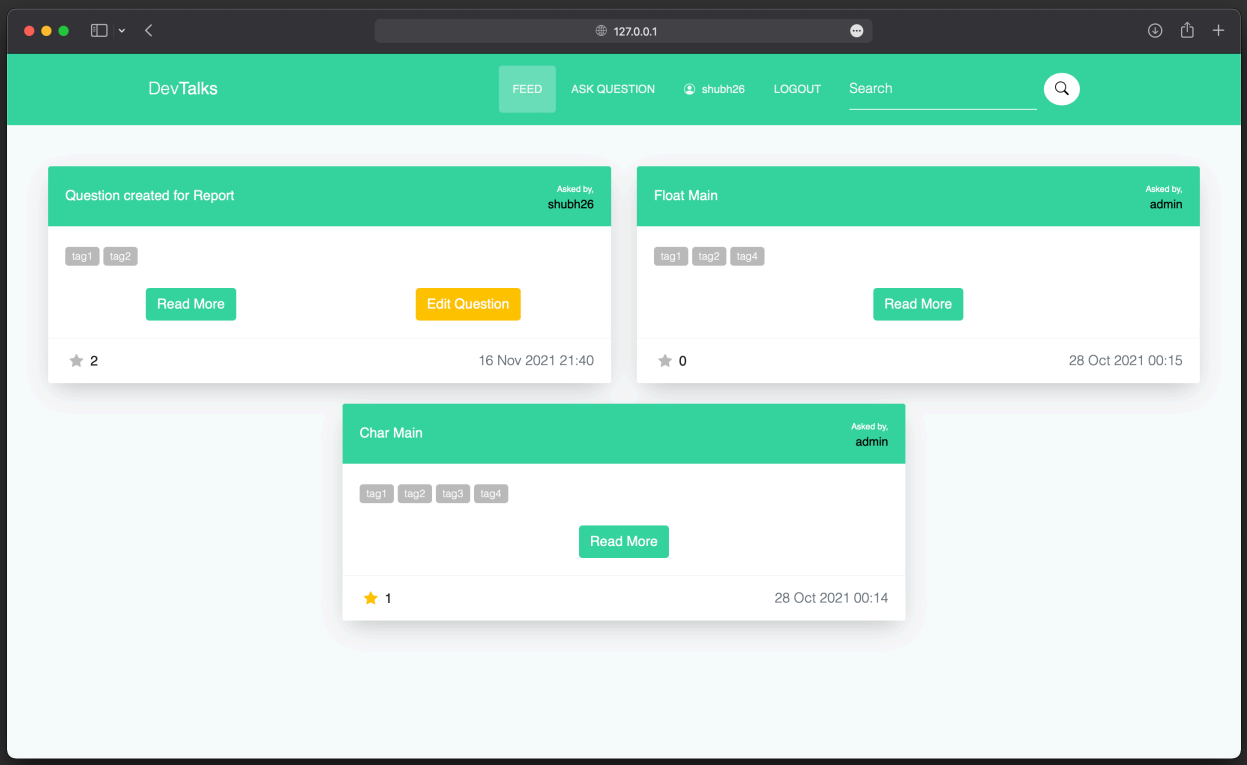
Feed Page



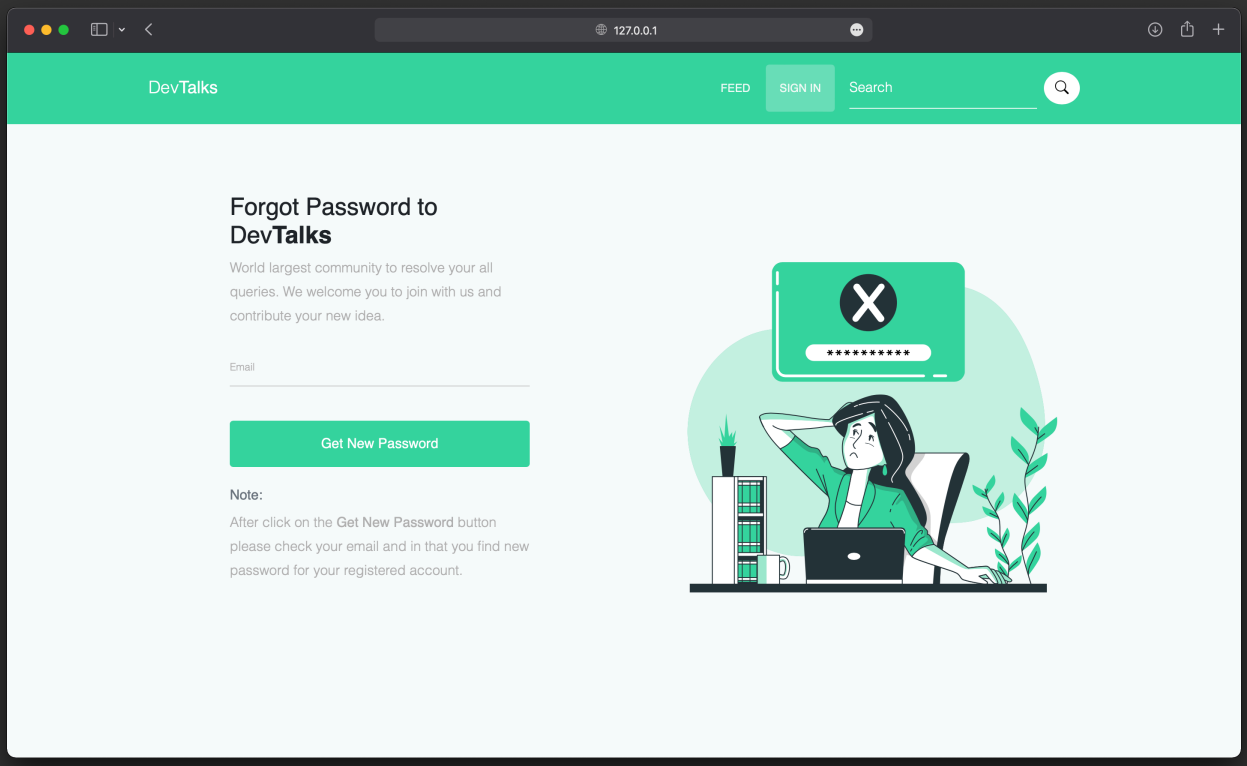
Answer Page



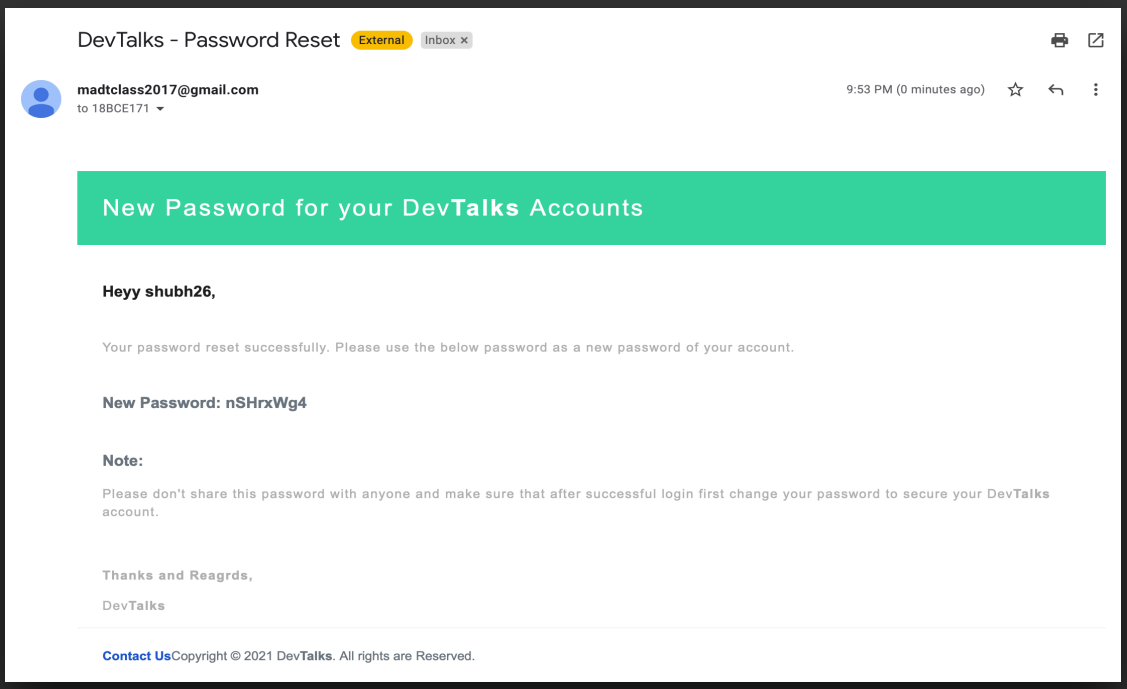
Tag Based Filtering (filter=tag1, tag2)



Forgot Password



Reset Password Email



GitHub Code

- Click Here to see the entire code on GitHub.

DevTalks App

settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'questions',
    'users',
]

AUTH_USER_MODEL = 'users.User'

ROOT_URLCONF = 'DevTalks.urls'

LOGIN_URL = 'signin'
LOGIN_REDIRECT_URL = 'home'
LOGOUT_REDIRECT_URL = 'signin'

EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '<email>'
EMAIL_HOST_PASSWORD = '<password>'

MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media/'

PROFILE_PICTURE_STORAGE = BASE_DIR / 'media/profile_pics/'
```

errorview.py

```
from django.contrib import messages
from django.shortcuts import redirect

def errorView(request):
     #messages.error(request,"Request Page not Exists")
     messages.warning(request,"Requested Page not Exists, Redirected to Home")
     #messages.info(request,"Redirected to Feed")
     return redirect('feed')
```

urls.py

```
from django.conf.urls import url
from django.contrib import admin
from django.urls import path,include
from questions import views
from DevTalks import settings
from django.conf.urls.static import static
from DevTalks.errorview import errorView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('feed/', views.get_feed, name='feed'),
    path('', views.get_feed),
    path('question/', include("questions.urls")),
    # url(r'^user.?/', include("users.urls")),
    path('user/', include("users.urls")),
]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

urlpatterns += [url(r'^.*$', errorView, name='catch-all')]
```

Users App

models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser

# Create your models here.

class User(AbstractUser):
    middle_name = models.CharField(max_length=30, null=True, blank=True)
    profile_picture = models.ImageField(default='profile_pics/default.svg',
upload_to='profile_pics')
    bio = models.CharField(max_length=255, blank=True)
    profession = models.CharField(max_length=255, blank=True)
    organization = models.CharField(max_length=255, blank=True)
    birth_date = models.DateField(blank=True, null=True)
    age = models.IntegerField(blank=True, null=True)
    gender = models.CharField(max_length=1, blank=True)
    mobile_number = models.CharField(max_length=15, null=True, blank=True)
    pincode = models.CharField(max_length=8, blank=True)
    city = models.CharField(max_length=255, blank=True)
    country = models.CharField(max_length=255, blank=True)
    skills = models.CharField(max_length=255, blank=True)
```

admin.py

```
from django.contrib import admin
from .models import User

# Register your models here.

admin.site.register(User)
```

urls.py

```
from django.urls import path
from django.conf.urls import url
from django.conf.urls.static import static
from django.conf import settings
from . import views

urlpatterns = [
    path('signup/', views.signup, name='signup'),
    path('signin/', views.signin, name='signin'),
    path('signout/', views.signout, name='signout'),
    path('profile/', views.profile, name='profile'),
```

```
    path('change-password/',views.change_password, name='change_password'),
    path('forgot-password/',views.forgot_password, name='forgot_password'),
    url(r'^activate/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-
[0-9A-Za-z]{1,40})/$',
        views.activate, name='activate'),
    path('check_username/<username>/',views.check_username,
name='check_username'),
    path('check_email/<email>/',views.check_email, name='check_email'),
    path('', views.signin),
]

# if settings.DEBUG:
#   urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

views.py

```
from typing import overload
from django.http.response import JsonResponse
from django.shortcuts import redirect, render
from django.contrib.auth import login, logout, authenticate
from django.contrib.sites.shortcuts import get_current_site
from django.contrib import messages
from django.utils.encoding import force_bytes, force_text
from django.utils.http import urlsafe_base64_encode, urlsafe_base64_decode
from django.template.loader import get_template
from django.core.mail import EmailMultiAlternatives
from django.contrib.auth.decorators import login_required
from django.contrib.auth.hashers import check_password
from django.views.decorators.csrf import csrf_exempt
from django.conf import settings
from .tokens import account_activation_token
from .models import User
from questions.models import Question, Answer
from django.db.models import Count
import json
import datetime

def signin(request):
    if(request.user.username != ""):
        messages.info(request,"Signout First inorder to Signin")
        return redirect('feed')
    if request.method == 'POST':
        username = request.POST['username'].strip()
        password = request.POST['password'].strip()

        if username == "" or username is None:
            messages.error(request,'Username must not be empty')
            return redirect('signin')

        if password == "" or password is None:
```

```
            messages.error(request,'Password must not be empty')
            return redirect('signin')

        try:
            user = User.objects.get(username=username)
        except(TypeError, ValueError, OverflowError, User.DoesNotExist):
            user = None

        if user and authenticate(request, username=username, password=password):
            login(request, user)
        else:
            messages.error(request,'Invalid username or password')
            return redirect('signin')

        return redirect('feed')

    return render(request, 'signin.html')


@csrf_exempt
def check_username(request, username):
    if request.method == 'POST':
        user_exists = False
        if username and username != "":
            user_exists = User.objects.filter(username=username).exists()
        return JsonResponse({ 'exists': user_exists })

@csrf_exempt
def check_email(request, email):
    if request.method == 'POST':
        email_exists = False
        if email and email != "":
            email_exists = User.objects.filter(email=email).exists()
        return JsonResponse({ 'exists': email_exists })

def signup(request):
    if request.method == 'POST':
        firstname = request.POST['firstname'].strip()
        lastname = request.POST['lastname'].strip()
        username = request.POST['username'].strip()
        email = request.POST['email'].strip()
        password = request.POST['password'].strip()
        retypepassword = request.POST['retypepassword'].strip();

        if User.objects.filter(username=username):
            messages.error(request, "Username already exist! Please try some
other username.")
            return redirect('signin')

        if User.objects.filter(email=email).exists():
            messages.error(request, "Email Already Registered!!")
```

```
            return redirect('signin')

        if len(username)>20:
            messages.error(request, "Username must be under 20 charcters!!")
            return redirect('signin')

        if password != retypepassword:
            messages.error(request, "Passwords didn't matched!!")
            return redirect('signin')

        if not username.isalnum():
            messages.error(request, "Username must be Alpha-Numeric!!")
            return redirect('signin')

        user = User.objects.create_user(first_name=firstname, last_name=lastname,
username=username,
                                        email=email, password=password)
        user.is_active = False

        current_site = get_current_site(request)
        email_subject = "Welcome to DevTalks!! Please Confirm your Email
Address ..."
        ctx = dict({
            'username': user.username,
            'domain': current_site.domain,
            'uid': urlsafe_base64_encode(force_bytes(user.pk)),
            'token': account_activation_token.make_token(user)
        })
        message = get_template('welcome.html').render(ctx)
        email = EmailMultiAlternatives(
            email_subject,
            message,
            settings.EMAIL_HOST_USER,
            [user.email],
        )
        email.content_subtype = 'html'
        email.fail_silently = True
        email.send()
        # Save User if Verification Mail Successfully Sent
        user.save()

        messages.success(request,'A confirmation link has been sent to your Email
Id.')
        return redirect('signin')

    return render(request, 'signin.html')


def activate(request, uidb64, token):
    try:
```

```
        uid = force_text(urlsafe_base64_decode(uidb64))
        user = User.objects.get(pk=uid)
    except(TypeError, ValueError, OverflowError, User.DoesNotExist):
        user = None

    if user is not None and account_activation_token.check_token(user, token):
        user.is_active = True
        user.save()
        login(request, user)
        messages.success(request,'Account has been activated!!')
        return render(request, "profile.html", { 'user': request.user })
    else:
        messages.error(request,'Activation link is invalid!')
        return render(request, 'signin.html')

@login_required(login_url='signin')
def signout(request):
    logout(request)
    return redirect('signin')

@login_required(login_url='signin')
def profile(request):
    if request.method == 'POST':
        firstname = request.POST['firstname'].strip()
        middlename = request.POST['middlename'].strip()
        lastname = request.POST['lastname'].strip()
        username = request.POST['username'].strip()
        email = request.POST['email'].strip()
        birthdate = request.POST['birthdate'].strip()
        try :
            age = int(request.POST['age'].strip().split()[0])
        except:
            age = 0
        bio = request.POST['bio'].strip()
        profession = request.POST['profession'].strip()
        organization = request.POST['organization'].strip()
        gender = request.POST['gender'].strip()
        mobilenumber = request.POST['mobilenumber'].strip()
        pincode = request.POST['pincode'].strip()
        city = request.POST['city'].strip()
        country = request.POST['country'].strip()
        skills = json.dumps(request.POST['skills'])
        # print(firstname, middlename, lastname, username, email, birthdate, age,
bio, profession, organization, gender, mobilenumber, pincode, city, country,
skills)

        current_user = request.user

        if current_user.id is not None:
            try:
                user = User.objects.get(pk=current_user.id)
```

```
            except(TypeError, ValueError, OverflowError, User.DoesNotExist):
                user = None

            if user is not None:
                if birthdate:
                    date_format = '%d/%m/%Y'
                    user.birth_date = datetime.datetime.strptime(birthdate,
date_format).date()
                    if( user.birth_date > datetime.date.today()):
                        # print('here')
                        user.birth_date = None
                        age = 0
                        messages.warning(request, "Birthdate must be less than
today.")
                        return redirect('profile')
                else:
                    user.birth_date = None
                user.first_name = firstname
                user.middle_name = middlename
                user.last_name = lastname
                user.username = username
                user.email = email
                if age:
                    user.age = age
                else:
                    user.age = None
                user.bio = bio
                user.profession = profession
                user.organization = organization
                user.gender = gender
                user.mobile_number = mobilenumber
                user.pincode = pincode
                user.city = city
                user.country = country
                user.skills = skills
                # print(user)

                if request.FILES.get('profilepicture', False):
                    profile_picture = request.FILES['profilepicture']
                    # print(profile_picture)
                    if user.profile_picture != 'profile_pics/default.svg':

user.profile_picture.storage.delete(user.profile_picture.name)
                    # fss =
FileSystemStorage(location=settings.PROFILE_PICTURE_STORAGE)
                    # print(profile_picture.name)
                    # filename = fss.save(profile_picture.name, profile_picture)
                    # user.profile_picture = profile_picture
                    # user.profile_picture.name = filename
                    user.profile_picture = profile_picture
```

```
                user.save()

                messages.success(request,'Profile Updated Successfully')
                return redirect('profile')

        messages.error(request,'Failed to update Profile... Try Again')

    current_user = request.user
    users =
Answer.objects.filter(is_accepted=True).values('user').annotate(ac_count=Count('u
ser')).order_by('-ac_count')
    num_of_users = len(users)
    current_user_index = next((index for index in range(num_of_users) if
users[index]['user'] == current_user.id), None)

    # print(users)
    # print(num_of_users, current_user_index)

    badge = None
    if current_user_index is not None:
        rank = round(current_user_index / num_of_users * 100)
        badge = 'Gold' if rank <= 10 else 'Silver' if rank <= 25 else 'Bronze' if
rank <= 50 else None

    # print(rank, badge)
    return render(request, 'profile.html', { 'user': current_user, 'badge': badge
})

@login_required(login_url='signin')
def change_password(request):
    if request.method == 'POST':
        currentpassword = request.POST['current-password'].strip()
        newpassword = request.POST['new-password'].strip()
        newretypepassword = request.POST['new-retypepassword'].strip()

        if newpassword != newretypepassword:
            messages.error(request, "New Password didn't matched!!")
            return redirect('profile')

        current_user = request.user

        if current_user.id is not None:
            try:
                user = User.objects.get(pk=current_user.id)
            except(TypeError, ValueError, OverflowError, User.DoesNotExist):
                user = None

            # print(user.password)
            if user is not None:
                # print(check_password(currentpassword, user.password))
                if check_password(currentpassword, user.password):
```

```
                    if currentpassword == newpassword:
                        messages.error(request, "New Password and Current
Password must be different!!")
                        return redirect('profile')
                    user.set_password(newpassword)
                    user.save()
                    messages.success(request, "Password Updated Successfully...")
                    return redirect('profile')
                else:
                    messages.error(request, "Current Password not Matched!! Try
Again...")
                    return redirect('profile')

        messages.error(request, "User not found!!")

    return redirect('profile')


def forgot_password(request):
    if request.method == 'POST':
        email = request.POST['email'].strip()

        try:
            user = User.objects.get(email=email)
        except(TypeError, ValueError, OverflowError, User.DoesNotExist):
            user = None

        if user is not None:
            password = User.objects.make_random_password(length=8)

            email_subject = "DevTalks - Password Reset"
            ctx = dict({
                'username': user.username,
                'password': password,
            })
            message = get_template('forgotPasswordEmail.html').render(ctx)
            email = EmailMultiAlternatives(
                email_subject,
                message,
                settings.EMAIL_HOST_USER,
                [user.email],
            )
            email.content_subtype = 'html'
            email.fail_silently = True
            email.send()

            # Set New Password if Verification Mail Successfully Sent
            user.set_password(password)
            user.save()

            messages.success(request,'New Password sent successfully on your
```

```
Email.')

            return redirect('signin')

        messages.error(request,'No account found with given Email ID.')

    # print(request.method)
    return render(request, 'forgotPassword.html')
```

tokens.py

```
from django.contrib.auth.tokens import PasswordResetTokenGenerator
import six


class AccountActivationTokenGenerator(PasswordResetTokenGenerator):
    def _make_hash_value(self, user, timestamp):
        return (
            six.text_type(user.pk) + six.text_type(timestamp)  +
six.text_type(user.is_active)
        )

account_activation_token = AccountActivationTokenGenerator()
```

Questions App

models.py

```
from django.db import models
from users.models import User
import uuid
# Create your models here


class Question(models.Model):
     id        = models.CharField(verbose_name='QuestionID',name='id',
primary_key=True, unique=True, editable=False, default=uuid.uuid4, max_length=40)
     user      = models.ForeignKey(User, on_delete=models.CASCADE,
blank=True, null=True, related_name="user")
     title     = models.CharField(verbose_name='Title',     name='title',
blank=False, null=False, max_length=100)
     body      = models.TextField(verbose_name='Body',      name='body',
blank=False, null=False)
     createdAt = models.DateTimeField(auto_now_add=True)
     updatedAt = models.DateTimeField(auto_now=True)

     class Meta:
          ordering = ['-updatedAt']

class Tag(models.Model):
     question      = models.ForeignKey(verbose_name='QuestionID',
name='question_id', to=Question, related_name='tags', on_delete=models.CASCADE,
blank=False, null=False)
     tag           = models.CharField(verbose_name='TagName',       name='tag',
max_length=30, blank=False, null=False)

     class Meta:
          unique_together = (('question_id', 'tag'),)

class Answer(models.Model):
     id          = models.CharField(verbose_name='AnswerID',      name='id',
primary_key=True,    default=uuid.uuid4,   unique=True,     editable=False,
max_length=40)
     question    = models.ForeignKey(verbose_name='QuestionID',
name='question_id',   to=Question, null=False, blank=False,
related_name='answers', on_delete=models.CASCADE, to_field='id')
     body        = models.TextField(verbose_name='Body',          name='body',
blank=False, null=False)
     user        = models.ForeignKey(User, on_delete=models.CASCADE, blank=True,
null=True)
     is_accepted = models.BooleanField(verbose_name='IsAccepted',
name='is_accepted', default=False, blank=False, null=False)
     createdAt = models.DateTimeField(auto_now_add=True)
     updatedAt = models.DateTimeField(auto_now=True)
```

```
class Like(models.Model):
     user     = models.ForeignKey(User, on_delete=models.CASCADE, blank=True,
null=True)
     question = models.ForeignKey(verbose_name='QuestionID',
name='question_id' ,to=Question,   null=False,  blank=False,
related_name='likes', on_delete=models.CASCADE, to_field='id')

class Upvote(models.Model):
     user   = models.ForeignKey(User, on_delete=models.CASCADE, blank=True,
null=True)
     answer = models.ForeignKey(verbose_name='AnswerID', name='answer',
to=Answer, to_field='id', null=False, blank=False, related_name='upvotes',
on_delete=models.CASCADE)

class Image(models.Model):
     id = models.CharField(verbose_name='ImageID', name='id',
primary_key=True,    default=uuid.uuid4,   unique=True,     editable=False,
max_length=40)
     img = models.ImageField(verbose_name='Image', name='image',
upload_to='images/')


# ForeignKey is same as DBMS, except a term, related_name="<NAME>"
# we can access one 2 many relationship from parent entity using related_name,
defined in the foreignKey of child entity

# child_objects = parent_object.related_name.all()
```

admin.py

```
from django.contrib import admin
from questions.models import Like, Upvote, Question, Tag, Answer, Image

admin.site.register(Question)
admin.site.register(Answer)
admin.site.register(Tag)
admin.site.register([Like, Upvote, Image])
# Register your models here.
```

urls.py

```
from django.urls import path
from questions import views
from django.views.generic import TemplateView

urlpatterns = [
     path('read/<uuid>/',views.read,name='read'),
     path('read/',views.readall,name='readall'),
     path('create/', TemplateView.as_view(template_name='questionCreate.html'),
```

```
    path('edit/<uuid>/', views.edit_question, name='question_edit'),
    path('question/',views.add_question,name='add_question'),
    path('answer/<question_id>/',views.add_answer,name='add_answer'),
    path('feed1/',views.get_feed,name='feed1'),
    path('question-
read/',TemplateView.as_view(template_name='questionRead.html')),

    path('like/<question_id>/', views.toggle_like, name='like'),
    path('answer/upvote/<answer_id>/', views.toggle_upvote, name='upvote'),
    path('answer/verify/<answer_id>/', views.toggle_verify, name='verify'),

    # Paths for EditorJS
    path('uploadImg/',views.upload_img, name='imageupload'),
    path('fileresp/<id>',views.fileresp),
    path('output/',views.output, name="output"),
]
```

views.py

```
from django.http.response import HttpResponse, JsonResponse, FileResponse
from django.shortcuts import redirect, render
from django.views.decorators.csrf import csrf_exempt
from questions.models import Question, Tag, Answer, Like, Upvote, Image
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.db.models import Q,Count

import io, time
from django.core.files.storage import default_storage


# Question Save Data Temp View
def output(request):
    if request.method == 'POST':
        # print(request.POST)
        question_title = request.POST['questionTitle']
        question_tag = request.POST['questionTag']
        json_data = request.POST['jsonData']
        # print(jsonData);
        return HttpResponse(question_title + "<br>" + question_tag + "<br>" +
json_data)

def fileresp(request,id):
    img = Image.objects.get(id=id)
    obj = default_storage.open(str(img.image.name),'rb')
    # ip = io.StringIO()
    # ip.write("Hello")
    # op = io.BufferedReader(ip)
    # print(obj.read())
    return FileResponse(obj)
```

```
def read(request,uuid):
     data = Question.objects.filter(id=uuid).first()
     if data is None:
          return HttpResponse('None')
     # print(data.user_id)
     # data.delete()
     # print(data.title)
     answers = data.answers.all()\
               .annotate(num_likes=Count('upvotes'))\
               .order_by('-num_likes')
     # print(answers)
     return render(request, 'questionRead.html', { 'question' : data, 'answers':
answers })

@csrf_exempt
def upload_img(request):
     response_json = {
          'success': 0
     }
     if request.method == 'POST':
          #print("POST Method Called")
          name = request.FILES['image'].name
          request.FILES['image'].name = str(time.time())+ '.' +name.split('.')
[-1]
          data = Image(image=request.FILES['image'])
          data.save()
          print(data.id)
          response_json['success'] = 1
          response_json['file'] = {
               'name': str(data.id),
               'url': '/question/fileresp/'+str(data.id)
          }
     else:
          print("Get Method Called")
     return JsonResponse(response_json)

def readall(request):
     data = Question.objects.all()
     st = "<ol>"
     for d in data:
          t = '<li>' + str(d.id) + '<ul>'
          for a in d.answers.all():
               t = t + '<li>' + str(a.id) + '</li>'
          # print(d.answers.all())
          st = st + t + '</ul></li>'

     return HttpResponse(st+'</ol>')

@login_required(login_url='signin')
def add_question(request):
     if request.method == "GET":
```

```
          return redirect('signin')

     myuser = request.user
     user_id = myuser.id

     if user_id is not None:
          title = request.POST['questionTitle']
          body = request.POST['jsonData']
          tags = request.POST['questionTag'].split(",")
          question = Question(user_id=user_id, title=title, body=body)
          question.save()

          for tag in tags:
               obj = Tag(question_id=question, tag=tag.strip())
               obj.save()

          messages.success(request,'Question added successfully')

     return redirect('feed')

@login_required(login_url='signin')
def edit_question(request, uuid):
     data = Question.objects.filter(id=uuid).first()
     if request.user.username != data.user.username:
          messages.error(request,'You are not Authorized to Edit the Question')
          return redirect('read', uuid=uuid)

     if request.method == "POST":
          data.title = request.POST['questionTitle']
          # if user_id data.user.id:
          data.body = request.POST['jsonData']
          tags = request.POST['questionTag'].split(",")
          Tag.objects.filter(question_id=data).delete()
          for tag in tags:
               obj = Tag(question_id=data, tag=tag.strip())
               obj.save()
          data.save()
          messages.success(request,'Question Updated successfully')
          return redirect('question_edit', uuid=uuid)

     return render(request, 'questionEdit.html', { 'question': data })


@login_required(login_url='signin')
def add_answer(request, question_id):
     if request.method == "POST":
          question = Question.objects.filter(id=question_id).first()
          myuser = request.user
          # print(myuser.username)
          body = request.POST['jsonData'].strip()
          print(body)
```

```
            answer = Answer(user=myuser, body=body, question_id=question)
            answer.save()
            messages.success(request, 'Answer added successfully')

            return redirect('read', uuid=question.id)

      return redirect('feed')

def get_feed(request):
      questions = Question.objects.all()
      # print(questions)
      tags = request.GET.get('tags',None)
      if tags is not None:
            tags = tags.split(',')
            for idx in range(len(tags)):
                  tags[idx] = tags[idx].strip()
            #print(tags)
            #q = Q(tag='tag1') & Q(tag='tag2')
            #print(q)
            #Tag.objects.complex_filter()

            # 1. Fetch Questions based on the given tag list, 2. select only
necessary fields
            # 3. add another field called annotate, which will group by based on
the available fields in the QuerySet, here only Question
            # 4. Filter only those which are Exact match
            # 5. Select only question_id field
            # 6. Fetch the questions which were filtered based on the tags
            filtered_questions = Tag.objects.filter(tag__in=tags)\
                                    .values('question_id')\
                                    .annotate(cnt=Count('tag'))\
                                    .filter(cnt=len(tags))\
                                    .values('question_id')
            questions = Question.objects.filter(id__in=filtered_questions)
            #print(questions)


      # print(questions[0].tags)
      return render(request, 'feed.html', { 'questions' : questions })

@csrf_exempt
@login_required(login_url='signin')
def toggle_like(request,question_id):
      if request.method == "POST":
            # print(request.user.id)
            user = request.user
            like_status = Like.objects.filter(question_id=question_id,
user=user).first()
            # print(like_status)
            try:
                  if like_status is None:
```

```
                    add_like = Like(question_id_id=question_id, user=user)
                    add_like.save()
                    # return redirect('feed')
                    return JsonResponse({'Success':1})
               else:
                    like_status.delete()
                    # return redirect('feed')
                    return JsonResponse({'Success':2})
          except:
               # return redirect('feed')
               return JsonResponse({'Success':0})

@csrf_exempt
@login_required(login_url='signin')
def toggle_upvote(request,answer_id):
     # print(request.user.id)
     user = request.user
     upvote_status = Upvote.objects.filter(answer_id=answer_id,
user=user).first()
     # print(upvote_status)
     try:
          if upvote_status is None:
               add_upvote = Upvote(answer_id=answer_id, user=user)
               add_upvote.save()
               return JsonResponse({'Success':1})
          else:
               upvote_status.delete()
               return JsonResponse({'Success':2})
     except:
          return JsonResponse({'Success':0})

@csrf_exempt
@login_required(login_url='signin')
def toggle_verify(request,answer_id):
     if request.method == "POST":
          # print(request.user.id)
          user = request.user
          answer = Answer.objects.filter(id=answer_id).first()
          if answer.question_id.user.username == user.username:
               answer.is_accepted = not answer.is_accepted
               answer.save()
               if answer.is_accepted:
                    return JsonResponse({'Success':1})
               else:
                    return JsonResponse({'Success':2})

          return JsonResponse({'Success':0})
```