Module	Portfolio	Assessment Type
Collaborative Development (5CS024)	1	Individual Report

Tour Management System - Developer

Student Id : 2059624

Student Name : Pallaw Rana Magar

Section : L5CG7

Group : L5CG7 Group 4

Module Leader : Er. Uday Kandel

Lecturer : Er. Raj Prasad Shrestha

Acknowledgement

I would like to thank the University of Wolverhampton and Herald College Kathmandu for providing with this opportunity to write this report.

I would also like to express my gratitude towards Mr. Biraj Dulal and Er. Raj Prasad Shrestha for assisting and guiding me in completing this report. Finally, I would like to thank my team members on their effort for completing this report.

Table of Contents

Table of Figures

Self-appraisal form

Student number	2059624	Name	Pallaw Rana Magar
Project	Tour Management System	Date	
Role	Backend Developer	Team	L5CG7 (group 4)
Sprint (1 or 2)	2		

Personal objectives – performance measurement

These should be copied from your role description.

Objectives	**Evidence provided** *(E.g. appendix A, file name etc.)*	**Evaluation** *Student / tutor*	
Implementing functional requirement	For this sprint, we have created many applications. First, we were able to make a user model. We could create different kind of users and add, delete, and edit a user. Also, we added blog and comments in our system. Users could view these blogs and admin could add, delete these blogs and users could add a comment in these blogs. Appendix A	9	
Tutor feedback:			
Bug Fixing	For this sprint, we have faced many bugs such as bug regarding permissions classes where everyone could see the account details and edit them which I fixed by adding proper permissions classes. I also faced a bug regarding packages and bookings not having date which I fixed by adding date field. I also had problem viewing blog detail which I fixed by adding slug. Appendix A	9	
Tutor feedback:			
		/20	/20

Evidence Provided: The reference of the appendix in which the evidence can be found (*Appendix A, appendix B, screenshot.jpg, myreport.docx* etc. please attach to this form). Don't hesitate to add comments and notes to your appendices to highlight particular sections, relevant pieces of code etc.

Collaboration Document

Evidence of good collaboration

Good communication and file sharing

Receivable evidence includes:

- Emails and other types of messages (Facebook messenger, WhatsApp etc.)
- Screenshots of Basecamp conversations in which you actively participate.
- Screenshots showing files (designs, reports) that **you** shared with your team on Basecamp

***Important**: Please include no more than 5 items*

I was able to communicate with my team through different apps. We mostly used Google meet, discord, and messenger for communicating. We also focused on physical communication more on this sprint. We usually talked in messenger informally about the problems we were facing and how to fix it. We talked in discord and basecamp about the works we are doing and asked for feedback about our work. Appendix 2

Continuing Personal Development (CPD)

Receivable evidence includes:

- Course/seminar attendance register
- Online course: certificate of completion
- Word document summarising what was learnt and how it can be used on the project

***Important**: Please include no more than 5 items*

I was more confident after completing sprint 1 and was able to my work more efficiently. I also learned about custom user model, authentication, jwt tokens and search filters in this sprint. I learnt them through researching, reading Django documentation and watching YouTube video about these topics. I also learned about permissions classes through documentation, YouTube, and Django API book. Appendix 2

Issue tracking

Receivable evidence includes:

- Screenshots of **personal contributions** to GitHub issues.

***Important**: Please include no more than 5 items*

In the first issue, I simply updated my urls.py file and changed the detail to slug. In one issue, I simply added a date field in my models and serializers for tour packages and blogs. I also had an issue regarding permissions classes missing and permissions not working which I fixed by adding proper permission classes and making only the authenticated users to view the data. Appendix 2

Work to deadlines

No evidence required. Your tutor will decide whether you have worked to deadlines based on various factors (team meetings, discussions with other team members, discussion with client etc.)

Appendix A

1) Implementing functional requirements

```python
from asyncio.windows_events import NULL
from multiprocessing.sharedctypes import Value
from django.db import models
from django.contrib.auth.models import (
    AbstractBaseUser, BaseUserManager
)

class UserManager(BaseUserManager):
    def create_user(self,email, first_name, last_name, phone_number, password=None,
                    is_active = True, is_staff = False, is_admin = False):
        if not email:
            raise ValueError("Users must have an email address")
        if not password:
            raise ValueError("Users must have a password")
        if not first_name:
            raise ValueError("Users must enter their name")
        if phone_number == NULL:
            raise ValueError("Users must enter their phone number")
        user_obj = self.model(
            email = self.normalize_email(email),
            first_name = first_name,
            last_name = last_name,
            phone_number = phone_number,
        )
        user_obj.set_password(password) #change user password
        user_obj.staff = is_staff
        user_obj.admin = is_admin
        user_obj.active = is_active
        user_obj.save(using=self._db)
        return user_obj
```

Figure 1 models.py

```python
    def create_staffuser(self, email, first_name, last_name, phone_number, password=None):
        user = self.create_user(
            email,
            first_name,
            last_name,
            phone_number,
            password=password,
            is_staff=True,
        )
        return user

    def create_superuser(self, email, first_name, last_name, phone_number, password=None):
        user = self.create_user(
            email,
            first_name,
            last_name,
            phone_number,
            password=password,
            is_staff=True,
            is_admin=True,
        )
        return user

class User(AbstractBaseUser):

    email = models.EmailField(max_length=255, unique=True)
    first_name = models.CharField(max_length=255, blank=True, null = True)
    last_name = models.CharField(max_length=255, blank=True, null = True)
    phone_number = models.IntegerField(blank=True, null = True)
    active = models.BooleanField(default=True) #can login
```

Figure 2 models.py

```python
    staff = models.BooleanField(default=False) #staff user non super user
    admin = models.BooleanField(default=False) #superuser

    USERNAME_FIELD = 'email' #username
    REQUIRED_FIELDS = ['first_name', 'last_name', 'phone_number']

    objects = UserManager()

    def __str__(self):
        return self.email

    def get_first_name(self):
        return self.email

    def get_last_name(self):
        return self.email

    def get_phone_number(self):
        return self.email

    def has_perm(self, perm, obj=None):
        return True

    def has_module_perms(self, app_label):
        return True

    @property
    def is_staff(self):
        return self.staff
```

Figure 3 models.py

```python
@property
def is_admin(self):
    return self.admin

@property
def is_active(self):
    return self.active
```

Figure 4 models.py

This is the models.py of accounts I made. In the first class UserManager, we check if the information is correct or not such as email, password, etc and this class helps us

to create different kind of users. And in class User, we have fields which are used to create a database to store the accounts with the provided information.

```python
from django.contrib import admin
from django.contrib.auth import get_user_model
from django.contrib.auth.models import Group
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin

from .forms import UserAdminCreationForm, UserAdminChangeForm

User = get_user_model()

# Remove Group Model from admin. We're not using it.
# admin.site.unregister(Group)


class UserAdmin(BaseUserAdmin):
    # The forms to add and change user instances
    form = UserAdminChangeForm
    add_form = UserAdminCreationForm

    # The fields to be used in displaying the User model.
    # These override the definitions on the base UserAdmin
    # that reference specific fields on auth.User.
    list_display = ['email', 'admin']
    list_filter = ['admin', 'staff', 'active']
    fieldsets = (
        (None, {'fields': ('email', 'password')}),
        ('Personal info', {'fields': ('first_name', 'last_name', 'phone_number',)}),
        ('Permissions', {'fields': ('admin', 'staff', 'active')}),
    )
    # add_fieldsets is not a standard ModelAdmin attribute. UserAdmin
```

Figure 5 admin.py

```python
    # overrides get_fieldsets to use this attribute when creating a user.
    add_fieldsets = (
        (None, {
            'classes': ('wide',),
            'fields': ('first_name', 'last_name', 'phone_number','email', 'password', 'password_2')}
        ),
    )
    search_fields = ['email']
    ordering = ['email']
    filter_horizontal = ()


admin.site.register(User, UserAdmin)
```

Figure 6 admin.py

Here, is the screenshot of admin.py where in UserAdmin class, we have forms which we can use to add and edit the user instances. And below we can see the fields we can display in the user model and the admin model.

```python
from rest_framework import serializers
from .models import User

class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        fields = ('email', 'first_name', 'last_name', 'phone_number', 'password')
        model = User
        extra_kwargs = {
            'password' : {'write_only': True}
        }
    def create(self, validated_data):
        password = validated_data.pop("password", None)
        instance = self.Meta.model(**validated_data)
        if password is not None:
            instance.set_password(password)
        instance.save()
        return instance
```

Figure 7 serializers.py

Here is the screenshot of serializers.py which takes the fields given can converts the database object from data to JSON.

```python
from datetime import date, datetime
from http.client import ResponseNotReady
from tokenize import Token
from urllib import response
from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework import generics, permissions
from rest_framework.exceptions import AuthenticationFailed
from .models import User
from rest_framework.response import Response
from .serializers import AccountSerializer
import jwt, datetime
from .permissions import IsAuthorOrIsAuthenticated



class AccountList(generics.ListCreateAPIView):
    permission_classes = [permissions.IsAuthenticated,]
    queryset = User.objects.all()
    serializer_class = AccountSerializer

class AccountDetail(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = [IsAuthorOrIsAuthenticated]


    queryset = User.objects.all()
    serializer_class = AccountSerializer

```

Figure 8 views.py

```python
class RegisterView(APIView):
    def post(self, request):
        serializer = AccountSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data)

class LoginView(APIView):
    def post(self, request):
        email = request.data['email']
        password = request.data['password']

        user = User.objects.filter(email = email).first()

        if user is None:
            raise AuthenticationFailed("User not found!")

        if not user.check_password(password):
            raise AuthenticationFailed("Incorrect password!")

        payload = {
            'id': user.id,
            'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60),
            'iat': datetime.datetime.utcnow()
        }

        token = jwt.encode(payload, 'secret', algorithm= 'HS256').decode('utf-8')

        response = Response()
```

Figure 9 views.py

```
        response.set_cookie(key='jwt', value=token, httponly=True)
        response.data = {
            'jwt': token
        }
        return response

class UserView(APIView):

    def get(self, request):
        token = request.COOKIES.get('jwt')

        if not token:
            raise AuthenticationFailed('Unauthenticated!')

        try:
            payload = jwt.decode(token, 'secret', algorithm=['HS256'])
        except jwt.ExpiredSignatureError:
            raise AuthenticationFailed('Unauthenticated!')

        user = User.objects.filter(id=payload['id']).first()
        serializer = AccountSerializer(user)
        return Response(serializer.data)

class LogoutView(APIView):
    def post(self, request):
        response = Response()
        response.delete_cookie('jwt')
        response.data = {
            'message': 'success'
        }
```

Figure 10 views.py

This is the screenshot of views.py where we are able to take a web request and helps us return a web response. There are classes like login, logout and register which redirects us to different page.

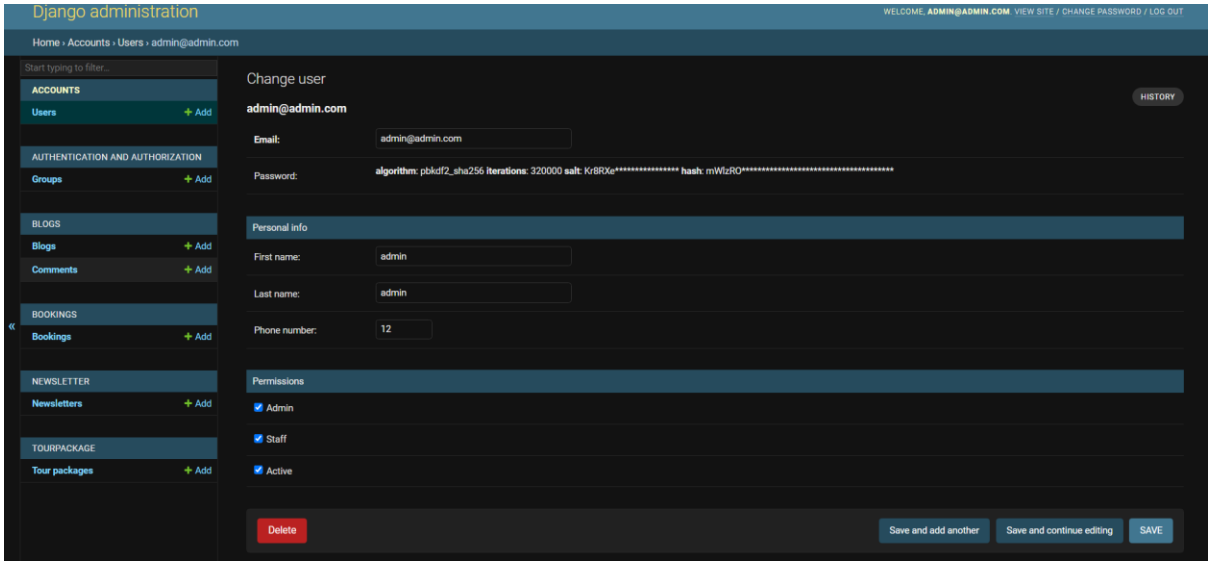


Figure 11 admin panel

This is the admin panel, we can edit the user data, add, and delete users from here.

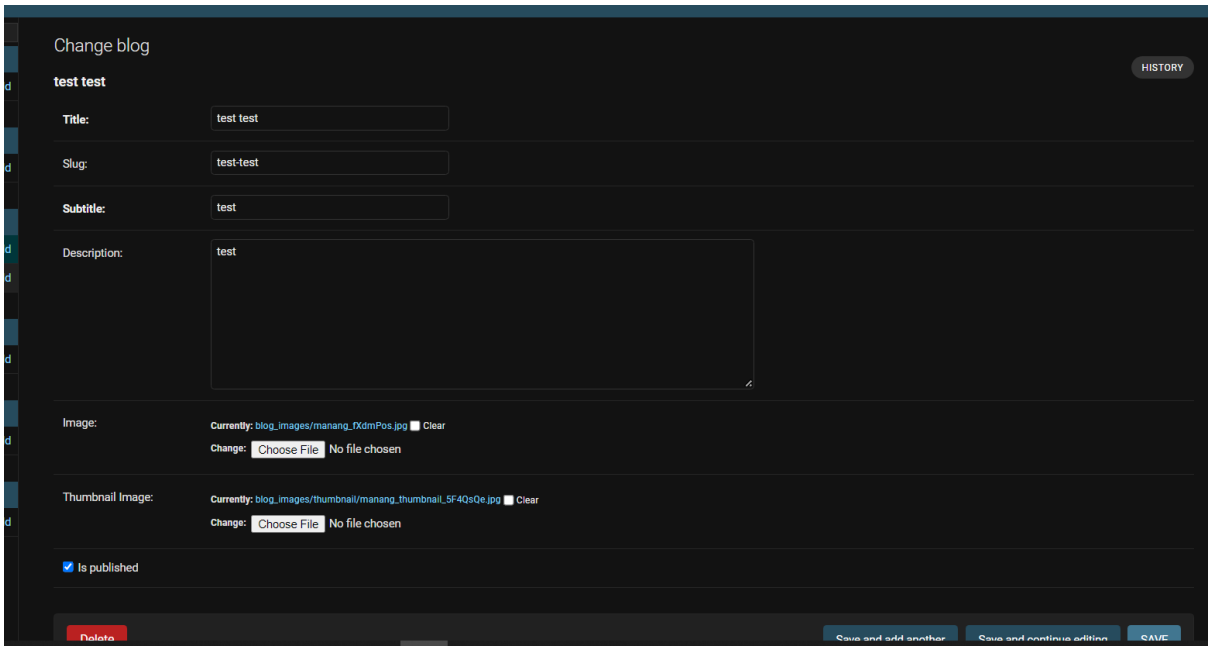


Figure 12 blog panel

This is the blog panel, where we can edit, add, and delete the blog.

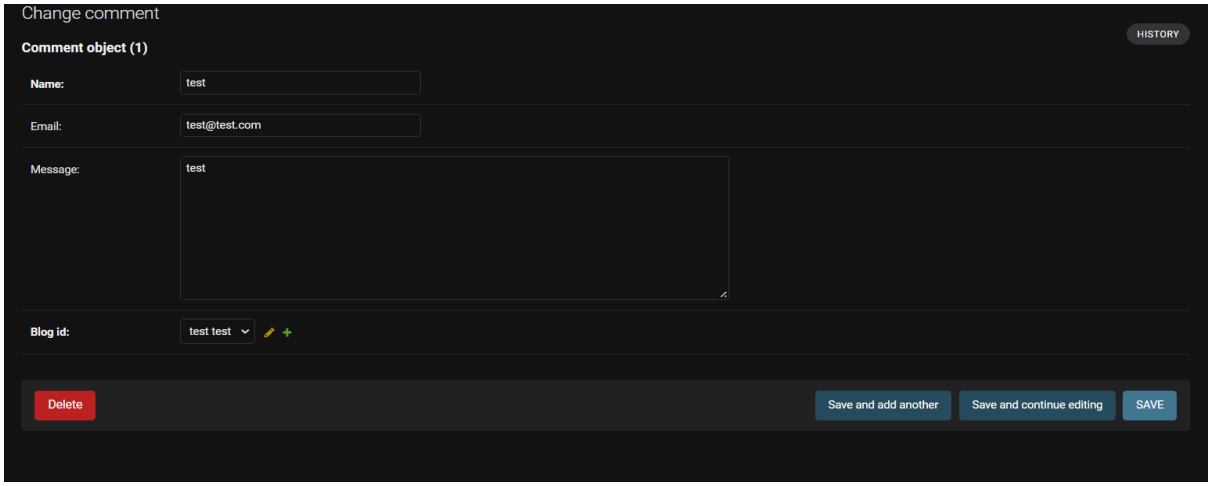


Figure 13 comment panel

Here is the comment panel where we can add, delete and edit the comments we add in blog.

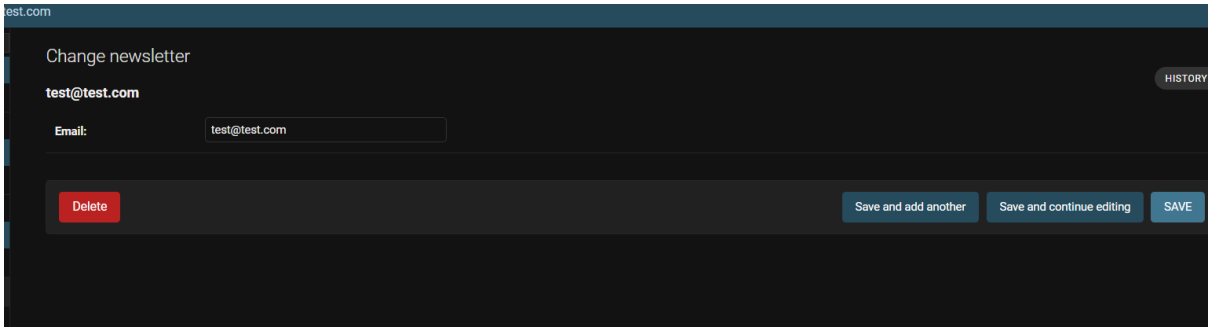


Figure 14 newsletter panel

This is the newsletter panel, we can see the email of people who have subscribed.

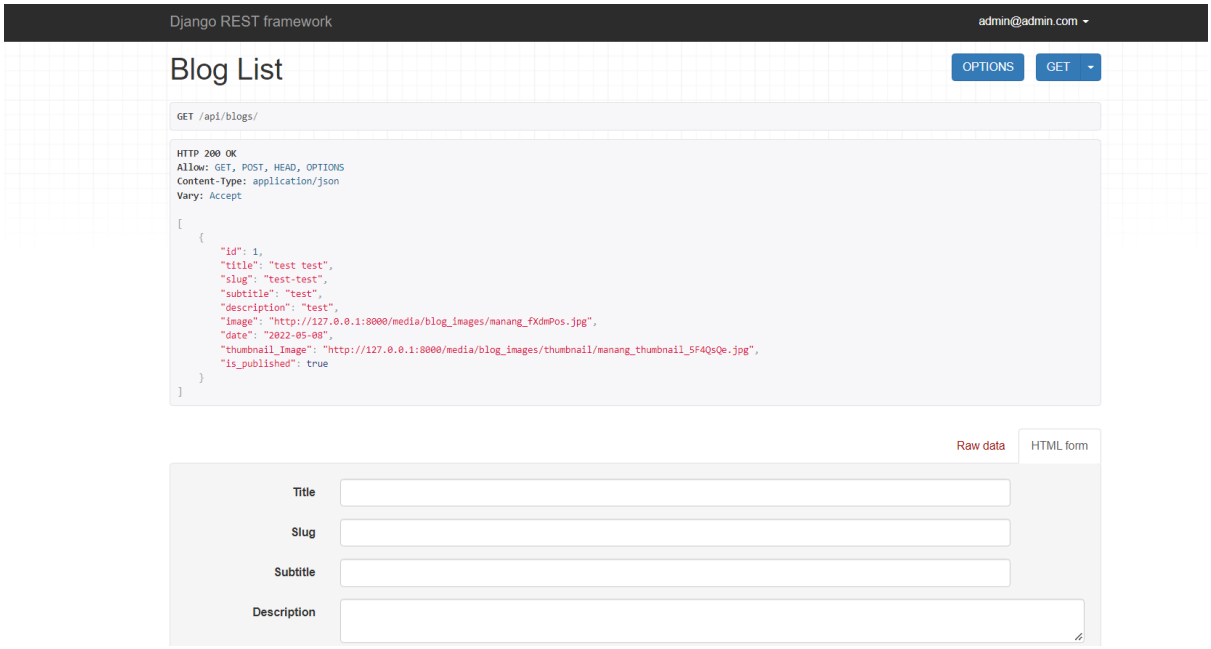


Figure 15 blog list

Here, is the API endpoint for blog.

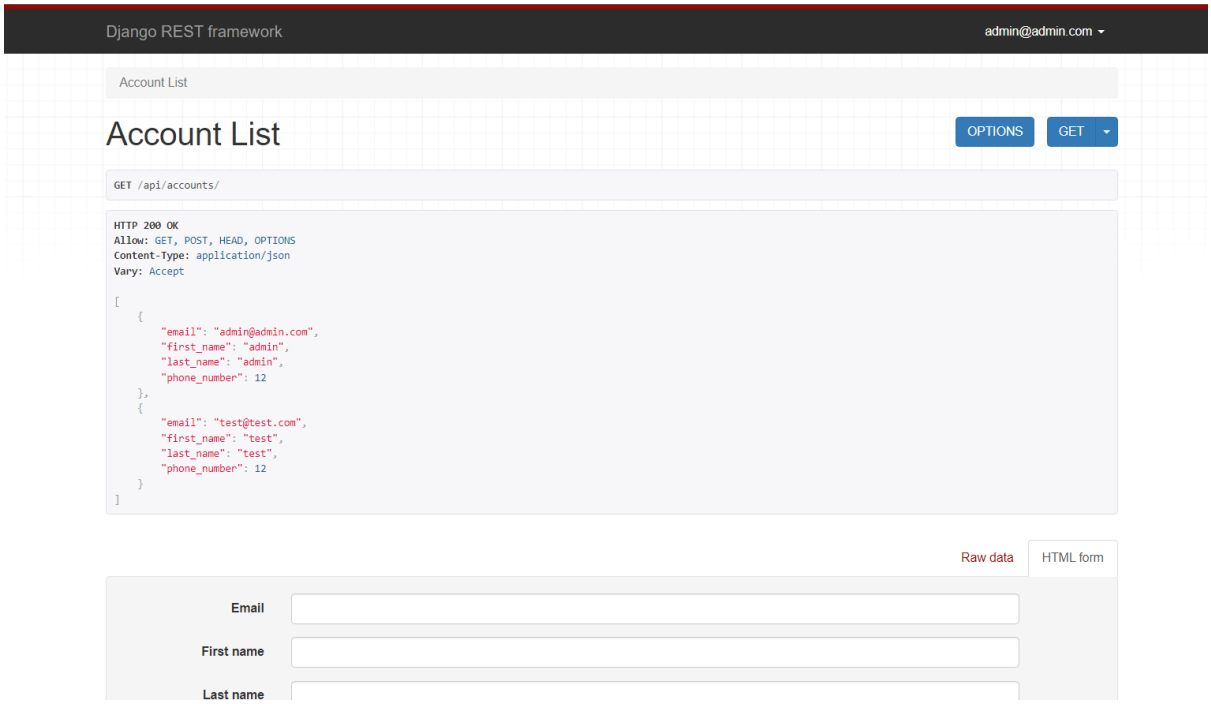


Figure 16 account list

Here, is the API endpoint for accounts.

```python
from django.contrib import admin
from django.urls import include, path
from django.conf import settings
from django.conf.urls.static import static

#urls of API
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/tour-packages/', include('tourPackage.urls')),
    path('api/bookings/', include('bookings.urls')),
    path('api/newsletter/', include('newsletter.urls')),
    path('api/blogs/', include('blogs.urls')),
    path('api/accounts/', include('accounts.urls')),
    path('api-auth', include('rest_framework.urls'))
]+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Figure 17 URLs of APIs

And these are the URLs for API's that I created.

Use of Version Control

This screenshot shows my git log.

```
(env) PS D:\herald\Year 2\4th semester\Collaborative Development\Tour\tour_management_system_api> git log --author "kenma-zks"
commit 32c8c396b222c4088776e361844fb993881e5fc5
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Sun May 8 13:32:16 2022 +0545

    updated the authentication

commit 239bcd8bff6236fc67b5c85a109924fb22f90e4a
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Sun May 8 11:54:49 2022 +0545

    updated default permissions

commit 8fcc97f83f94e6f9133708424039ac13fcb2a83a
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Sun May 8 11:09:43 2022 +0545

    added permissions and updated some models.

commit d08a7e411843f7c0a451699e1224574b05e284f7
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Mon May 2 15:39:23 2022 +0545

    added user authentication

:...skipping...
commit 32c8c396b222c4088776e361844fb993881e5fc5
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Sun May 8 13:32:16 2022 +0545

    updated the authentication

commit 239bcd8bff6236fc67b5c85a109924fb22f90e4a
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Sun May 8 11:54:49 2022 +0545

    updated default permissions
```

Figure 18 git log

```
commit d1c6b9b0fb15a82aee220def0c7c44d38485d6f8
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Mon May 2 11:21:49 2022 +0545

    added user register api

commit cab2f9b08c6ae3784f160cdd62d65eb3f4935478
Author: kenma-zks <pallawmgr20@gmail.com>
Date:   Sun May 1 10:19:41 2022 +0545

    added slug field and filter

.
```

```
(env) PS D:\herald\Year 2\4th semester\Collaborative Development\Tour\tour_management_system_api> git log --graph
*   commit 7fd6c75b9ba152d33c85ccd6c6ec8502c9ce94c7 (HEAD -> pallaw, origin/main, origin/HEAD)
|\  Merge: 7b87c35 32c8c39
| | Author: Nishant Shrestha <49269819+sNishant011@users.noreply.github.com>
| | Date:   Sun May 8 13:36:45 2022 +0545
| |
| |     Merge pull request #26 from sNishant011/pallaw
| |
| |     prevented update of user
| |
| * commit 32c8c396b222c4088776e361844fb993881e5fc5
| | Author: kenma-zks <pallawmgr20@gmail.com>
| | Date:   Sun May 8 13:32:16 2022 +0545
| |
| |     updated the authentication
| |
| * commit 239bcd8bff6236fc67b5c85a109924fb22f90e4a
| | Author: kenma-zks <pallawmgr20@gmail.com>
| | Date:   Sun May 8 11:54:49 2022 +0545
| |
| |     updated default permissions
| |
| * commit 8fcc97f83f94e6f9133708424039ac13fcb2a83a
| | Author: kenma-zks <pallawmgr20@gmail.com>
| | Date:   Sun May 8 11:09:43 2022 +0545
| |
| |     added permissions and updated some models.
| |
| * commit d08a7e411843f7c0a451699e1224574b05e284f7
| | Author: kenma-zks <pallawmgr20@gmail.com>
| | Date:   Mon May 2 15:39:23 2022 +0545
| |
| |     added user authentication
| |
| * commit d1c6b9b0fb15a82aee220def0c7c44d38485d6f8
| | Author: kenma-zks <pallawmgr20@gmail.com>
```

Figure 19 git log graph

```
| * commit cab2f9b08c6ae3784f160cdd62d65eb3f4935478
| | Author: kenma-zks <pallawmgr20@gmail.com>
| | Date:   Sun May 1 10:19:41 2022 +0545
| |
| |     added slug field and filter
| |
| * commit 185723ec4f0b721e924266728431c202283618df
|/  Author: kenma-zks <pallawmgr20@gmail.com>
|   Date:   Tue Apr 26 21:39:11 2022 +0545
|
|       added blog and comment api
|
*   commit 7b87c3513d2ad3947010e2681cf2f27c728ec46c (main)
|\  Merge: c07e959 a7bc2a2
| | Author: Nishant Shrestha <49269819+sNishant011@users.noreply.github.com>
| | Date:   Sun Apr 10 13:36:23 2022 +0545
| |
| |     Merge pull request #14 from sNishant011/pallaw
| |
| |     added more display fields in display models
| |
```

This is the screenshot of my git log which I did for the project.

2) Bug Fixing

Bugs are very common when are coding. I had many bugs in many stages of my project. I fixed most of them by communicating with my team members and by researching on the internet.

2.1 Bug regarding permissions

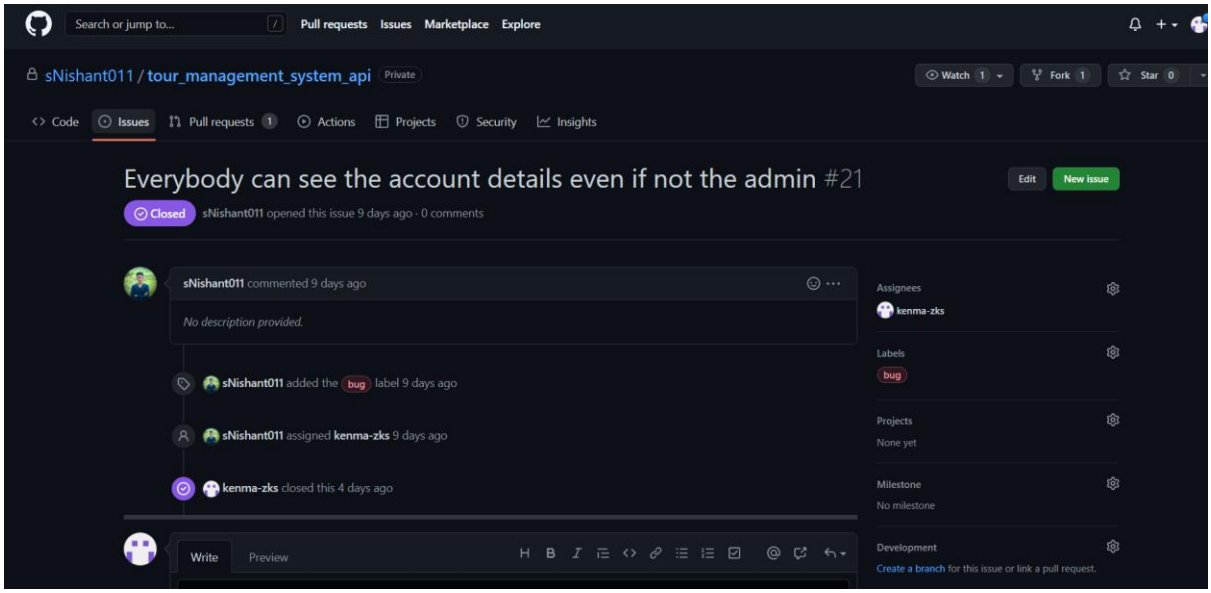


Figure 20 issue regarding permissions

I had a bug where everyone could see the account details of someone even if they are not logged in. They could also change the accounts details of people.

The bug happened because I hadn't added permissions for viewing account list and details and also because the default permissions were AllowAny.

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ],
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
}
```

Figure 21 default permissions class

To fix this, first I imported permission to views.py. I also imported IsAuthorOrIsAuthenticated from permissions.py which I made. Then I added

permission classes to AccountList with isAuthenticated which makes only the logged in user to see others information.

```
class AccountList(generics.ListCreateAPIView):
    permission_classes = [permissions.IsAuthenticated,]
    queryset = User.objects.all()
    serializer_class = AccountSerializer
```

Figure 22 screenshot of account list

For Account Detail, first I made a permissions.py file and imported permissions there. Now I made a class which made users unable to edit others information.

```
from rest_framework import permissions

from accounts.models import User

class IsAuthorOrIsAuthenticated(permissions.BasePermission):

    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return bool(request.user and request.user.is_authenticated)
        return User.id == request.user
```

Figure 23 screenshot of permissions.py

After making the new file, I added it to account detail in views.py.

```
class AccountDetail(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = [IsAuthorOrIsAuthenticated]
    queryset = User.objects.all()
    serializer_class = AccountSerializer
```

Figure 24 of account detail

Doing these steps fixed the bug of others being able to view or edit users information.

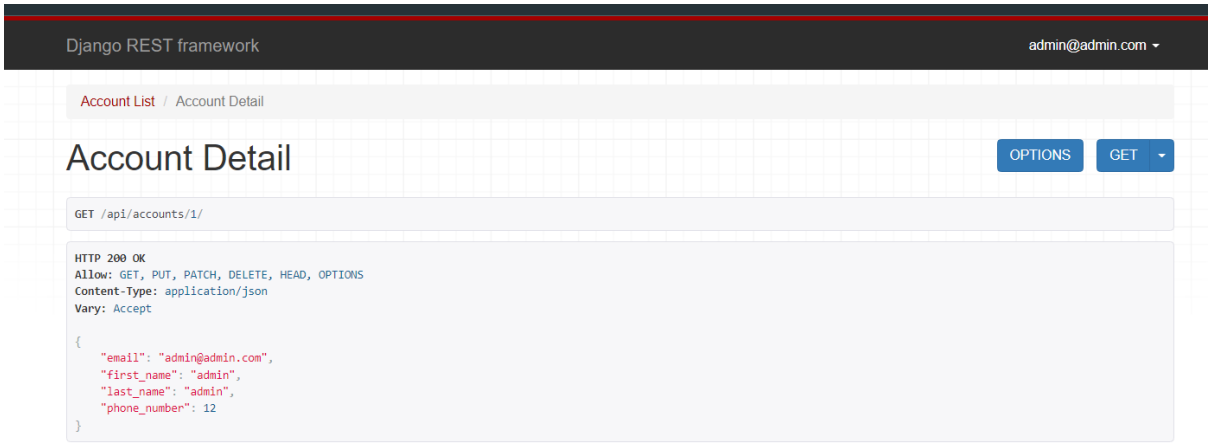
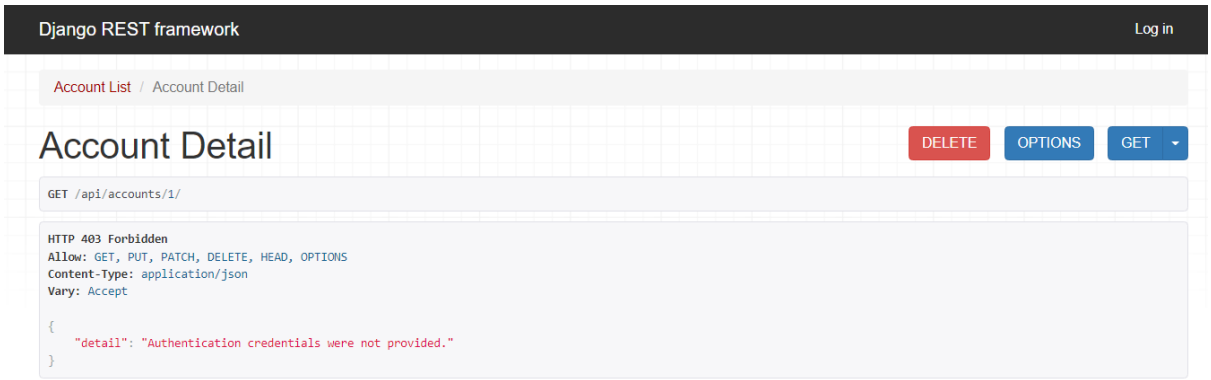


Figure 25 account detail panel



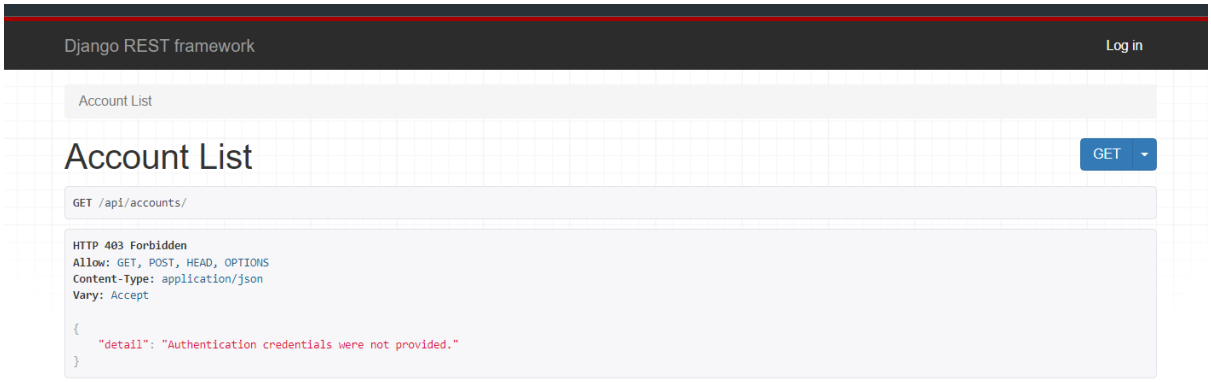


Figure 26 screenshot of account list panel

2.2) Bug regarding Date

I had an issue with tour packages and bookings. Even though I had added many fields for them like name of package, price, etc. We couldn't for when the package was and the time limit for booking of the package. This made the users to only view the package with no date given. This was a major flaw in our system.

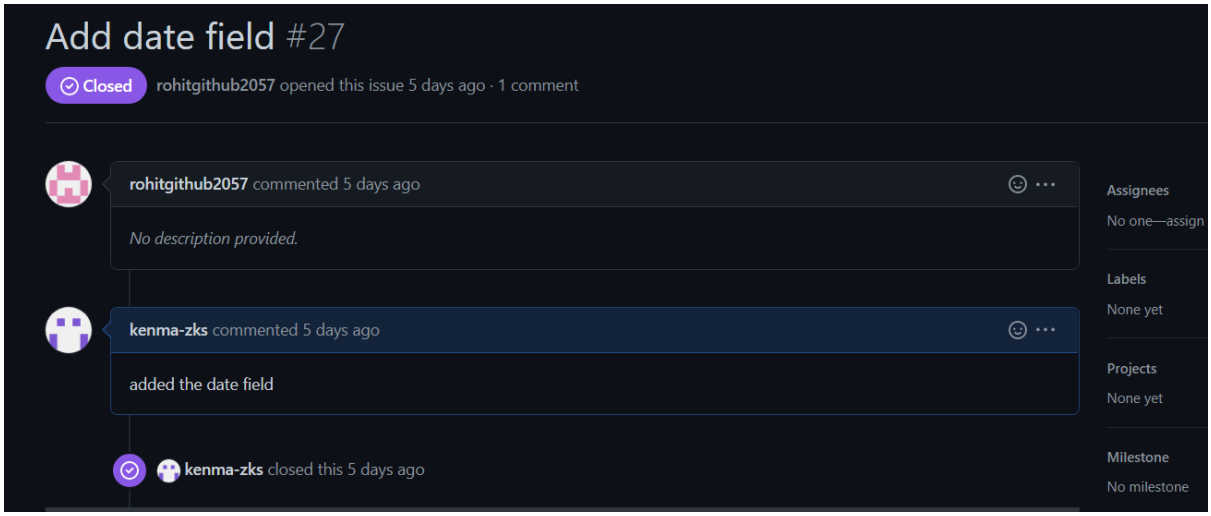


Figure 27 date field bug

To fix this, first I added a date field in my models.py in tourpackage app. I also added date in my serializers which would help me covert the database object in JSON which would help the user to view the date.

```
date = models.DateField(blank=False)
```

Figure 28 date field

```
fields = ('id', 'title', 'slug', 'image', 'thumbnail_Image', 'no_of_days', 'summary', 'date',
```

Figure 29 serializers.py with date

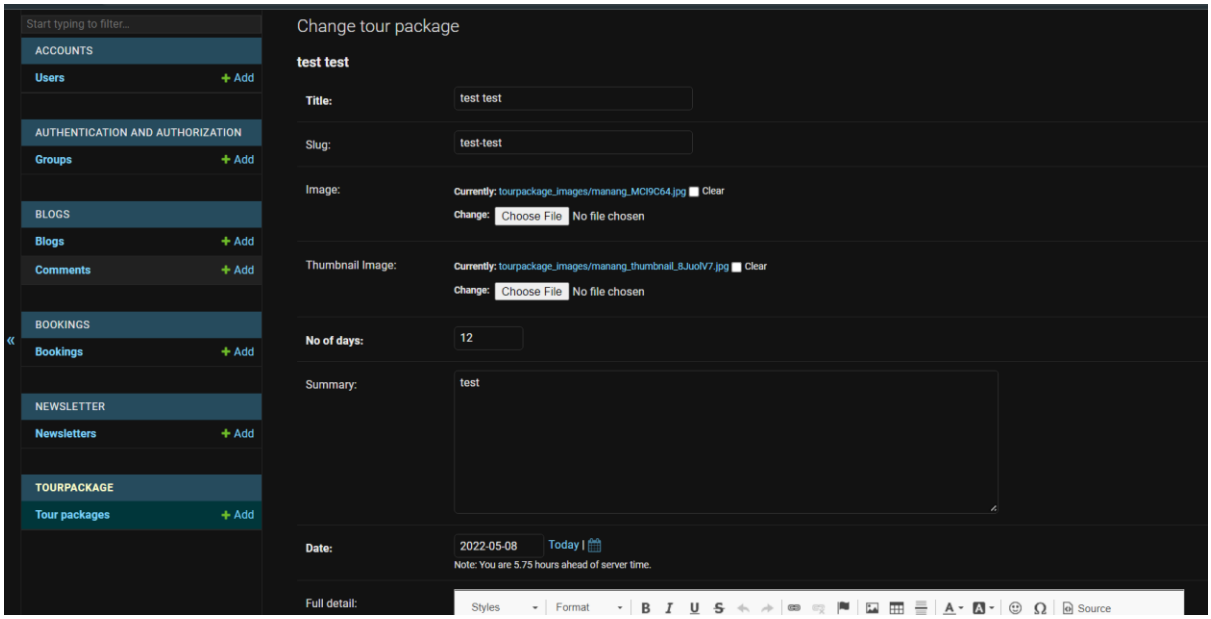


Figure 30 package panel

For the blog, I again created a new filed called date with datefield and add auto_now_add which automatically adds the date with its creation date. I also added the date field in serializers.py so that I could convert the database data to JSON.

```
date = models.DateField(auto_now_add= True)
```

```
class Meta:
    fields = ('firstname', 'lastname', 'email', 'phone_number', 'date',
```

2.3 Bug regarding Blog details

I had this bug which didn't allow me to view the Blog detail page. I couldn't go to my blog detail URL because in my urls.py I made the BlogDetail as int:pk which didn't let me search the blog detail.

Figure 31 bug regarding blog details

To fix this, first I went to my urls.py file and changes the URL pattern from int:pk to slug:slug which would take me to my detail page.

```
urlpatterns = [
    path('<int:pk>/', BlogDetail.as_view()),
    path('', BlogList.as_view()),
    path('comments/', CommentList.as_view()),
    path('comments/<int:pk>/', CommentDetail.as_view()),

]
```

Figure 32 URL patterns

```
urlpatterns = [
    path('<slug:slug>/', BlogDetail.as_view()),
    path('', BlogList.as_view()),
    path('comments/', CommentList.as_view()),
    path('comments/<int:pk>/', CommentDetail.as_view()),

]
```

Now, I could finally go to blogdetail and use it to add, view or delete the blogs.

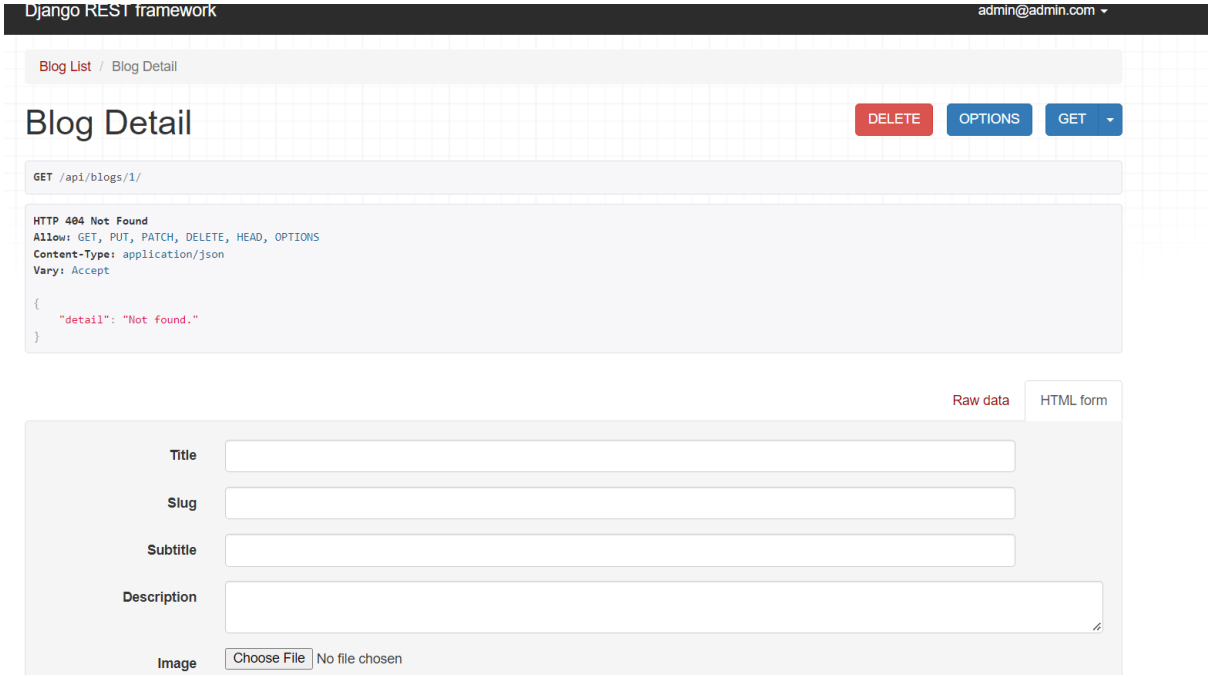


Figure 33 blog detail

2.4) Bug regarding permissions

I also had a small issue with viewing the account details and list. In default permissions settings I had written IsAuthenticated and also kept IsAuthenticated in serializers.py in both details and list but this raised an issue of not being able to login, register, etc and view in postman.

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
}
```

Figure 34 default permissions classes

So, I changed back the permissions to AllowAny in settings.py to be able to view accounts details, list.

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ],
    'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
}
```

2.5 Bug regarding searching comments

I had a bug where even though I had installed filtering in Django, it wouldn't work. I had learned about Django filtering and used for search blog with its id. But it would still who us every data instead of searching the specific comment.

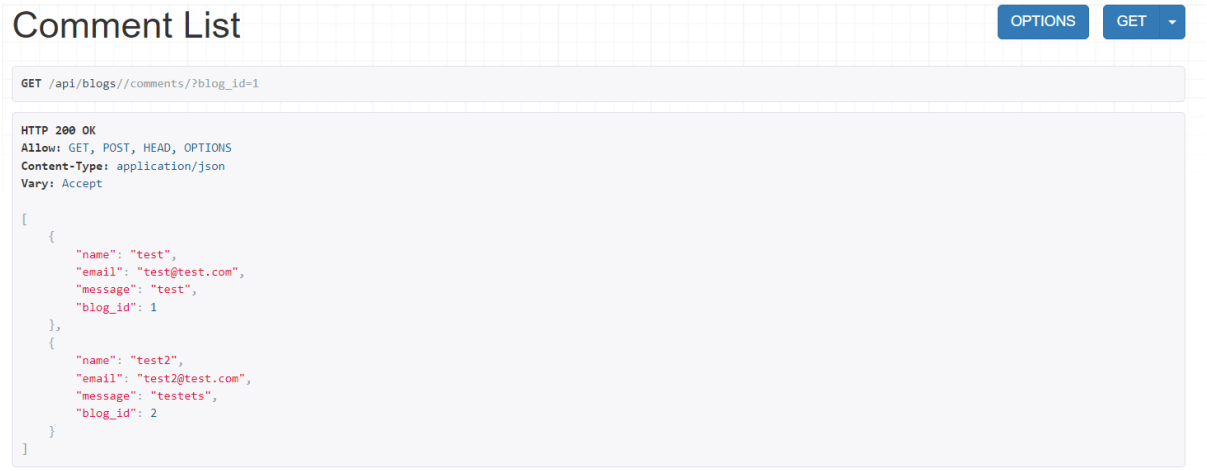


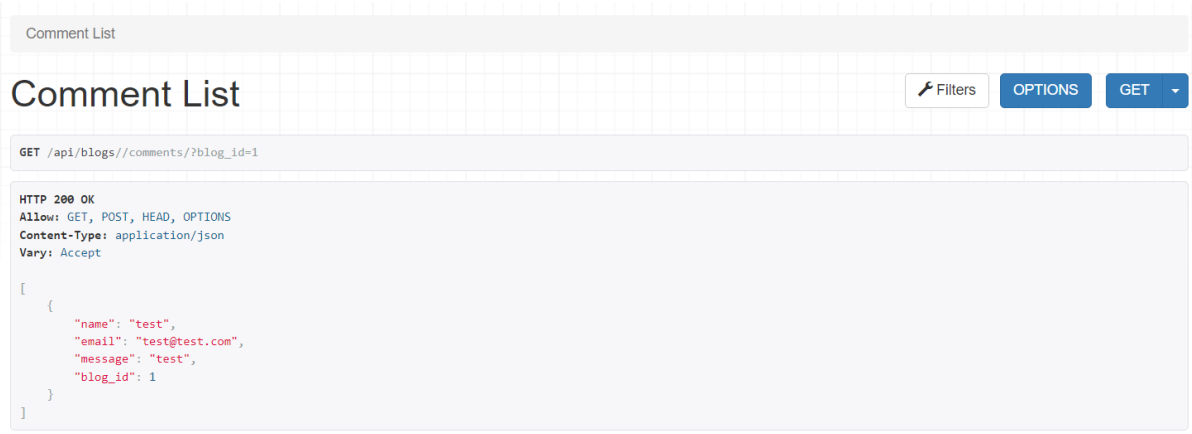
Figure 35 comment list

As we can see even if I put ?id = 1 in search , I was still showing all comments. To fix this I simply edited the filter fields in the views.py and added blog_id as the search field. This field was at the blog list before which created an error. So, I edited it and added the backend filter to comment list.

```
    lookup_field =  "slug"

class CommentList(generics.ListCreateAPIView):
    queryset = Comment.objects.all()
    serializer_class = CommentSerializer
    filter_backends = [DjangoFilterBackend]
    filter_fields = ['blog_id']
```

Figure 36 filter fields



Appendix 2

Evidences of Good communication and file sharing

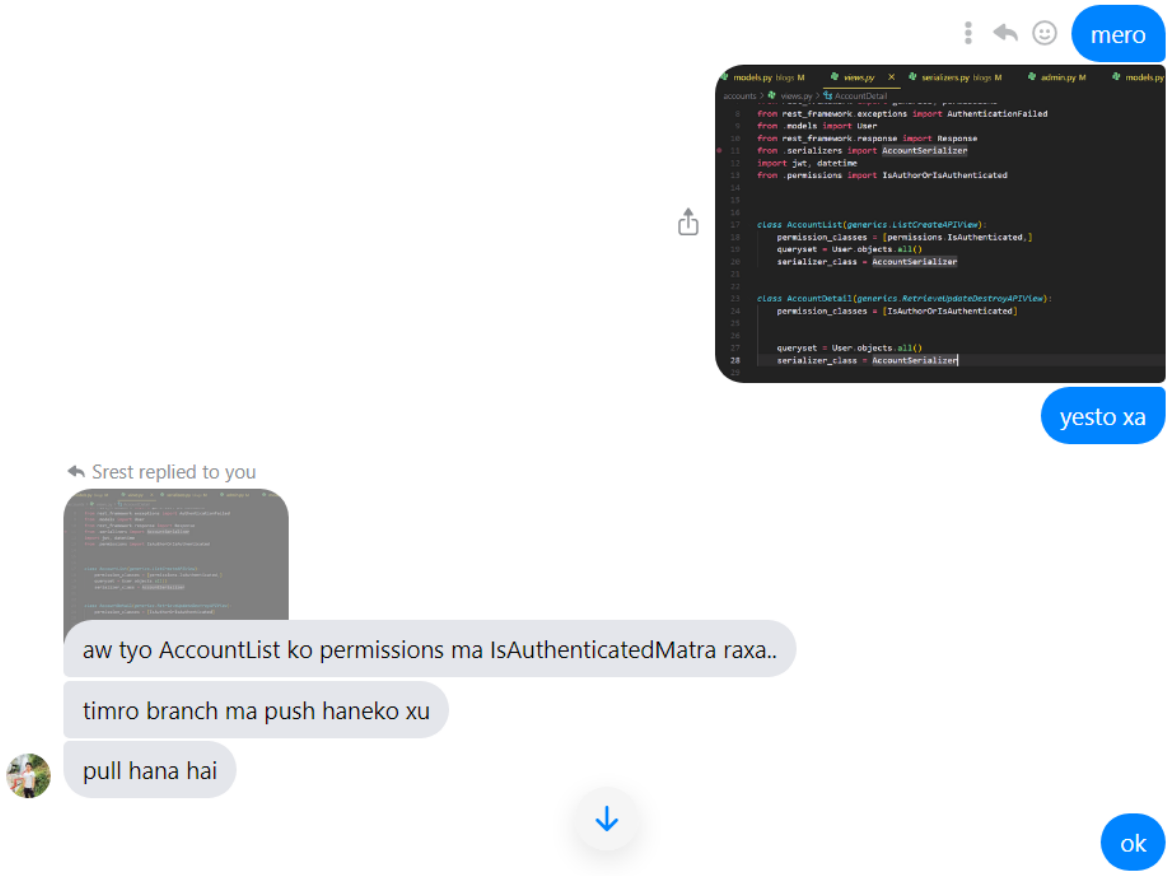


Figure 37 chatting in messenger

Figure 38 screenshot of chatting in messenger

Screenshots of members chatting in messenger.

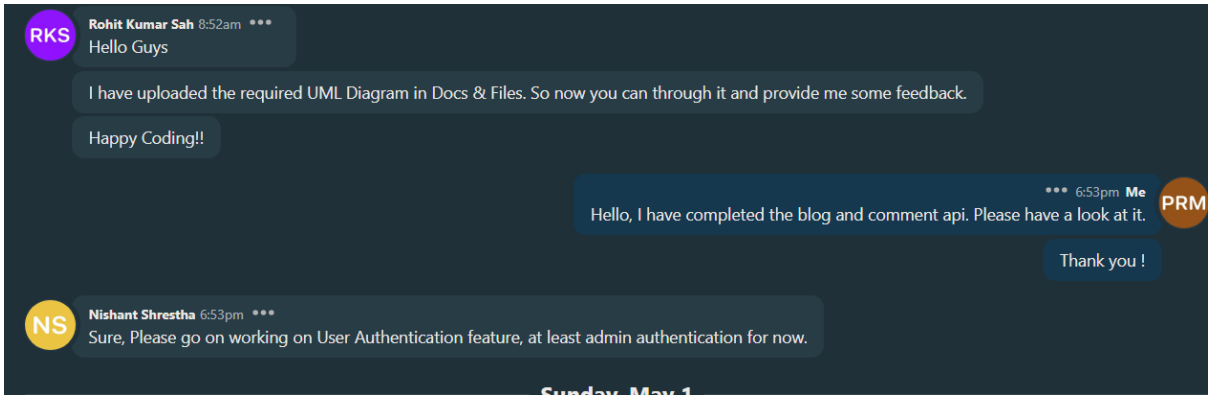


Figure 39 screenshot of chatting in basecamp

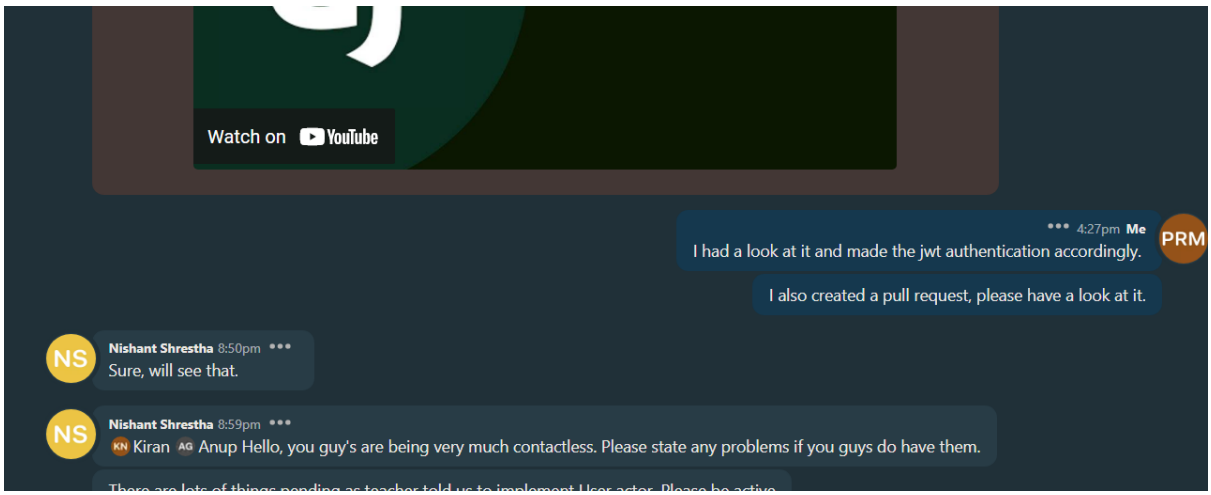
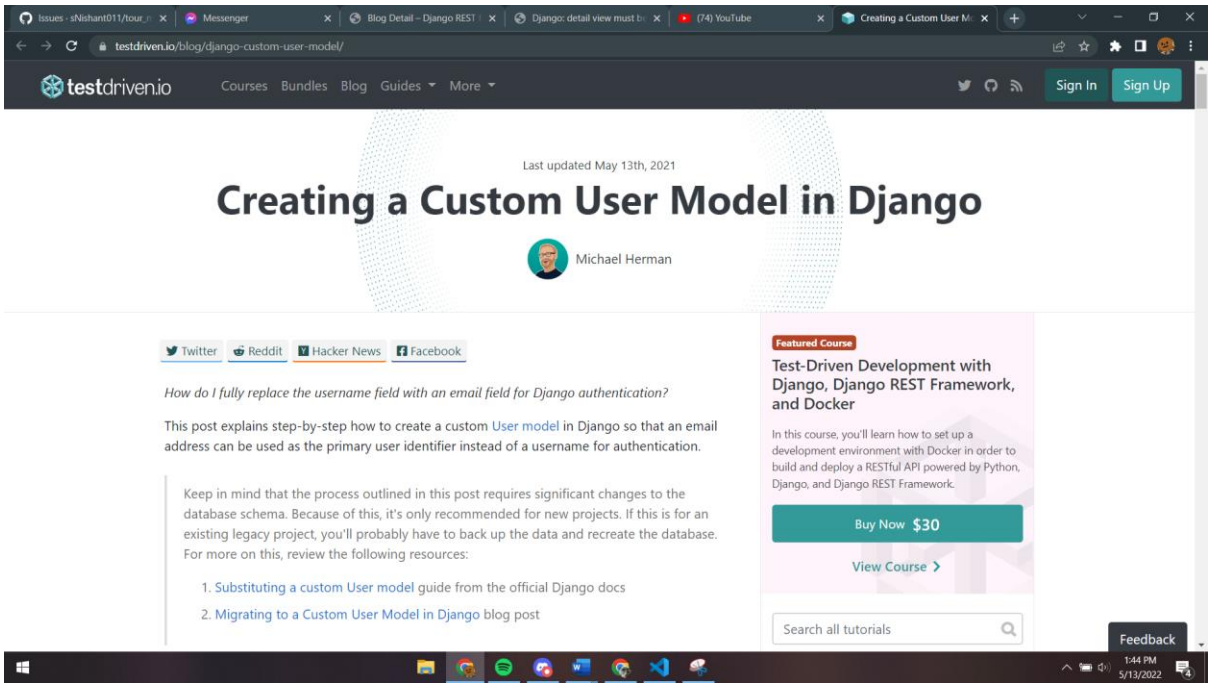


Figure 40 chatting in basecamp

In this sprint, our team members mostly had our communications physically by attending more meetings for better communication.

Evidences of Continuing Personal Development (CPD)

After doing some Django in Sprint 1, I was a little confident in doing this project than I was before. I was able to make APIs more feaster and was able to make less errors than before. I could also understand what each component of the project would do.
But still I had to learn many new things for this Sprint. I learned a lot about custom user model. Custom user model was very hard to hard and complex, I was able to learn it after watching YouTube and researching in google.



(Herman, 2021)

Figure 41 website

This website helped me the most while making a custom user model. This website had explained step by step process in making user model in detail. I was able to take some reference from here and make a user model.
I was able to learn filtering for APIs. Filtering was very necessary for the project. I could search about a blog by searching its name. I was able to learn filtering from

Django documentation.

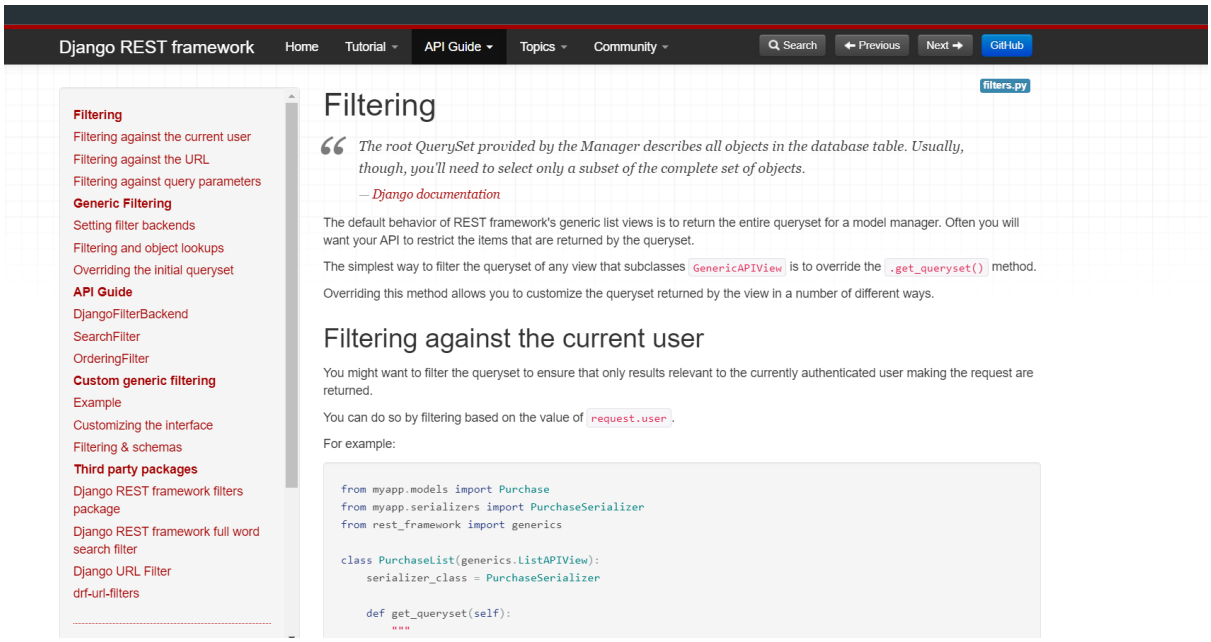
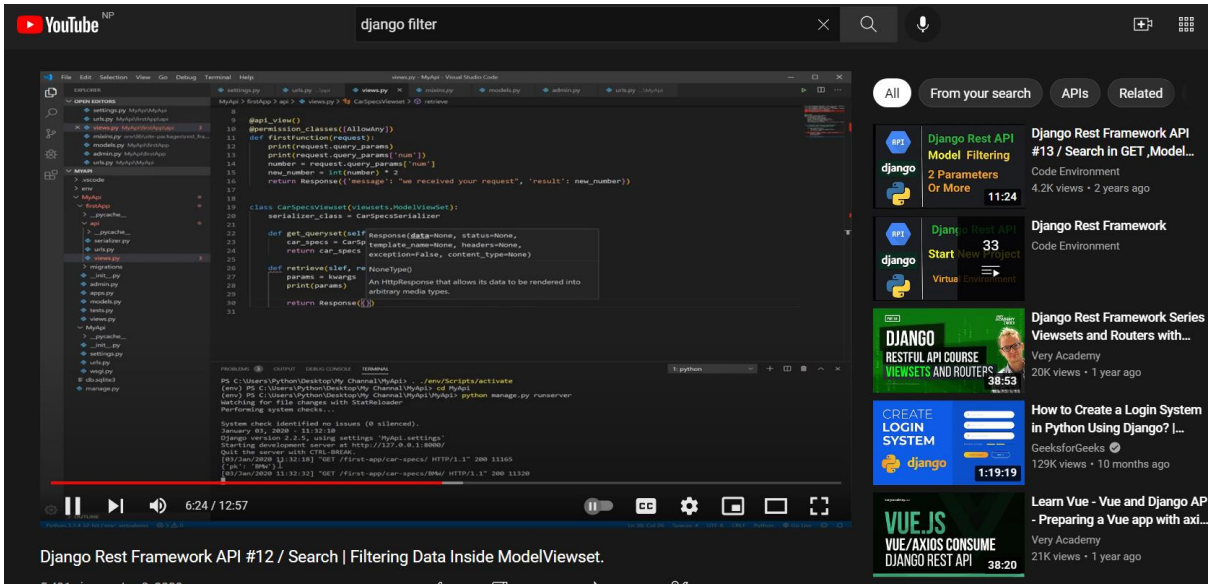


Figure 42 documentation

Django documentation had a detail guide on the filtering components it had and how to use it. Since this was not enough, I also looked into some YouTube videos on how other people used them.

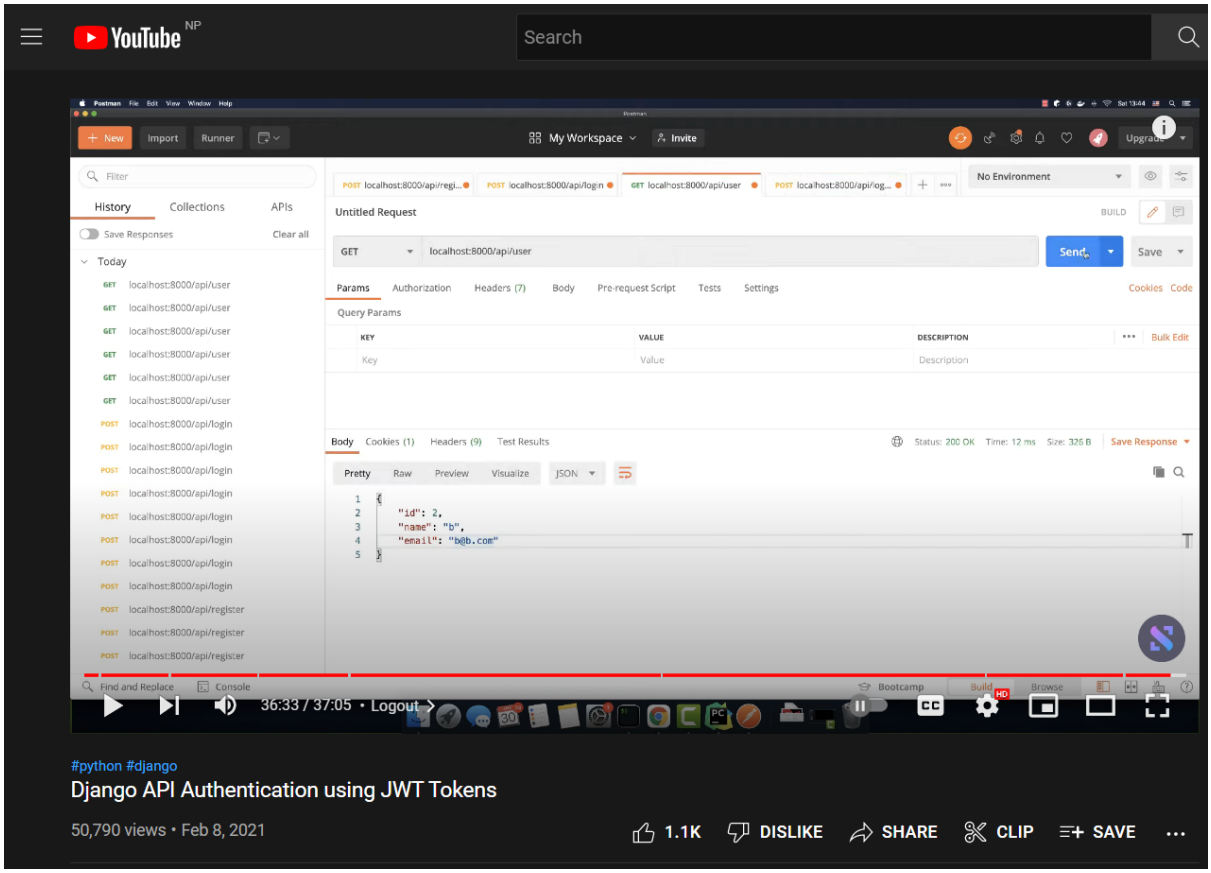


(Environment, 2020)

Figure 43 screenshot of watching YouTube video

This video was the most helpful for me. Since, it had talked about how to filter data in models which I needed. After taking the website and videos as references I was able to create a search filer for my blog.

I also learned about JWT token for Django authentication. This was the hardest to learn. Learning about jwt tokens was very hard for me since it was new to me. I learn it by watching YouTube video by Scalable Scripts.



(Scripts, 2021)

Figure 44 jwt token video

This video had a step-by-step process on making a jwt token. I learned how to use Postman with this too. I could make the login view, log out view and register view with the help of this video. I also learned about getting cookies and securing my passwords. With this I was able to make my project more secure and could timeout the logged-out users.

I also learned about authentication. I learned about permission classes through Django documentation. This documentation had detailed information about each permissions classes I could use in my project. BY taking the documentation as reference I was able to make my project more secure and make use only authenticated user could view the accounts details.

permissions.py

Permissions

> *Authentication or identification by itself is not usually sufficient to gain access to information or code. For that, the entity requesting access must have authorization.*
>
> — *Apple Developer Documentation*

Together with authentication and throttling, permissions determine whether a request should be granted or denied access.

Permission checks are always run at the very start of the view, before any other code is allowed to proceed. Permission checks will typically use the authentication information in the `request.user` and `request.auth` properties to determine if the incoming request should be permitted.

Permissions are used to grant or deny access for different classes of users to different parts of the API.

The simplest style of permission would be to allow access to any authenticated user, and deny access to any unauthenticated user. This corresponds to the `IsAuthenticated` class in REST framework.

A slightly less strict style of permission would be to allow full access to authenticated users, but allow read-only access to unauthenticated users. This corresponds to the `IsAuthenticatedOrReadOnly` class in REST framework.

How permissions are determined

Permissions in REST framework are always defined as a list of permission classes.

Before running the main body of the view each permission in the list is checked. If any permission check fails, an `exceptions.PermissionDenied` or `exceptions.NotAuthenticated` exception will be raised, and the main body of the view will not run.

When the permission checks fail, either a "403 Forbidden" or a "401 Unauthorized" response will be returned, according to the following rules:

- The request was successfully authenticated, but permission was denied. — *An HTTP 403 Forbidden response will be returned.*
- The request was not successfully authenticated, and the highest priority authentication class *does not* use `WWW-Authenticate`

(Documentation, n.d.)

Figure 45 documentation

eavesdrop on anything you do
sure why this change happened

Restore settings

Chapter 6: Permissions

Security is an important part of any website but it is doubly important with web APIs. Currently our *Blog* API allows anyone full access. There are no restrictions; any user can do anything which is extremely dangerous. For example, an anonymous user can create, read, update, or delete any blog post. Even one they did not create! Clearly we do not want this.

Django REST Framework ships with several out-of-the-box permissions settings that we can use to secure our API. These can be applied at a project-level, a view-level, or at any individual model level.

In this chapter we will add a new user and experiment with multiple permissions settings. Then we'll create our own custom permission so that only the author of a blog post has the ability to update or delete it.

Create a new user

(Vincent, 2020)

Figure 46 screenshot of permission chapter

I also looked in the Django API book written by William S. Vincent since it had a chapter regarding permissions which was helpful for learning about these permissions classes.

Evidences of Issue tracking

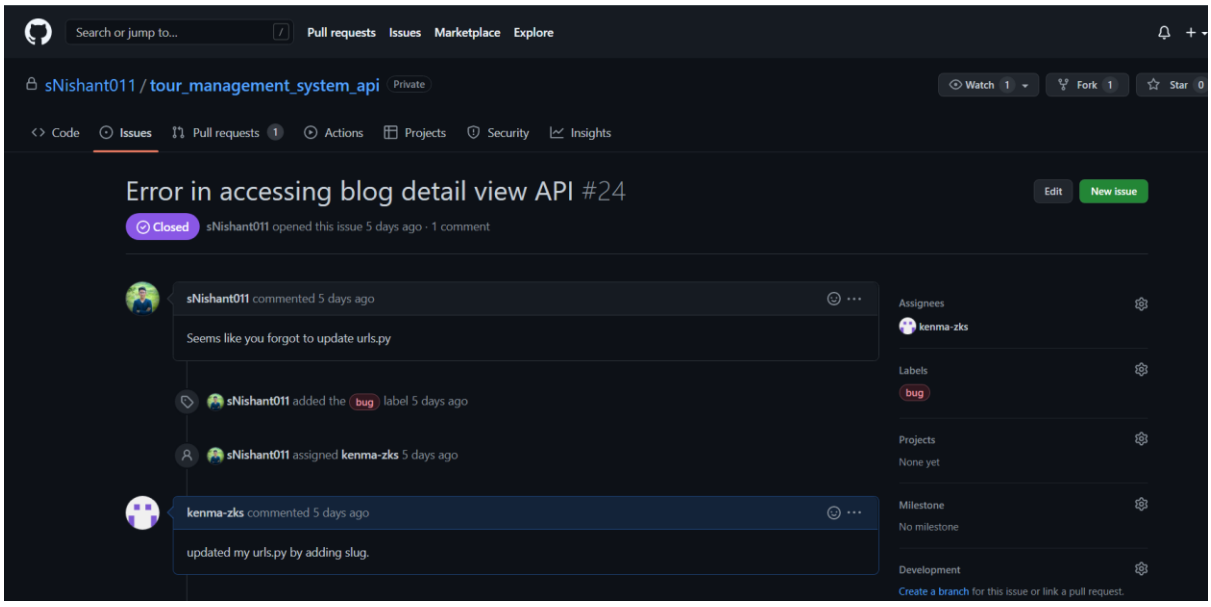


Figure 47 issue of blog detail

We had an issue regarding not being able to view the blog detail in admin panel. To fix this I simply edited the blog detail in urls.py of Blog and changed the <int:pk> to <slug:slug> in the URL patterns.

Figure 48 issue of permission classes missing

Figure 49 issue of group permissions missing

I had accidently removed the group from admin panel because I thought we were not using it. So, to fix it I simple commented my code which was to remove the group from admin panel.

```
# Remove Group Model from admin. We're not using it.
# admin.site.unregister(Group)
```

Figure 50 fixing group

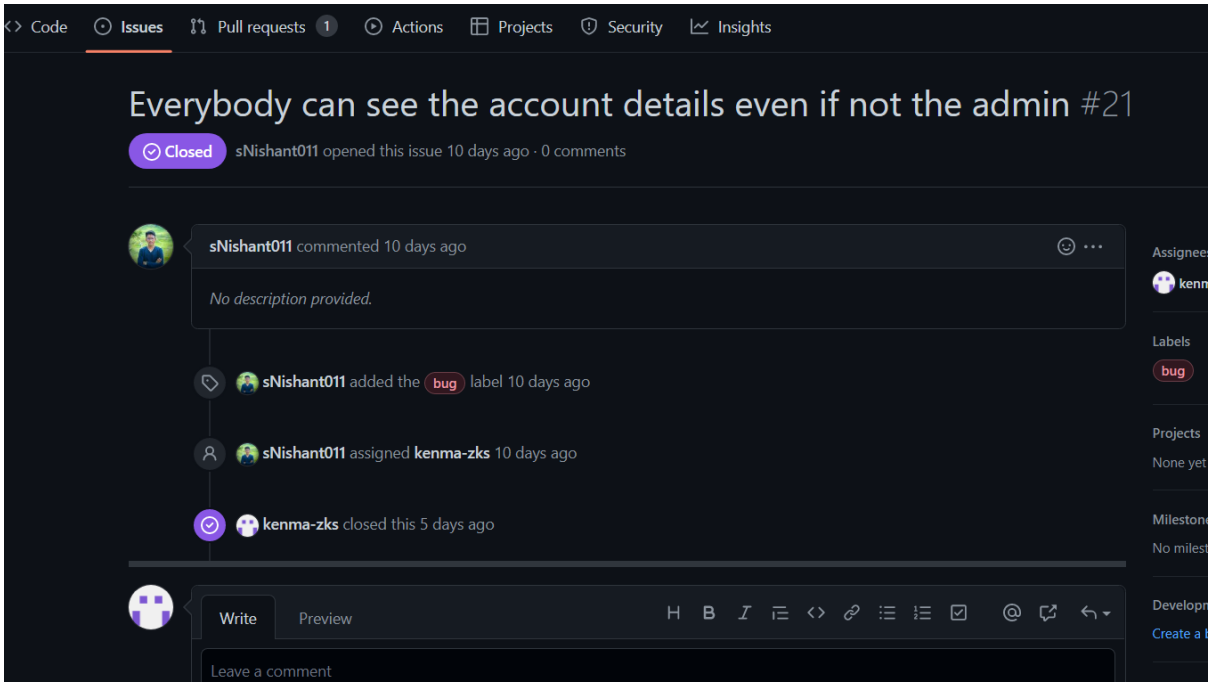


Figure 51 issue about authentication

We had this issue for not maintain proper permissions. To fix this, I created some permissions classes and made the accounts list to be viewed on if the user is authenticated and the account detail to be viewed only if the is author or authenticated. And I made a separate permissions.py file where I wrote code for class Is author or is authenticated. This made sure no one could edit other user's data and anonymous user couldn't view the data of others.

```
class AccountList(generics.ListCreateAPIView):
    permission_classes = [permissions.IsAuthenticated,]
    queryset = User.objects.all()
    serializer_class = AccountSerializer


class AccountDetail(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = [IsAuthorOrIsAuthenticated]
    queryset = User.objects.all()
    serializer_class = AccountSerializer
```

Figure 52 code to fix authentication

```python
from rest_framework import permissions

from accounts.models import User

class IsAuthorOrIsAuthenticated(permissions.BasePermission):

    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return bool(request.user and request.user.is_authenticated)
        return User.id == request.user


```

Figure 53 permissions.py

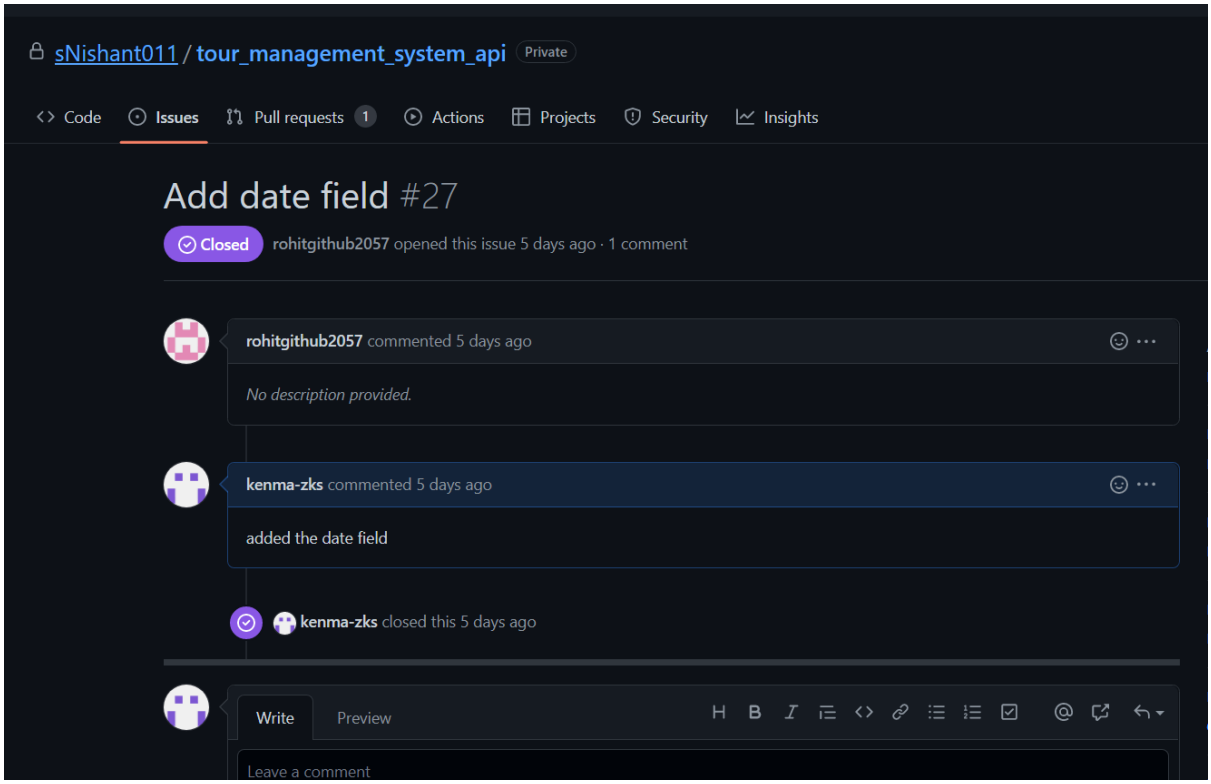


Figure 54 issue regarding date

This was a issue where even if we had a tour package or blog or booking, we could see the date. This was a major flaw since if the user wanted to book the user wouldn't see the date of the package. They wouldn't know when to book a package, when the package starts, etc. To fix this, I simply added a date field in each model of these app. I also added serializer in tour package, named date.

```python
class TourPackage(models.Model):
    title = models.CharField(max_length=50)
    slug = models.SlugField(max_length=50, blank=True)
    image = models.ImageField(blank = True, upload_to='tourpackage_images')
    thumbnail_Image = models.ImageField(blank = True, upload_to = 'tourpackag
    no_of_days = models.IntegerField()
    summary = models.TextField(blank= True)
    date = models.DateField(blank=False)
    full_detail = RichTextField(blank = True)
    is_featured = models.BooleanField(default=False)
    is_active = models.BooleanField()
    price = models.IntegerField()
```

Figure 55 adding date field

```python
TourPackageSerializer(serializers.ModelSerializer):

ass Meta:
  fields = ('id', 'title', 'slug', 'image', 'thumbnail_Image', 'no_of_days', 'summary', 'date','fu
  model = TourPackage
```

Figure 56 adding date in serializers.py

References

Documentation. (n.d.). Retrieved from https://docs.djangoproject.com/en/4.0/

Environment, C. (2020, January 3). Retrieved from Youtube: https://www.youtube.com/watch?v=ws0jwg1J0BU&t=383s

Herman, M. (2021, May 13). Retrieved from testdriven: https://testdriven.io/blog/django-custom-user-model/

Scripts, S. (2021, February 8). Retrieved from Youtube: https://www.youtube.com/watch?v=PUzgZrS_piQ

Vincent, W. S. (2020). Django For API.