**:Project Statement Assignment:**

**Library Management System**

**Brief**:

You have to build a library management system using Django where an admin can perform CRUD ( create, read, update and delete) operations like

● Add a Book

● Update an entry of a book

● Delete a book

● Get all books

Develop web services using Python (Django Framework)

and front end technologies which can be used for the front end.

1. **Admin Signup**:

a) Insert admin details in tables.

b) An admin records should be unique based on email, which means duplicates records should not be able to enter in DB.

2. **Admin Login**:

a) The Admin can log in based on email and password.

3. **Create an entry for Books**.

a) The admin can create a new entry of a book.

4. **Retrieve all the books**.

a) Retrieve all the books.

5. **Update a book**.

a) The book records should be able to update.

6. **Delete a book**.

a) The book record should be deleted

Student View:

1. View all the records of book

Conditions:

1. For Front end, you can use any library / framework to build the UI.

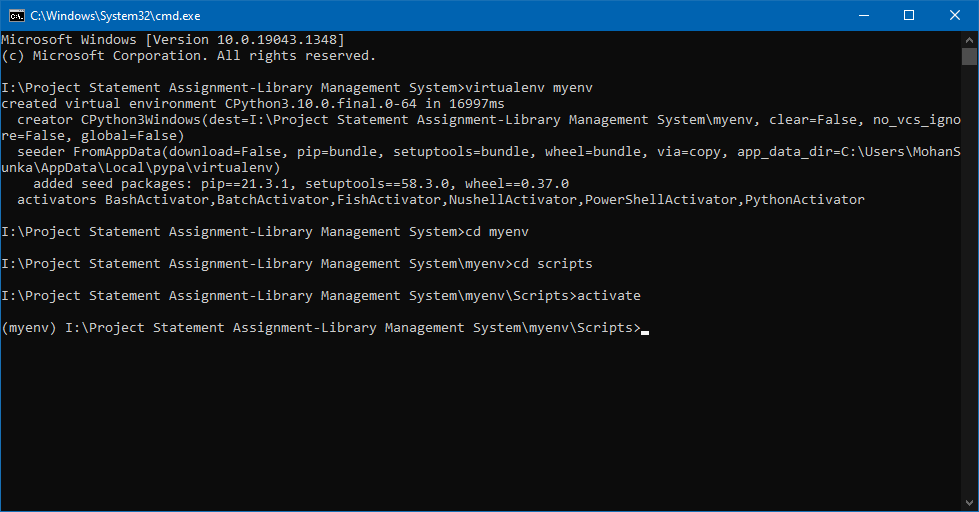
2. Write the documentation for backend code.

3. Use MySQL database to store the data

Only Backend Code for Django Rest framework

**Virtual env create**:

1. Pip install virtualenv
2. Go to cmd path open
3. Virtualenv myenv
4. Cd myenv
5. Cd scripts
6. Activate
7. pip3 freeze > requirements.txt # Python3
8. pip install –r requirements.txt
9. (or) And pip install django djangorestframework djangorestframework-simplejwt



**Projects create:**

1. **I:\ProjectStatementAssignment-LibraryManagementSystem>django-admin startproject Library\_Management\_System**
2. **Cd Library\_Management\_System**
3. **I:\ProjectStatementAssignment-LibraryManagementSystem\Library\_Management\_System>python manage.py runserver**
4. **I:\ProjectStatementAssignment-LibraryManagementSystem\Library\_Management\_System>python manage.py makemigratios**
5. **I:\ProjectStatementAssignment-LibraryManagementSystem\Library\_Management\_System>python manage.py migrate**
6. **I:\ProjectStatementAssignment-LibraryManagementSystem\Library\_Management\_System>python manage.py createsuperuser**

1. **Admin Signup**:

a) Insert admin details in tables.

b) An admin records should be unique based on email, which means duplicates records should not be able to enter in DB.

Default admin page login and register with username and password .

Custom user using so config in settings.py

# User Model Config

AUTH\_USER\_MODEL = 'ACCOUNTS.CustomUser'

And the basically using

In this admin and student user type is there so, I can create custom user create model and model manger

1. python manage.py startapp ACCOUNTS
2. python manage.py startapp BOOKS

**settings.py**

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    # APPS

    'ACCOUNTS.apps.AccountsConfig',

    'BOOKS.apps.BooksConfig',

    # 3rd party modules

    'rest\_framework',

    'rest\_framework\_simplejwt',

]

**Database settings.py in configuration.**

1. # this is using for pip install pymsql or pip3 install mysqlclient
2. DATABASES = {
3. 'default': {
4. 'ENGINE': 'django.db.backends.mysql',
5. 'NAME': 'lms\_db',
6. 'USER': 'root',
7. 'PASSWORD': '123456',
8. 'HOST': '127.0.0.1',
9. 'PORT': '3306',
10. }

2. **Admin Login**:

a) The Admin can log in based on email and password.

Api login with simple jwt using installed apps in 'rest\_framework\_simplejwt',

from django.contrib import admin

from django.urls import path,include

from rest\_framework\_simplejwt.views import (

    TokenObtainPairView,

    TokenRefreshView,

)

urlpatterns = [

    path('admin/', admin.site.urls),

    # api signup and signin

    path('login/token', TokenObtainPairView.as\_view(), name='email\_login'),

    path('login/token/refresh', TokenRefreshView.as\_view(), name='refresh\_token'),

    # app urls

    path('',include('ACCOUNTS.urls')),

    path('',include('BOOKS.urls')),

    path('api-auth/', include('rest\_framework.urls', namespace='rest\_framework'))

]

from datetime import timedelta

# REST Settings

APPEND\_SLASH = False

# JWT Settings

# REST Settings  
APPEND\_SLASH = False  
# JWT Settings  
# Django project settings.py  
  
from datetime import timedelta  
...  
  
SIMPLE\_JWT = {  
 'ACCESS\_TOKEN\_LIFETIME': timedelta(minutes=5),  
 'REFRESH\_TOKEN\_LIFETIME': timedelta(days=1),  
 'ROTATE\_REFRESH\_TOKENS': False,  
 'BLACKLIST\_AFTER\_ROTATION': False,  
 'UPDATE\_LAST\_LOGIN': False,  
  
 'ALGORITHM': 'HS256',  
 'SIGNING\_KEY': SECRET\_KEY,  
 'VERIFYING\_KEY': None,  
 'AUDIENCE': None,  
 'ISSUER': None,  
 'JWK\_URL': None,  
 'LEEWAY': 0,  
  
 'AUTH\_HEADER\_TYPES': ('Bearer',),  
  
}

REST\_FRAMEWORK = {

    'DEFAULT\_AUTHENTICATION\_CLASSES': ['rest\_framework\_simplejwt.authentication.JWTAuthentication'],

}

MVT IS DJANGO FALLOWS AND MVS IS DJANGO REST FRAMEWORK

**ACCOUNTS:**

1. **ACCOUNTS /MODELS.PY**
2. from django.db import models
3. from django.utils.translation import ugettext\_lazy as \_
4. from django.db import models
5. from django.contrib.auth.models import AbstractUser
6. from .managers import CustomUserManager
7. # Create your models here.
8. class CustomUser(AbstractUser):
9. """ custom user model """
10. USER = [
11. ('select', \_('SELECT')),
12. ('admin', \_('ADMIN USER')),
13. ('student', \_('STUDENT USER'))
14. ]
15. username = None
16. first\_name = None
17. last\_name = None
18. name = models.CharField(max\_length=255, null=True, blank=True)
19. email = models.EmailField(max\_length=254, unique=True)
21. type = models.CharField(max\_length=50, choices=USER,default='select')
22. USERNAME\_FIELD = 'email'
23. REQUIRED\_FIELDS = ['name',]
24. objects = CustomUserManager()
25. def \_\_str\_\_(self):
26. return str(self.email)
27. def create\_user(self):
28. pass
29. class Meta:
30. db\_table = 'tbl\_user'
31. verbose\_name = 'User'
32. ordering = ['-date\_joined']
33. **ACCOUNTS /VIEWS.PY**
34. from django.shortcuts import render
35. # Create your views here.
36. from ACCOUNTS.serializers import SignupSerializer
37. from ACCOUNTS.models import CustomUser
38. from rest\_framework import viewsets
39. from rest\_framework import permissions
40. from django.contrib.auth import get\_user\_model
41. User = get\_user\_model()
42. class UserViewSet(viewsets.ModelViewSet):
43. """
44. API endpoint that allows users to be viewed or edited.
45. """
46. queryset = User.objects.all().order\_by('-date\_joined')
47. serializer\_class =SignupSerializer
48. permission\_classes = [permissions.IsAdminUser]
49. class SignupUserViewSet(viewsets.ModelViewSet):
50. """
51. API endpoint that allows users to be viewed or edited.
52. """
53. queryset = User.objects.all().order\_by('-date\_joined')
54. serializer\_class =SignupSerializer
56. def get\_permissions(self):
57. permission\_classes = []
58. if self.action == 'create':
59. permission\_classes = [permissions.AllowAny] # signup all users with limited permissions allow only
60. elif self.action == 'list':
61. permission\_classes = [permissions.IsAdminUser | permissions.DjangoModelPermissions] # admin only .
62. elif self.action == 'retrieve' or self.action == 'update' or self.action == 'partial\_update':
63. permission\_classes = [permissions.IsAdminUser | permissions.DjangoModelPermissions ]  # admin only
64. elif self.action == 'destroy':
65. permission\_classes = [permissions.IsAdminUser | permissions.DjangoModelPermissions ]  # admin only
66. return [permission() for permission in permission\_classes]
67. **ACCOUNTS /SERIALIZERS.PY**
68. from rest\_framework import serializers
69. from ACCOUNTS.models import CustomUser
70. from django.contrib.auth.hashers import make\_password
71. from django.contrib.auth import get\_user\_model
72. User = get\_user\_model()
73. from rest\_framework.validators import UniqueValidator
74. from django.contrib.auth.password\_validation import validate\_password
75. class SignupSerializer(serializers.ModelSerializer):
76. email = serializers.EmailField(
77. required=True,
78. validators=[UniqueValidator(queryset=User.objects.all())]
79. )
80. password = serializers.CharField(write\_only=True, required=True, validators=[validate\_password])
81. password2 = serializers.CharField(write\_only=True, required=True)
82. class Meta:
83. model = User
84. fields = ('id','name', 'password', 'password2', 'email','type','groups')
85. extra\_kwargs = {
86. 'name': {'required': True},
87. 'type': {'required': True},
88. 'groups': {'required': True}
89. }
90. def validate(self, attrs):
91. if attrs['password'] != attrs['password2']:
92. raise serializers.ValidationError({"password": "Password fields didn't match."})
93. return attrs
94. def create(self, validated\_data):
95. groups = validated\_data.pop('groups', ())
96. user = User.objects.create(
97. name=validated\_data['name'],
98. email=validated\_data['email'],
99. password=validated\_data['password'],
100. type=validated\_data['type'],
102. )
103. if user.type=='admin':
104. user.type = 'admin'
105. user.is\_staff = True
106. # add groups to the user ↓
107. user.groups.add(\*groups)
108. user.set\_password(validated\_data['password'])
109. user.save()
110. return user
112. def validated\_name(self,value):
113. if value!=None:
114. raise serializers.ValidationError("name must be enter")
115. return value
116. def validated\_groups(self,value):
117. if value!=None:
118. raise serializers.ValidationError("groups must be enter")
119. return value

**ADMIN CAN (CURD) WITH BOOKS ,** STUDENT VIEW PERMISSION:

1. View all the records of book
2. Two ways are there
3. 1)one is group based django permissions
4. 2)two is custom permissions with django restframework and particular person is view so I will create type in custom user model.

**ACCOUNTS /permissions.py**

from rest\_framework.permissions import BasePermission

class IsAdmin(BasePermission):

   def has\_permission(self, request, view):

      return bool(request.user.type=="admin")

class IsStudent(BasePermission):

   def has\_permission(self, request, view):

      return bool(request.user.type=="student")

**ACCOUNTS/urls.py**

from django.urls import include, path

from rest\_framework import routers

from ACCOUNTS import views

router = routers.DefaultRouter()

router.register(r'users', views.UserViewSet)

router.register(r'Signup', views.SignupUserViewSet)

# Wire up our API using automatic URL routing.

# Additionally, we include login URLs for the browsable API.

urlpatterns = [

    path('', include(router.urls)),

]

**Books app**

3. **Create an entry for Books**.

a) The admin can create a new entry of a book.

4. **Retrieve all the books**.

a) Retrieve all the books.

5. **Update a book**.

a) The book records should be able to update.

6. **Delete a book**.

a) The book record should be deleted

Student View:

1. View all the records of book

**BOOKS/models.py**

Book have name ,price, etc

But I am using only title field .

from django.db import models

# Create your models here.

class Book(models.Model):

    title = models.CharField(max\_length=100)

    def \_\_str\_\_(self):

        return self.title

    def \_\_unicode\_\_(self):

        return self.title

BOOKS/serializers.py

from rest\_framework import serializers

from BOOKS.models import Book

class BookSerializer(serializers.ModelSerializer):

    class Meta:

        model = Book

        fields = "\_\_all\_\_"

BOOKS/views.py

from django.contrib.auth.models import Permission

from django.shortcuts import render

from ACCOUNTS.permissions import IsAdmin, IsStudent

from BOOKS.models import Book

from rest\_framework import viewsets

from rest\_framework import permissions

from BOOKS.serializers import BookSerializer

# Create your views here.

class BooksViewSet(viewsets.ModelViewSet):

    """

    API endpoint that allows books to be viewed or edited.

    """

    queryset = Book.objects.all()

    serializer\_class = BookSerializer

    # permission\_classes = [IsAdmin]

def get\_permissions(self):  
 permission\_classes = []  
 if self.action == 'create':  
 # permission\_classes = [permissions.IsAdminUser,IsAdmin] # admin can create retrive update delete only  
 permission\_classes = [IsAuthenticated | IsAdmin | IsAdminUser,  
 DjangoModelPermissions] # admin can create retrive update delete only  
 elif self.action == 'list':  
 permission\_classes = [  
 IsAuthenticated| IsStudent | IsAdmin | IsAuthenticatedOrReadOnly | IsAdminUser | DjangoModelPermissions] # only student books view.  
 elif self.action == 'retrieve' or self.action == 'update' or self.action == 'partial\_update':  
 permission\_classes = [IsAuthenticated| IsAdmin | IsAdminUser, DjangoModelPermissions] # admin only  
 elif self.action == 'destroy':  
 permission\_classes = [IsAuthenticated| IsAdmin | IsAdminUser, DjangoModelPermissions] # admin only  
 return [permission() for permission in permission\_classes]

BOOKS/urls.py

from django.urls import include, path

from rest\_framework import routers

from BOOKS import views

router = routers.DefaultRouter()

router.register(r'books', views.BooksViewSet)

# Wire up our API using automatic URL routing.

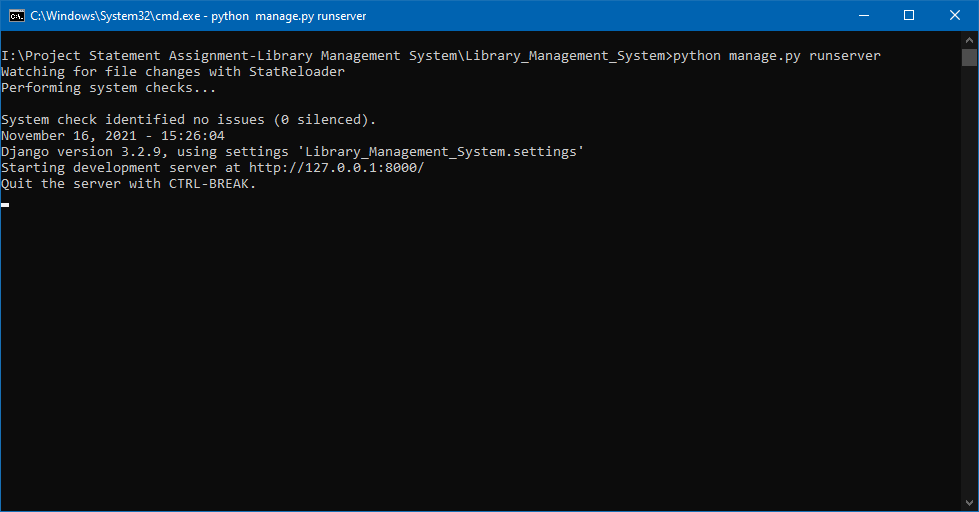
# Additionally, we include login URLs for the browsable API.

urlpatterns = [

    path('', include(router.urls)),

]

**python manage.py runserver:**

****

Api testing for postman

**Api end urls**:

ACCOUNTS APP END URLS:

1. POST FOR <http://127.0.0.1:8000/login/token>
2. GET ALL USER FOR <http://127.0.0.1:8000/users/>
3. POST , GET FOR <http://127.0.0.1:8000/users/>
4. GET ,PUT,PATCH FOR <http://127.0.0.1:8000/users/1/>
5. POST FOR <http://127.0.0.1:8000/Signup/>
6. POST <http://127.0.0.1:8000/login/token/refresh>

Note:

<http://127.0.0.1:8000/login/token>

it is post operation using give email,password then get the access token

which url you can hit then passing Autherization select in postman and access token pass and data will be accessing.

This token is time limit so time is over after then <http://127.0.0.1:8000/login/token/refresh>

Use. timedelta in days define.

# REST Settings

APPEND\_SLASH = False

# JWT Settings

SIMPLE\_JWT = {

    'ACCESS\_TOKEN\_LIFETIME': timedelta(days=30),

    'REFRESH\_TOKEN\_LIFETIME': timedelta(days=30)

}

BOOKS APP END URLS:

1. GET ALL BOOKS VIEW STUDENTS AND ADMIN <http://127.0.0.1:8000/books/>
2. POST BOOK ADMIN CAN CREATE <http://127.0.0.1:8000/books/>
3. PUT BOOK ADMIN CAN CREATE <http://127.0.0.1:8000/books/1/>
4. PATCH BOOK ADMIN CAN CREATE <http://127.0.0.1:8000/books/2/>
5. DELETE BOOK ADMIN CAN CREATE <http://127.0.0.1:8000/books/3/>

Note:

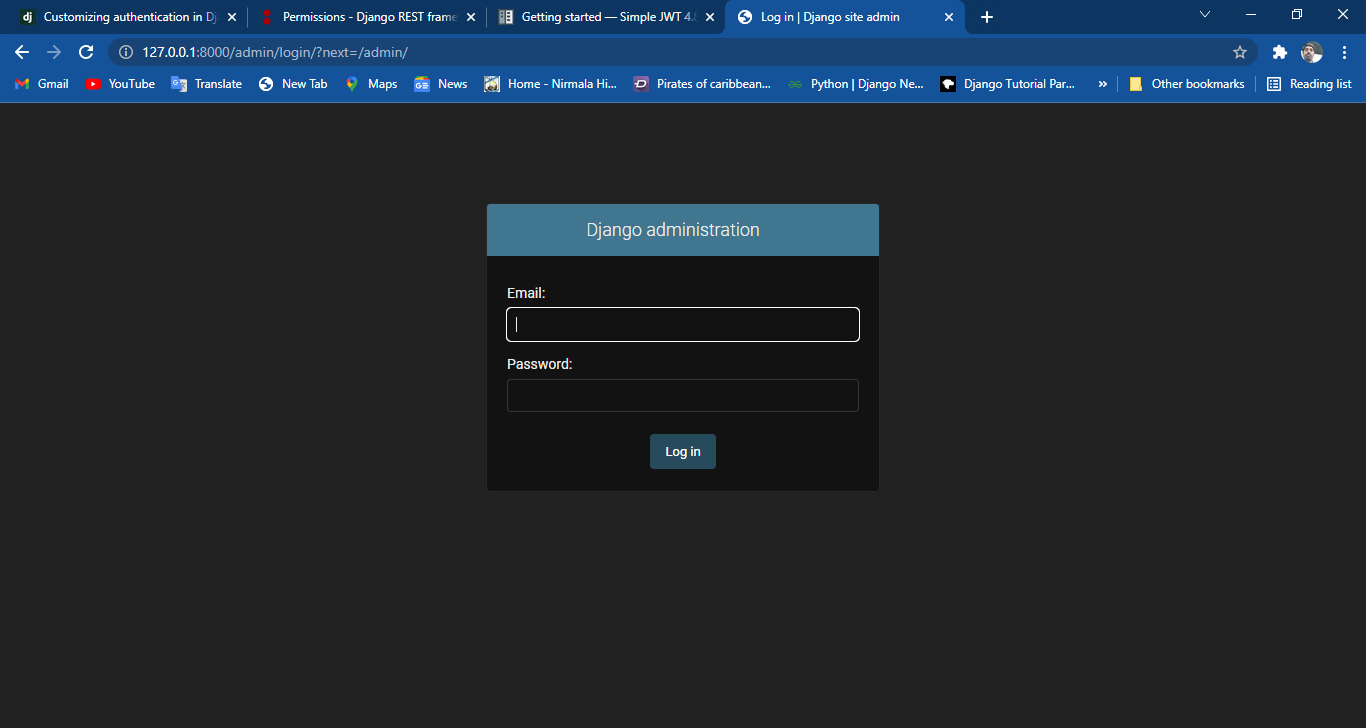
Student View:

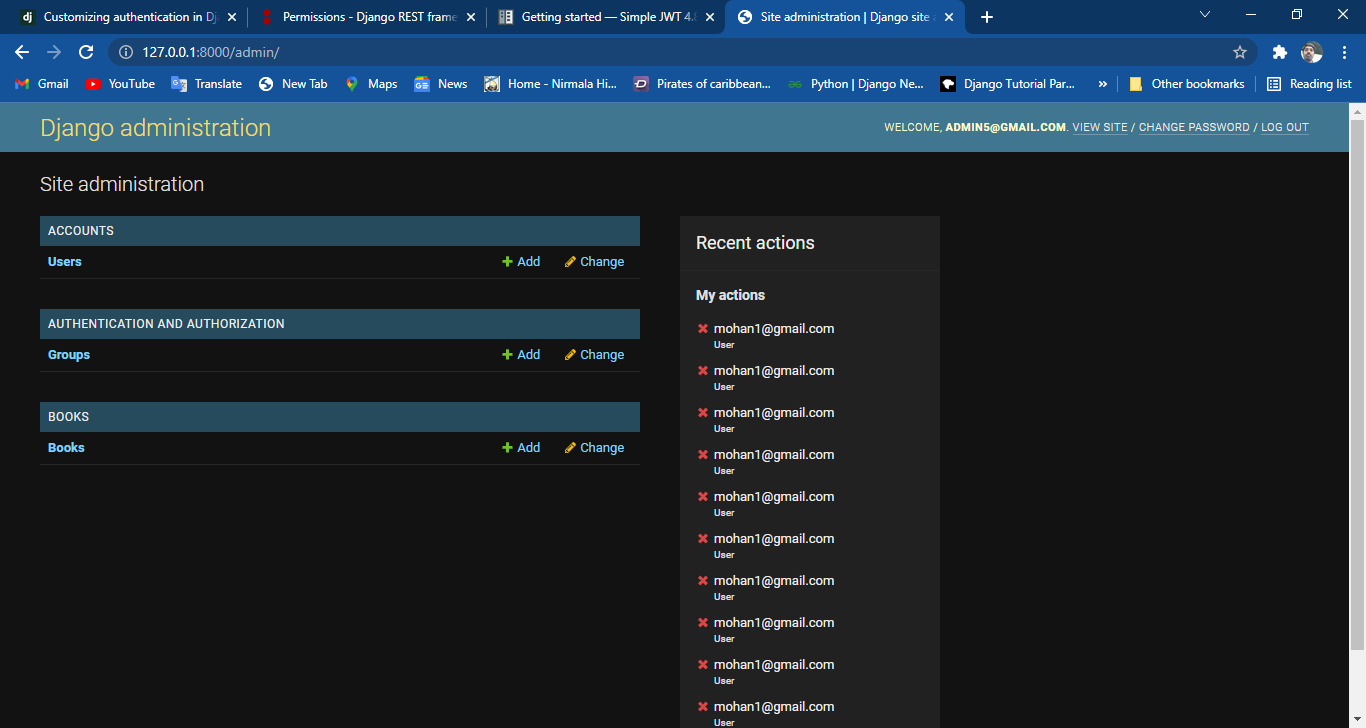
Using this url only

1. <http://127.0.0.1:8000/books/>

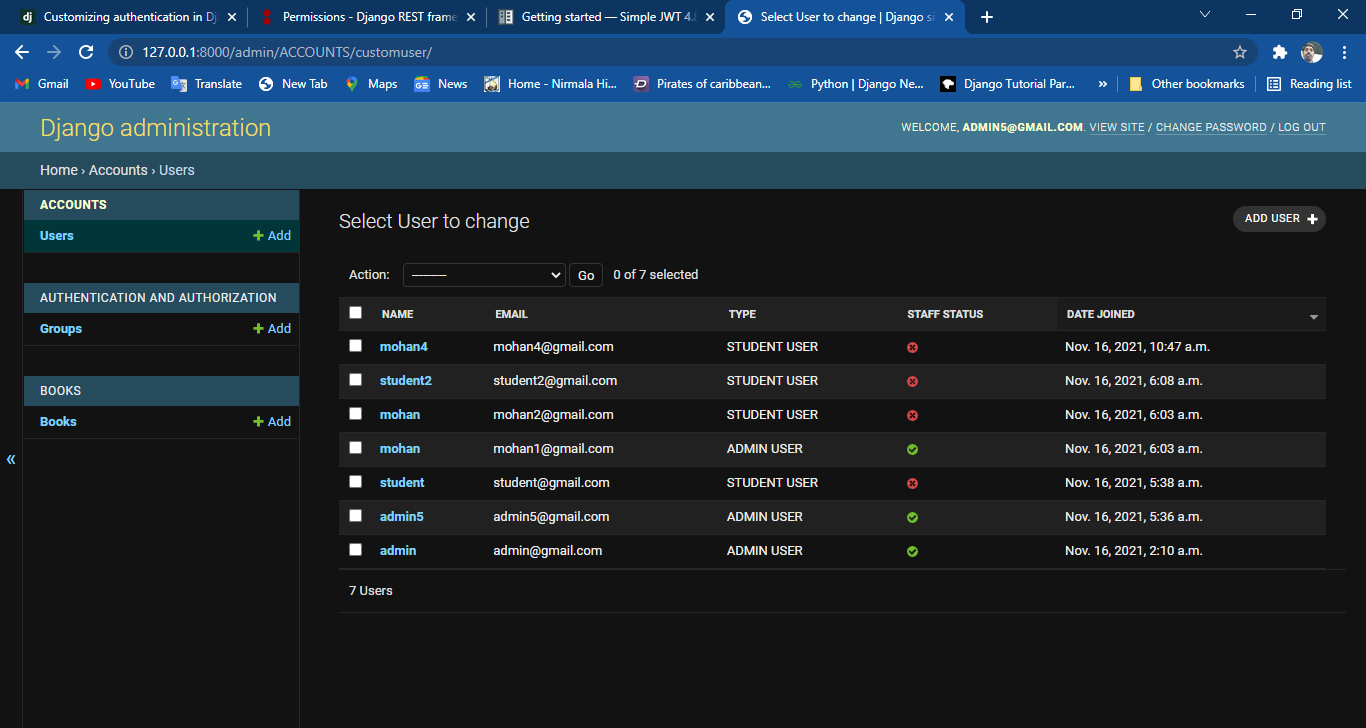
1. View all the records of book

Admin page

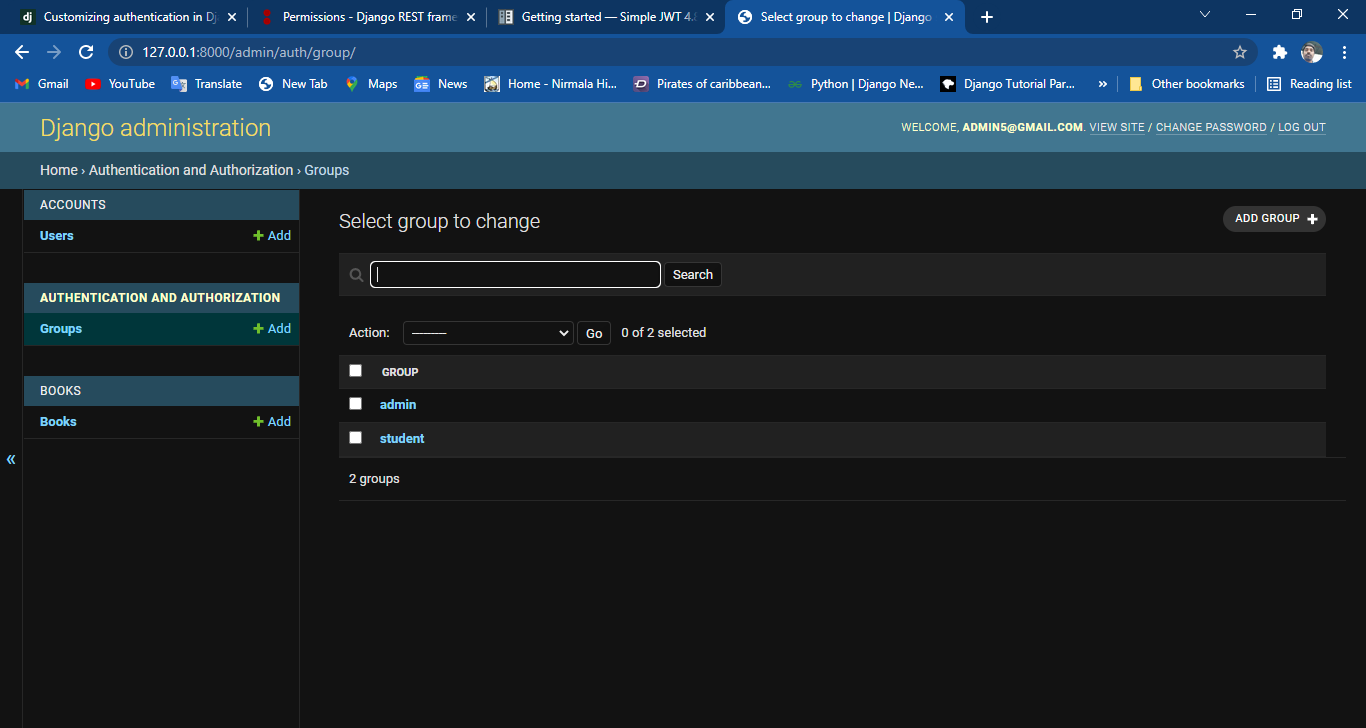


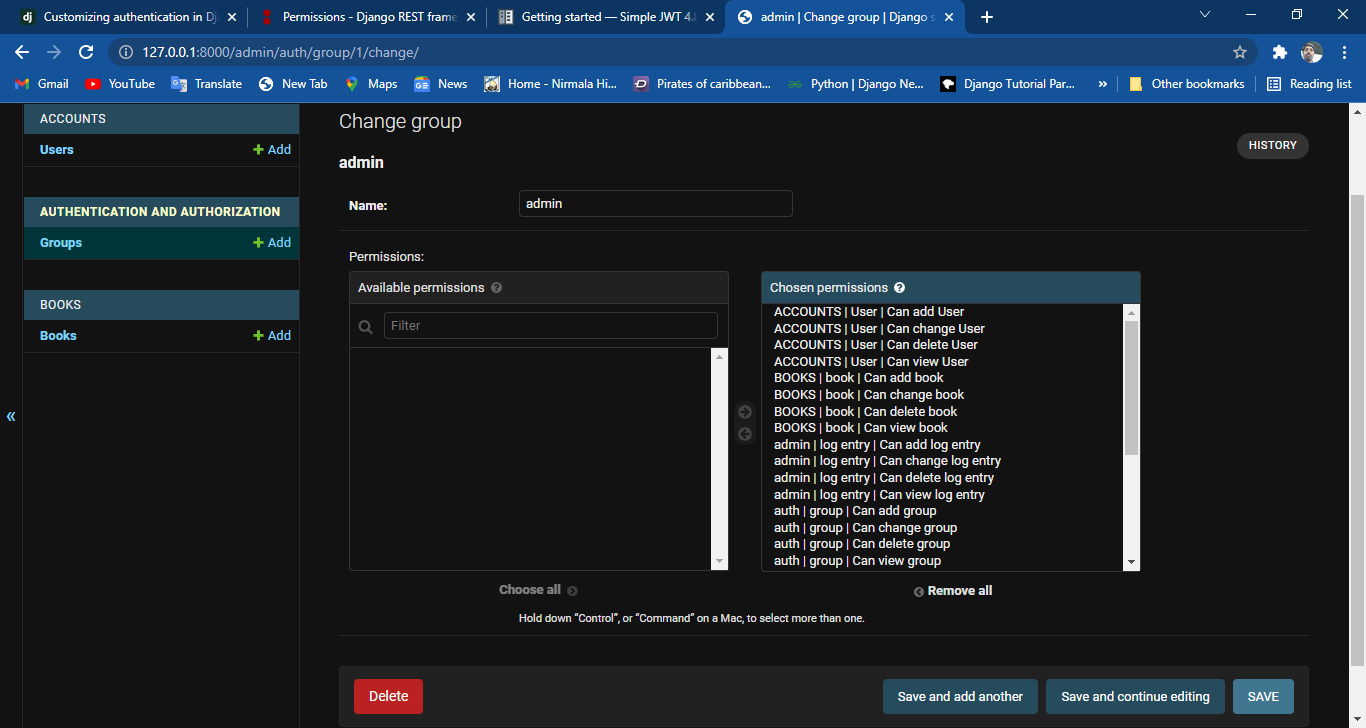


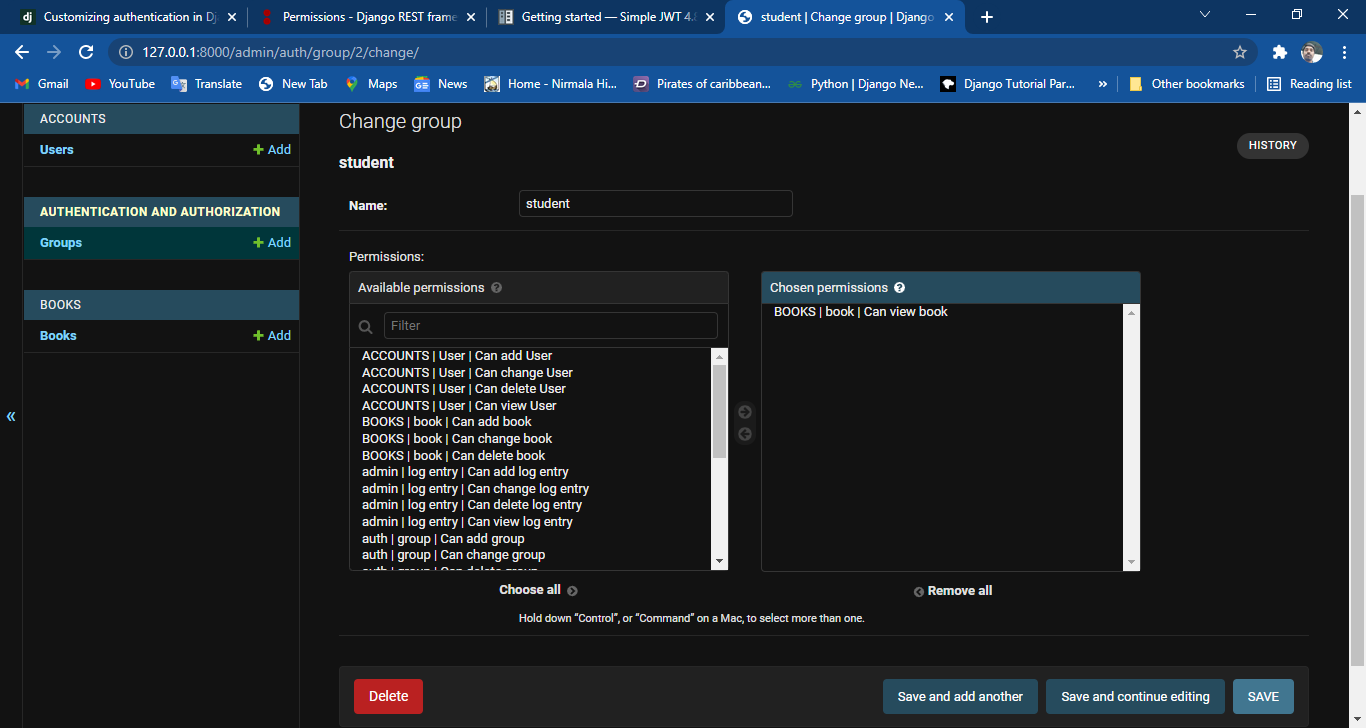
Users model



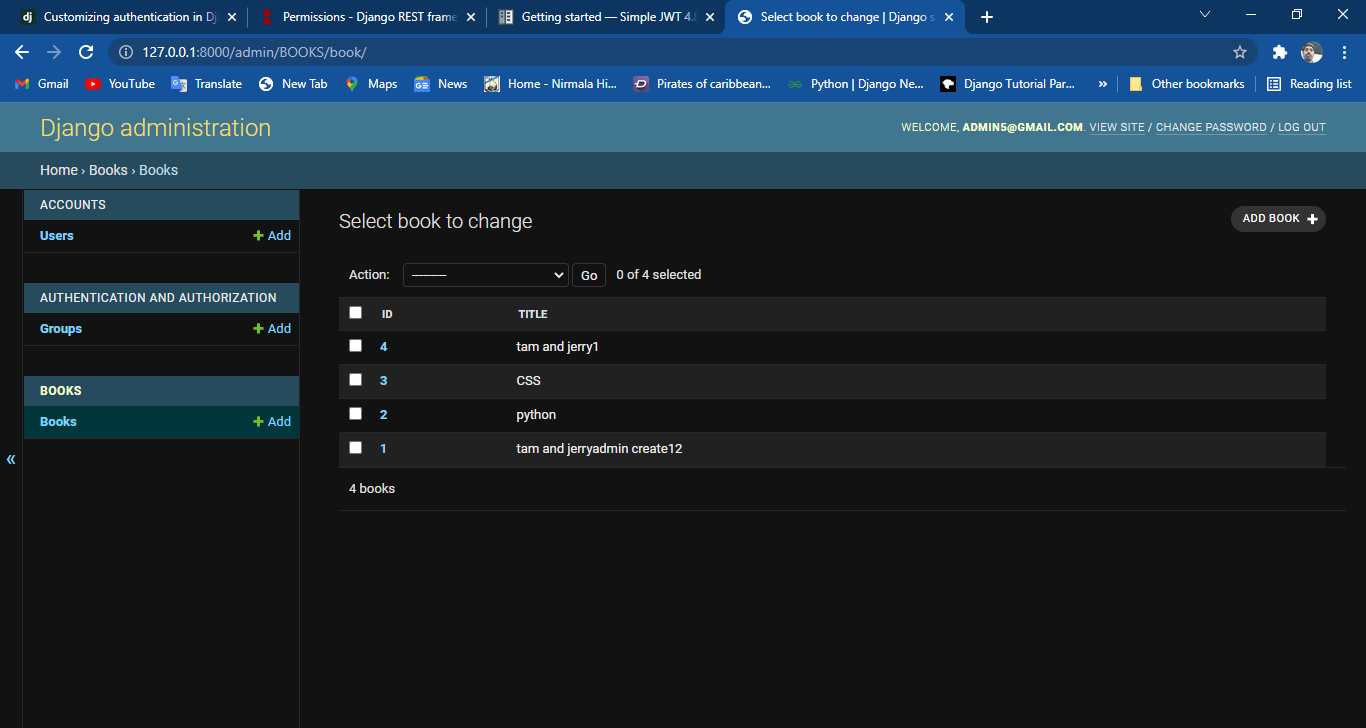
Group model





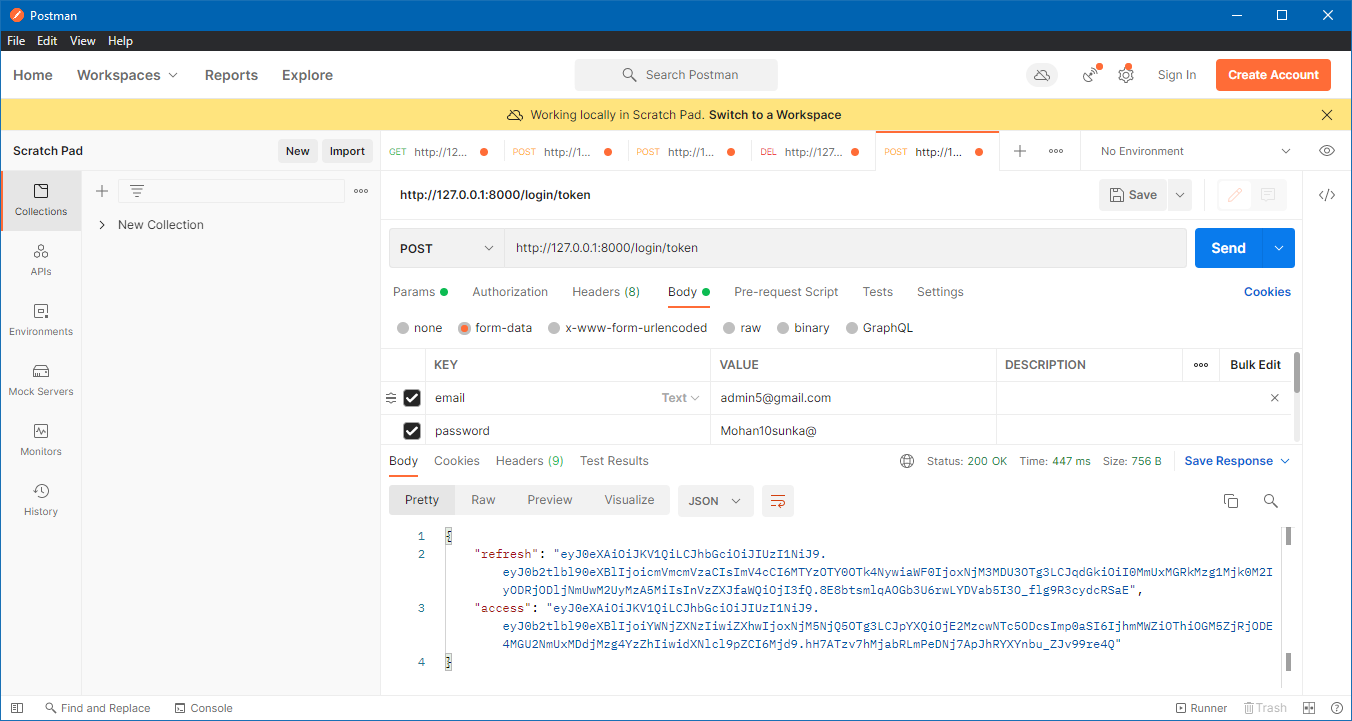


Book model

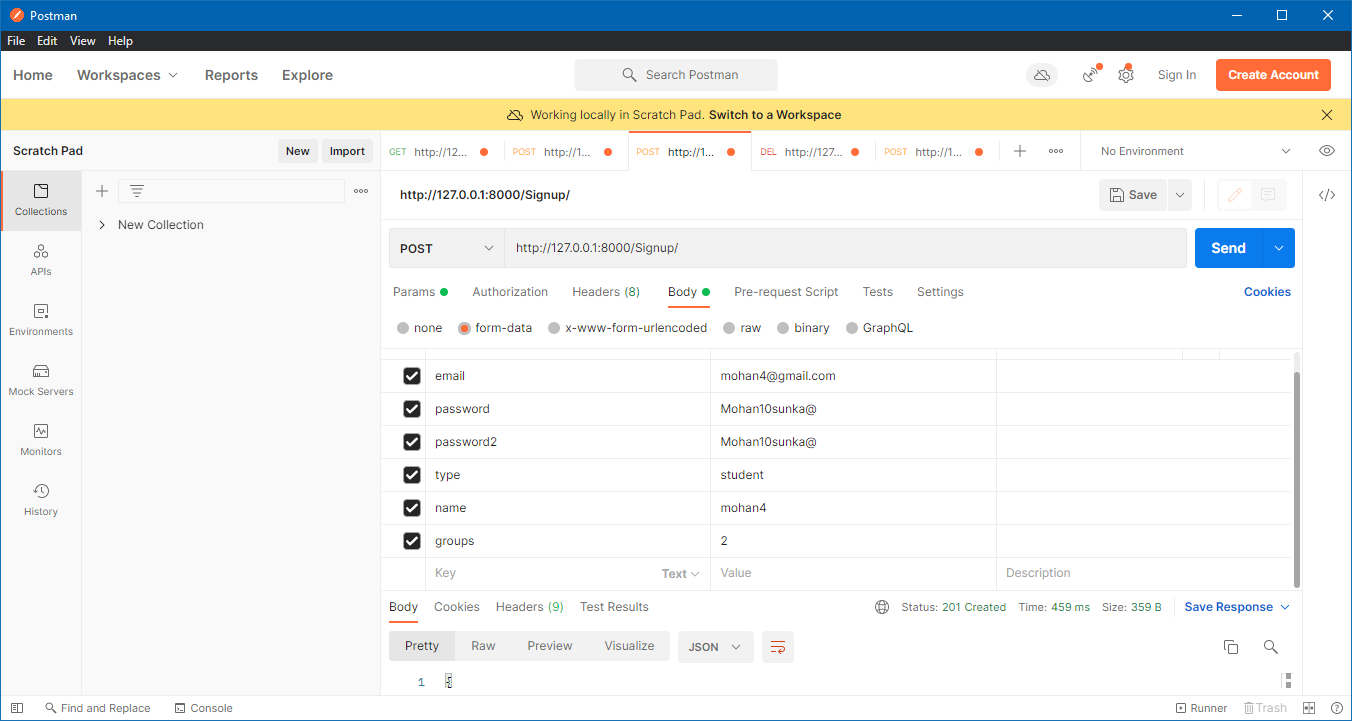


Post man

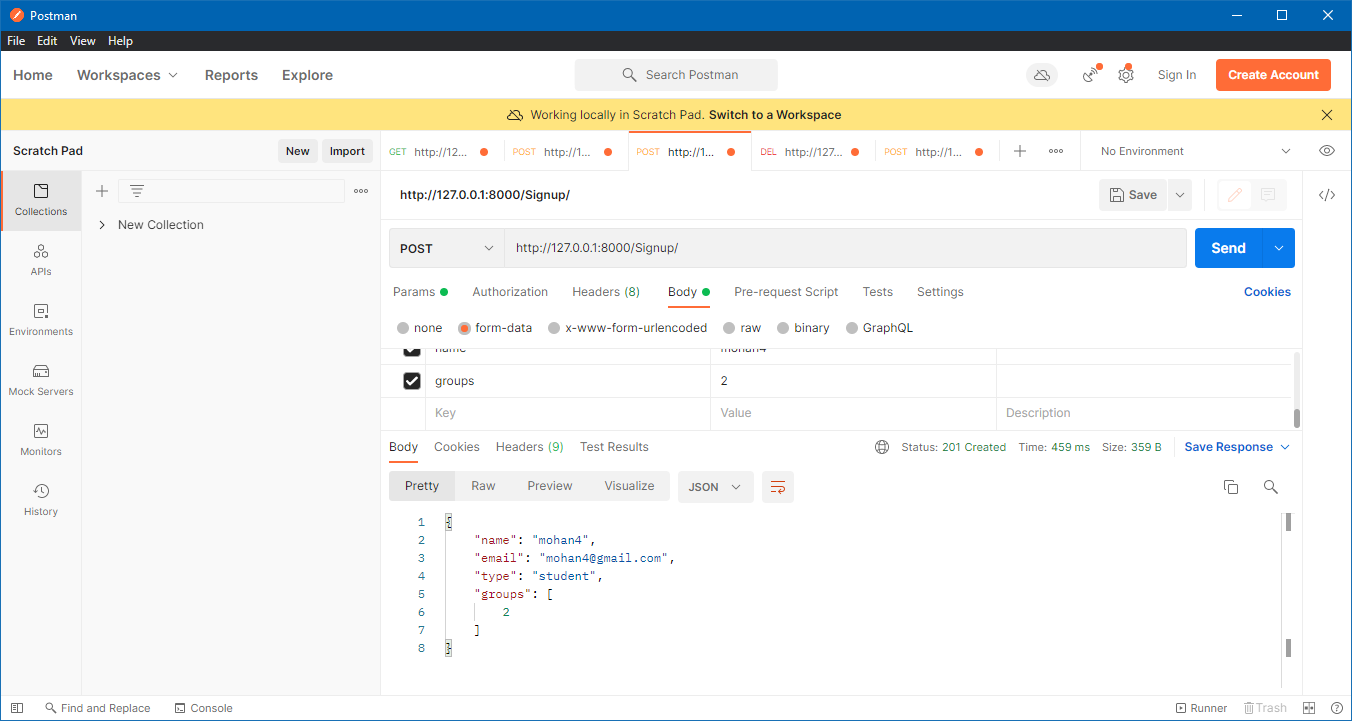
1. <http://127.0.0.1:8000/login/token>



1. <http://127.0.0.1:8000/Signup/>

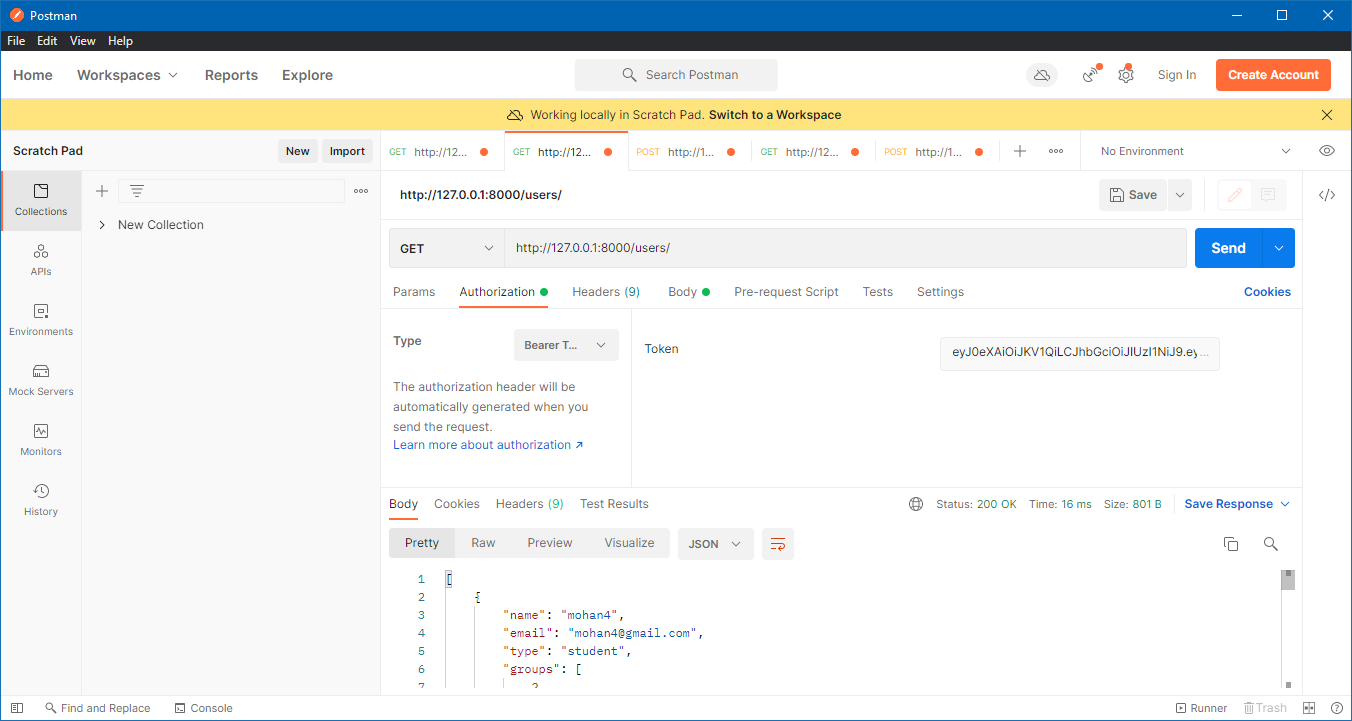


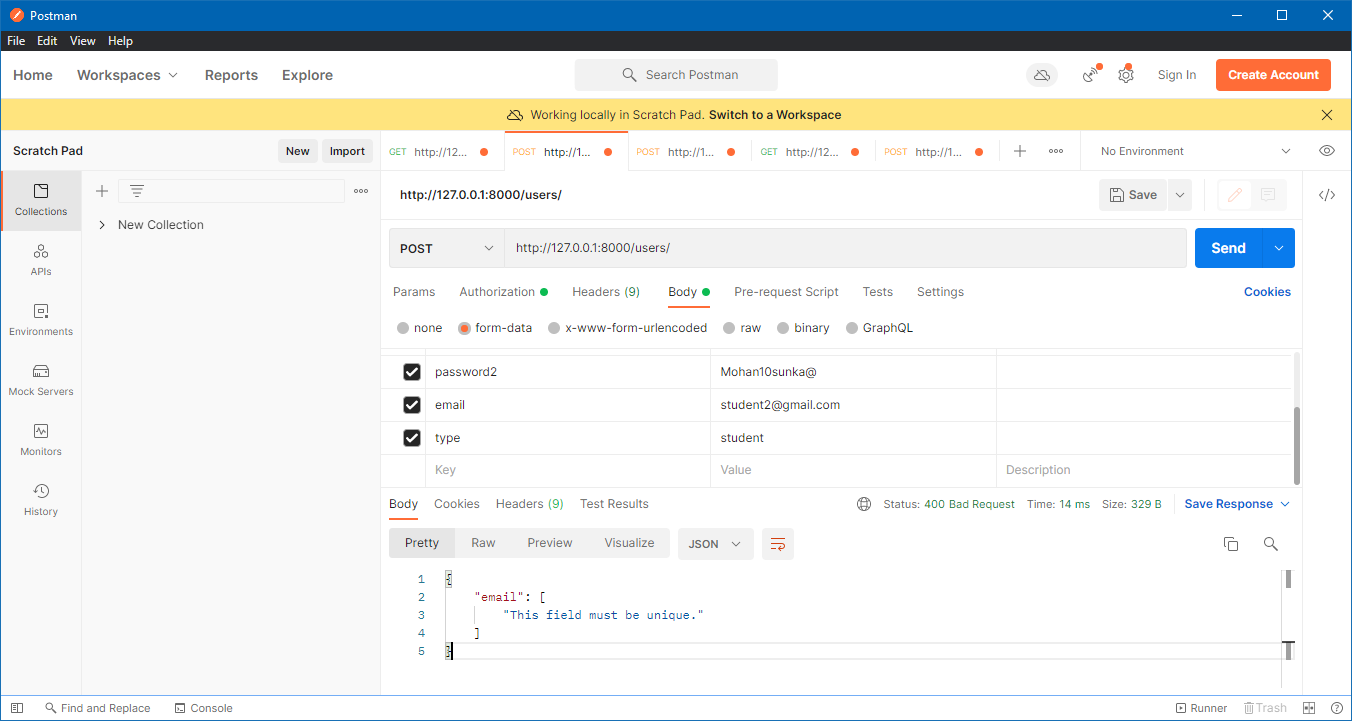
Here must be passing groups otherwise djangomodelpermissions not working so define

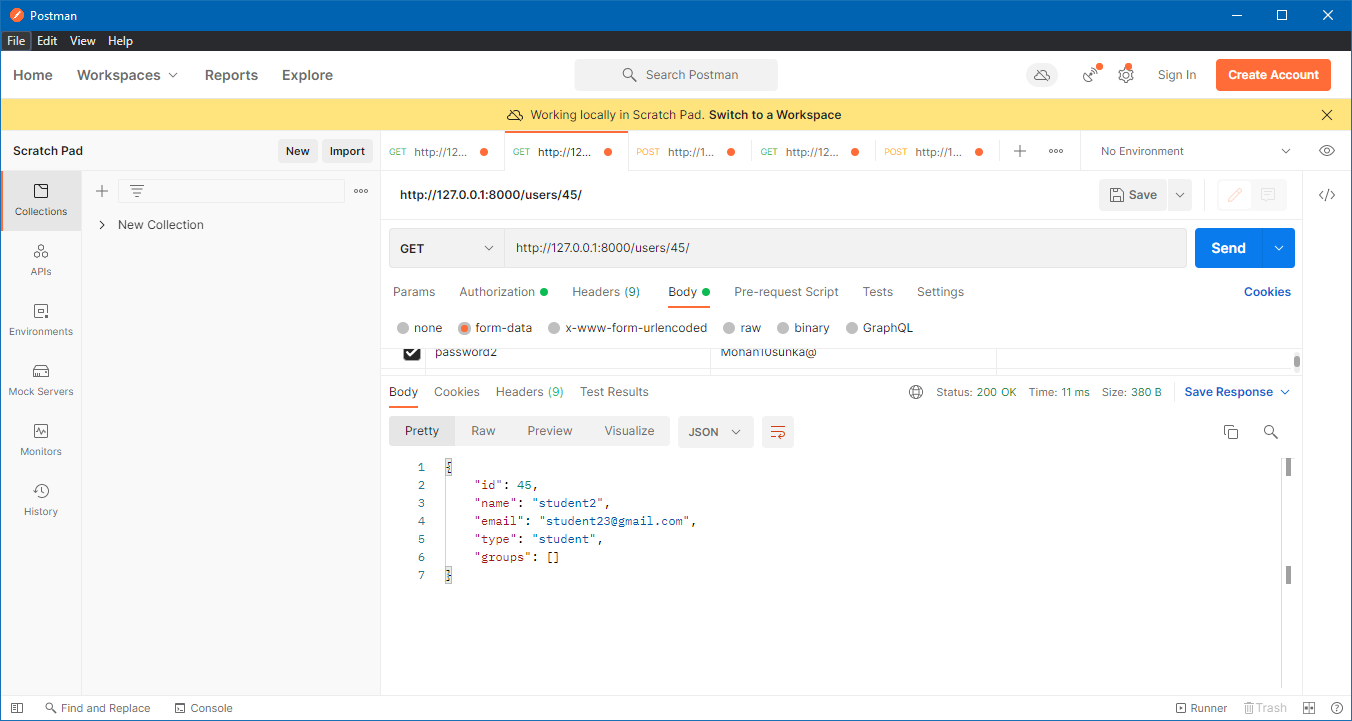


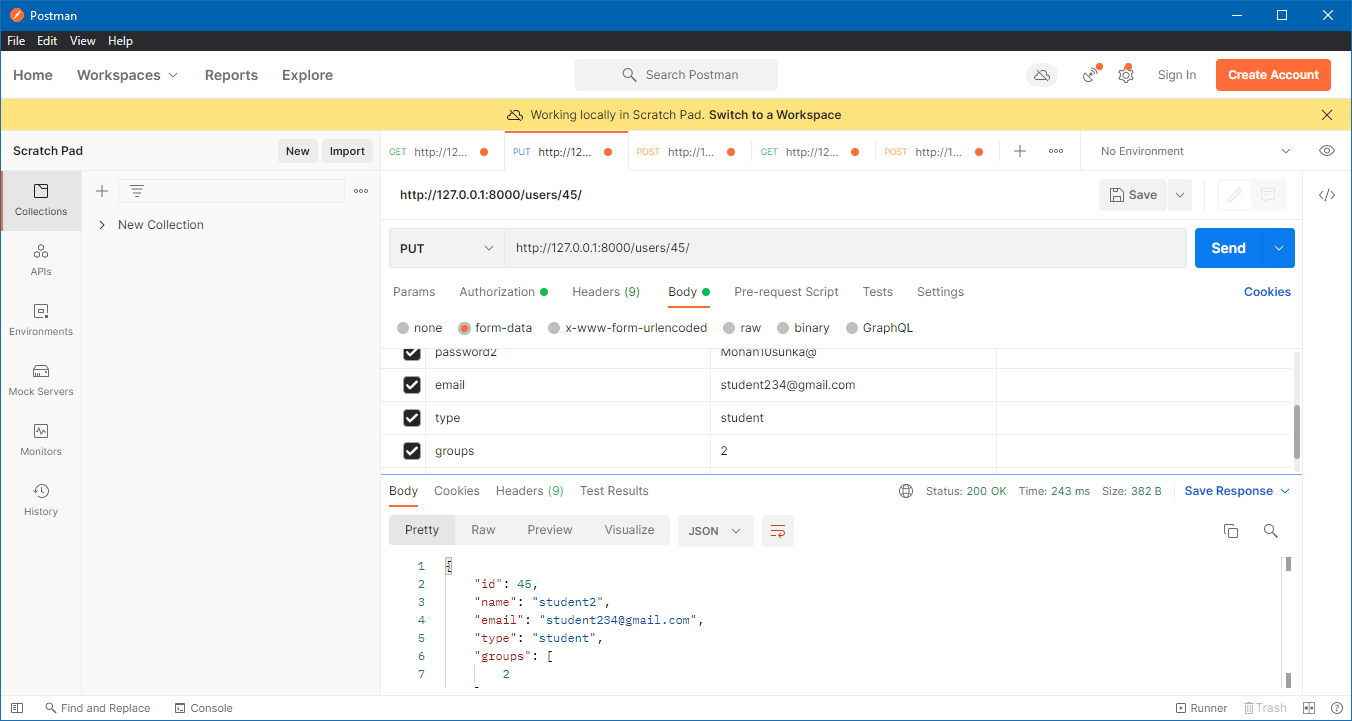
**ACCOUNTS APP**

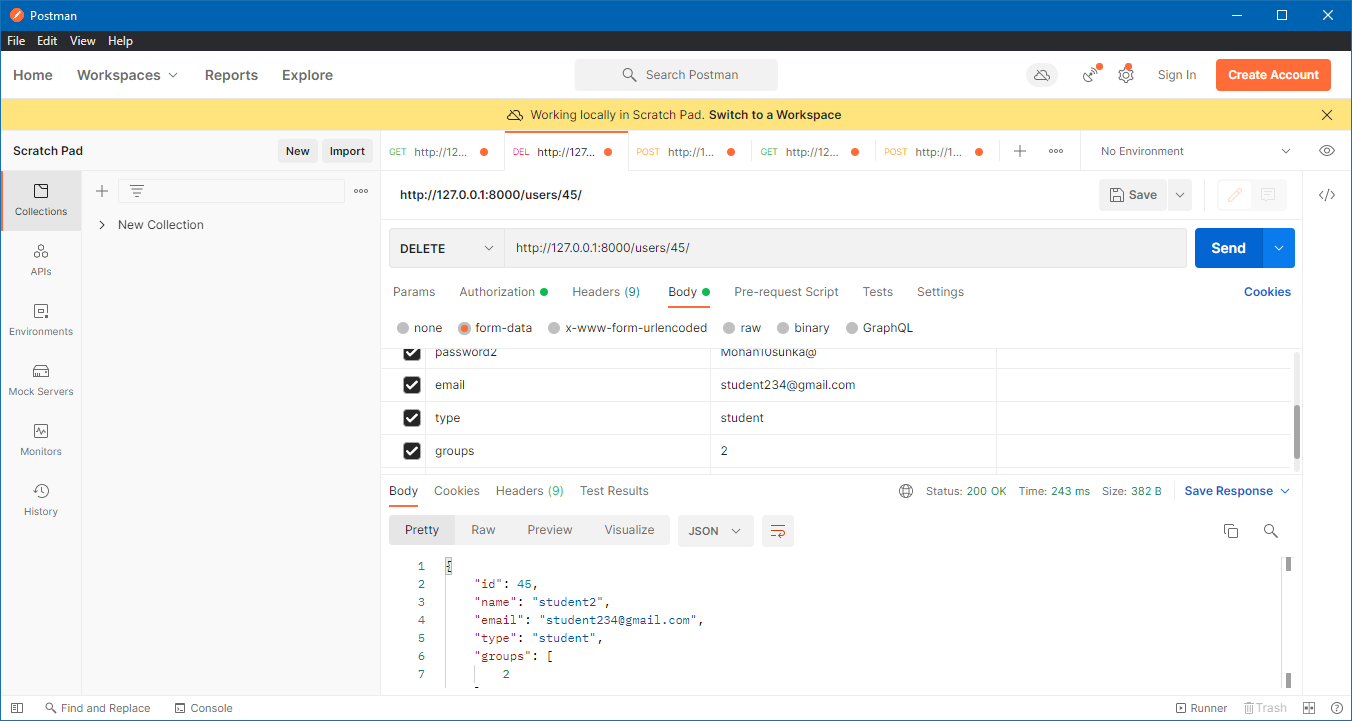
**Can user access for create,delete,update,details only admin user only**

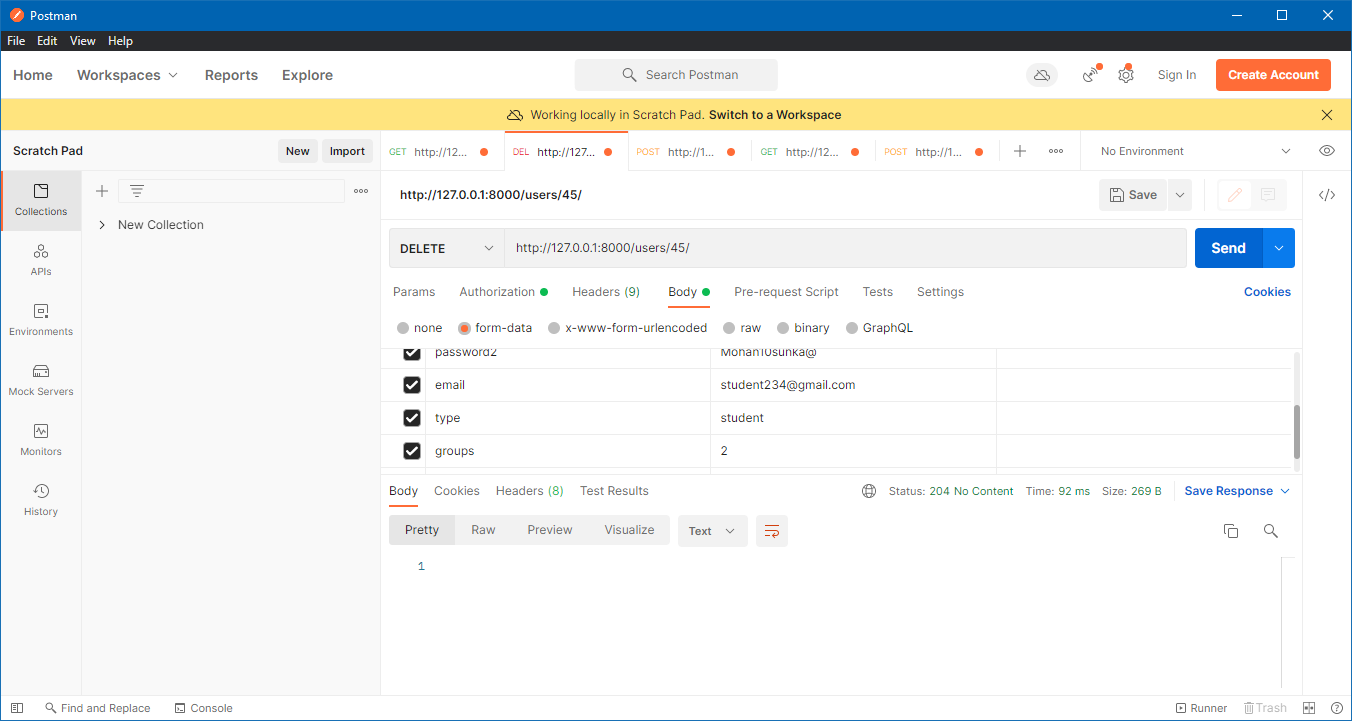
****

****

****

****

****

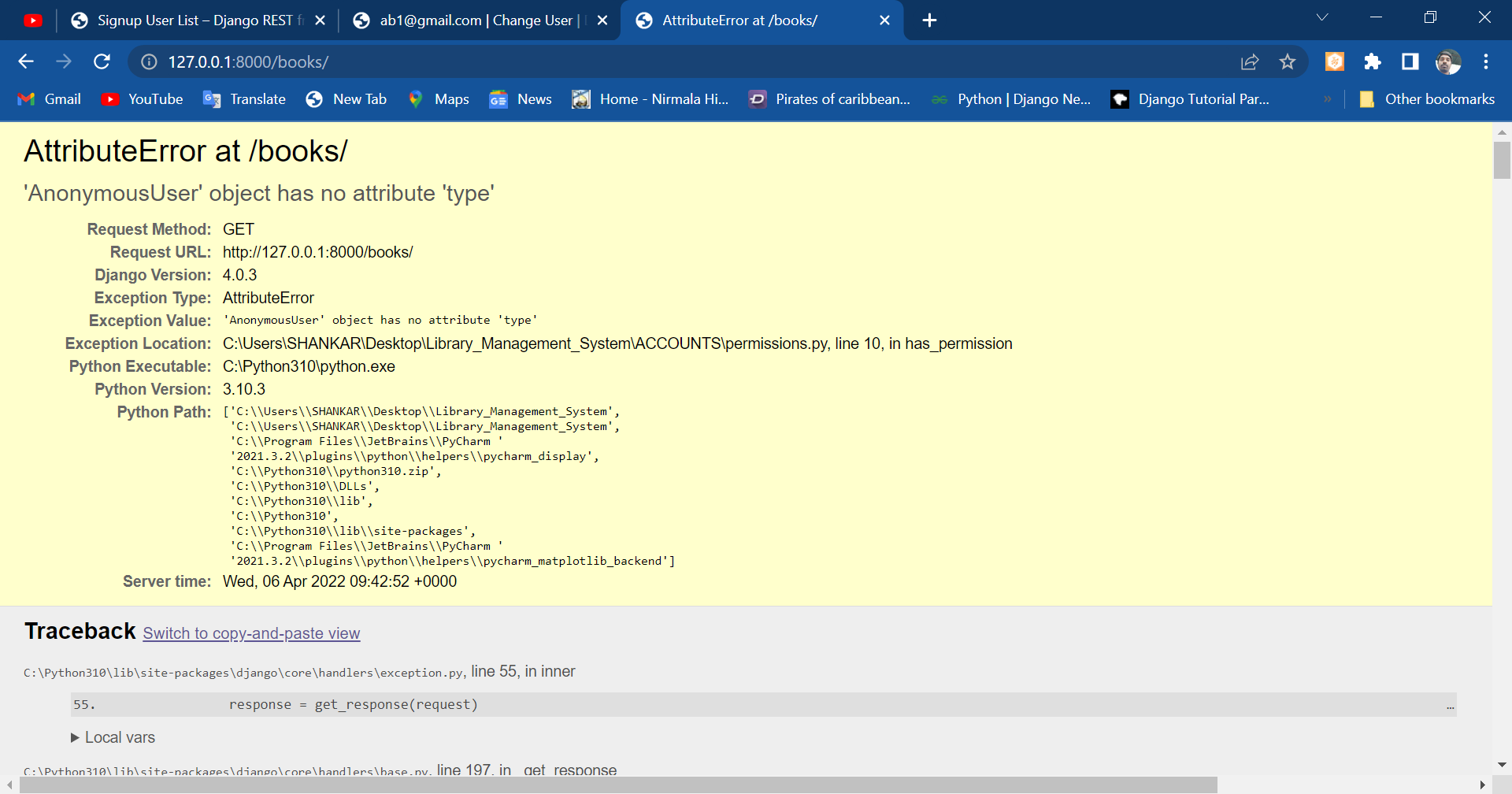
****

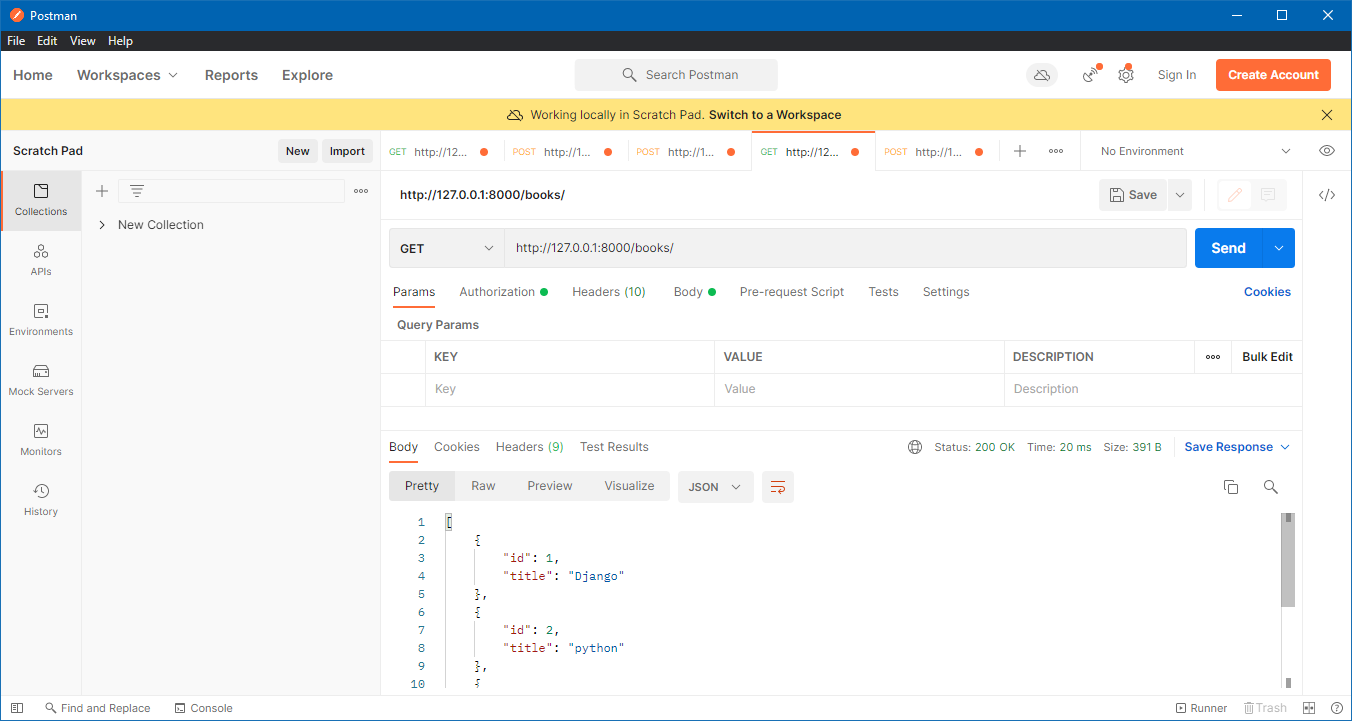
**Books app**

3.getting all books for GET admin user can do for GET,POST,PUT,DELETE,PATCH

student is only view

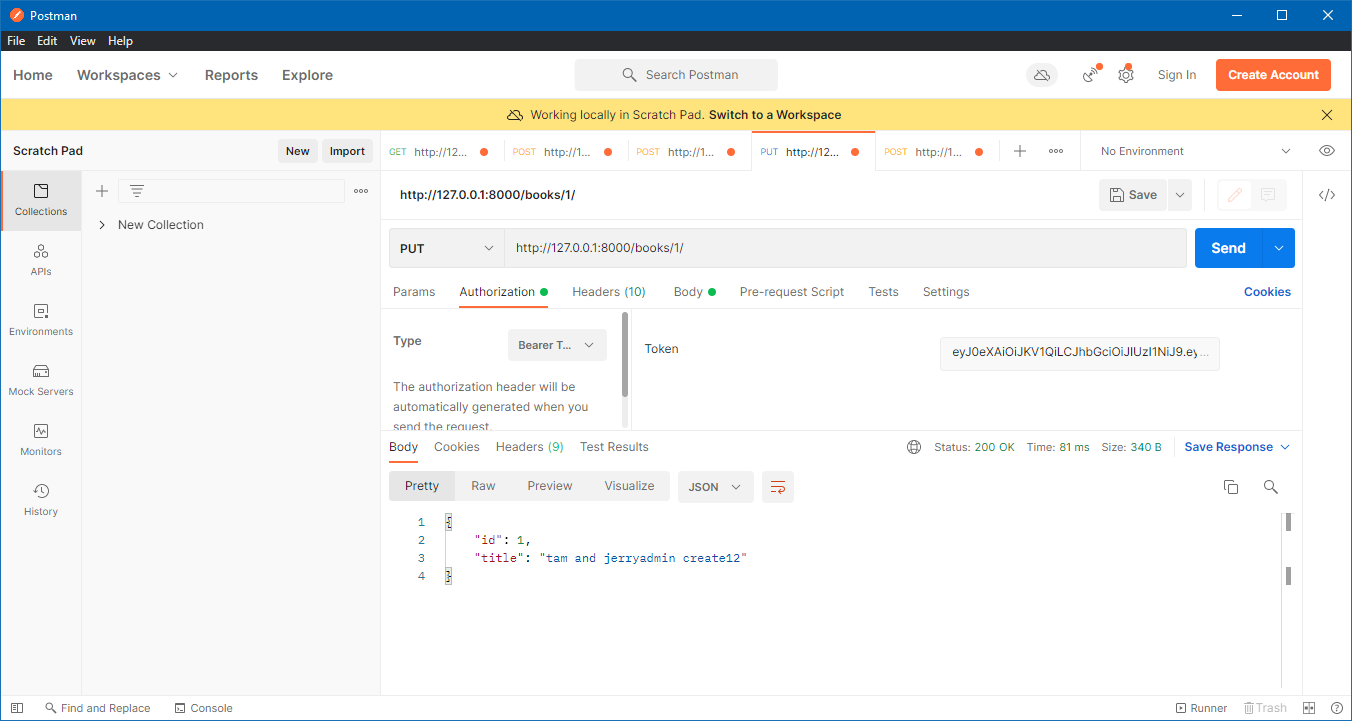
<http://127.0.0.1:8000/books/>

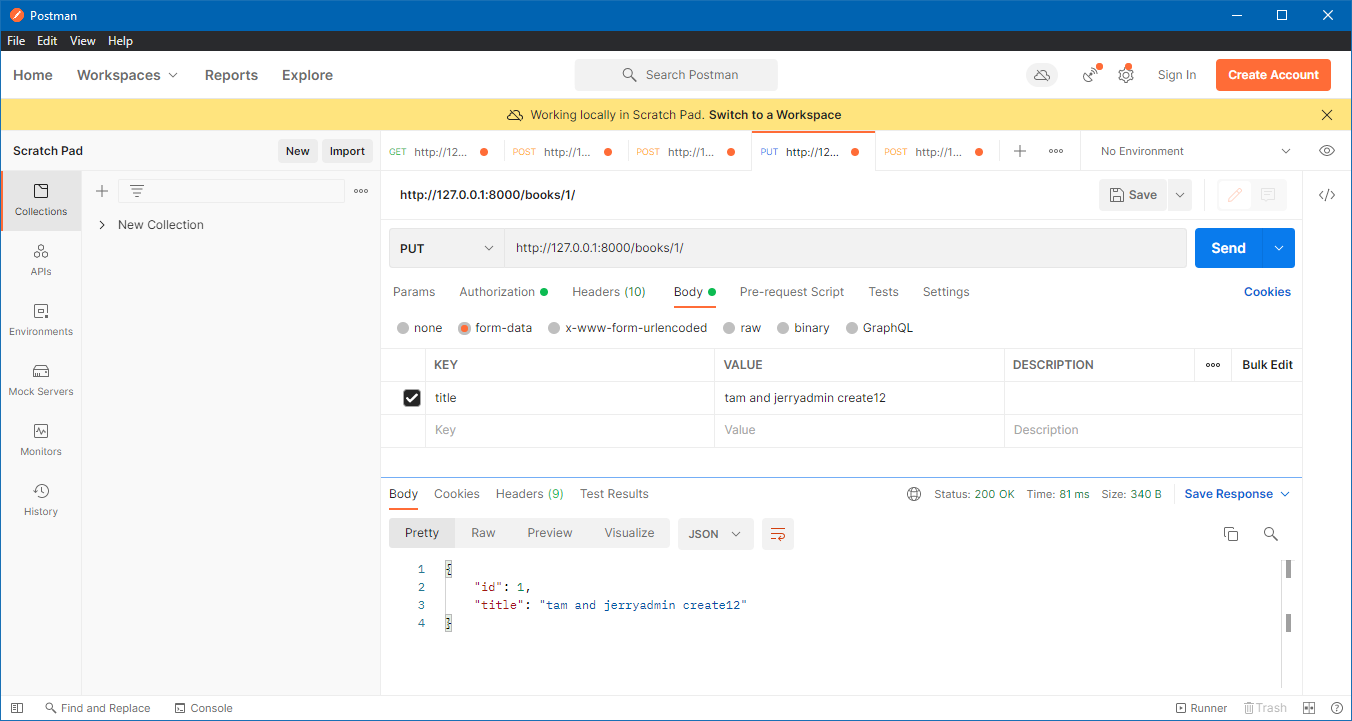


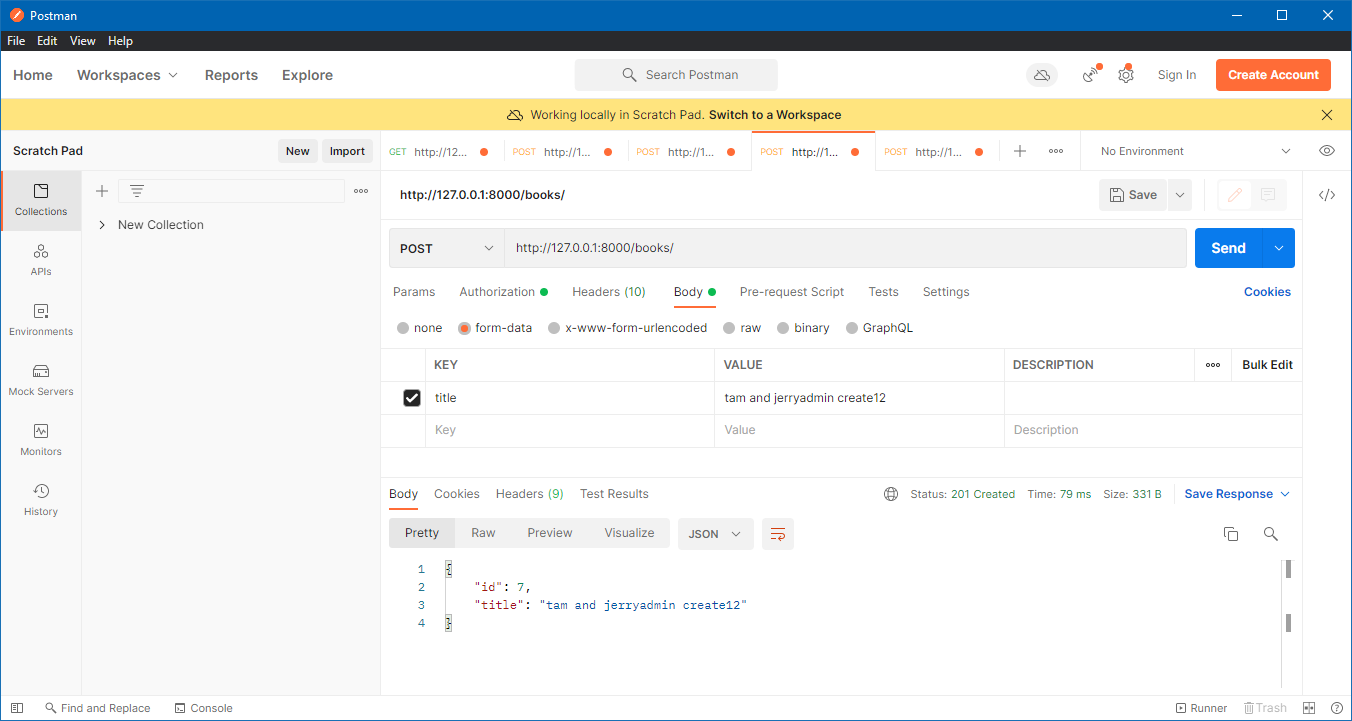


GET,ONE BOOKS BASED ON ID,PUT,PATCH,DELETE. CAN ADMIN USER

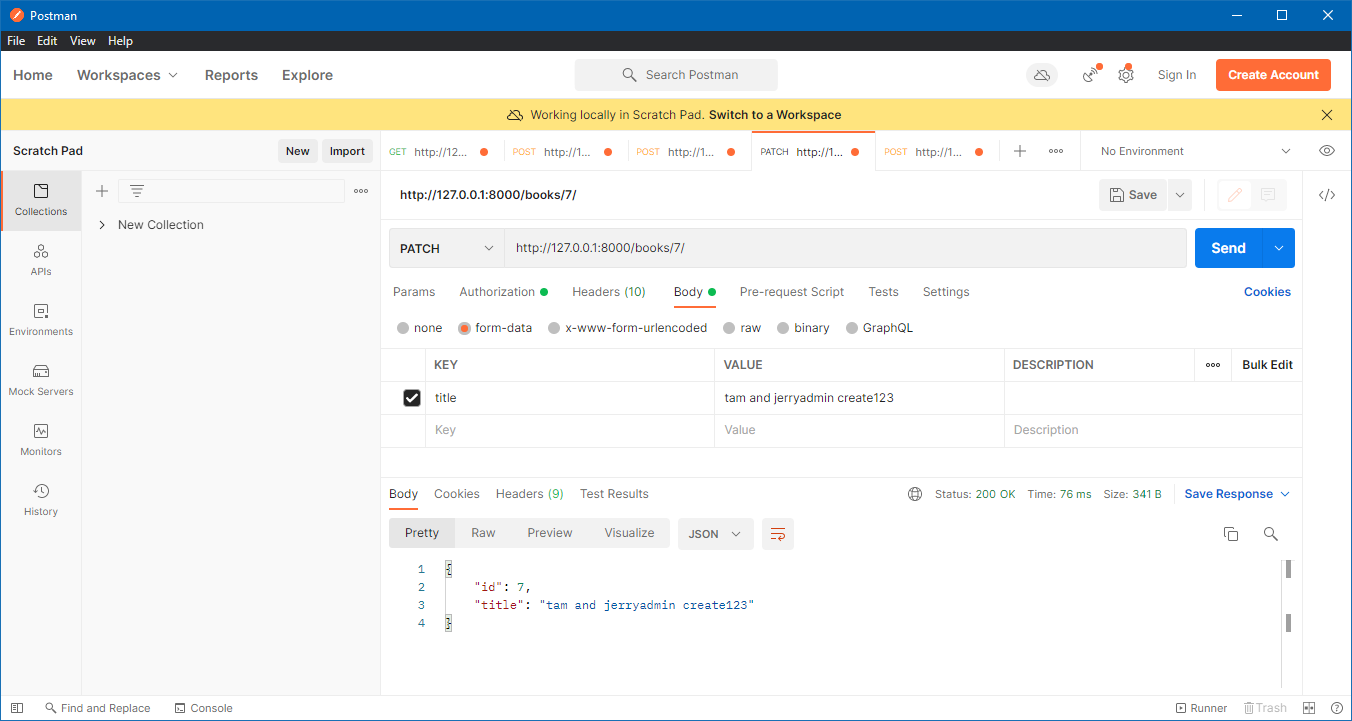
<http://127.0.0.1:8000/books/1/>

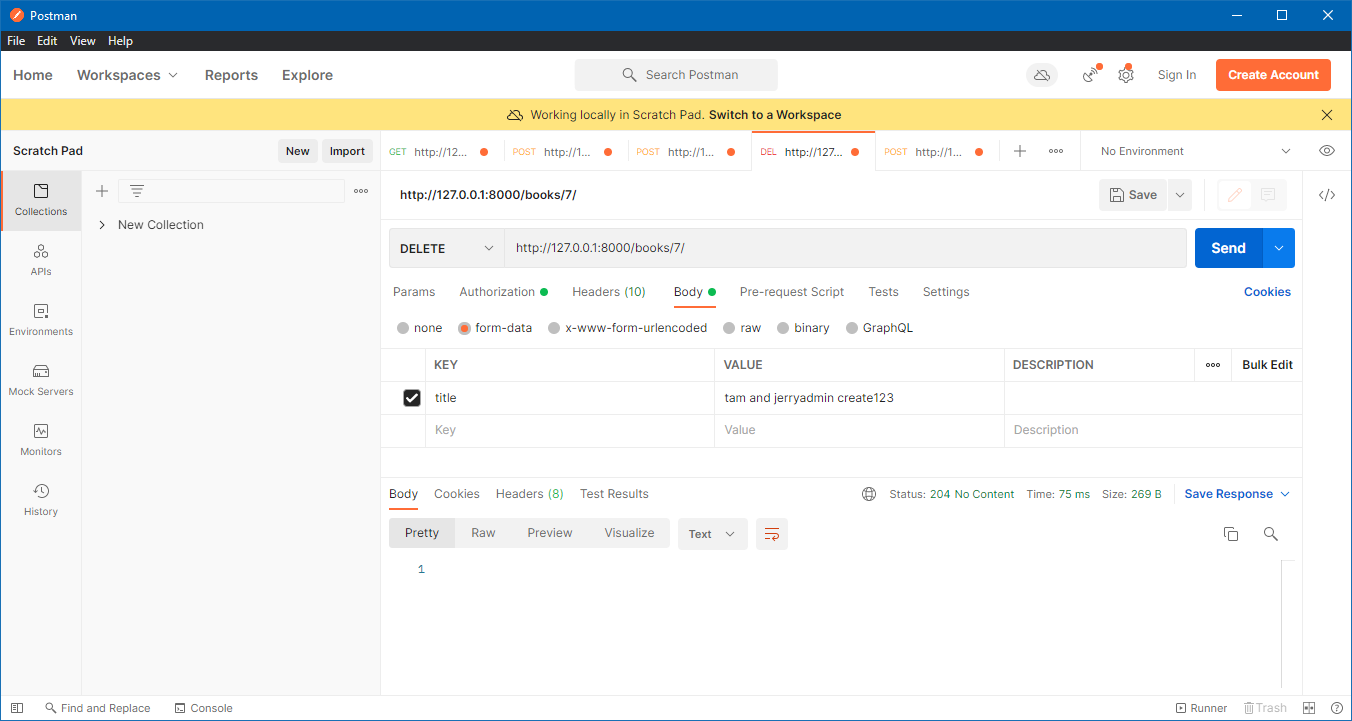






<http://127.0.0.1:8000/books/7/>





Thanks you sir