



4CP33 – FSEP

Python Internship– NSVTech

20CP011 – Jaimin Damor

Project_Title : EMR Data Exchange

Build a Electronic Medical Record Application which is a bi-directional interface using the FHIR Standard for seamless sharing and retrieval of electronic medical records (EMRs) among healthcare providers. This initiative ensures secure and standardized transmission of patient data, including demographics, medical history, and treatment plans.:

- Django
- Api's (Django Rest Framework)
- Creating models (SQLite3-database)
- FHIR Standard



FHIR Standards



It is a Resource Followed by Companies for Exchanging Medical Records of Patients, FHIR is designed to facilitate interoperability, making it easier for different healthcare systems to share and exchange data.

HL7 FHIR Release 5

Home | Getting Started | Documentation | Data Types | Resource Types | Terminologies | Artifacts | Implementation Guides |

Table of Contents > Resources

This page is part of the FHIR Specification (v5.0.0: R5 - STU). This is the current published version. For a full list of available versions, see the [Directory of published versions](#) of Page versions: **R5** R4 R3 R2

1.1 Resource Index

FHIR Infrastructure (FHIR Work Group)	Maturity Level: N/A	Standards Status: Informative
---------------------------------------	---------------------	-------------------------------

Types Framework Cross Reference: [Base Types](#) | [Datatypes](#) | [Resources](#) | [Patterns](#)

This page is provided to help find what you are looking for quickly - there are 137 resources. There is also a more detailed classification, ontology, and description. For background on the layout on the layers in this page, see the [Architect's Overview](#), and [Security Category Considerations](#). See also the abstract Base Resources [Resource](#) and [DomainResource](#).

[Categorized](#) | [Alphabetical](#) | [R3 Layout](#) | [By Maturity](#) | [Security Category](#) | [By Standards Status](#) | [By Work Group](#)

Conformance	Terminology	Security	Documents	Other
<ul style="list-style-type: none">• CapabilityStatement N• StructureDefinition N• ImplementationGuide 4• SearchParameter 3• MessageDefinition 1• OperationDefinition N• CompartmentDefinition 3• StructureMap 4• GraphDefinition 2	<ul style="list-style-type: none">• CodeSystem N• ValueSet N• ConceptMap 3• NamingSystem 4• TerminologyCapabilities 1	<ul style="list-style-type: none">• Provenance 4• AuditEvent 4• Permission 0• Consent 2	<ul style="list-style-type: none">• Composition 4• DocumentReference 4	<ul style="list-style-type: none">• Basic 3• Binary N• Bundle N• Linkage 0• MessageHeader 4• OperationOutcome N• Parameters N• Subscription 3• SubscriptionStatus 2• SubscriptionTopic 2

Individuals	Entities #1	Entities #2	Workflow	Management
<ul style="list-style-type: none">• Patient N• Practitioner 3• PractitionerRole 4• RelatedPerson 5	<ul style="list-style-type: none">• Organization 5• OrganizationAffiliation 1• HealthcareService 4• Endpoint 2	<ul style="list-style-type: none">• Substance 2• BiologicallyDerivedProduct 2• Device 2• DeviceMetric 1	<ul style="list-style-type: none">• Task 2• Transport 1• Appointment 2• AppointmentResponse 3	<ul style="list-style-type: none">• Encounter 4• EncounterHistory 0• EpisodeOfCare 2• Flag 1



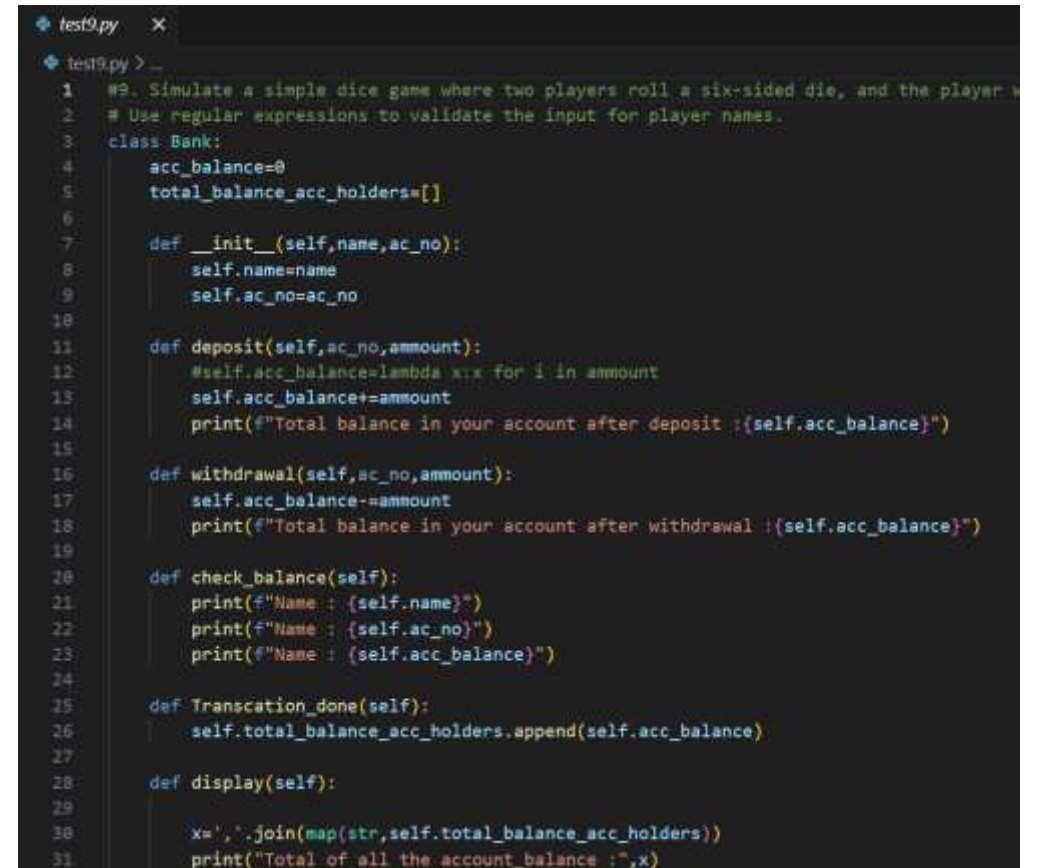
Work Done till now

1st Week:

- HR Induction Program
- Medical Device Demonstration

2nd Week:

- Revise Python language
- End of the week Exam(python)



```
test9.py X
test9.py > ...
1  #9. Simulate a simple dice game where two players roll a six-sided die, and the player
2  # Use regular expressions to validate the input for player names.
3  class Bank:
4      acc_balance=0
5      total_balance_acc_holders=[]
6
7      def __init__(self,name,ac_no):
8          self.name=name
9          self.ac_no=ac_no
10
11     def deposit(self,ac_no,ammount):
12         #self.acc_balance=lambda x:x for i in ammount
13         self.acc_balance+=ammount
14         print(f"Total balance in your account after deposit :{self.acc_balance}")
15
16     def withdrawal(self,ac_no,ammount):
17         self.acc_balance-=ammount
18         print(f"Total balance in your account after withdrawal :{self.acc_balance}")
19
20     def check_balance(self):
21         print(f"Name : {self.name}")
22         print(f"Name : {self.ac_no}")
23         print(f"Name : {self.acc_balance}")
24
25     def Transcation_done(self):
26         self.total_balance_acc_holders.append(self.acc_balance)
27
28     def display(self):
29
30         x=', '.join(map(str,self.total_balance_acc_holders))
31         print("Total of all the account_balance :",x)
```

Work Done till now

3rd Week:

- Django Framework
- Working in Virtual Environment
- Create Database
- Field Selection from FHIR standard

Modules

- `django.http` (sending Response)
- `rest_framework.parsers` (send data)
- `rest_framework.renderers` (retrieve data)
- `rest_framework.views` (API View)
- `django.views.decorators.csrf` (CSRF token)
- `rest_framework.authentication` (Valid Auth)
- `rest_framework.permissions` (Authentication permissions)
- `rest_framework.generics` (Generic API View)
- `rest_framework.routers` (router urls)
- `rest_framework_simplejwt.views` (Token generation/refresh/verification)

Model Creation

```
← → Django_project

models.py 0001_initial.py serializers.py myapp.py views.py urls.py ...EMRDATA

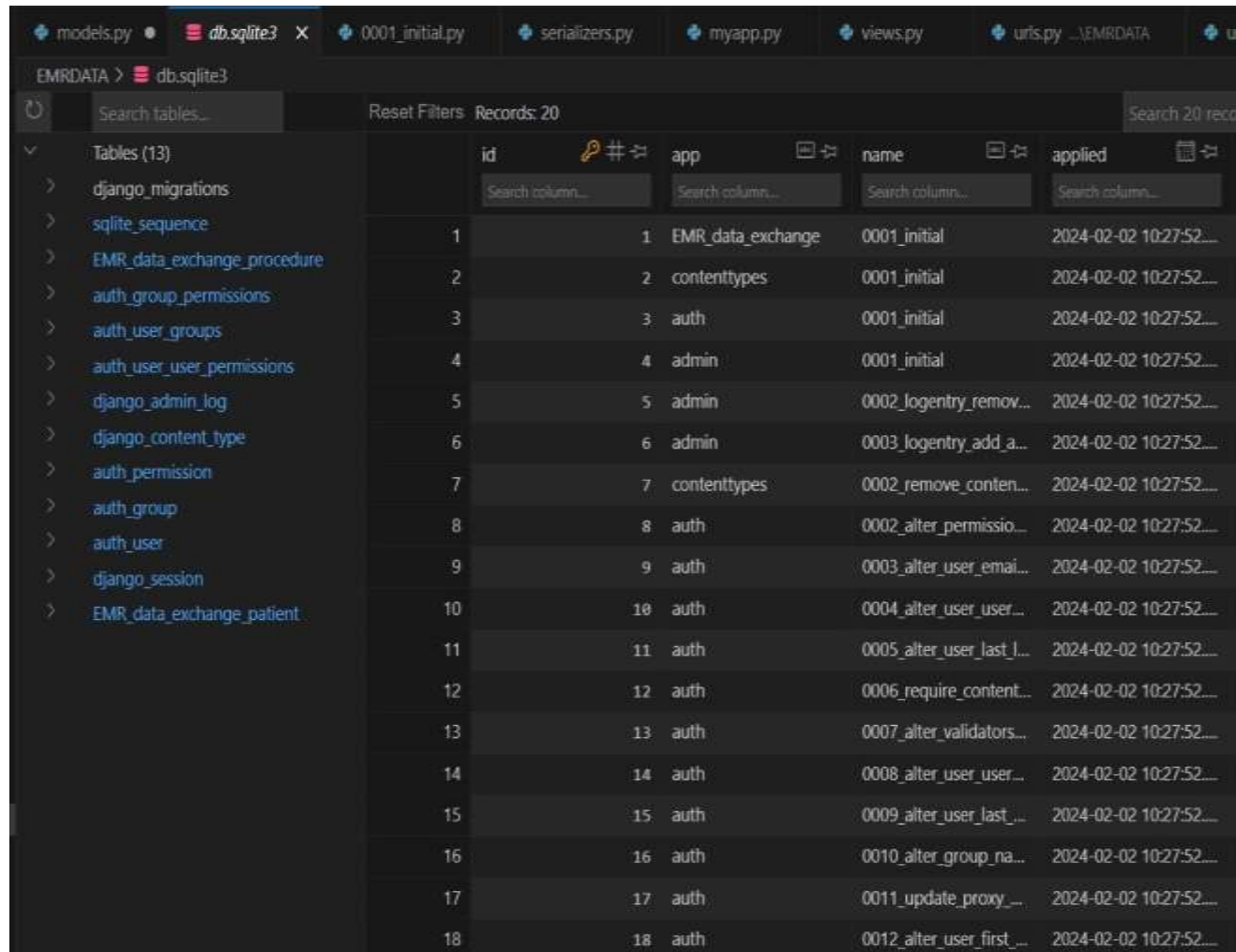
EMRDATA > EMR_data_exchange > models.py > ...
1 from django.db import models
2
3 # Create your models here.
4 class Patient(models.Model):
5     first_name=models.CharField(max_length=10,null=True,default="")
6     last_name=models.CharField(max_length=10,null=True,default="")
7     mobile_number=models.CharField(max_length=10,null=True)
8     gender=models.CharField(max_length=15,null=True,default="")
9     birthdate=models.DateField(blank=True, default='', null=True)
10    city=models.CharField(max_length=12,null=True,default="")
11    state=models.CharField(max_length=14,null=True,default="")
12    pincode=models.CharField(max_length=6,null=True,default="")
13    emergency_contant_name=models.CharField(max_length=12,null=True,default="")
14    emergency_contant_mobile_number=models.CharField(max_length=10,null=True,default="")
15    language=models.CharField(max_length=8,null=True,default="") #null=True,blank=True
16
17
18    def __str__(self):
19        return str(self.mobile_number)
20
21
22 class Procedure(models.Model):
23     #USER ,
24     patient=models.ForeignKey(Patient, on_delete=models.CASCADE)
25     status=models.CharField(max_length=50,default="")
26     statusReason=models.CharField(max_length=50,default="")
27     category=models.CharField(max_length=50,default="") # Surgical Procedure
28     type=models.CharField(max_length=50,default="") # Laparoscopic Appendectomy
29     clinic_address=models.CharField(max_length=50,default="")
30     notes=models.CharField(max_length=50,default="")
31     report=models.FileField(null=True,blank=True)
```

Call functions

```
myapp.py views.py X urls.py ...EMRDATA tests.py urls.py ...EMRDATA

EMRDATA > EMR_data_exchange > views.py > patientinfo
1 from django.shortcuts import render
2
3 # Create your views here.
4 from .models import Patient,Procedure
5 from .serializers import PatientSerializer
6 from rest_framework.parsers import JSONParser
7 from rest_framework.renderers import JSONRenderer
8
9 def patientinfo(request):
10     if request.method=='GET':
11         json_data=request.body
12         stream=io.BytesIO(json_data)
13         pythondata=JSONParser().parse(stream)
14         id=pythondata.get('id',None)
15         if id is not None:
16             pat=Patient.objects.get(id=id)
17             serializer=PatientSerializer(pat)
18             json_data=JSONRenderer().render(serializer)
19             return HttpResponse(json_data,content_type='application/text')
20
21     Pat=Patient.objects.all()
22     serializer=PatientSerializer(Pat)
23     json_data=JSONRenderer.render(serializer.data)
24     return HttpResponse(json_data,content_type='application/text')
```

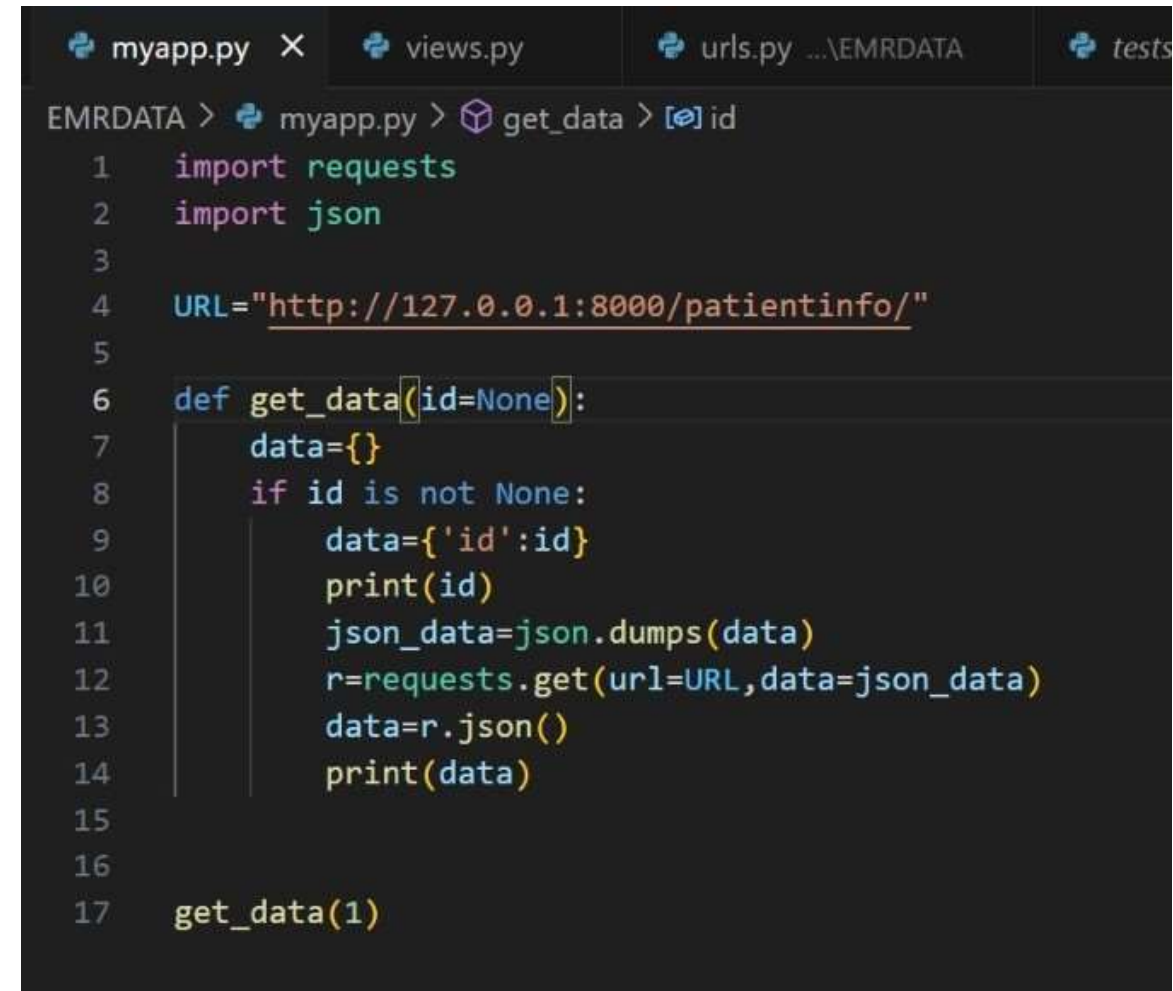
Migrations in DB



The screenshot shows a database management interface with a table of migrations. The table has columns for id, app, name, and applied. The left sidebar lists various database tables. The main table displays 18 migration records, each with a unique ID, the app it belongs to, the migration name, and the time it was applied.

id	app	name	applied
1	EMR_data_exchange	0001_initial	2024-02-02 10:27:52...
2	contenttypes	0001_initial	2024-02-02 10:27:52...
3	auth	0001_initial	2024-02-02 10:27:52...
4	admin	0001_initial	2024-02-02 10:27:52...
5	admin	0002_logentry_remov...	2024-02-02 10:27:52...
6	admin	0003_logentry_add_a...	2024-02-02 10:27:52...
7	contenttypes	0002_remove_conten...	2024-02-02 10:27:52...
8	auth	0002_alter_permissio...	2024-02-02 10:27:52...
9	auth	0003_alter_user_email...	2024-02-02 10:27:52...
10	auth	0004_alter_user_user...	2024-02-02 10:27:52...
11	auth	0005_alter_user_last_l...	2024-02-02 10:27:52...
12	auth	0006_require_content...	2024-02-02 10:27:52...
13	auth	0007_alter_validators...	2024-02-02 10:27:52...
14	auth	0008_alter_user_user...	2024-02-02 10:27:52...
15	auth	0009_alter_user_last_...	2024-02-02 10:27:52...
16	auth	0010_alter_group_na...	2024-02-02 10:27:52...
17	auth	0011_update_proxy_...	2024-02-02 10:27:52...
18	auth	0012_alter_user_first_...	2024-02-02 10:27:52...

Reading through Api



The screenshot shows a Python script in a code editor. The script imports the requests and json modules, defines a URL, and creates a function get_data that takes an id parameter. The function constructs a JSON payload and sends a GET request to the specified URL. The response is parsed as JSON and printed. Finally, the function is called with the argument 1.

```
myapp.py X views.py urls.py ...\EMRDATA tests

EMRDATA > myapp.py > get_data > id

1 import requests
2 import json
3
4 URL="http://127.0.0.1:8000/patientinfo/"
5
6 def get_data(id=None):
7     data={}
8     if id is not None:
9         data={'id':id}
10        print(id)
11        json_data=json.dumps(data)
12        r=requests.get(url=URL,data=json_data)
13        data=r.json()
14        print(data)
15
16
17 get_data(1)
```


Class Based View

```
← → EMRDATA
```

```
py views.py settings.py Create_data.py serializers.py custompermission.py
```

```
ata_exchange > views.py > PatientViewSet
```

```
# CLASS based API''''''ssssssssssssssssssss !!!!!!!!!!!!!!!!!!!!!!!
```

```
##### Procedure API's
```

```
@method_decorator(csrf_exempt,name='dispatch')
```

```
class update_procedure_dataAPI(View):
```

```
    def patch(self,request,*args,**kwargs):
```

```
        json_data=request.body
```

```
        stream=io.BytesIO(json_data)
```

```
        pythondata=JSONParser().parse(stream)
```

```
        id=pythondata.get('id',None)
```

```
        procedure_data=Procedure.objects.get(id=id)
```

```
        serializer=ProcedureSerializer(instance=procedure_data,data=pythondata,partial=True)
```

```
        if serializer.is_valid():
```

```
            serializer.save()
```

```
            res={'msg':' Data Updated !!!!!!!!!'}
```

```
            json_data=JSONRenderer().render(res)
```

```
            return HttpResponse(json_data,content_type='application/text')
```

```
    def put(self,request,*args,**kwargs):
```

```
        json_data=request.body
```

```
        stream=io.BytesIO(json_data)
```

```
        pythondata=JSONParser().parse(stream)
```

```
        id=pythondata.get('id',None)
```

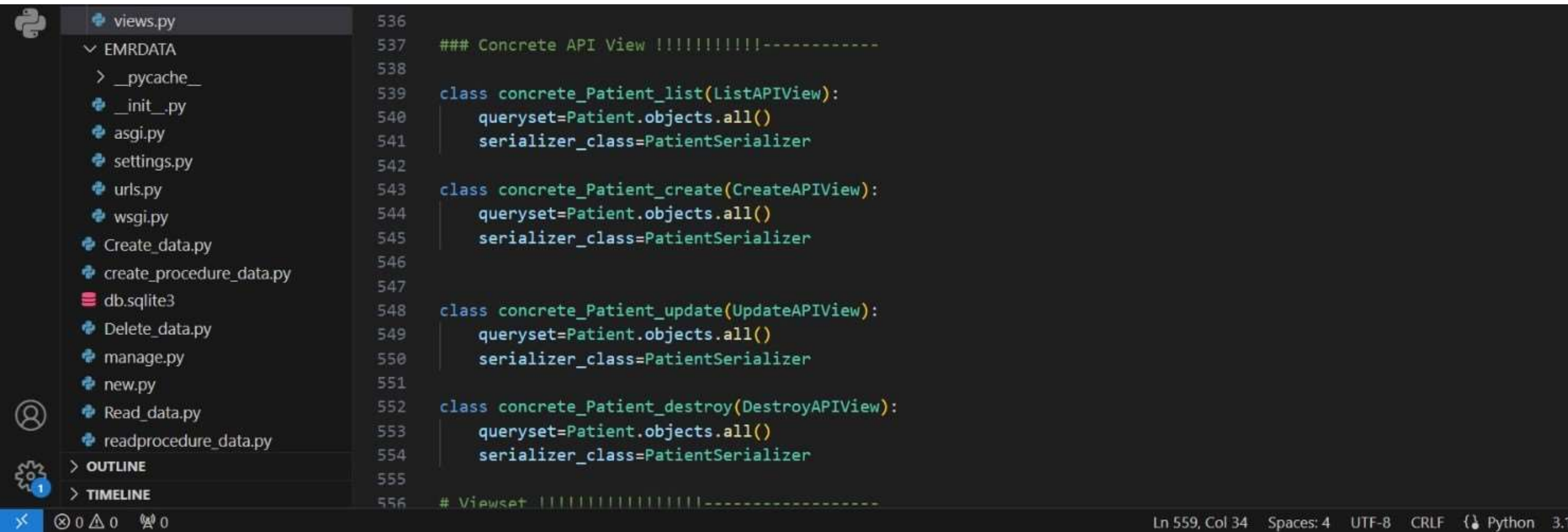
```
        procedure_data=Procedure.objects.get(id=id)
```

```
        serializer=ProcedureSerializer(instance=procedure_data,data=pythondata,partial=True)
```

```
        if serializer.is_valid():
```

```
            serializer.save()
```

Concrete Based View



The image shows a code editor with a dark theme. On the left is a file explorer showing a project structure with files like `views.py`, `EMRDATA`, `__pycache__`, `__init__.py`, `asgi.py`, `settings.py`, `urls.py`, `wsgi.py`, `Create_data.py`, `create_procedure_data.py`, `db.sqlite3`, `Delete_data.py`, `manage.py`, `new.py`, `Read_data.py`, and `readprocedure_data.py`. Below these are sections for `OUTLINE` and `TIMELINE`. The main editor area shows Python code for concrete API views. The code includes comments and class definitions for `concrete_Patient_list`, `concrete_Patient_create`, `concrete_Patient_update`, and `concrete_Patient_destroy`, all inheriting from Django's `ListAPIView`, `CreateAPIView`, `UpdateAPIView`, and `DestroyAPIView` respectively. Each class sets `queryset=Patient.objects.all()` and `serializer_class=PatientSerializer`. The code is line-numbered from 536 to 556. At the bottom, a status bar indicates the current position is `Ln 559, Col 34` with `Spaces: 4`, `UTF-8` encoding, `CRLF` line endings, and the interpreter is `Python 3.1`.

```
536
537  ### Concrete API View !!!!!!!!!!!!!-----
538
539  class concrete_Patient_list(ListAPIView):
540      queryset=Patient.objects.all()
541      serializer_class=PatientSerializer
542
543  class concrete_Patient_create(CreateAPIView):
544      queryset=Patient.objects.all()
545      serializer_class=PatientSerializer
546
547
548  class concrete_Patient_update(UpdateAPIView):
549      queryset=Patient.objects.all()
550      serializer_class=PatientSerializer
551
552  class concrete_Patient_destroy(DestroyAPIView):
553      queryset=Patient.objects.all()
554      serializer_class=PatientSerializer
555
556  # Viewset !!!!!!!!!!!!!-----
```

Ln 559, Col 34 Spaces: 4 UTF-8 CRLF Python 3.1

Generic View



The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like migrations, __init__.py, admin.py, apps.py, custompremission.py, models.py, serializers.py, tests.py, urls.py, and views.py. The views.py file is selected. The code editor shows the following Python code:

```
505
506 ##### Generic API View !!!!!!!!!!!!!!!!!!!!!-----
507
508 class patientlist(GenericAPIView,ListModelMixin):
509     queryset=Patient.objects.all()
510     serializer_class=PatientSerializer
511     print(queryset)
512     def get(self,request,*args,**kwargs):
513         return self.list(request,*args,**kwargs)
514
515 class patientcreate(GenericAPIView,CreateModelMixin):
516     queryset=Patient.objects.all()
517     serializer_class=PatientSerializer
518     print(queryset)
519     def post(self,request,*args,**kwargs):
520         return self.create(request,*args,**kwargs)
521
522 class patientupdate(GenericAPIView,UpdateModelMixin):
523     queryset=Patient.objects.all()
524     serializer_class=PatientSerializer
525     print(queryset)
526     def put(self,request,*args,**kwargs):
527         return self.update(request,*args,**kwargs)
528
529 class patientdelete(GenericAPIView,DestroyModelMixin):
530     queryset=Patient.objects.all()
531     serializer_class=PatientSerializer
532     print(queryset)
533     def delete(self,request,*args,**kwargs):
534         return self.destroy(request,*args,**kwargs)
535
536
```

The code defines four generic API views: patientlist, patientcreate, patientupdate, and patientdelete. Each view inherits from GenericAPIView and a specific ModelMixin (ListModelMixin, CreateModelMixin, UpdateModelMixin, and DestroyModelMixin respectively). Each view sets a queryset to Patient.objects.all() and a serializer_class to PatientSerializer. The patientlist view has a get method that returns self.list(request,*args,**kwargs). The patientcreate view has a post method that returns self.create(request,*args,**kwargs). The patientupdate view has a put method that returns self.update(request,*args,**kwargs). The patientdelete view has a delete method that returns self.destroy(request,*args,**kwargs). Each view also prints the queryset.

Normal URL's

```
path('patientlist/', views.patientlist.as_view(), name='patientlist'),
path('patientcreate/', views.patientcreate.as_view(), name='patientcreate'),
path('patientupdate/<int:pk>', views.patientupdate.as_view(), name='patientupdate'),
path('patientdelete/<int:pk>', views.patientdelete.as_view(), name='patientdelete'),
```

Router URL's

```
from rest_framework.routers import DefaultRouter
router = DefaultRouter()

router.register('PatientViewSet', views.PatientViewSet, basename="doctor")
router.register('PatientModelViewSet', views.PatientModelViewSet, basename="None")
```

```
def patientcreate(self, request):
    serializer = PatientSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors)

def patientupdate(self, request, pk):
    id = pk
    pat = Patient.objects.get(id=id)
    serializer = PatientSerializer(pat, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors)
```



Authentication/Authorization

Start typing to filter...

AUTH TOKEN

Tokens [+ Add](#)

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

EMR_DATA_EXCHANGE

Patients [+ Add](#)

Procedures [+ Add](#)

Select group to change

Search

Action: Go 0 of 2 selected

☐ GROUP

☐ Doctor's

☐ Nurse's

2 groups

Doctor's

Name:

Permissions:

Available permissions

Filter

Auth Token | Token | Can add Token
Auth Token | Token | Can change Token
Auth Token | Token | Can delete Token
Auth Token | Token | Can view Token
Auth Token | token | Can add token
Auth Token | token | Can change token
Auth Token | token | Can delete token
Auth Token | token | Can view token

Chosen permissions

Filter

Emr_Data_Exchange | patient | Can add patient
Emr_Data_Exchange | patient | Can change patient
Emr_Data_Exchange | patient | Can delete patient
Emr_Data_Exchange | patient | Can view patient
Emr_Data_Exchange | procedure | Can add procedure
Emr_Data_Exchange | procedure | Can change procedure
Emr_Data_Exchange | procedure | Can delete procedure
Emr_Data_Exchange | procedure | Can view procedure
Administration | log entry | Can add log entry
Administration | log entry | Can change log entry
Administration | log entry | Can delete log entry

Nurse's

Name:

Permissions:

Available permissions

Filter

Emr_Data_Exchange | patient | Can add patient
Emr_Data_Exchange | patient | Can delete patient
Emr_Data_Exchange | procedure | Can add procedure
Emr_Data_Exchange | procedure | Can delete procedure
Administration | log entry | Can add log entry
Administration | log entry | Can change log entry
Administration | log entry | Can delete log entry
Administration | log entry | Can view log entry
Authentication and Authorization | group | Can add group
Authentication and Authorization | group | Can change group
Authentication and Authorization | group | Can delete group

Chosen permissions

Filter

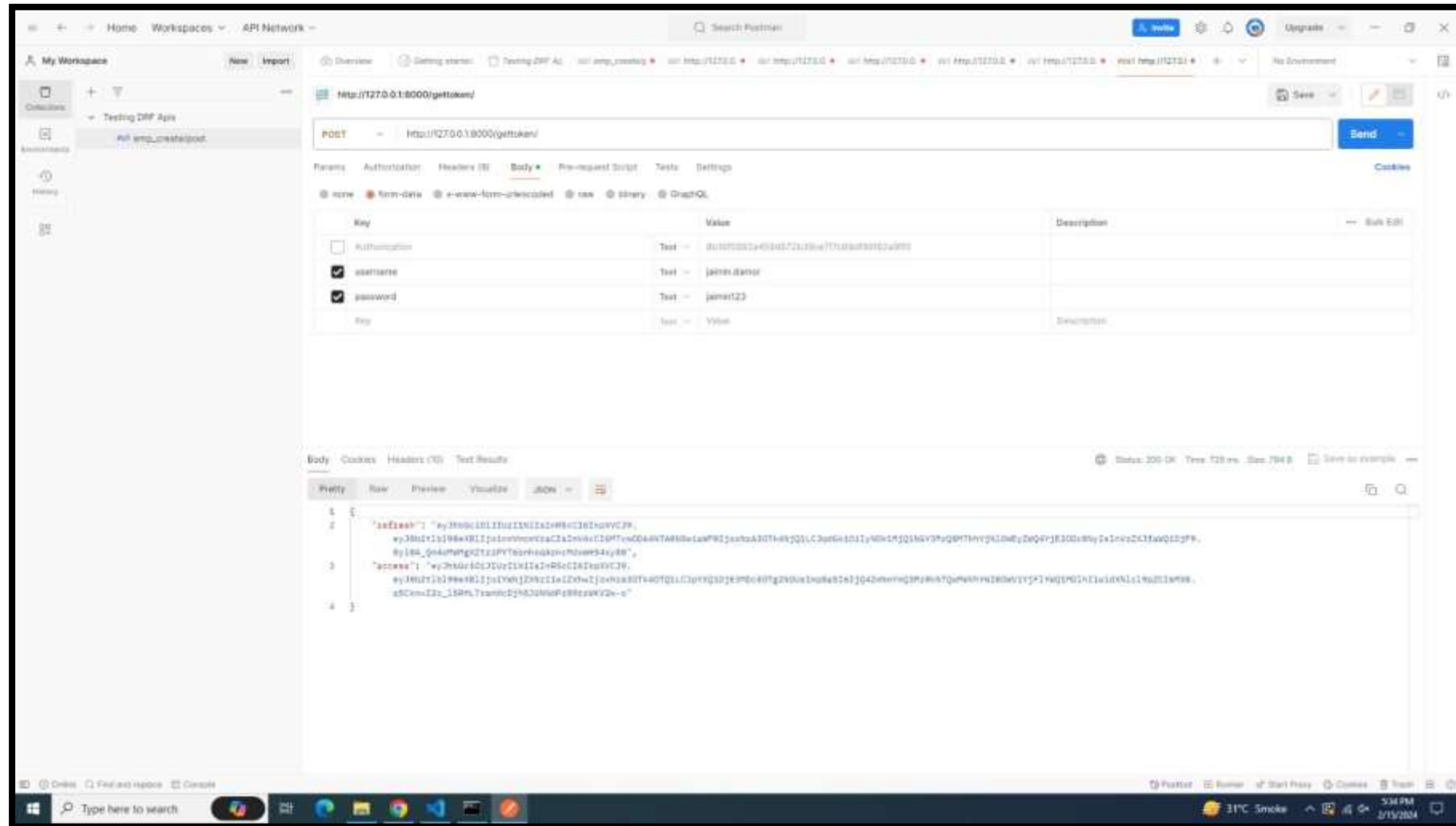
Emr_Data_Exchange | patient | Can change patient
Emr_Data_Exchange | patient | Can view patient
Emr_Data_Exchange | procedure | Can change procedure
Emr_Data_Exchange | procedure | Can view procedure

ViewSet View

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer lists files under the 'EMRDATA' directory, including '_pycache_', 'migrations', '__init__.py', 'admin.py', 'apps.py', 'custompermission.py', 'models.py', 'serializers.py', 'tests.py', 'urls.py', and 'views.py'. The 'views.py' file is selected. The code editor displays the following Python code:

```
552 # Viewset !!!!!!!!!!!!!!!!-----
553 class PatientViewSet(viewsets.ViewSet):
554     authentication_classes=[SessionAuthentication]
555     permission_class=[IsAdminUser] #IsAuthenticated,AllowAny,|jangoModelPermissions
556
557     def list(self,request):
558         pat=Patient.objects.all()
559         serializer=PatientSerializer(pat,many=True)
560         return Response(serializer.data)
561
562     def create(self,request):
563         serializer=PatientSerializer(data=request.data)
564         if serializer.is_valid():
565             serializer.save()
566             return Response(serializer.data)
567         return Response(serializer.errors)
568
569     def partialupdate(self,request,pk):
570         id=pk
571         pat=Patient.objects.get(id=id)
572         serializer=PatientSerializer(pat,data=request.data)
573         if serializer.is_valid():
574             serializer.save()
575             return Response(serializer.data)
576         return Response(serializer.errors)
577
578     def delete(self,request):
579         id=request.data.get('id')
580         pat=Patient.objects.get(id=id)
581         pat.delete()
582         return Response("Data Deleted")
583
```


Token generation



Retrieve Data (Token Auth)

The screenshot shows the Postman interface with a GET request to `http://127.0.0.1:8000/PatientModelViewSet/`. The request is configured with the following headers:

Key	Value	Description
<input checked="" type="checkbox"/> Postman-Token	<calculated when request is sent>	
<input checked="" type="checkbox"/> Host	<calculated when request is sent>	
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.36.3	
<input checked="" type="checkbox"/> Accept	*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	
<input checked="" type="checkbox"/> Authorization	Token 8b10f5992a459d672b39ce7f7c89d98962a89f0	

The response status is 200 OK. The response body is a JSON array of patient data:

```
[{"id": 8, "first_name": "kathan", "last_name": "Patel", "mobile_number": "6743483847", "gender": "male", "birthdate": "2024-02-01", "city": "Valsad", "state": "Gujarat", "pincode": "331801", "emergency_contact_name": "Dean", "emergency_contact_mobile_number": "8888888888", "language": "English"}, {"id": 12, "first_name": "param", "last_name": "Patel", "mobile_number": "6743483842", "gender": "male", "birthdate": "2023-02-01", "city": "Navagam", "state": "Gujarat", "pincode": "331801", "emergency_contact_name": "Dean", "emergency_contact_mobile_number": "2222222222", "language": "English"}, {"id": 17, "first_name": "Dean", "last_name": "Embrose", "mobile_number": "1234567890", "gender": "male", "birthdate": "2024-02-01", "city": "new york", "state": "CALIFORNIA", "pincode": "123456", "emergency_contact_name": "Dean", "emergency_contact_mobile_number": "8888888888", "language": "English"}]
```

Token Verification

The screenshot shows the JWT.io website interface for token verification. The browser's address bar displays `jwt.io/#debugger-io`. The page features a dark header with the JWT logo and navigation links: Debugger, Libraries, Introduction, and Ask. It also credits 'Auth0 by Ckta' as the creator.

The main content area is divided into two panels:

- Encoded**: Displays a long JWT string: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2t1b190eXB1IjoiaYWNjZXNzIiwiaXhwIjozNzA4MDAwNTYzLCJpYXQiOiJlOTQyYzVhZDk3MjIwN2I0WZhnNGQwIiwidXN1c19pZCI6MX0.3kWdunC09spUIm2t_-1JgSnzp20H4MHd5mkBajgK-kw`
- Decoded**: Shows the token's structure:
 - HEADER: ALGORITHM & TOKEN TYPE**: `{ "alg": "HS256", "typ": "JWT" }`
 - PAYLOAD: DATA**: `{ "token_type": "access", "exp": 1708008583, "iat": 1707999483, "jti": "96197595b2e942c5ed972287b49fa400", "user_id": 1 }`
 - VERIFY SIGNATURE**: Shows the signature verification process using `HMACSHA256` with the header and payload, and a secret key `ymv7-256-B11-secret`. The result is `secret base64 encoded`.

At the bottom of the 'Decoded' panel, a blue button labeled **SHARE JWT** is visible. Below the 'Encoded' panel, a green checkmark and the text **Signature Verified** are displayed.

THANK YOU