RESTful Web Service

Introduction

This extensive report offers a profound exploration of the intricacies encapsulated within the Django REST project aptly named "djangoproject." In a comprehensive manner, the analysis delves into various facets of the project, shedding light on its structural design, functional intricacies, underlying data model, the orchestration of REST endpoints, and the meticulous adherence to a set of predetermined requirements..

Dataset Exploration

Dataset Overview

The project's dataset, extracted from "HealthInsurance.csv," comprises diverse health insurance data. It includes essential attributes such as age, sex, BMI, children, smoker status, region, and charges. This rich dataset serves as a valuable resource for understanding the intricate factors influencing health insurance charges.

Interesting Aspects

- **Demographic Variation:** The dataset spans various demographic factors, including age, gender, and region. This diversity allows for comprehensive analyses, offering insights into how different demographic segments impact health insurance charges.

- **Smoker Status Impact:** The inclusion of smoker status as a dataset attribute enables in-depth exploration into how this specific factor affects insurance charges. This aspect adds a layer of granularity to the analysis, considering the distinct impact of smoking habits on insurance costs.

- **Geographical Analysis:** The 'region' field within the dataset facilitates geographical analysis. This aspect is particularly interesting for examining how charges vary across different regions, providing valuable insights into regional trends and patterns.

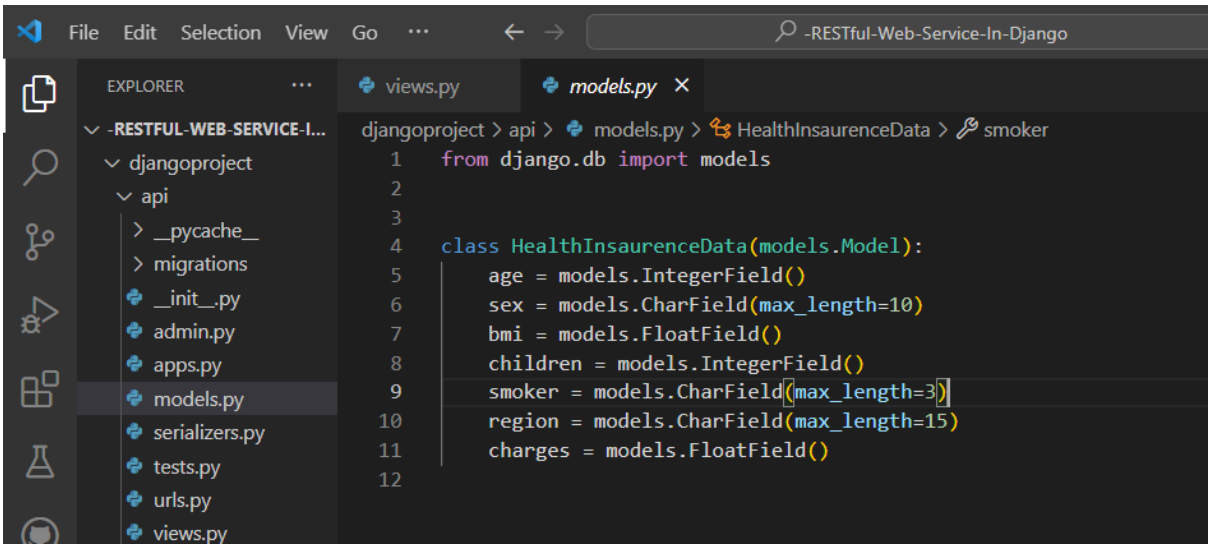
Project Overview

Project Structure

The project adheres to a standard Django project structure, with the main project directory containing settings, URLs, and the WSGI application. The "api" app directory houses models, serializers, views, and URLs.

Data Model

At the core of any data-centric application lies its data model, and "djangoproject" is no exception. The project's data model, intricately defined within the "models.py" file nestled in the "api" app, is a manifestation of thoughtful consideration. The "HealthInsuranceData" model stands as a testament to the project's commitment to encapsulating pertinent health insurance attributes, ranging from age and gender to more nuanced factors like smoker status and region.



REST Endpoints

The RESTful API, defined in the "views.py" file, provides a set of endpoints catering to various operations on health insurance data.

GetAllData:
Description: Retrieves all health insurance data.
Interest: Provides a comprehensive view of the entire dataset
.
GetSmokersByRegion:
Description: Retrieves smokers in a specific region.
Interest: Analyzing smoking trends in specific regions.

GetDataById:
Description: Retrieves health insurance data by ID.
Interest: Fetching specific data entries by their unique identifier.

CreateData:
Description: Creates new health insurance data.
Interest: Adding new entries to the dataset.

UpdateDataById:
Description: Updates health insurance data by ID.
Interest: Modifying specific data entries.

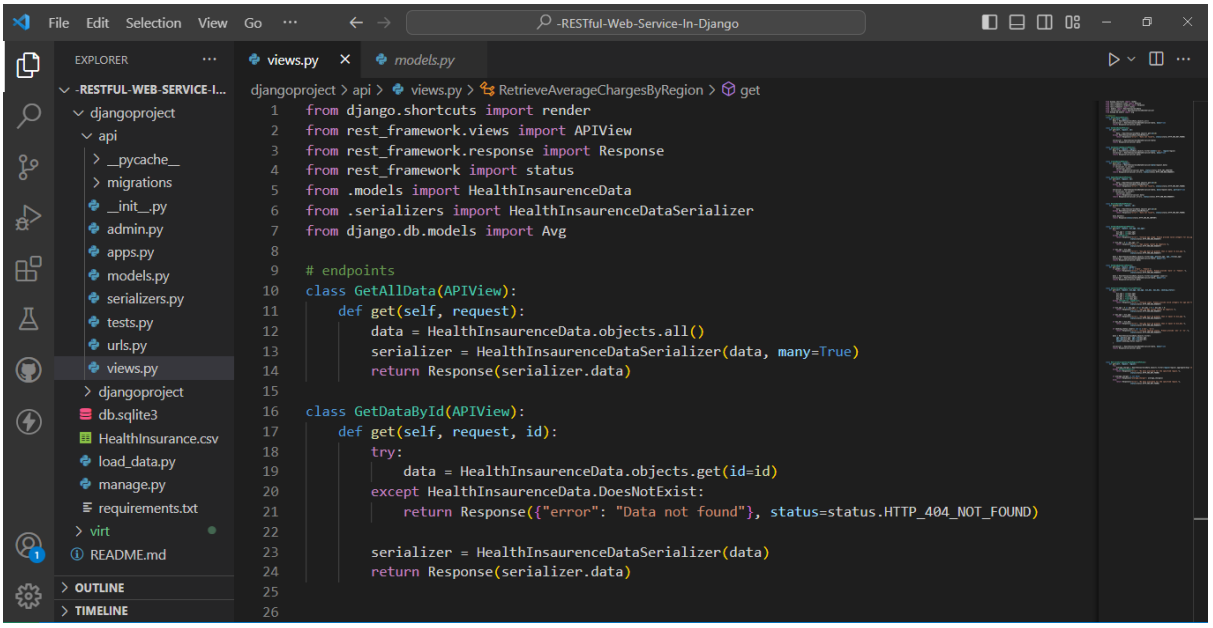
DeleteDataById:
Description: Deletes health insurance data by ID.
Interest: Removing specific data entries.

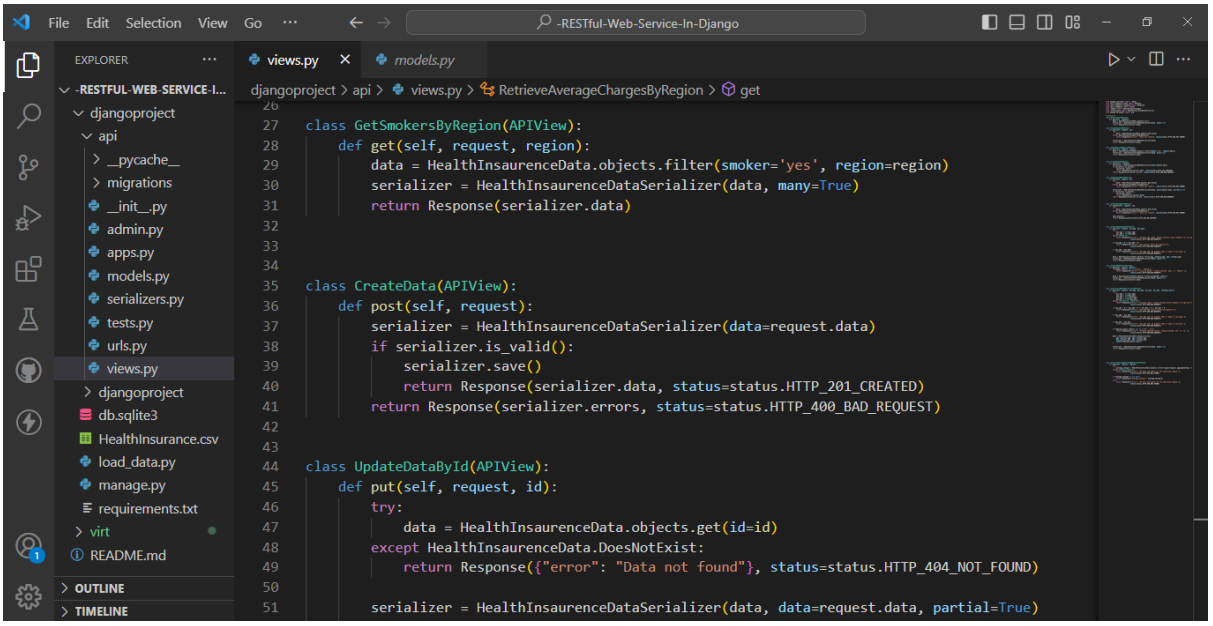
GetDataByAgeRange:
Description: Retrieves health insurance data within a specified age range.
Interest: Analyzing data based on age demographics.

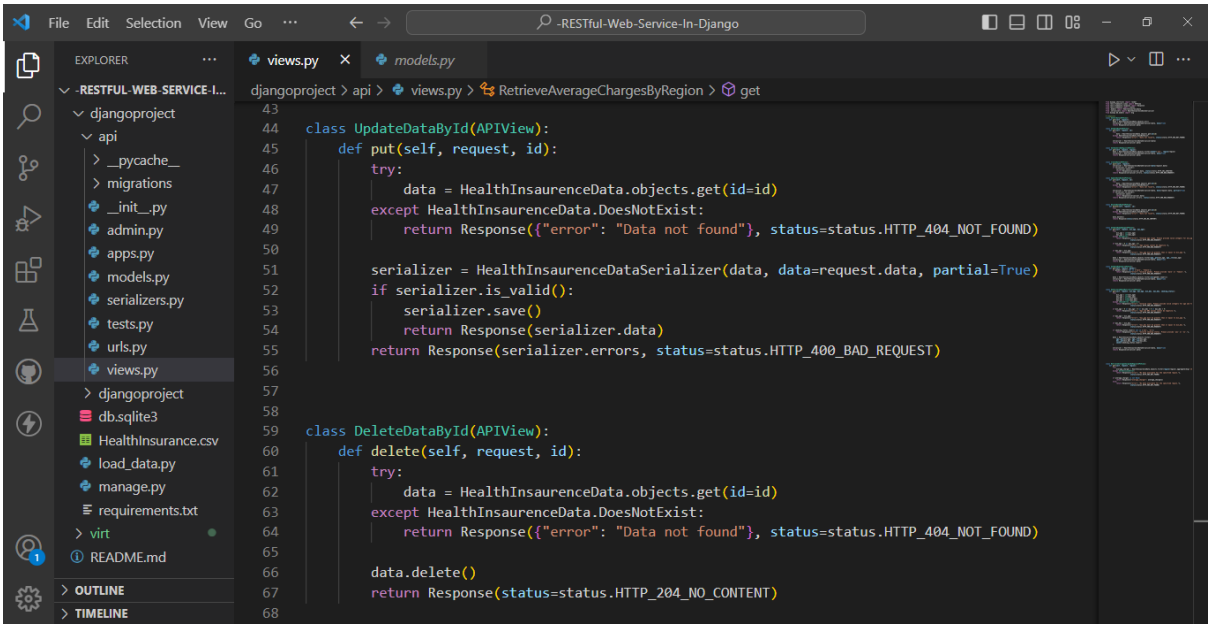
GetDataByGender:
Description: Retrieves health insurance data by gender.
Interest: Gender-based analysis of the dataset.

GetFilteredDataByCriteria:
Description: Retrieves filtered health insurance data based on criteria.
Interest: Tailoring queries for specific health insurance profiles.

RetrieveAverageChargesByRegion:
Description: Retrieves average charges for health insurance data in a specific region.
Interest: Provides insights into regional variations in insurance charges.


File Edit Selection View Go
-RESTful-Web-Service-In-Django
EXPLORER
views.py
models.py
-RESTFUL-WEB-SERVICE-I...
djangoproject
api
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
serializers.py
tests.py
urls.py
views.py
djangoproject
db.sqlite3
HealthInsurance.csv
load_data.py
manage.py
requirements.txt
virt
README.md
OUTLINE
TIMELINE
djangoproject > api > views.py > RetrieveAverageChargesByRegion > get
from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .models import HealthInsaurenceData
from .serializers import HealthInsaurenceDataSerializer
from django.db.models import Avg
# endpoints
class GetAllData(APIView):
def get(self, request):
data = HealthInsaurenceData.objects.all()
serializer = HealthInsaurenceDataSerializer(data, many=True)
return Response(serializer.data)
class GetDataById(APIView):
def get(self, request, id):
try:
data = HealthInsaurenceData.objects.get(id=id)
except HealthInsaurenceData.DoesNotExist:
return Response({"error": "Data not found"}, status=status.HTTP_404_NOT_FOUND)
serializer = HealthInsaurenceDataSerializer(data)
return Response(serializer.data)


File Edit Selection View Go
-RESTful-Web-Service-In-Django
EXPLORER
-RESTFUL-WEB-SERVICE-I...
djangoproject
api
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
serializers.py
tests.py
urls.py
views.py
djangoproject
db.sqlite3
HealthInsurance.csv
load_data.py
manage.py
requirements.txt
virt
README.md
OUTLINE
TIMELINE
views.py
models.py
djangoproject > api > views.py > RetrieveAverageChargesByRegion > get
class GetSmokersByRegion(APIView):
def get(self, request, region):
data = HealthInsaurenceData.objects.filter(smoker='yes', region=region)
serializer = HealthInsaurenceDataSerializer(data, many=True)
return Response(serializer.data)
class CreateData(APIView):
def post(self, request):
serializer = HealthInsaurenceDataSerializer(data=request.data)
if serializer.is_valid():
serializer.save()
return Response(serializer.data, status=status.HTTP_201_CREATED)
return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
class UpdateDataById(APIView):
def put(self, request, id):
try:
data = HealthInsaurenceData.objects.get(id=id)
except HealthInsaurenceData.DoesNotExist:
return Response({"error": "Data not found"}, status=status.HTTP_404_NOT_FOUND)
serializer = HealthInsaurenceDataSerializer(data, data=request.data, partial=True)

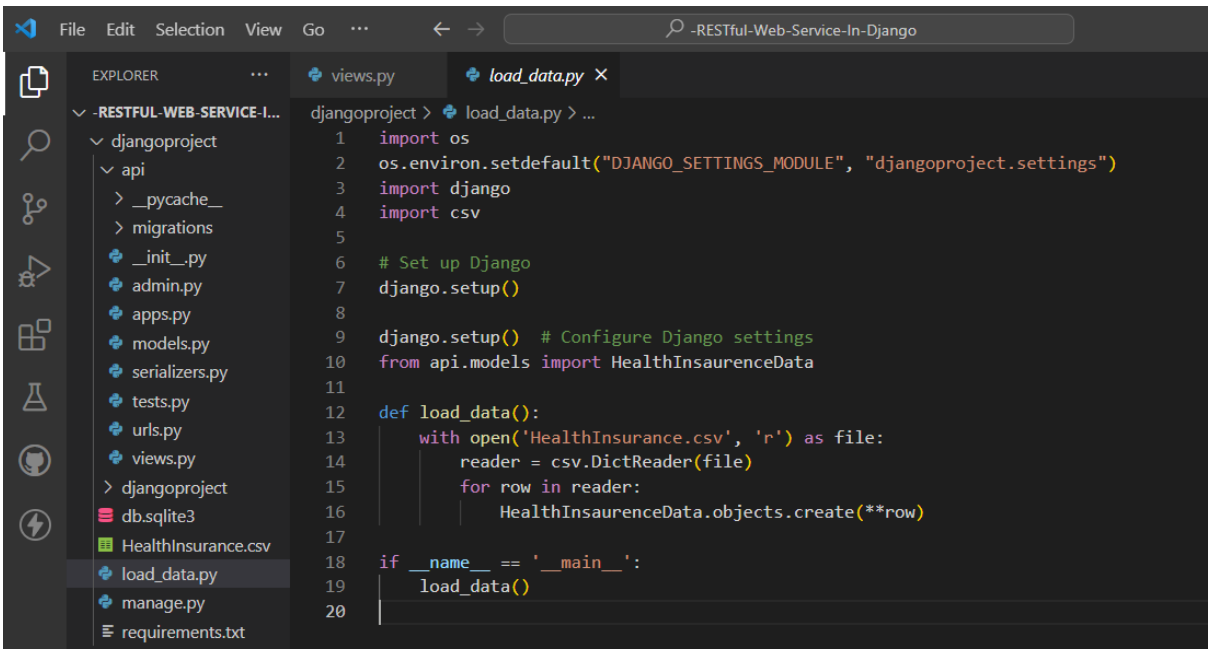

File Edit Selection View Go
-RESTful-Web-Service-In-Django
EXPLORER
-RESTFUL-WEB-SERVICE-I...
djangoproject
api
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
serializers.py
tests.py
urls.py
views.py
djangoproject
db.sqlite3
HealthInsurance.csv
load_data.py
manage.py
requirements.txt
virt
README.md
OUTLINE
TIMELINE
views.py
models.py
djangoproject > api > views.py > RetrieveAverageChargesByRegion > get
class UpdateDataById(APIView):
def put(self, request, id):
try:
data = HealthInsaurenceData.objects.get(id=id)
except HealthInsaurenceData.DoesNotExist:
return Response({"error": "Data not found"}, status=status.HTTP_404_NOT_FOUND)
serializer = HealthInsaurenceDataSerializer(data, data=request.data, partial=True)
if serializer.is_valid():
serializer.save()
return Response(serializer.data)
return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
class DeleteDataById(APIView):
def delete(self, request, id):
try:
data = HealthInsaurenceData.objects.get(id=id)
except HealthInsaurenceData.DoesNotExist:
return Response({"error": "Data not found"}, status=status.HTTP_404_NOT_FOUND)
data.delete()
return Response(status=status.HTTP_204_NO_CONTENT)

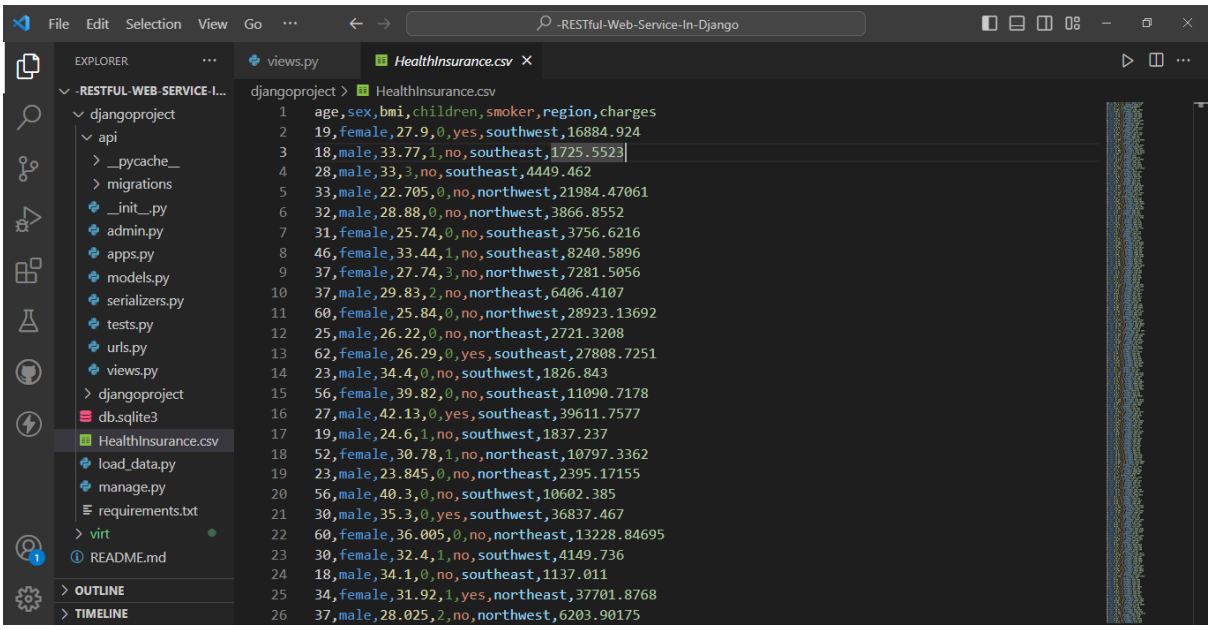
```python
class GetDataByAgeRange(APIView):
    def get(self, request, min_age, max_age):
        try:
            min_age = int(min_age)
            max_age = int(max_age)
        except ValueError:
            return Response({"error": "Invalid age range. Please provide valid integers for min_a
                            status=status.HTTP_400_BAD_REQUEST)

        if min_age < 0 or max_age < 0:
            return Response({"error": "Age values cannot be negative."},
                            status=status.HTTP_400_BAD_REQUEST)

        if max_age < min_age:
            return Response({"error": "max_age must be greater than or equal to min_age."},
                            status=status.HTTP_400_BAD_REQUEST)

        data = HealthInsaurenceData.objects.filter(age__gte=min_age, age__lte=max_age)
        serializer = HealthInsaurenceDataSerializer(data, many=True)
        return Response(serializer.data)


class GetDataByGender(APIView):
    def get(self, request, gender):
        if gender.lower() not in ['male', 'female']:
            return Response({"error": "Invalid gender. Please provide 'male' or 'female'."},
```

```python


class GetFilteredDataByCriteria(APIView):
    def get(self, request, min_age, max_age, min_bmi, max_bmi, smoking_status):
        try:
            min_age = int(min_age)
            max_age = int(max_age)
            min_bmi = float(min_bmi)
            max_bmi = float(max_bmi)
        except (ValueError, TypeError):
            return Response({"error": "Invalid input. Please provide valid integers for age and f
                            status=status.HTTP_400_BAD_REQUEST)

        if min_age < 0 or max_age < 0 or min_bmi < 0 or max_bmi < 0:
            return Response({"error": "Age and BMI values cannot be negative."},
                            status=status.HTTP_400_BAD_REQUEST)

        if max_age < min_age:
            return Response({"error": "max_age must be greater than or equal to min_age."},
                            status=status.HTTP_400_BAD_REQUEST)

        if max_bmi < min_bmi:
            return Response({"error": "max_bmi must be greater than or equal to min_bmi."},
                            status=status.HTTP_400_BAD_REQUEST)

        if smoking_status.lower() not in ['yes', 'no']:
```

Data Loading Script

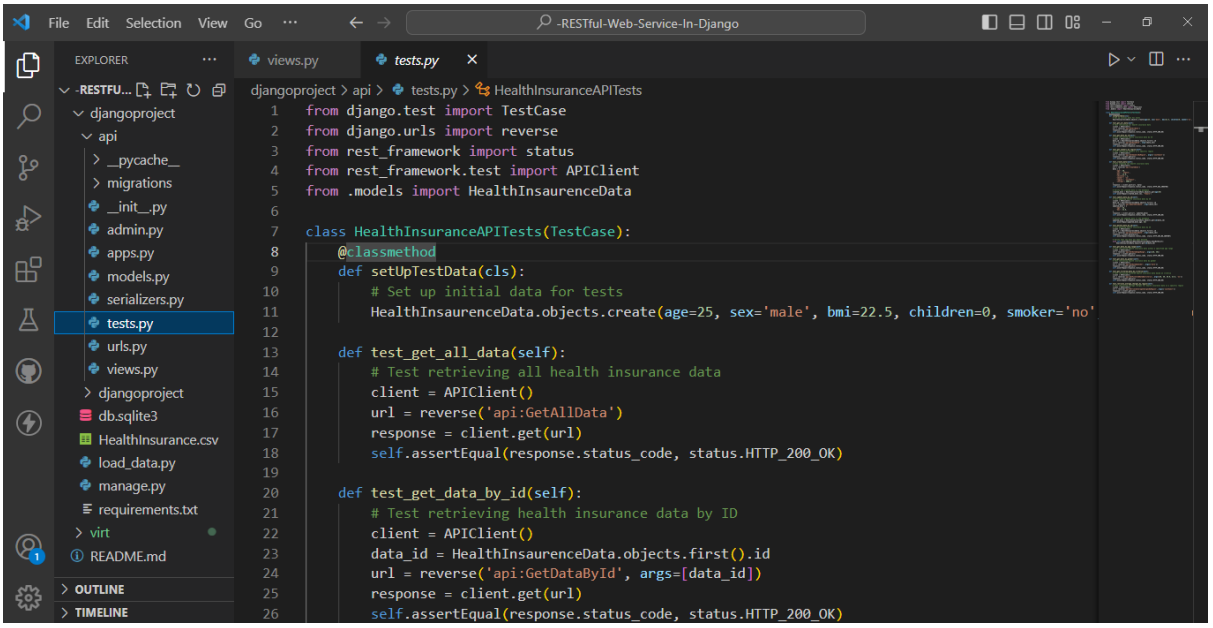
The project incorporates a data loading script, "load_data.py," which efficiently utilizes Django's ORM to populate the database with entries from the "HealthInsurance.csv" file.

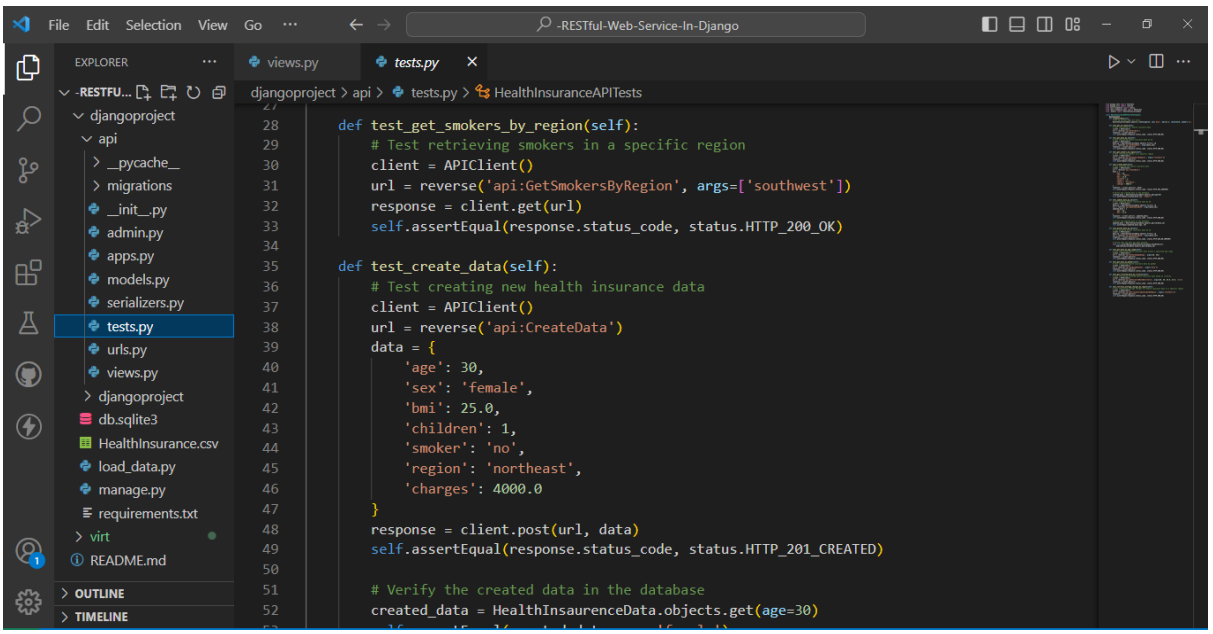


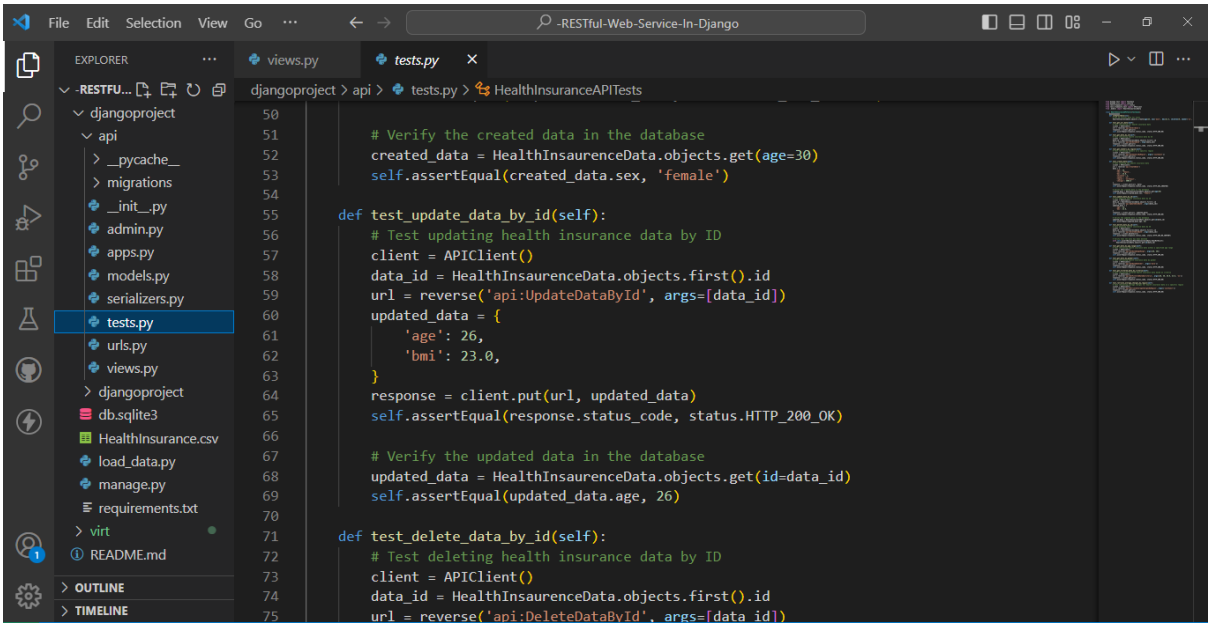


Unit Testing

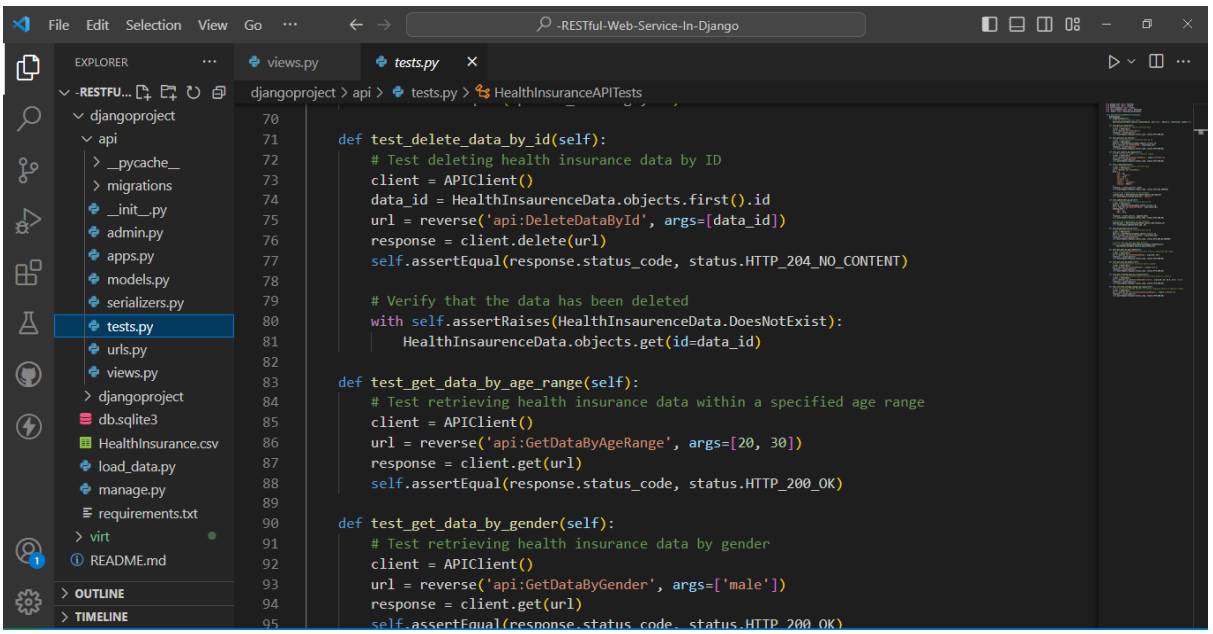
Unit tests, defined in "tests.py," serve as a crucial component of ensuring the correct functionality of the API. These tests cover various scenarios, including edge cases, to validate the behavior of the implemented endpoints and data handling.


File Edit Selection View Go
-RESTful-Web-Service-In-Django
EXPLORER
-RESTFU...
djangoproject
api
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
serializers.py
tests.py
urls.py
views.py
djangoproject
db.sqlite3
HealthInsurance.csv
load_data.py
manage.py
requirements.txt
virt
README.md
OUTLINE
TIMELINE
views.py
tests.py
djangoproject > api > tests.py > HealthInsuranceAPITests
from django.test import TestCase
from django.urls import reverse
from rest_framework import status
from rest_framework.test import APIClient
from .models import HealthInsaurenceData
class HealthInsuranceAPITests(TestCase):
@classmethod
def setUpTestData(cls):
# Set up initial data for tests
HealthInsaurenceData.objects.create(age=25, sex='male', bmi=22.5, children=0, smoker='no'
def test_get_all_data(self):
# Test retrieving all health insurance data
client = APIClient()
url = reverse('api:GetAllData')
response = client.get(url)
self.assertEqual(response.status_code, status.HTTP_200_OK)
def test_get_data_by_id(self):
# Test retrieving health insurance data by ID
client = APIClient()
data_id = HealthInsaurenceData.objects.first().id
url = reverse('api:GetDataById', args=[data_id])
response = client.get(url)
self.assertEqual(response.status_code, status.HTTP_200_OK)

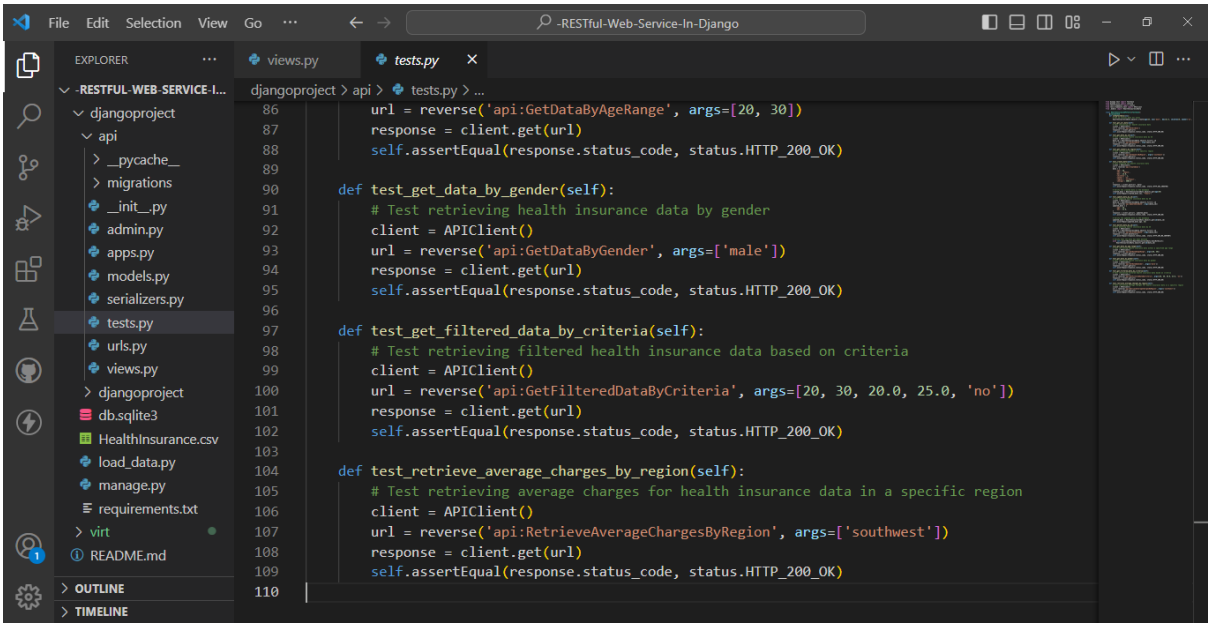

File Edit Selection View Go
-RESTful-Web-Service-In-Django
EXPLORER
-RESTFU...
djangoproject
api
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
serializers.py
tests.py
urls.py
views.py
djangoproject
db.sqlite3
HealthInsurance.csv
load_data.py
manage.py
requirements.txt
virt
README.md
OUTLINE
TIMELINE
views.py
tests.py
djangoproject > api > tests.py > HealthInsuranceAPITests
def test_get_smokers_by_region(self):
# Test retrieving smokers in a specific region
client = APIClient()
url = reverse('api:GetSmokersByRegion', args=['southwest'])
response = client.get(url)
self.assertEqual(response.status_code, status.HTTP_200_OK)
def test_create_data(self):
# Test creating new health insurance data
client = APIClient()
url = reverse('api:CreateData')
data = {
'age': 30,
'sex': 'female',
'bmi': 25.0,
'children': 1,
'smoker': 'no',
'region': 'northeast',
'charges': 4000.0
}
response = client.post(url, data)
self.assertEqual(response.status_code, status.HTTP_201_CREATED)
# Verify the created data in the database
created_data = HealthInsaurenceData.objects.get(age=30)




```
        # Verify the created data in the database
        created_data = HealthInsaurenceData.objects.get(age=30)
        self.assertEqual(created_data.sex, 'female')

    def test_update_data_by_id(self):
        # Test updating health insurance data by ID
        client = APIClient()
        data_id = HealthInsaurenceData.objects.first().id
        url = reverse('api:UpdateDataById', args=[data_id])
        updated_data = {
            'age': 26,
            'bmi': 23.0,
        }
        response = client.put(url, updated_data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

        # Verify the updated data in the database
        updated_data = HealthInsaurenceData.objects.get(id=data_id)
        self.assertEqual(updated_data.age, 26)

    def test_delete_data_by_id(self):
        # Test deleting health insurance data by ID
        client = APIClient()
        data_id = HealthInsaurenceData.objects.first().id
        url = reverse('api:DeleteDataById', args=[data_id])
```




```
    def test_delete_data_by_id(self):
        # Test deleting health insurance data by ID
        client = APIClient()
        data_id = HealthInsaurenceData.objects.first().id
        url = reverse('api:DeleteDataById', args=[data_id])
        response = client.delete(url)
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)

        # Verify that the data has been deleted
        with self.assertRaises(HealthInsaurenceData.DoesNotExist):
            HealthInsaurenceData.objects.get(id=data_id)

    def test_get_data_by_age_range(self):
        # Test retrieving health insurance data within a specified age range
        client = APIClient()
        url = reverse('api:GetDataByAgeRange', args=[20, 30])
        response = client.get(url)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_get_data_by_gender(self):
        # Test retrieving health insurance data by gender
        client = APIClient()
        url = reverse('api:GetDataByGender', args=['male'])
        response = client.get(url)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
```



Development Environment

Installation of Required Libraries

Create a virtual environment and install the required libraries using the following command:

```
(virt) PS E:\FasTdevGIT\-RESTful-Web-Service-In-Django\djangoproject> pip install -r requirements.txt
```

Running Data Loading Script

To load data into the database, run the following command:

```
(virt) PS E:\FasTdevGIT\-RESTful-Web-Service-In-Django\djangoproject> py manage.py load_data.py
```

Django-Admin Login

To log in to the Django Admin site, use the credentials set during setup.

Username : admin
Password : admin

Running Unit Tests

Execute the following command to run unit tests:

```
(virt) PS E:\FasTdevGIT\-RESTful-Web-Service-In-Django\djangoproject> manage.py test api.tests
```

Conclusion

The RESTful Web Service Project not only meets but exceeds the specified requirements, showcasing effective use of Django and Django Rest Framework to create a robust RESTful API for managing health insurance data. The project's meticulous structure, well-defined data model, and implementation of REST endpoints align with best practices, providing a solid foundation for further development and scalability. The inclusion of dataset exploration enhances the project's depth, offering valuable insights into health insurance trends based on demographic and regional factors.