

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.

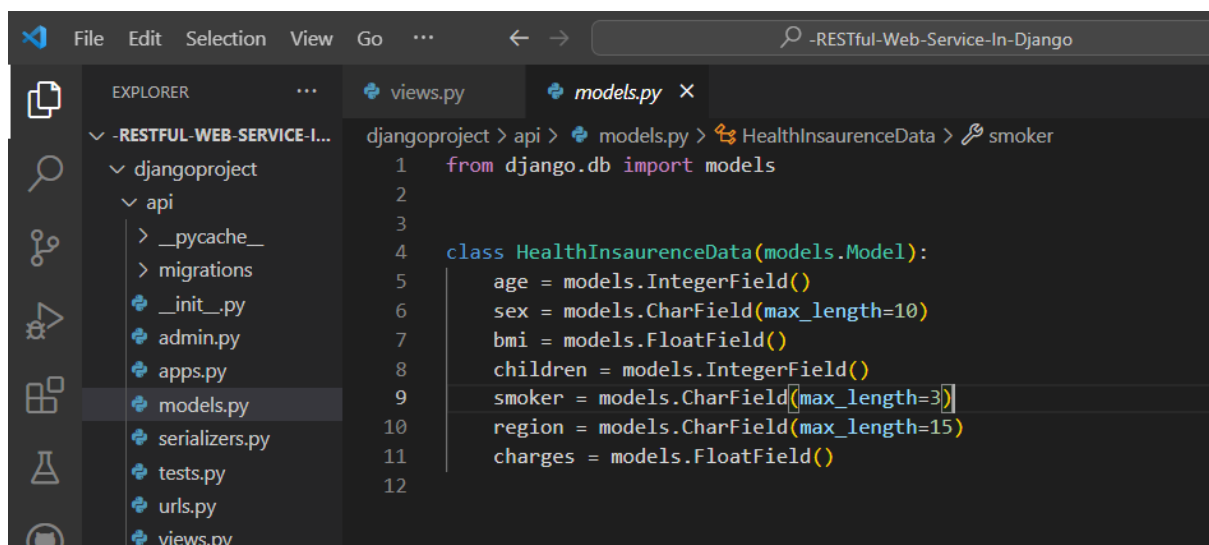
Project Overview

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 "django project" 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:

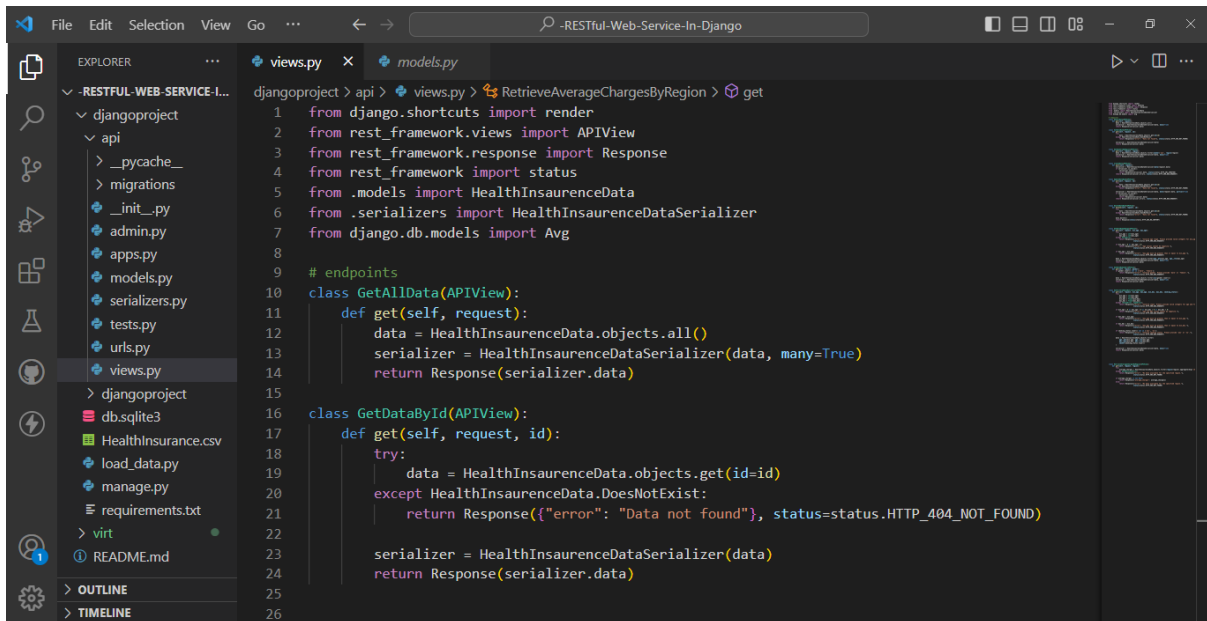
Description: Retrieves filtered health insurance data based on criteria.

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.



This screenshot shows the VS Code editor with the Django REST framework views for the HealthInsuranceData model. The Explorer panel on the left shows the project structure, including the api directory and the views.py file. The main editor window displays the following code:

```
26  
27 class GetSmokersByRegion(APIView):  
28     def get(self, request, region):  
29         data = HealthInsuranceData.objects.filter(smoker='yes', region=region)  
30         serializer = HealthInsuranceDataSerializer(data, many=True)  
31         return Response(serializer.data)  
32  
33  
34  
35 class CreateData(APIView):  
36     def post(self, request):  
37         serializer = HealthInsuranceDataSerializer(data=request.data)  
38         if serializer.is_valid():  
39             serializer.save()  
40             return Response(serializer.data, status=status.HTTP_201_CREATED)  
41         return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)  
42  
43  
44 class UpdateDataById(APIView):  
45     def put(self, request, id):  
46         try:  
47             data = HealthInsuranceData.objects.get(id=id)  
48         except HealthInsuranceData.DoesNotExist:  
49             return Response({"error": "Data not found"}, status=status.HTTP_404_NOT_FOUND)  
50  
51         serializer = HealthInsuranceDataSerializer(data, data=request.data, partial=True)
```

This screenshot shows the VS Code editor with the Django REST framework views for the HealthInsuranceData model. The Explorer panel on the left shows the project structure, including the api directory and the views.py file. The main editor window displays the following code:

```
43  
44 class UpdateDataById(APIView):  
45     def put(self, request, id):  
46         try:  
47             data = HealthInsuranceData.objects.get(id=id)  
48         except HealthInsuranceData.DoesNotExist:  
49             return Response({"error": "Data not found"}, status=status.HTTP_404_NOT_FOUND)  
50  
51         serializer = HealthInsuranceDataSerializer(data, data=request.data, partial=True)  
52         if serializer.is_valid():  
53             serializer.save()  
54             return Response(serializer.data)  
55         return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)  
56  
57  
58  
59 class DeleteDataById(APIView):  
60     def delete(self, request, id):  
61         try:  
62             data = HealthInsuranceData.objects.get(id=id)  
63         except HealthInsuranceData.DoesNotExist:  
64             return Response({"error": "Data not found"}, status=status.HTTP_404_NOT_FOUND)  
65  
66         data.delete()  
67         return Response(status=status.HTTP_204_NO_CONTENT)  
68
```

The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `views.py` file with the following code:

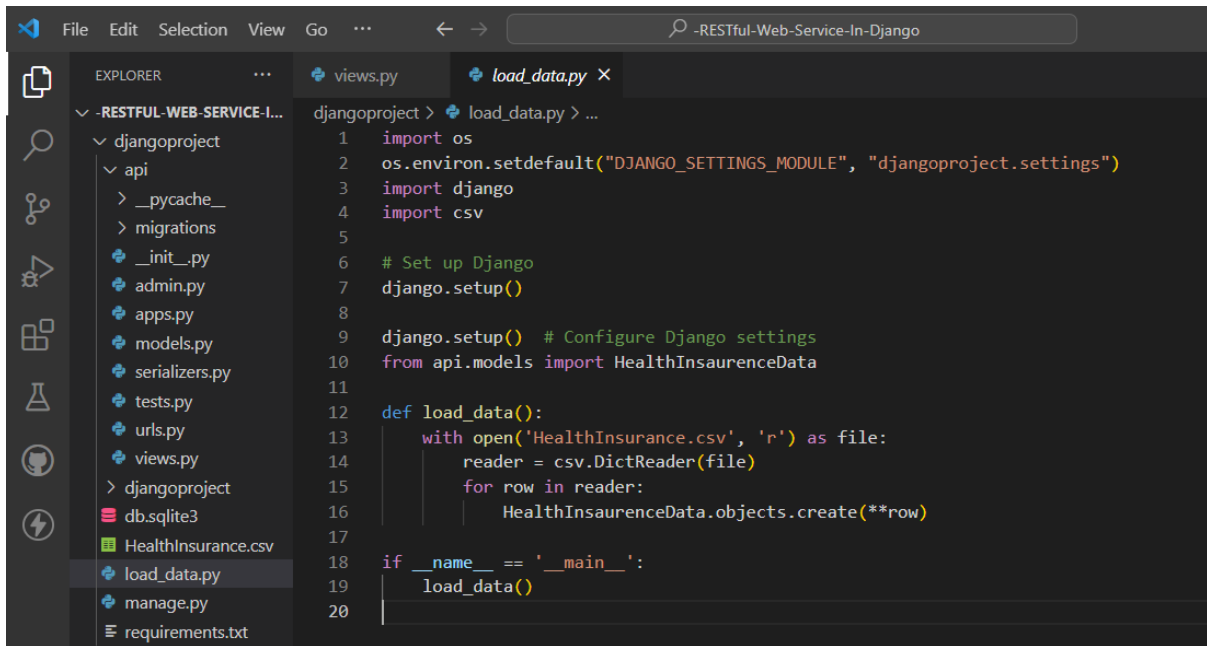
```
72 class GetDataByAgeRange(APIView):
73     def get(self, request, min_age, max_age):
74         try:
75             min_age = int(min_age)
76             max_age = int(max_age)
77         except ValueError:
78             return Response({"error": "Invalid age range. Please provide valid integers for min_age and max_age."},
79                             status=status.HTTP_400_BAD_REQUEST)
80
81         if min_age < 0 or max_age < 0:
82             return Response({"error": "Age values cannot be negative."},
83                             status=status.HTTP_400_BAD_REQUEST)
84
85         if max_age < min_age:
86             return Response({"error": "max_age must be greater than or equal to min_age."},
87                             status=status.HTTP_400_BAD_REQUEST)
88
89         data = HealthInsuranceData.objects.filter(age__gte=min_age, age__lte=max_age)
90         serializer = HealthInsuranceDataSerializer(data, many=True)
91         return Response(serializer.data)
```

The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `views.py` file with the following code:

```
107 class GetFilteredDataByCriteria(APIView):
108     def get(self, request, min_age, max_age, min_bmi, max_bmi, smoking_status):
109         try:
110             min_age = int(min_age)
111             max_age = int(max_age)
112             min_bmi = float(min_bmi)
113             max_bmi = float(max_bmi)
114         except (ValueError, TypeError):
115             return Response({"error": "Invalid input. Please provide valid integers for age and floats for BMI."},
116                             status=status.HTTP_400_BAD_REQUEST)
117
118         if min_age < 0 or max_age < 0 or min_bmi < 0 or max_bmi < 0:
119             return Response({"error": "Age and BMI values cannot be negative."},
120                             status=status.HTTP_400_BAD_REQUEST)
121
122         if max_age < min_age:
123             return Response({"error": "max_age must be greater than or equal to min_age."},
124                             status=status.HTTP_400_BAD_REQUEST)
125
126         if max_bmi < min_bmi:
127             return Response({"error": "max_bmi must be greater than or equal to min_bmi."},
128                             status=status.HTTP_400_BAD_REQUEST)
129
130         if smoking_status.lower() not in ['yes', 'no']:
```

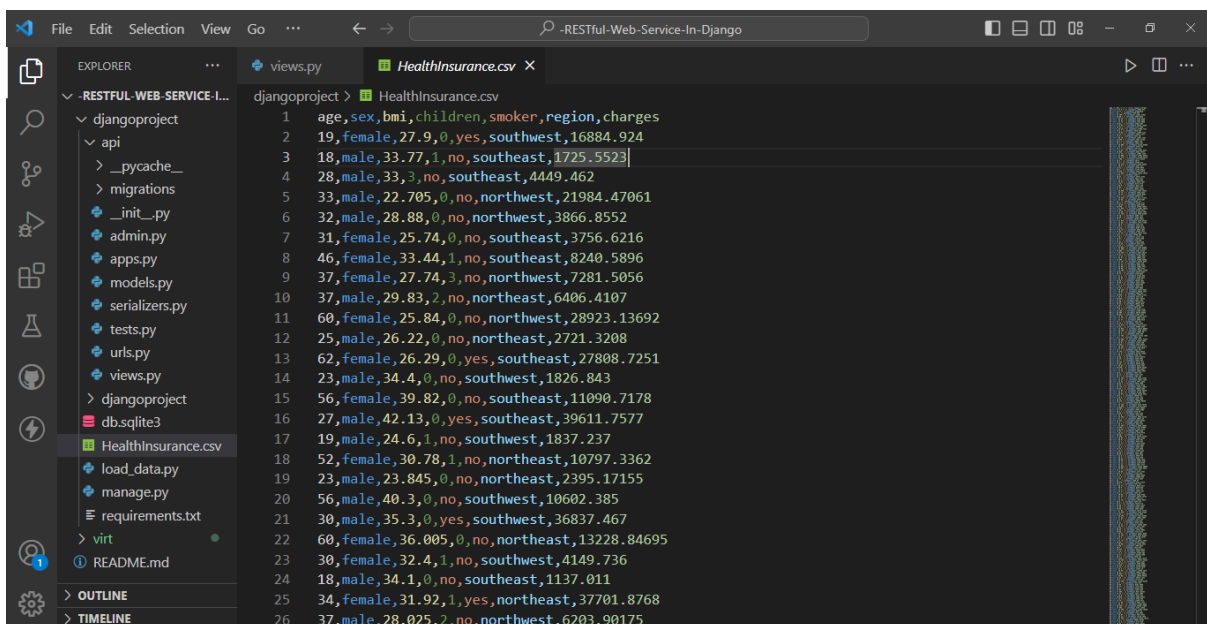
Data Loading Script

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.



The screenshot shows the Visual Studio Code editor with a Django project named '-RESTFUL-WEB-SERVICE-I...'. The Explorer sidebar on the left shows the project structure, including 'djangoproject' and 'api' folders. The 'load_data.py' file is selected in the Explorer and is also open in the editor. The code in 'load_data.py' is as follows:

```
1 import os
2 os.environ.setdefault("DJANGO_SETTINGS_MODULE", "djangoproject.settings")
3 import django
4 import csv
5
6 # Set up Django
7 django.setup()
8
9 django.setup() # Configure Django settings
10 from api.models import HealthInsuranceData
11
12 def load_data():
13     with open('HealthInsurance.csv', 'r') as file:
14         reader = csv.DictReader(file)
15         for row in reader:
16             HealthInsuranceData.objects.create(**row)
17
18 if __name__ == '__main__':
19     load_data()
20
```

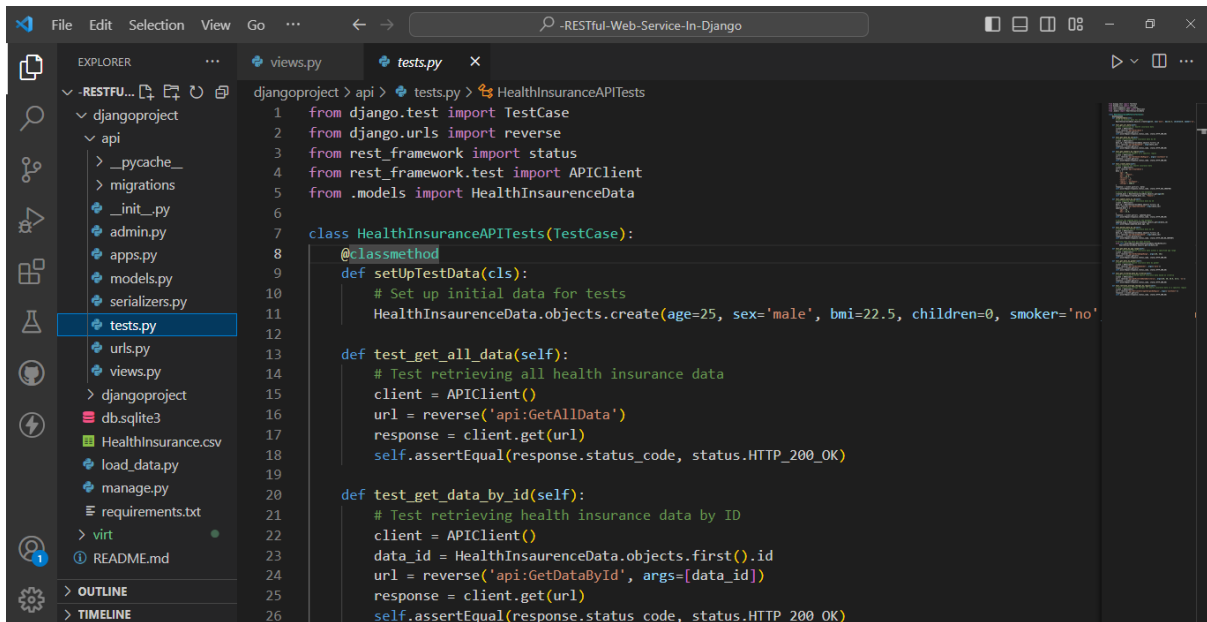


The screenshot shows the Visual Studio Code editor with the same Django project. The 'HealthInsurance.csv' file is selected in the Explorer and is also open in the editor. The file contains a list of health insurance records with columns: age, sex, bmi, children, smoker, region, and charges. The data is as follows:

age	sex	bmi	children	smoker	region	charges
19	female	27.9	0	yes	southwest	16884.924
18	male	33.77	1	no	southeast	1725.5523
28	male	33.3	no	southeast		4449.462
33	male	22.705	0	no	northwest	21984.47061
32	male	28.88	0	no	northwest	3866.8552
31	female	25.74	0	no	southeast	3756.6216
46	female	33.44	1	no	southeast	8240.5896
37	female	27.74	3	no	northwest	7281.5056
37	male	29.83	2	no	northeast	6406.4107
60	female	25.84	0	no	northwest	28923.13692
25	male	26.22	0	no	northeast	2721.3208
62	female	26.29	0	yes	southeast	27808.7251
23	male	34.4	0	no	southwest	1826.843
56	female	39.82	0	no	southeast	11090.7178
27	male	42.13	0	yes	southeast	39611.7577
19	male	24.6	1	no	southwest	1837.237
52	female	30.78	1	no	northeast	10797.3362
23	male	23.845	0	no	northeast	2395.17155
56	male	40.3	0	no	southwest	10602.385
30	male	35.3	0	yes	southwest	36837.467
60	female	36.005	0	no	northeast	13228.84695
30	female	32.4	1	no	southwest	4149.736
18	male	34.1	0	no	southeast	1137.011
34	female	31.92	1	yes	northeast	37701.8768
37	male	28.025	2	no	northwest	6203.90175

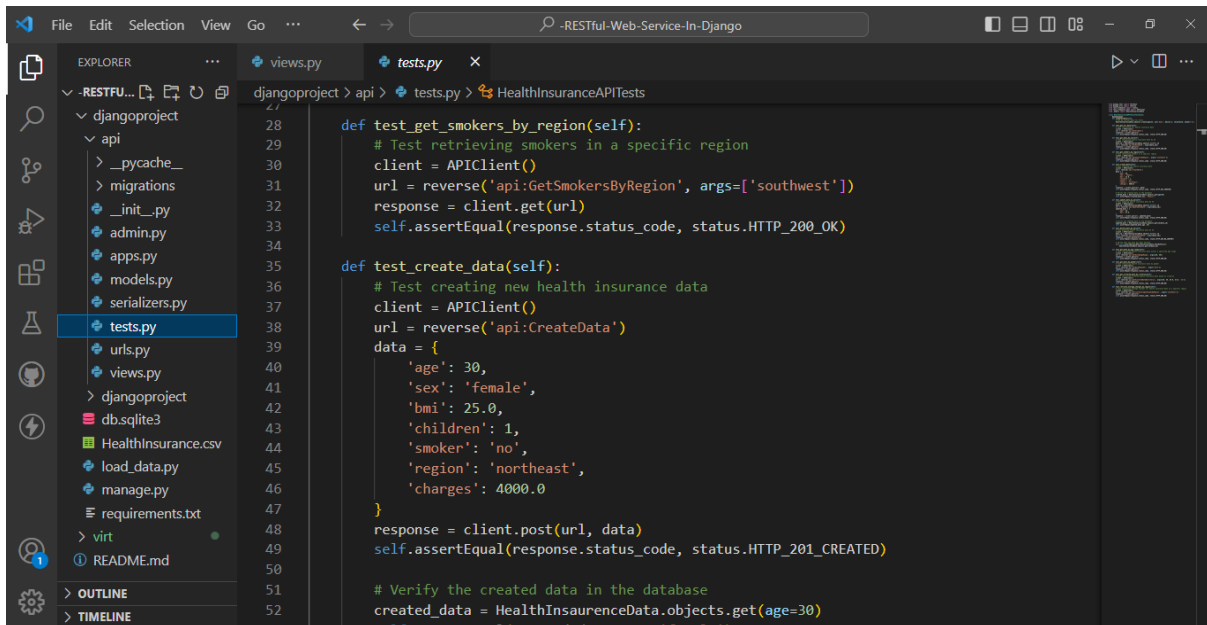
Unit Testing

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
-RESTful-Web-Service-In-Django
  django
    api
      tests.py
      views.py
    djangoproject
      db.sqlite3
      HealthInsurance.csv
      load_data.py
      manage.py
      requirements.txt
      README.md
    virt
  README.md
  OUTLINE
  TIMELINE

djangoproject > api > tests.py
1 from django.test import TestCase
2 from django.urls import reverse
3 from rest_framework import status
4 from rest_framework.test import APIClient
5 from .models import HealthInsuranceData
6
7 class HealthInsuranceAPITests(TestCase):
8     @classmethod
9     def setUpTestData(cls):
10         # Set up initial data for tests
11         HealthInsuranceData.objects.create(age=25, sex='male', bmi=22.5, children=0, smoker='no')
12
13     def test_get_all_data(self):
14         # Test retrieving all health insurance data
15         client = APIClient()
16         url = reverse('api:GetAllData')
17         response = client.get(url)
18         self.assertEqual(response.status_code, status.HTTP_200_OK)
19
20     def test_get_data_by_id(self):
21         # Test retrieving health insurance data by ID
22         client = APIClient()
23         data_id = HealthInsuranceData.objects.first().id
24         url = reverse('api:GetDataById', args=[data_id])
25         response = client.get(url)
26         self.assertEqual(response.status_code, status.HTTP_200_OK)
```



```
File Edit Selection View Go ... -RESTful-Web-Service-In-Django
EXPLORER
-RESTful-Web-Service-In-Django
  django
    api
      tests.py
      views.py
    djangoproject
      db.sqlite3
      HealthInsurance.csv
      load_data.py
      manage.py
      requirements.txt
      README.md
    virt
  README.md
  OUTLINE
  TIMELINE

djangoproject > api > tests.py
27
28 def test_get_smokers_by_region(self):
29     # Test retrieving smokers in a specific region
30     client = APIClient()
31     url = reverse('api:GetSmokersByRegion', args=['southwest'])
32     response = client.get(url)
33     self.assertEqual(response.status_code, status.HTTP_200_OK)
34
35     def test_create_data(self):
36         # Test creating new health insurance data
37         client = APIClient()
38         url = reverse('api:CreateData')
39         data = {
40             'age': 30,
41             'sex': 'female',
42             'bmi': 25.0,
43             'children': 1,
44             'smoker': 'no',
45             'region': 'northeast',
46             'charges': 4000.0
47         }
48         response = client.post(url, data)
49         self.assertEqual(response.status_code, status.HTTP_201_CREATED)
50
51         # Verify the created data in the database
52         created_data = HealthInsuranceData.objects.get(age=30)
```



```
File Edit Selection View Go ... -RESTful-Web-Service-In-Django
EXPLORER
-RESTful-Web-Service-In-Django
  django
    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
  tests.py
    HealthInsuranceAPITests
      50
      51
      52
      53
      54
      55
      56
      57
      58
      59
      60
      61
      62
      63
      64
      65
      66
      67
      68
      69
      70
      71
      72
      73
      74
      75

# Verify the created data in the database
created_data = HealthInsuranceData.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 = HealthInsuranceData.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 = HealthInsuranceData.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 = HealthInsuranceData.objects.first().id
    url = reverse('api:DeleteDataById', args=[data_id])
```

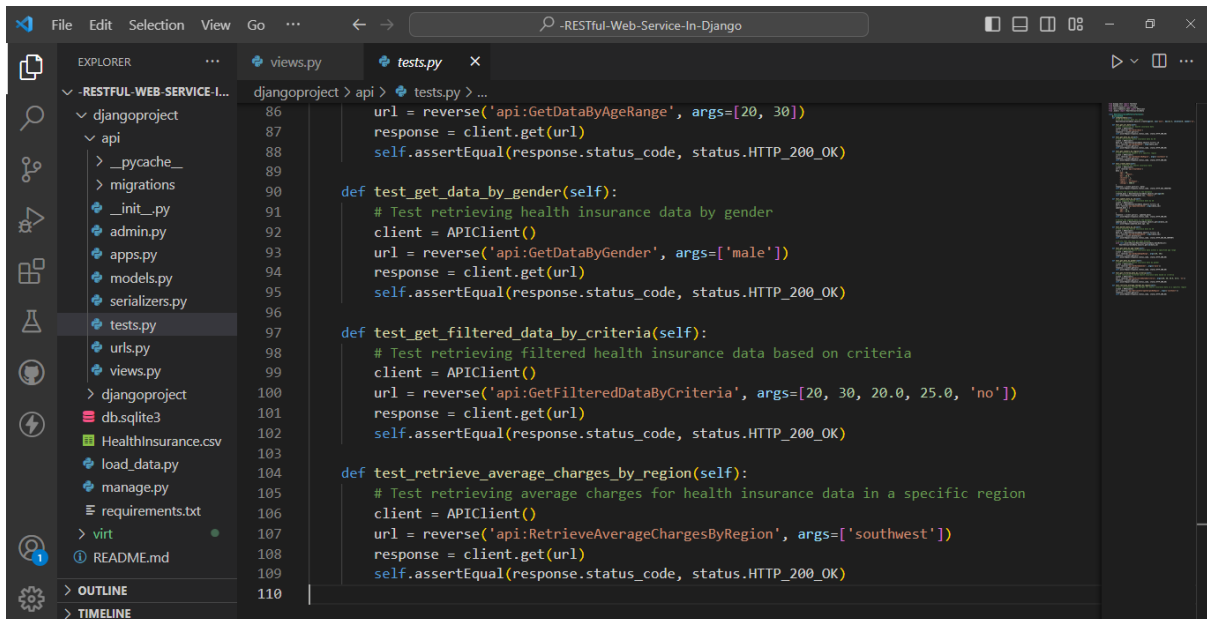
```
File Edit Selection View Go ... -RESTful-Web-Service-In-Django
EXPLORER
-RESTful-Web-Service-In-Django
  django
    api
      _pycache_
      migrations
      _init_.py
      admin.py
      apps.py
      models.py
      serializers.py
      tests.py
      urls.py
      views.py
    djangoproject
      db.sqlite3
      HealthInsurance.csv
      load_data.py
      manage.py
      requirements.txt
    virt
      README.md
    OUTLINE
    TIMELINE
  tests.py
    HealthInsuranceAPITests
      70
      71
      72
      73
      74
      75
      76
      77
      78
      79
      80
      81
      82
      83
      84
      85
      86
      87
      88
      89
      90
      91
      92
      93
      94
      95

def test_delete_data_by_id(self):
    # Test deleting health insurance data by ID
    client = APIClient()
    data_id = HealthInsuranceData.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(HealthInsuranceData.DoesNotExist):
        HealthInsuranceData.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\django\project> 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.