Báo cáo kiến trúc và thiết kế phần mềm

Chương 2:

Phần 2.1 trình bày mô hình kiến trúc microservice

a. Các khái niệm

Kiến trúc microservice là một phương pháp xây dựng ứng dụng phần mềm bằng cách chia nhỏ thành các dịch vụ nhỏ và độc lập. Mỗi dịch vụ nhỏ này được gọi là microservice và thường chạy trên một quy trình hoặc một máy chủ độc lập.

Các microservice trong kiến trúc này được thiết kế để thực hiện một chức năng cụ thể trong ứng dụng và có khả năng làm việc độc lập với nhau thông qua giao tiếp qua mạng. Mỗi microservice có thể được phát triển, triển khai và mở rộng độc lập với nhau, cho phép các nhóm phát triển tập trung vào việc phát triển một phần nhỏ của ứng dụng mà không cần phải quan tâm đến toàn bộ hệ thống.

Kiến trúc microservice có một số lợi ích, bao gồm:

1. Tính linh hoạt: Các dịch vụ nhỏ có thể được phát triển và triển khai độc lập, cho phép các nhóm phát triển làm việc theo phương pháp Agile và nhanh chóng thay đổi.
2. Dễ dàng mở rộng: Do mỗi microservice là độc lập, nên có thể mở rộng riêng từng dịch vụ để đáp ứng nhu cầu tải cao mà không ảnh hưởng đến các phần khác của hệ thống.
3. Dễ dàng tái sử dụng: Với kiến trúc microservice, các dịch vụ nhỏ có thể được sử dụng lại trong các ứng dụng khác, giúp tăng tính tái sử dụng và hiệu quả trong quá trình phát triển.

Tuy nhiên, kiến trúc microservice cũng có một số thách thức như:

1. Quản lý và theo dõi: Vì có nhiều microservice chạy độc lập, việc quản lý và theo dõi toàn bộ hệ thống trở nên phức tạp hơn.
2. Giao tiếp mạng: Việc giao tiếp qua mạng giữa các microservice đòi hỏi hiệu năng và độ tin cậy cao để đảm bảo sự hoạt động chính xác và hiệu quả của hệ thống.
3. Quản lý khả năng tương thích: Các microservice có thể được phát triển bằng ngôn ngữ và công nghệ khác nhau, do đó, quản lý khả năng tương thích giữa chúng có thể trở thành một thách thức.

Tóm lại, kiến trúc microservice cung cấp một phương pháp linh hoạt và mở rộng để xây dựng các ứng dụng phức tạp. Nó giúp tách biệt các chức năng khác nhau thành các dịch vụ nhỏ và độc lập, tạo điều kiện cho phát triển và triển khai linh hoạt hơn.

Để cung cấp thêm thông tin chi tiết về kiến trúc microservice, hãy xem xét các yếu tố và thành phần quan trọng sau đây:

1. Dịch vụ độc lập: Trong kiến trúc microservice, ứng dụng được chia thành các dịch vụ nhỏ, mỗi dịch vụ đảm nhận một chức năng cụ thể. Ví dụ, trong một ứng dụng thương mại điện tử, có thể có các dịch vụ riêng biệt cho quản lý khách hàng, quản lý sản phẩm, thanh toán, vận chuyển, đánh giá sản phẩm và nhiều hơn nữa.

2. Giao tiếp qua mạng: Các microservice trong kiến trúc microservice giao tiếp với nhau thông qua một cơ chế giao tiếp qua mạng, thường là giao thức HTTP/REST. Mỗi microservice có một giao diện API công khai để cho phép các microservice khác truy cập vào chức năng của nó. Giao tiếp qua mạng giữa các microservice cho phép chúng hoạt động độc lập và tương tác để thực hiện các nhiệm vụ cụ thể.

3. Độc lập triển khai và mở rộng: Mỗi microservice có thể được triển khai và quản lý độc lập với nhau. Điều này có nghĩa là các nhóm phát triển có thể làm việc độc lập trên một hoặc nhiều microservice mà không ảnh hưởng đến các phần khác của hệ thống. Điều này tạo điều kiện cho sự linh hoạt và tăng khả năng mở rộng của ứng dụng, vì bạn có thể mở rộng chỉ những phần cần thiết mà không cần phải mở rộng toàn bộ hệ thống.

4. Quản lý dữ liệu: Trong kiến trúc microservice, mỗi microservice có thể có cơ sở dữ liệu riêng của mình. Điều này giúp tách biệt dữ liệu của từng dịch vụ và giải quyết vấn đề quản lý dữ liệu phức tạp. Mỗi microservice có thể sử dụng loại cơ sở dữ liệu phù hợp như cơ sở dữ liệu quan hệ, NoSQL hoặc bộ nhớ đệm, tùy thuộc vào yêu cầu của chức năng cụ thể.

5. Quản lý lỗi và giám sát: Với nhiều microservice chạy độc lập, việc quản lý lỗi và giám sát trở nên quan trọng. Các công cụ giám sát và khắc phục sự cố như trình giám sát hệ thống, trình nhật ký và thông báo lỗi có thể được triển khai để theo dõi và xử lý các sự cố xảy ra trong các microservice.

6. Phân phối công việc và khả năng tương thích: Trong kiến trúc microservice, công việc được phân chia giữa các microservice. Mỗi dịch vụ nhỏ có thể được phát triển bằng ngôn ngữ lập trình và công nghệ khác nhau, tùy thuộc vào yêu cầu và sự chuyên môn của nhóm phát triển. Điều này cho phép sử dụng những công nghệ tốt nhất cho từng phần của ứng dụng.

Tổng quan, kiến trúc microservice tách biệt các thành phần của ứng dụng thành các dịch vụ nhỏ, độc lập, cho phép phát triển và triển khai linh hoạt, mở rộng dễ dàng và quản lý dễ dàng. Mô hình này đặc biệt hữu ích trong các ứng dụng lớn và phức tạp với nhiều chức năng và yêu cầu khác nhau.

b. Ví dụ về ứng dụng quản lý sinh viên bằng django

Tóm tắt bài toán:

Để tạo một ứng dụng Django quản lý sinh viên nhỏ với kiến trúc microservice, bạn có thể áp dụng một số nguyên tắc sau:

1. Phân tích yêu cầu chức năng: Đầu tiên, xác định các chức năng cụ thể mà ứng dụng quản lý sinh viên của bạn cần. Ví dụ: quản lý thông tin sinh viên, quản lý khoá học, đăng ký khóa học, v.v.

2. Xác định các microservice: Dựa trên yêu cầu chức năng, xác định các microservice mà bạn muốn tạo. Mỗi microservice sẽ đảm nhận một chức năng cụ thể trong ứng dụng. Ví dụ: một microservice để quản lý thông tin sinh viên, một microservice để quản lý khoá học, v.v.

3. Thiết kế giao diện API: Đối với mỗi microservice, thiết kế giao diện API công khai để cho phép giao tiếp với các microservice khác. Bạn có thể sử dụng Django Rest Framework để tạo các API RESTful cho mỗi microservice.
4. Triển khai và quản lý các microservice: Triển khai và quản lý các microservice độc lập. Bạn có thể sử dụng các công cụ như Docker hoặc Kubernetes để triển khai và quản lý các microservice.
5. Giao tiếp giữa các microservice: Các microservice cần giao tiếp với nhau để thực hiện các chức năng. Bạn có thể sử dụng giao thức HTTP/REST để gửi yêu cầu và nhận phản hồi giữa các microservice. Các yêu cầu có thể được thực hiện qua URL và các phương thức HTTP như GET, POST, PUT, DELETE.
6. Quản lý dữ liệu: Mỗi microservice có thể có cơ sở dữ liệu riêng của mình để lưu trữ dữ liệu liên quan đến chức năng của nó. Trong trường hợp này, bạn có thể sử dụng cơ sở dữ liệu Django theo mô hình ORM (Object-Relational Mapping) để quản lý dữ liệu trong từng microservice.
7. Quản lý lỗi và giám sát: Sử dụng các công cụ và kỹ thuật giám sát hệ thống để theo dõi hiệu suất và xử lý sự cố trong các microservice. Đảm bảo rằng bạn có cơ chế xử lý lỗi và ghi lại các thông báo lỗi từ các microservice.

Tóm lại, để tạo một ứng dụng Django quản lý sinh viên với kiến trúc microservice, bạn cần phân tích yêu cầu, xác định các microservice, thiết kế giao diện API, triển khai và quản lý các microservice, giao tiếp giữa chúng, quản lý dữ liệu và giám sát hệ thống. Quá trình này yêu cầu sự kiên nhẫn và hiểu biết về Django và kiến thức về kiến trúc microservice.

Bước 1: tạo project

Sau khi cài đặt Django thành công, bạn có thể tạo một dự án Django mới bằng cách làm theo các bước dưới đây:

1. **Di chuyển đến thư mục nơi bạn muốn tạo dự án Django.**
2. **Chạy lệnh sau để tạo một dự án Django mới:**

```
$ django-admin startproject quan_ly_sinh_vien
```

3. Chạy project

```
$ python manage.py runserver
```

Kết quả:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttype
s, sessions.
Run 'python manage.py migrate' to apply them.
June 07, 2023 - 21:03:36
Django version 4.0, using settings 'quan_ly_sinh_vien.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Migrate ra database

```
$ python manage.py migrate
```

Kết quả:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Bước 2: tạo service giangvien

1. **Tạo service giangvien**

```
$ python manage.py startapp giangvien
```

2. **Tạo model giangvien**

```
from django.db import models


# Create your models here.
class GiangVien(models.Model):
    id = models.AutoField(primary_key=True)
    username = models.CharField(max_length=20)
    ma_gv = models.CharField(max_length=20)
    ten_gv = models.CharField(max_length=100)
```

```
    email = models.CharField(max_length=50)
    dob = models.CharField(max_length=20)
    address = models.CharField(max_length=200)

    def __str__(self):
        return self.username
```

3. **Migrate database**

```
python manage.py makemigrations giangvien

python manage.py migrate
```

4. **Định nghĩa API cho microservice**:

- Trong thư mục **giangvien**, tạo một file **views.py**.
- Trong file **views.py**, định nghĩa các API cho quản lý giảng viên.

```
  from rest_framework import status

from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework.generics import CreateAPIView, RetrieveAPIView,
UpdateAPIView, DestroyAPIView
from .models import GiangVien
from .serializers import GiangVienSerializer

class GiangVienCreateAPIView(CreateAPIView):
    serializer_class = GiangVienSerializer
    def post(self, request, *args, **kwargs):
        data = {
            'username': request.data.get('username'),
            'ma_gv': request.data.get('ma_gv'),
            'ten_gv': request.data.get('ten_gv'),
            'email': request.data.get('email'),
            'dob': request.data.get('dob'),
            'address': request.data.get('address'),
        }
        serializer = GiangVienSerializer(data=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 GiangVienDetail(RetrieveAPIView, UpdateAPIView, DestroyAPIView):
    queryset = GiangVien.objects.all()
    serializer_class = GiangVienSerializer
```

5. Tiếp theo, tạo một file **serializers.py** trong thư mục **giangvien** và định nghĩa trình phân giải JSON cho mô hình **GiangVien**:

```
   from rest_framework import serializers

from .models import GiangVien

class GiangVienSerializer(serializers.ModelSerializer):
    class Meta:
        model = GiangVien
        fields = '__all__'
```

6. Cấu hình URL cho microservice:

- Trong thư mục gốc của dự án Django, mở file **urls.py**.
- Import **DefaultRouter** và **include** từ Django Rest Framework và import **GiangVienViewSet** từ **giangvien.views**.
- Thêm các đường dẫn URL cho microservice giảng viên vào **urlpatterns**.

```
  from django.urls import path, include

from rest_framework.routers import DefaultRouter
from giangvien.views import GiangVienCreateAPIView, GiangVienDetail


urlpatterns = [
  path('create', GiangVienCreateAPIView.as_view()),
  path('<int:pk>',GiangVienDetail.as_view() ),
]
```

7. Migrate service giangvien

Khai báo ‘giangvien’ vào project

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',

     'giangvien',
]
```

Chạy lệnh

```
$ python manage.py migrate
```

Bước 3: Tạo service sinhvien

```
$ python manage.py startapp sinhvien
```

1. Tạo model sinhvien

```
    from django.db import models

# Create your models here.
class SinhVien(models.Model):
    id = models.AutoField(primary_key=True)
    username = models.CharField(max_length=20)
    ma_sv = models.CharField(max_length=20)
    ten_sv = models.CharField(max_length=100)
    email = models.CharField(max_length=50)
    dob = models.CharField(max_length=20)
    address = models.CharField(max_length=200)
    gv_id = models.IntegerField(max_length=100)

    def __str__(self):
        return self.username
```

2. Định nghĩa API cho microservice:

- Trong thư mục **sinhvien**, tạo một file **views.py**.
- Trong file **views.py**, định nghĩa các API cho quản lý giảng viên.

```
    from rest_framework import status

from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework.generics import CreateAPIView, RetrieveAPIView,
UpdateAPIView, DestroyAPIView
from .models import SinhVien
from .serializers import SinhVienSerializer
```

```
class SinhVienCreateAPIView(CreateAPIView):
    serializer_class = SinhVienSerializer
    def post(self, request, *args, **kwargs):
        data = {
            'username': request.data.get('username'),
            'ma_sv': request.data.get('ma_sv'),
            'ten_sv': request.data.get('ten_sv'),
            'email': request.data.get('email'),
            'dob': request.data.get('dob'),
            'address': request.data.get('address'),
            'gv_id': request.data.get('gv_id'),
        }
        serializer = SinhVienSerializer(data=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 SinhVienDetail(RetrieveAPIView, UpdateAPIView, DestroyAPIView):
    queryset = SinhVien.objects.all()
    serializer_class = SinhVienSerializer
```

3. Tiếp theo, tạo một file **serializers.py** trong thư mục **sinhvien** và định nghĩa trình phân giải JSON cho mô hình **SinhVien**:

```
  from rest_framework import serializers

from .models import SinhVien

class SinhVienSerializer(serializers.ModelSerializer):
    class Meta:
        model = SinhVien
        fields = '__all__'
```

4. Cấu hình URL cho microservice:
    - Trong thư mục gốc của dự án Django, mở file **urls.py**.
    - Import **DefaultRouter** và **include** từ Django Rest Framework và import **SinhVienViewSet** từ **sinhvien.views**.
    - Thêm các đường dẫn URL cho microservice giảng viên vào **urlpatterns**.

```
    from django.urls import include, path

from rest_framework import routers
```

```
from sinhvien.views import SinhVienCreateAPIView, SinhVienDetail

urlpatterns = [
    path('create', SinhVienCreateAPIView.as_view()),
    path('<int:pk>', SinhVienDetail.as_view()),
]
```

5. Migrate service sinhvien

Khai báo ‘sinhvien’ vào project

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
      ...
     'giangvien',
     'sinhvien',

]
```

Chạy lệnh

```
python manage.py makemigrations sinhvien

python manage.py migrate
```

```
$ python manage.py makemigrations sinhvien
System check identified some issues:

WARNINGS:
sinhvien.SinhVien.gv_id: (fields.W122) 'max_length' is ignored when used with IntegerField.
        HINT: Remove 'max_length' from field
Migrations for 'sinhvien':
  sinhvien\migrations\0001_initial.py
    - Create model SinhVien
```

```
$ python manage.py migrate
System check identified some issues:

WARNINGS:
sinhvien.SinhVien.gv_id: (fields.W122) 'max_length' is ignored when used with IntegerField.
        HINT: Remove 'max_length' from field
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, giangvien, sessions, sinhvien
Running migrations:
  Applying giangvien.0001_initial... OK
  Applying sinhvien.0001_initial... OK
```

Bước 4: Tạo service user

```
$ python manage.py startapp user_service
```

6. Tạo model user

```
    from django.db import models


# Create your models here.
class User(models.Model):
    id = models.AutoField(primary_key=True)
    username = models.CharField(max_length=20)
    password = models.CharField(max_length=20, blank = False)
    confirm_password = models.CharField(max_length=20, blank = False)
    role = models.IntegerField(max_length=10)
    # role = 1 cho giangvien
    # role = 2 cho sinhvien
    def __str__(self):
        return self.username
```

7. Định nghĩa API cho microservice:

- Trong thư mục **user_service**, tạo một file **views.py**.
- Trong file **views.py**, định nghĩa các API cho quản lý giảng viên.

```
from rest_framework.generics import CreateAPIView, ListCreateAPIView,
RetrieveAPIView, UpdateAPIView, DestroyAPIView

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .serializers import UserSerializer, LoginSerializer
```

```python
from .models import User

class UserCreateAPIView(CreateAPIView):
    serializer_class = UserSerializer

    def post(self, request, *args, **kwargs):
        data = {
            'username': request.data.get('username'),
            'password': request.data.get('password'),
            'confirm_password': request.data.get('confirm_password'),
            'role': request.data.get('role')
        }
        if data['confirm_password']!=data['password'] :
            return Response("error confirm_password",
status=status.HTTP_400_BAD_REQUEST)
        serializer = UserSerializer(data=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 UserLoginAPIView(CreateAPIView):
    serializer_class = LoginSerializer

    def post(self, request, *args, **kwargs):
        data = {
            'username' : request.data.get('username'),
            'password' : request.data.get('password')

        }
        user = User.objects.filter(username=data['username'], password =
data['password']).first()
        if user is not None:
            return Response('ok', status=status.HTTP_200_OK)
        else:
            return Response({"error": "Invalid credentials"},
status=status.HTTP_400_BAD_REQUEST)


class UserDetail(RetrieveAPIView, UpdateAPIView, DestroyAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

8. Tiếp theo, tạo một file **serializers.py** trong thư mục **user** và định nghĩa trình phân giải JSON cho mô hình **User**:

```
  from rest_framework import serializers

from .models import User
from rest_framework import serializers
from .models import User

# from django.contrib.auth import authenticate

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = '__all__'

class LoginSerializer(serializers.Serializer):
    username = serializers.CharField()
    password = serializers.CharField()
```

9. Cấu hình URL cho microservice:
   - Trong thư mục gốc của dự án Django, mở file **urls.py**.
   - Import **DefaultRouter** và **include** từ Django Rest Framework và import **UserViewSet** từ **user.views**.
   - Thêm các đường dẫn URL cho microservice giảng viên vào **urlpatterns**.

```
from django.urls import path

from .views import(
    UserCreateAPIView,
    UserLoginAPIView,
    UserDetail
)
urlpatterns = [
    path('register',UserCreateAPIView.as_view(), name='register'),
    path('login', UserLoginAPIView.as_view(), name='login'),
    path('<int:pk>', UserDetail.as_view(), name='user-list'),
]
```

10. Migrate service user

    Khai báo ‘user_service’ vào project

```
# Application definition

INSTALLED_APPS = [
```

```
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
      ...
     'giangvien',
     'sinhvien',
      'user_service',
]
```

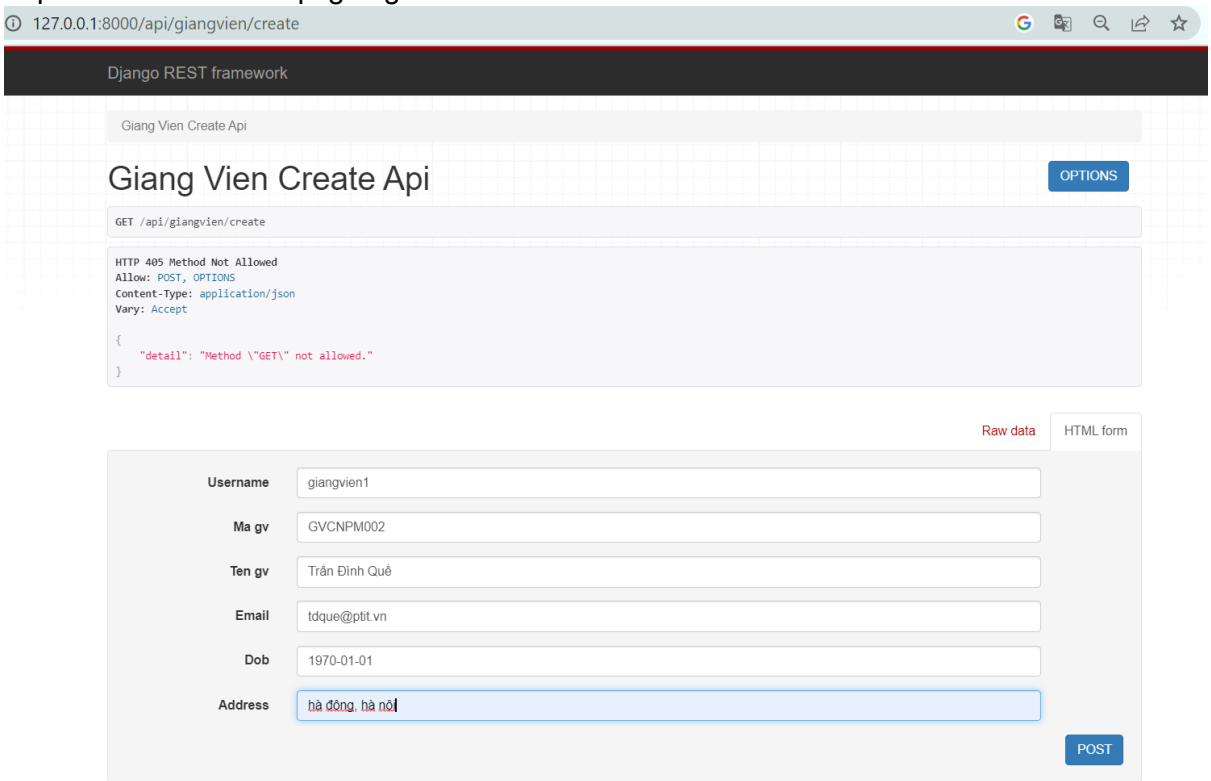
Chạy lệnh

```
python manage.py makemigrations user_service

python manage.py migrate
```

Bước 4: chạy project

1. Create giảng viên
   http://127.0.0.1:8000/api/giangvien/create

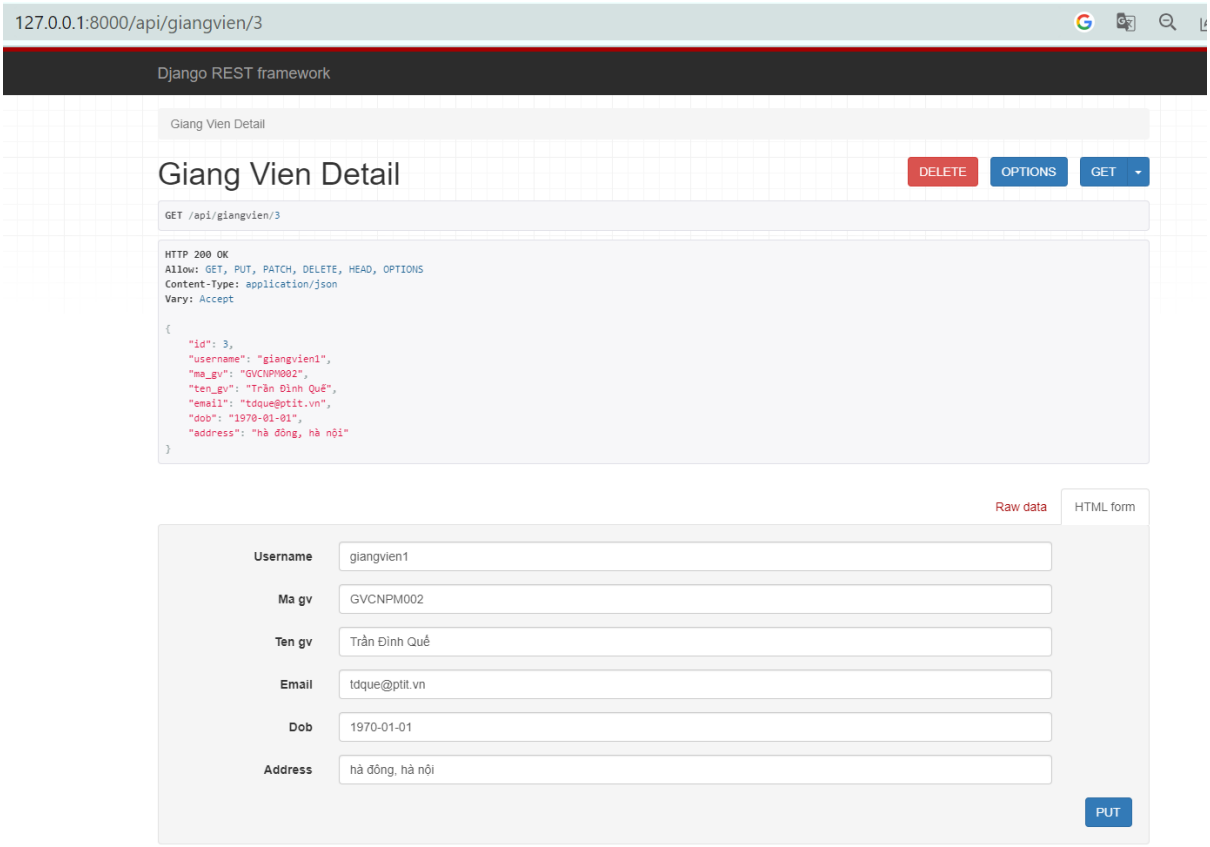


Kết quả:

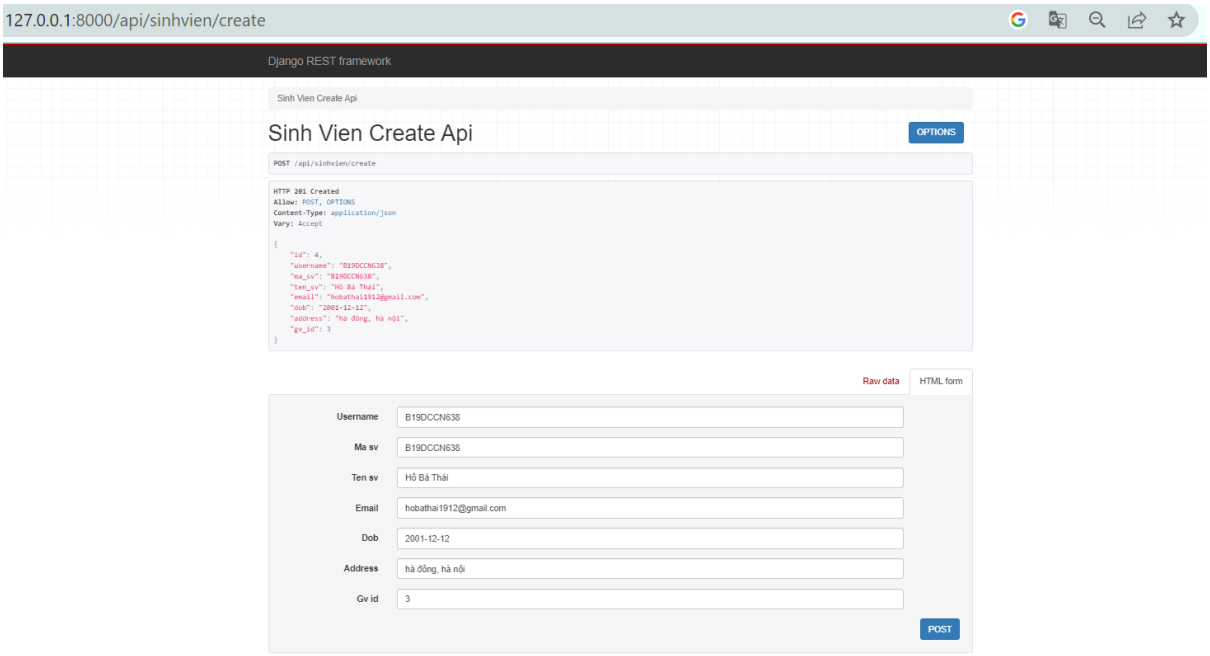
```
HTTP 201 Created
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 3,
    "username": "giangvien1",
    "ma_gv": "GVCNPM002",
    "ten_gv": "Trần Đình Quế",
    "email": "tdque@ptit.vn",
    "dob": "1970-01-01",
    "address": "hà đông, hà nội"
}
```

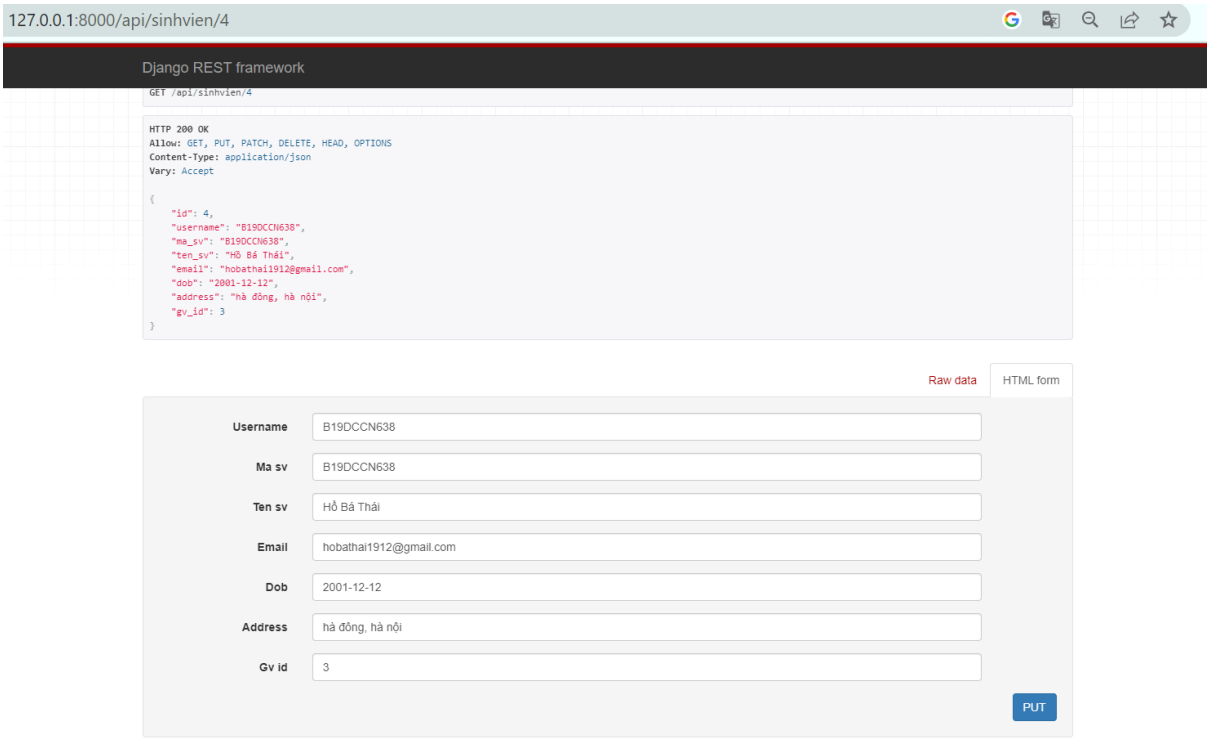
2. CRUD giảng viên
   http://127.0.0.1:8000/api/giangvien/3



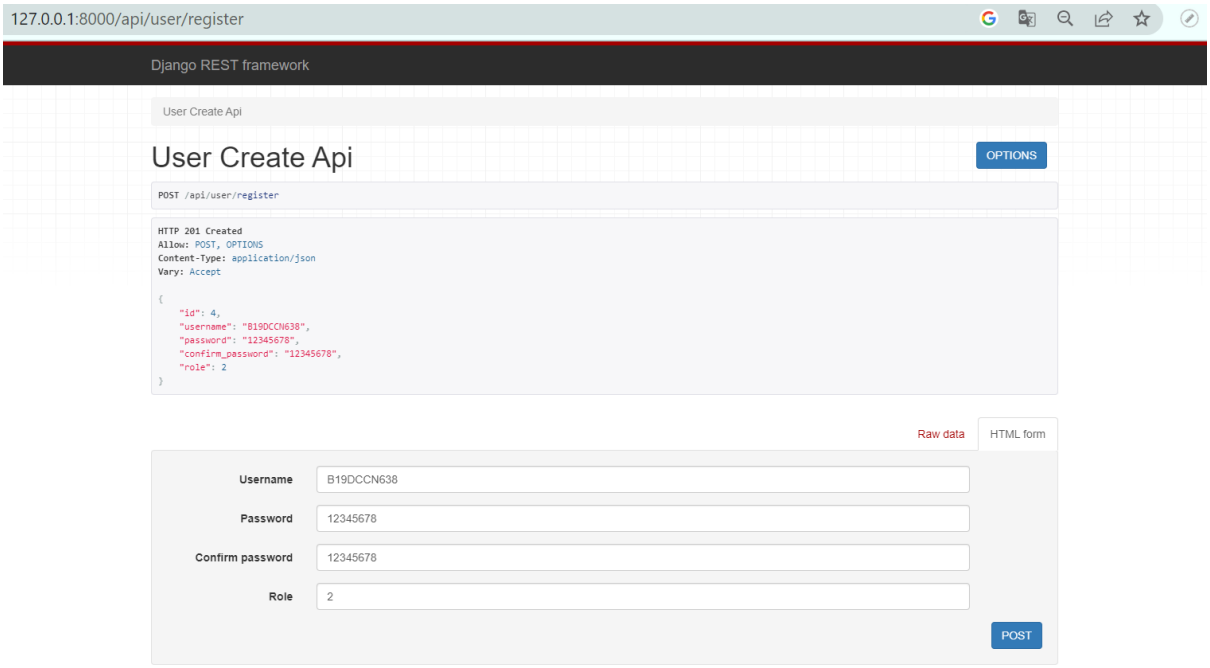
3. Create sinh viên
   http://127.0.0.1:8000/api/sinhvien/create



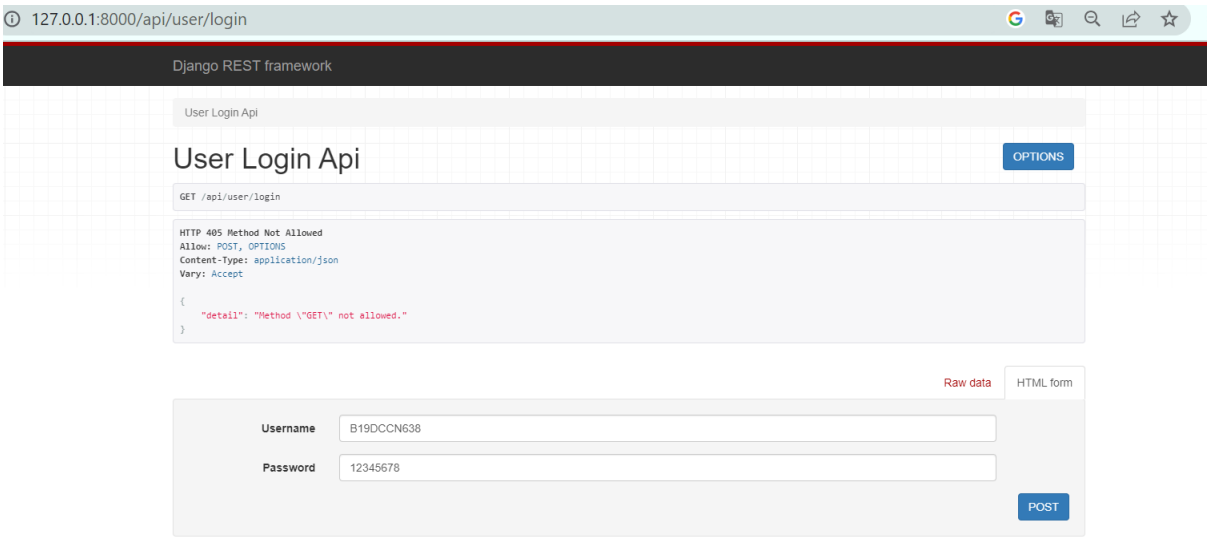
4. CRUD sinh viên
   http://127.0.0.1:8000/api/sinhvien/4



5. Register user
   http://127.0.0.1:8000/api/user/register



6. Login user
   http://127.0.0.1:8000/api/user/login



Kết quả: