https://dttvn0010.github.io/python\_tutorial/django/

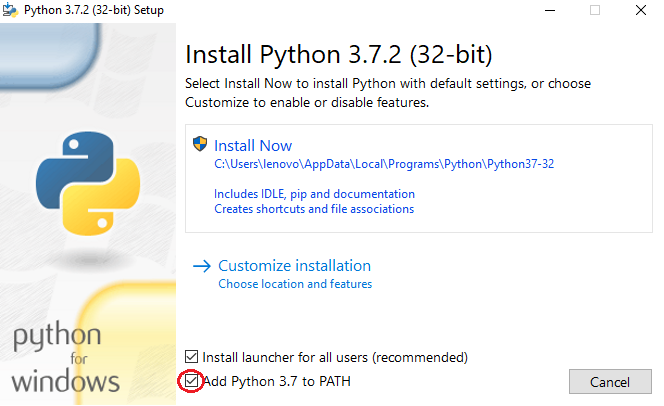
Giới thiệu

[Django](https://www.google.com/url?q=https://www.djangoproject.com/&sa=D&ust=1550420032658000) là web framework phổ biến nhất trong các web framework dựa trên python

Cài đặt

Cài đặt python

* Download và cài đặt python tại :[https://www.python.org/downloads/](https://www.google.com/url?q=https://www.python.org/downloads/&sa=D&ust=1550420032660000)
* Lưu ý khi cài đặt: Tại màn hình cửa sổ đầu tiên khi chạy chương trình cài đăt, nên chọn option “**Add Python 3.x to PATH**”, để thuận lợi cho việc chạy python từ cửa sổ cmd sau khi cài đặt

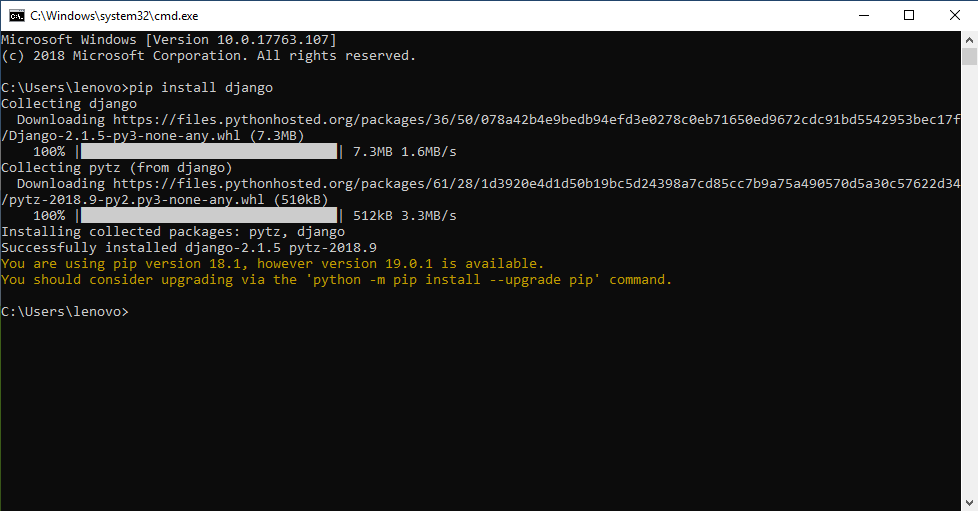


Thêm lựa chọn “Add Python to PATH” khi cài đặt Python

Cài đặt thư viện django

* Sử dụng pip để cài đặt django. Mở cửa sổ cmd, từ cửa sổ cmd gõ lệnh:

pip install django



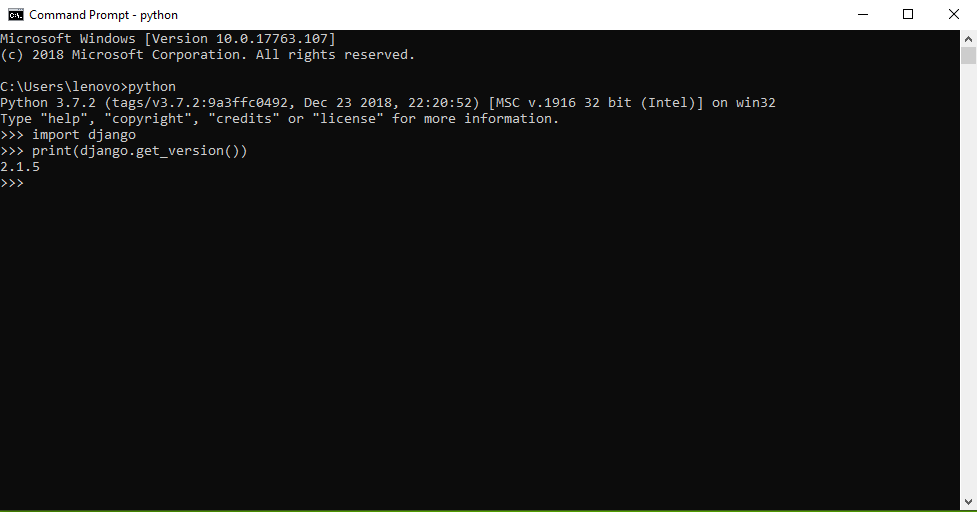
Cài đặt django bằng pip từ cửa sổ cmd

* Kiểm tra django đã được cài đặt:

Từ cửa sỗ cmd, chạy python, sau đó kiểm tra phiên bản django đã cài đặt với các lệnh:

>> import django

>> print(django.get\_version())



Kiểm tra django đã được cài đặt thành công

Phát triển ứng dụng web với django

Tạo mới project

* Mở cửa sổ cmd
* Từ cửa sổ cmd, tạo mới project bằng lệnh:

django-admin startproject mysite

Project mới sẽ được tạo ra trong thư mục mysite với các file như sau:

mysite/

    manage.py

    mysite/

        \_\_init\_\_.py

        settings.py

        urls.py

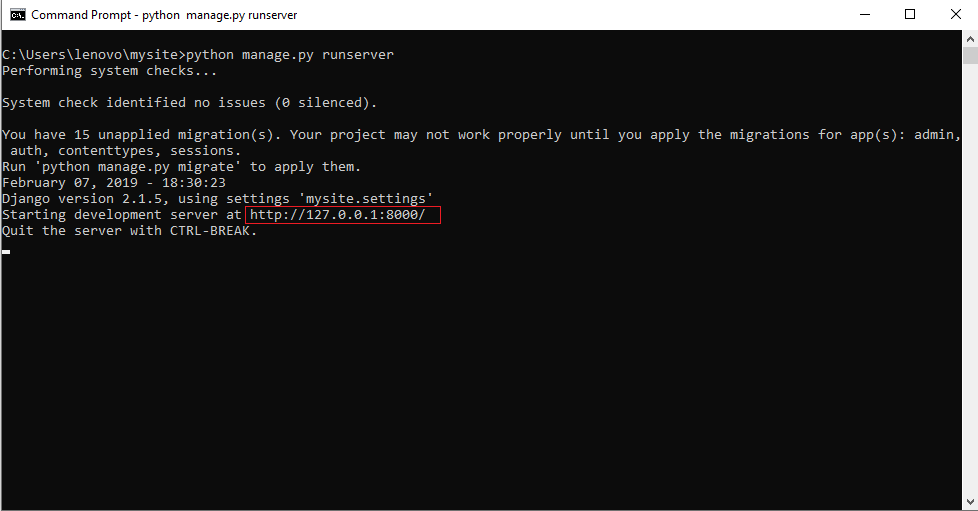
        wsgi.py

* Khởi động server:

Từ cửa sổ cmd, di chuyển vào trong thư mục project (mysite) và khởi động server với lệnh:

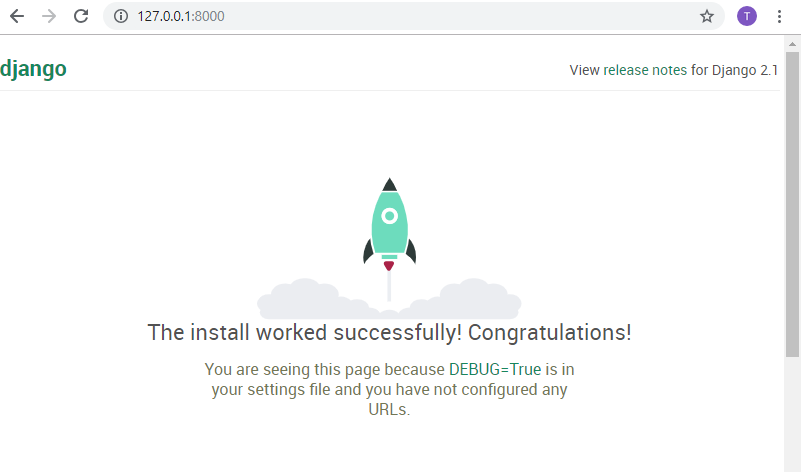
cd mysite

python manage.py runserver



Khởi động server django

Theo mặc định, server sẽ lắng nghe tại địa chỉ http://127.0.0.1:8000



Truy cập ứng dụng web django tại địa chỉ http://127.0.0.1:8000

Để thay đổi địa chỉ ip và cổng mà server lắng nghe, có thể dùng lệnh:

python manage.py runserver <ip:port>

     Ví dụ:

python manage.py runserver 8080          # nghe tại cổng 8080

python manage.py runserver 0.0.0.0:8080  # nghe tại cổng 8080, tất cả ip

python manage.py runserver 0:8080        # nghe tại cổng 8080, tất cả ip

Xây dựng ứng dụng truy xuất database

Trong phần này, chúng ta xây dựng một ứng dụng ví dụ về quản lý học sinh với các chức năng truy xuất database cơ bản : thêm mới, sửa thông tin, xóa bản ghi, xem danh sách.

Tạo mới ứng dụng

Sau khi tạo mới project (như đã trình bày ở phần trên), từ cửa sổ cmd, di chuyển vào trong thư mục project (mysite) và tạo mới một ứng dụng với lệnh:

python manage.py startapp students

Một project django sẽ có nhiều ứng dụng. Trong ví dụ này, project là mysite và ứng dụng là students. Ứng dụng này nằm trong thư mục students với cấu trúc như sau:

students/

    \_\_init\_\_.py

    admin.py

    apps.py

    migrations/

        \_\_init\_\_.py

    models.py

    tests.py

    views.py

Cấu hình thư mục static

Thư mục static chứa các file css, js, jpg, … Để có thể truy nhập các file dạng này, trước hết trong thư mục gốc của project (mysite), cần tạo một thư mục có tên static, sau đó thêm dòng sau vào cuối file mysite/settings.py:

STATICFILES\_DIRS = [

    os.path.join(BASE\_DIR, "static"),

]

Trong ứng dụng này, chúng ta sử dụng thư mục static để chứa ảnh profile của các học sinh. Để đơn giản, ảnh profile của mỗi học sinh sẽ được đặt tên theo cấu trúc:

static/images/{{student\_id}}.jpg

Như vậy cấu trúc của project sau khi tạo thư mục static (và thư mục con images) sẽ có dạng như sau:

mysite/

    manage.py

    mysite/

    static/

        images/

    students/

Thiết lập database:

Có thể dùng django với nhiều loại database khác nhau. Thông tin cấu hình database nằm trong mục DATABASES của file settings.py của project. Cấu hình mặc định sử dụng database engine là sqlite3:

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

    }

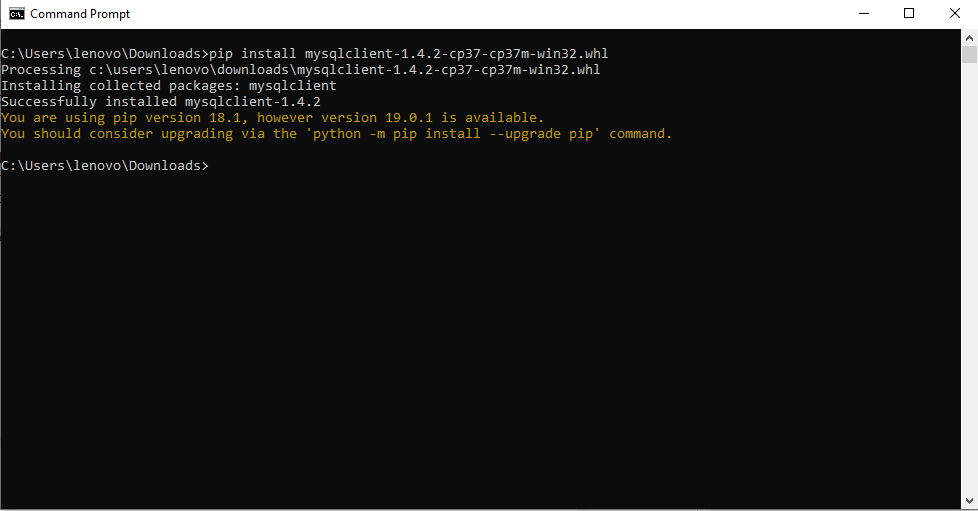
}

Ở ví dụ này, chúng ta sẽ sử dụng database engine là MySQL. Trước hết cần cài đặt thư viện mysql-client cho python. Cách thực hiện như sau:

* Download thư viện tại : https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient  
    
  Lựa chọn theo đúng phiên bản Python và kiến trúc cpu (32 bit/64bit) của phiên bản Python đã cài đặt.
* Sử dụng pip để cài đặt thư viện:

Mở cửa sổ cmd, di chuyển vào trong thư mục chứa file thư viện đã download, gõ lệnh sau để cài đặt thư viện:

pip install <tên file thư viện đã download>



Cài đặt thư viện mysqlclient

Sau khi cài xong thư viện Mysql client, chúng ta sửa cấu hình database trong file mysite/settings.py như sau:

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'testdb',

        'USER': 'admin',

        'PASSWORD': '1234',

        'HOST': 'localhost',

        'PORT': '',

    }

}

Tạo model

Model được khai báo trong file students/model.py:

|  |
| --- |
| **from** django.db **import** models  **class** **Student**(models.Model):     studentNo = models.CharField(db\_column='student\_no',  max\_length=20, unique=**True**)                                    studentName = models.CharField(db\_column='student\_name', max\_length=50)                                        address = models.CharField(db\_column='address', max\_length=100) |

Khai báo các models trong file students/models.py

Trong ví dụ này, ứng dụng của chúng ta chỉ có một bảng duy nhất (Student) với các trường sau:

* student\_no : VARCHAR(20), UNIQUE=TRUE
* student\_name : VARCHAR(50)
* address : VARCHAR(100)

Sau khi khai báo xong model, để cập nhật model vào database, cần thực hiện các bước sau:

* Chỉnh sửa file mysite/settings.py để thêm khai báo cấu hình ứng dụng như sau:

INSTALLED\_APPS = [

    'students.apps.StudentsConfig',  # new

    'django.contrib.admin',

    ....

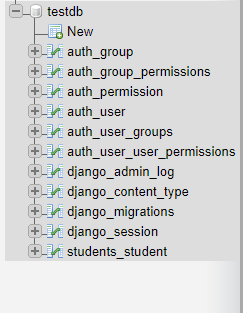
Dòng được thêm mới chứa giá trị students.apps.StudentsConfig, tương đương với class StudentsConfig nằm trong file students/apps.py

* Chạy lệnh sau để tạo model trong database:

python manage.py makemigrations students

python manage.py migrate

Sau bước trên, các bảng dữ liệu sẽ được tạo ra trong database, nếu sử dụng các tool sql client, chúng ta sẽ thấy django sinh ra các bảng như sau:



Các bảng được django tạo ra khi thực hiện lệnh migrate

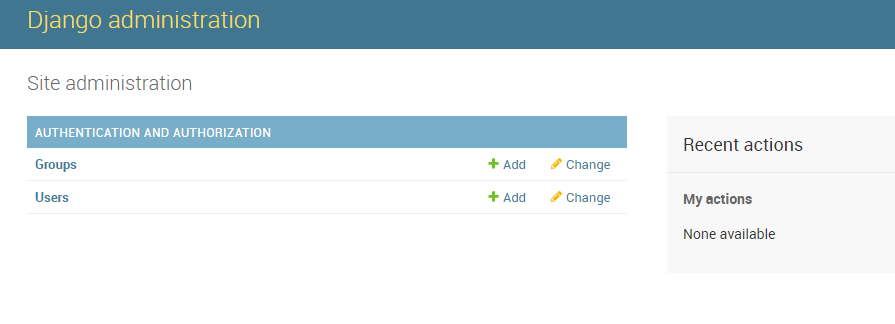
Thêm model vào màn hình admin

Theo mặc định, django cung cấp giao diện admin tại địa chỉ http://127.0.0.1:8000/admin/. Giao diện này cho phép thêm mới/ chỉnh sửa/ xóa các đối tượng dữ liệu do django quản lý như User, Group. Ngoài ra, giao diện này còn cho phép thêm các đối tượng dữ liệu mà các ứng dụng tạo ra vào danh sách quản lý để thực hiện các chức năng thêm mới/ chỉnh sửa/ xóa một cách nhanh chóng.

Trước hết, để vào màn hình admin cần tài khoản cho user admin. Tài khoản này được tạo ra bằng lệnh sau:

python manage.py createsuperuser

Sau khi tài khoản admin được tạo ra, chúng ta có thể dùng để đăng nhập vào màn hình admin của django (http://127.0.0.1:8000/admin ), màn hình admin có dạng như sau:



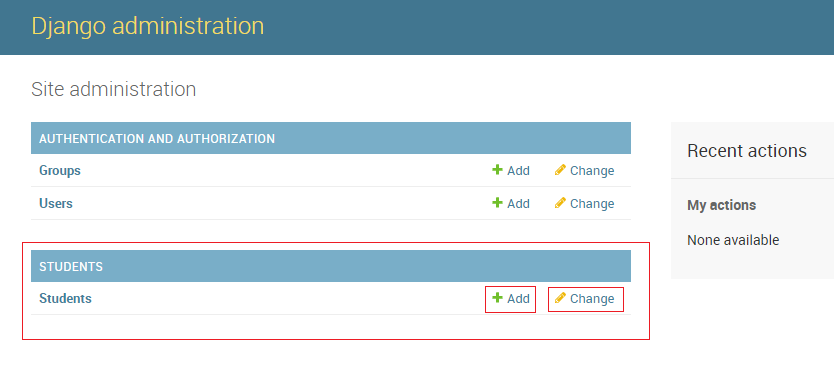
Giao diện admin của django

Trong màn hình admin chỉ có 2 đối tượng có sẵn của hệ thống (Groups, Users). Để thêm đối tượng Student của ứng dụng vào màn hình admin, chúng ta mở file students/admin.py và sửa nội dung thành như sau:

|  |
| --- |
| **from** django.contrib **import** admin **from** .models **import** Student  # new  admin.site.register(Student)  # new |

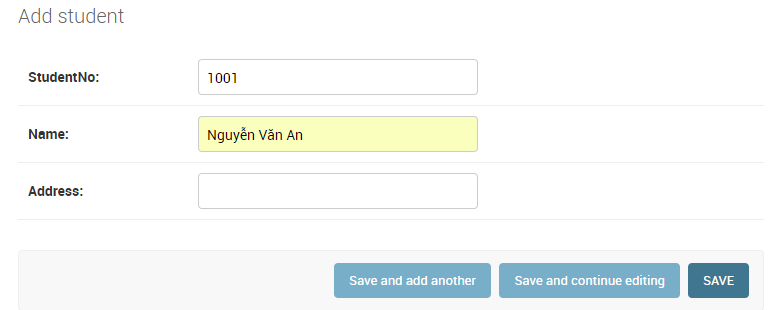
Thay đổi nội dung file students/admin.py để thêm đối tượng Student vào màn hình admin

Sau khi khởi động lại server và đăng nhập lại, đối tượng Student sẽ xuất hiện trong màn hình admin :



Màn hình admin sau khi thêm đối tượng Student vào danh sách quản lý

Từ màn hình này, chúng ta có thể thực hiện tạo mới và sửa thông tin các bản ghi trong bảng Student :



Sử dụng màn hình admin để thêm mới bản ghi cho đối tượng

Xây dựng các màn hình của ứng dụng

Màn hình admin cho phép thêm mới/chỉnh sửa các bản ghi một cách nhanh chóng, tuy nhiên các chức năng của ứng dụng vẫn cần có các màn hình riêng. Phần này chúng ta thực hiện xây dựng các màn hình ứng dụng, cụ thể gồm 2 màn hình sau:

* Màn hình danh sách học sinh : Hiển thị toàn bộ các học sinh trong hệ thống. Cuối danh sách học sinh có nút thêm mới để chuyển sang màn hình thêm mới học sinh. Mỗi học sinh trong danh sách có link để xóa và chỉnh sửa thông tin

* Màn hình thêm mới/chỉnh sửa thông tin học sinh : Cho phép thêm mới học sinh vào hệ thống hoặc cập nhật thông tin các học sinh đã nằm trong hệ thống.

Màn hình danh sách học sinh

Trước hết, chúng ta xây dựng phần truy xuất dữ liệu (controller). Phần này nằm trong file students/views.py:

|  |
| --- |
| **from** django.http **import** HttpResponse **from** django.shortcuts **import** render  **from** .models **import** Student  **def** **listStudent**(request):    student\_list = Student.objects.all()        context = {        'student\_list': student\_list,    }    **return** render(request, 'students/list\_student.html', context) |

File students/views.py sau khi thêm phần truy xuất dữ liệu

Tiếp theo chúng ta tạo template hiển thị : tạo mới folder templates/students trong thư mục students, sau đó tạo file list\_student.html đặt trong thư mục vừa tạo với nội dung sau :

|  |
| --- |
| {% **if** student\_list %}    <**table** border=1>        <**tr**>            <**th**>Student Number</**th**>            <**th**>Student Name</**th**>            <**th**>Address</**th**>        </**tr**>    {% **for** student **in** student\_list %}            <**tr**>            <**td**> {{ student.studentNo }} </**td**>            <**td**> {{ student.studentName }} </**td**>            <**td**> {{ student.address }} </**td**>        </**tr**>            {% **endfor** %}    </**table**> {% **else** %}    <**p**>No student available.</**p**> {% **endif** %} |

File template students/templates/students/list\_student.html

Cuối cùng, phải khai báo đường dẫn (url) cho trang vừa được tạo ra. Các khai báo url được đặt trong file mysite/urls.py, tuy nhiên để dễ quản lý, chúng ta tạo riêng một file mới students/urls.py và include nội dung của file mới tạo này vào file mysite/urls.py

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path, include  # new  urlpatterns = [    path('students/', include('students.urls')),  # new    path('admin/', admin.site.urls), ] |

File mysite/urls.py được sửa để include file url của ứng dụng (students.urls)

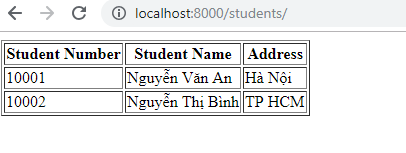
Dòng được thêm mới trong file mysite/urls.py có nghĩa : thực hiện mapping các urls có pattern students/\* theo nội dung của file ở đường dẫn students.urls (tức students/urls.py). Nội dung file này như sau:

|  |
| --- |
| **from** django.urls **import** path  **from** . **import** views  urlpatterns = [    path('', views.listStudent), ] |

File students/urls.py được tạo ra để thực hiện các mapping url cho ứng dụng students

File students/urls.py ở trên thực hiện gắn url với đường dẫn tương đối path='' (đường dẫn tuyệt đối sẽ là 'students/' + ' ' = 'students/') vào hàm xử lý views.listStudent (hàm listStudent trong file views.py)

Sau khi hoàn thành các phần trên, khởi động server và truy cập địa chỉ http://localhost:8000/students, chúng ta sẽ thấy nội dung như sau:



Màn hình danh sách học sinh (draft)

Màn hình thêm mới/chỉnh sửa thông tin học sinh

Với màn hình thêm mới chỉnh sửa, trước hết chúng ta tạo một form html cho phép nhập thông tin học sinh. Form này nằm trong file template tại đường dẫn students/templates/students/add\_edit\_student.html :

|  |
| --- |
| <**h3**>Student information</**h3**>  <**div** id='error\_list'> {% if errorList %}  <**ul**>    {% for error in errorList %}        <**li** style='color:red'>{{ error }}</**li**>    {% endfor %}      </**ul**> {% endif %} </**div**>  <**form** method="post" enctype='multipart/form-data'>  {% csrf\_token %} <**input** type='hidden' name='studentId' value='{{ form.studentId }}'> <**table**>  <**tr**>    <**td**>Student Number</**td**>    <**td**>      <**input** name='studentNo' value='{{ form.studentNo }}'>    </**td**>  </**tr**>  <**tr**>    <**td**>Student Name</**td**>    <**td**>      <**input** name='studentName' value='{{ form.studentName }}'>    </**td**>  </**tr**>  <**tr**>    <**td**>Address</**td**>    <**td**>      <**input** name='address' value='{{ form.address }}'>    </**td**>  <**tr**>    <**td**>Profile picture</**td**>    <**td**><**input** type='file' name='profilePicture'></**td**>  </**tr**> </**table**> <**input** type='submit' value='Save'> <**input** type='button' value='Back'  onclick='location.href="{% url "list\_student" %}"'> </**form**> |

File template students/templates/students/ add\_edit\_student.html

Khi form html được submit lên server, chúng ta cần lưu trữ và validate các trường dữ liệu trước khi ghi vào database. Để thực hiện việc này, chúng ta tạo mới file forms.py trong thư mục students với nội dung như sau:

|  |
| --- |
| **import** os **from** django.core.files.storage **import** FileSystemStorage **from** .models **import** Student  IMG\_DIR = 'static/images'  **class** **StudentForm**():     **def** **\_\_init\_\_**(self, request=None, dbModel=None):                   **if** request:            self.studentId = request.POST['studentId']            self.studentNo = request.POST['studentNo']            self.studentName = request.POST['studentName']            self.address = request.POST['address']                        self.profilePictureFile = request.FILES.get('profilePicture')                   **elif** dbModel:            self.studentId = dbModel.id            self.studentNo = dbModel.studentNo            self.studentName = dbModel.studentName            self.address = dbModel.address                          **def** **validate**(self):        errorList = []               **if** **not** self.studentNo:            errorList.append('Student number is required')                   **if** **not** self.studentName:            errorList.append('Student name is required')                   **if** **not** self.address:            errorList.append('Address is required')                   **if** **not** self.studentId **and** **not** self.profilePictureFile:             errorList.append('Profile picture is required')                   **return** errorList               **def** **save**(self):        student = Student()        student.id = int(self.studentId) **if** self.studentId **else** **None**        student.studentNo = self.studentNo        student.studentName = self.studentName        student.address = self.address        student.save()                **if** student **and** self.profilePictureFile:            fs = FileSystemStorage()                        filepath = os.path.join(IMG\_DIR, str(student.id) + '.jpg')                        **if** os.path.exists(filepath):                os.remove(filepath)                            fs.save(filepath, self.profilePictureFile) |

File students/forms.py

Phần controller trong file students/views.py được update như sau:

|  |
| --- |
| **import** traceback **from** django.http **import** HttpResponse **from** django.shortcuts **import** render, redirect, get\_object\_or\_404 **import** time  **from** .models **import** Student **from** .forms **import** StudentForm  **def** **listStudent**(request):    studentList = Student.objects.all()        context = {'studentList': studentList, 'time' : int(time.time())}        **return** render(request, 'students/list\_student.html', context)  **def** **addEditStudent**(request, studentId):    errorList = []    form = StudentForm()       **if** request.method == 'POST':        **try**:            form = StudentForm(request)            errorList = form.validate()                       **if** **not** errorList:                form.save()                **return** redirect('list\_student')                       **except** Exception **as** e:            traceback.print\_exc()            errorList.append(str(e))               **elif** studentId > 0:        student = get\_object\_or\_404(Student, pk=studentId)                    form = StudentForm(dbModel=student)               context = {'form' : form, 'errorList' : errorList}       **return** render(request, 'students/add\_edit\_student.html', context)    **def** **deleteStudent**(request, studentId):    student = get\_object\_or\_404(Student, pk=studentId)    student.delete()    **return** redirect('list\_student') |

File students/views.py sau khi bổ sung các hàm cho chức năng thêm mới/chỉnh sửa thông tin

Cuối cùng, cập nhật file students/urls.py để thêm các mapping url cho màn hình thêm mới/chỉnh sửa thông tin học sinh :

|  |
| --- |
| **from** django.urls **import** path  **from** . **import** views  urlpatterns = [    path('', views.listStudent, name='list\_student'),    path('<int:studentId>/add\_edit', views.addEditStudent, name='add\_edit\_student'),        path('<int:studentId>/delete', views.deleteStudent, name='delete\_student'), ] |

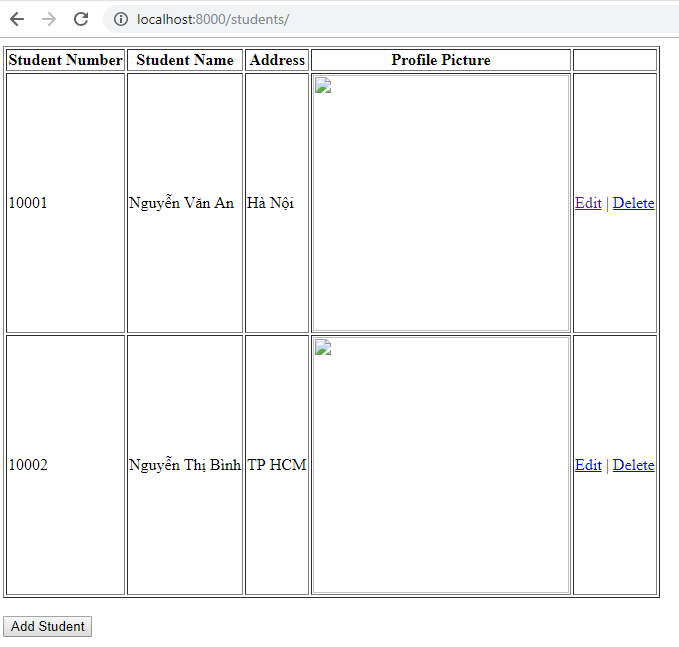
File students/urls.py sau khi thêm các mapping cho phần thêm mới/chỉnh sửa thông tin

Sau khi xây dựng xong  màn hình thêm mới/chỉnh sửa thông tin, chúng ta link màn hình danh sách sang màn hình thêm mới/chỉnh sửa bằng cách bổ sung các button và link như sau:

|  |
| --- |
| {% load static %}  {% if studentList %}  <**table** border=1>    <**tr**>      <**th**>Student Number</**th**>      <**th**>Student Name</**th**>      <**th**>Address</**th**>           <**th**>Profile Picture</**th**>      <**th**></**th**>    </**tr**>  {% for student in studentList %}      <**tr**>      <**td**> {{ student.studentNo }} </**td**>      <**td**> {{ student.studentName }} </**td**>      <**td**> {{ student.address }} </**td**>           <**td**> <**image** width="256" src='/static/images/{{student.id}}.jpg?t={{time}}'></**td**>      <**td**>        <**a** href='{% url "add\_edit\_student" student.id %}'>Edit</**a**> |        <**a** href='javascript:confirmDelete("{% url "delete\_student" student.id %}")'>Delete</**a**>      </**td**>    </**tr**>      {% endfor %}  </**table**>     <**script**>    **function** **confirmDelete**(url) {      **if**(confirm('Do you want to delete?')){        location.href = url;      }          }  </**script**> {% else %}  <**p**>No student available.</**p**> {% endif %}  <**br**> <**input** type='button' value='Add Student'   onclick='location.href="{% url "add\_edit\_student" 0 %}"'> |

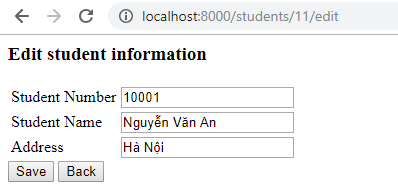
Chỉnh sửa file list\_student.html để thêm các button và link để liên kết 2 màn hình

Màn hình danh sách mới sẽ như sau:



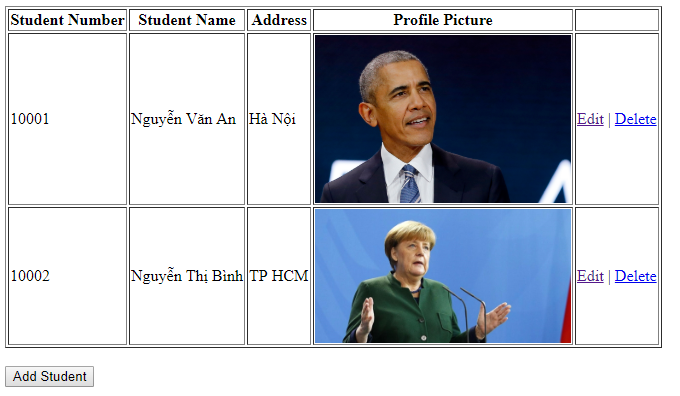
Màn hình danh sách học sinh sau khi thêm button và link sang màn hình thêm mới/chỉnh sửa

Từ màn hình danh sách, có thể chuyển sang màn hình thêm mới/chỉnh sửa thông tin học sinh:



Màn hình thêm mới/chỉnh sửa thông tin học sinh

Sau khi update ảnh profile của các học sinh, màn hình danh sách sẽ có dạng như sau:



Màn hình danh sách sau khi update ảnh profile

Tích hợp và mở rộng chức năng quản lý User của django

Thông thường, các ứng dụng web đều có chức năng quản lý thông tin User và cho phép mỗi loại User xem những nội dung nhất định của site. Django cung cấp sẵn hệ thống quản lý User và cho phép ứng dụng mở rộng mô hình User của django để quản lý các thông tin bổ sung.

Trong phần này chúng ta sẽ xây dựng chức năng authentication cho ứng dụng. Phần này tích hợp vào project quản lý học sinh đã xây dựng ở phần trước, nhưng có thể sử dụng cho tất cả các hệ thống cần chức năng authentication.

Xây dựng ứng dụng quản lý User

Tạo mới ứng dụng

* Trong thư mục project (mysite), tạo một ứng dụng mới với tên accounts :

python manage.py startapp accounts

Ứng dụng mới được tạo ra nằm trong thư mục accounts ngang hàng với thư mục students. Cấu trúc của thư mục projects có dạng sau:

mysite/

    manage.py

    accounts/

    mysite/

    static/

    students/

* Thêm ứng dụng vào file cấu hình mysite/settings.py:

INSTALLED\_APPS = [

    'students.apps.StudentsConfig',  # new

    'accounts.apps.AccountsConfig',  # new

    …..

Cấu hình url

* Tạo mới file accounts/urls.py:

|  |
| --- |
| **from** django.urls **import** path  **from** . **import** views  urlpatterns = [ ] |

Tạo mới file accounts/urls.py

Nội dung urlpatterns trong file accounts/urls.py sẽ được bổ sung ở phần sau

* Update file mysite/urls.py :

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path, include  urlpatterns = [    path('students/', include('students.urls')),    path('accounts/', include('accounts.urls')),             # new    path('accounts/', include('django.contrib.auth.urls')),  # new    path('admin/', admin.site.urls), ] |

Update nội dung file mysite/urls.py để include file accounts/urls.py mới tạo ra

Trong 2 dòng mới thêm vào file mysite/urls.py, một dòng là include nội dung của file accounts/urls.py, dòng còn lại là include các đường dẫn url của chức năng quản lý account của django, trong đó 2 url quan trọng nhất là:

* /accounts/login : login url
* /accounts/logout : logout url

Tạo thư mục chứa templates

Theo mặc định, django sẽ tìm kiếm template cho các chức năng quản lý User tại thư mục templates/registration. Do đó chúng ta tạo mới thư mục templates/registration trong thư mục accounts, sau đó tạo mới các template cho các màn hình đăng ký, đăng nhập trong thư mục account/templates/registration/

mysite/

    manage.py

    accounts/

        templates

            registration

                login.html

                signup.html

Tạo database model

Django đã có sẵn bảng Users và các bảng liên quan cho chức năng quản lý User. Các trường thông tin của User được django lưu trữ gồm:

* Username
* Email
* Password

Tuy nhiên Django cho phép mở rộng mô hình User này để lưu thêm các thông tin bổ sung. Ở đây, chúng ta xây dựng ví dụ quản lý User có thêm các trường thông tin sau:

* Fullname
* Address

Để thực hiện việc này, cần thêm database model trong file accounts/models.py :

|  |
| --- |
| **from** django.db **import** models **from** django.contrib.auth.models **import** User **from** django.db.models.signals **import** post\_save **from** django.dispatch **import** receiver  **class** **Profile**(models.Model):    user = models.OneToOneField(User, on\_delete=models.CASCADE)    fullname = models.TextField(db\_column='fullname', max\_length=50)        address = models.CharField(db\_column='address', max\_length=100)     @receiver(post\_save, sender=User) **def** **update\_user\_profile**(sender, instance, created, \*\*kwargs):    **if** created:        Profile.objects.create(user=instance)                     instance.profile.save() |

Thêm database model trong file accounts/models.py để mở rộng mô hình User của django

Để tạo mới model trong database, chúng ta chạy lệnh sau:

python manage.py makemigrations accounts

python manage.py migrate

Xây dựng các màn hình chức năng

Màn hình đăng ký tài khoản

Về cơ bản, màn hình đăng ký tài khoản là một form nhập dữ liệu, do đó các bước thực hiện tương tự với ứng dụng quản lý học sinh đã trình bày ở phần trước.

* Tạo form html nhập dữ liệu trong file accounts/templates/registration/signup.html :

|  |
| --- |
| <**h2**>Sign up</**h2**>  <**div** id='error\_list'> {% if errorList %}   <**ul**>     {% for error in errorList %}         <**li** style='color:red'>{{ error }}</**li**>     {% endfor %}       </**ul**> {% endif %} </**div**>  <**form** method="post">    {% csrf\_token %}    <**table**>        <**tr**>            <**td**>Username</**td**>            <**td**><**input** name='username' value='{{ form.username }}'></**td**>        </**tr**>        <**tr**>            <**td**>Email</**td**>            <**td**><**input** name='email' value='{{ form.email }}'></**td**>        </**tr**>        <**tr**>            <**td**>Full name</**td**>            <**td**><**input** name='fullname' value='{{ form.fullname }}'></**td**>        </**tr**>        <**tr**>            <**td**>Address</**td**>            <**td**><**input** name='address' value='{{ form.address }}'></**td**>        </**tr**>        <**tr**>            <**td**>Password</**td**>            <**td**><**input** type='password' name='password1' value='{{ form.password1 }}'></**td**>        </**tr**>        <**tr**>            <**td**>Confirm Password</**td**>            <**td**><**input** type='password' name='password2' value='{{ form.password2 }}'></**td**>        </**tr**>    </**table**>    <**input** type="submit" value='Sign up'> </**form**> |

File accounts/templates/registration/signup.html

* Tạo file accounts/forms.py để lưu trữ các trường dữ liệu của request:

|  |
| --- |
| **class** **SignUpForm**():        **def** **\_\_init\_\_**(self, request=None):        **if** request:            self.username = request.POST['username']            self.email = request.POST['email']            self.fullname = request.POST['fullname']            self.address = request.POST['address']            self.password1 = request.POST['password1']            self.password2 = request.POST['password2']                        **def** **validate**(self):        errorList = []                **if** **not** self.username:            errorList.append('Username is required')                    **if** **not** self.email:            errorList.append('Email is required')                    **if** len(self.password1) < 6:            errorList.append('Password is too short')                    **if** self.password1 != self.password2:            errorList.append('Incorrect confirmed password')                    **return** errorList |

File accounts/forms.py

* Thêm phần controller trong file accounts/views.py :

|  |
| --- |
| **from** django.contrib.auth **import** login, authenticate **from** django.shortcuts **import** render, redirect **from** django.contrib.auth.models **import** User **from** .forms **import** SignUpForm  **def** **signup**(request):        **if** request.method == 'POST':        form = SignUpForm(request)                errorList = form.validate()                **if** **not** errorList:                    **try**:                user = User.objects.create\_user(                                    username=form.username,                                    email=form.email,                                    password=form.password1)                                                     user.profile.fullname = form.fullname                user.profile.address = form.address                            user.save()                                            user = authenticate(username=form.username,                                       password=form.password1)                 login(request, user)                **return** redirect('list\_student')                            **except** Exception **as** e:                                errorList.append(str(e))                    **else**:        form = SignUpForm()        errorList = []     **return** render(request, 'registration/signup.html', {'form': form, 'errorList' : errorList}) |

File accounts/view.py

Phần controller ở trên thực hiện tạo mới user sau đó sẽ thực hiện đăng nhập luôn. Khi đăng nhập thành công, ứng dụng sẽ chuyển hướng tới trang mặc định. Để cấu hình trang mặc định này, chúng ta bổ sung vào cuối file mysite/settings.py như sau:

STATIC\_URL = '/static/'

# new

LOGIN\_REDIRECT\_URL = 'list\_student'

LOGOUT\_REDIRECT\_URL = 'list\_student'

Bổ sung update vào cuối file mysite/settings.py

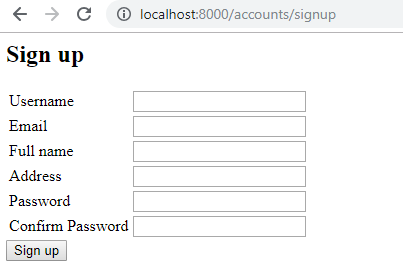
* Cấu hình url trong file accounts/urls.py :

|  |
| --- |
|  |

|  |
| --- |
| **from** django.urls **import** path  **from** . **import** views  urlpatterns = [   path('signup', views.signup, name='signup'),  # new ] |

File accounts/urls.py

Sau khi xây dựng xong màn hình đăng ký tài khoản, khởi động server và truy nhập vào địa chỉ [http://localhost:8000/accounts/signup](https://www.google.com/url?q=http://localhost:8000/accounts/signup&sa=D&ust=1550420032755000) chúng ta sẽ thấy màn hình như sau:



Màn hình đăng ký tài khoản

Sau khi điền đầy đủ thông tin vào form đăng ký và submit, nếu thành công, trình duyệt sẽ chuyển sang màn hình chính của ứng dụng quản lý học sinh đã xây dựng trong phần trước.

Màn hình đăng nhập

Với màn hình đăng nhập, chỉ cần tạo file login.html nằm trong thư mục

accounts/template/registration với nội dung sau:

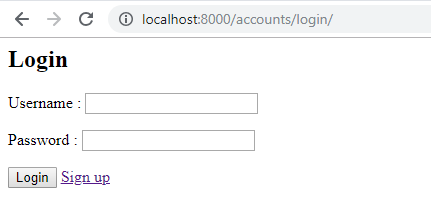
|  |
| --- |
|  |

|  |
| --- |
| <**h2**>Login</**h2**> <**form** method="post">  {% csrf\_token %}    <**p**>Username : <**input** name='username'></**p**>  <**p**>Password : <**input** type='password' name='password'></**p**>  <**input** type="submit" value='Login'>  <**a** href="{% url 'signup' %}">Sign up</**a**> </**form**> |

File accounts/template/registration/login.html

Với màn hình đăng nhập, phần controller sẽ do django xử lý.

Khởi động lại server và truy nhập vào địa chỉ [http://localhost:8000/accounts/login](https://www.google.com/url?q=http://localhost:8000/accounts/login&sa=D&ust=1550420032759000), chúng ta sẽ thấy màn hình đăng nhập như sau :



Màn hình đăng nhập

Sử dụng tài khoản đã tạo ra ở phần đăng ký để đăng nhập. Nếu chưa có tài khoản, có thể dùng link Sign up ở dưới cùng để tạo tài khoản.

Tích hợp chức năng đăng nhập cho ứng dụng quản lý học sinh

Để minh họa việc tích hợp chức năng đăng nhập vào một ứng dụng, chúng ta sửa lại ứng dụng quản lý học sinh để khi user chưa đăng nhập thì chỉ được phép xem danh sách học sinh, còn các chức năng thêm mới/chỉnh sửa/xóa chỉ được phép thực hiện khi user đã đăng nhập.

Với phần template:

Sử dụng điều kiện {{ user.is\_authenticated }} để biết user đã đăng nhập chưa, từ đó ẩn đi các nội dung mà chỉ user đăng nhập mới được thấy. Cụ thể, file students/templates/students/list\_student.html được sửa như sau:

|  |
| --- |
| {% load static %}  {% if studentList %}  <**table** border=1>    <**tr**>      <**th**>Student Number</**th**>      <**th**>Student Name</**th**>      <**th**>Address</**th**>      <**th**>Profile Picture</**th**>      {% if user.is\_authenticated %}        <**th**></**th**>      {% endif %}        </**tr**>  {% for student in studentList %}      <**tr**>      <**td**> {{ student.studentNo }} </**td**>      <**td**> {{ student.studentName }} </**td**>      <**td**> {{ student.address }} </**td**>      <**td**> <**image** width="256" src='/static/images/{{student.id}}.jpg?t={{time}}'></**td**>      {% if user.is\_authenticated %}        <**td**>          <**a** href='{% url "add\_edit\_student" student.id %}'>Edit</**a**> |          <**a** href='javascript:confirmDelete("{% url "delete\_student" student.id %}")'>Delete</**a**>        </**td**>      {% endif %}    </**tr**>      {% endfor %}  </**table**>     <**script**>    **function** **confirmDelete**(url) {      **if**(confirm('Do you want to delete?')){        location.href = url;      }          }  </**script**> {% else %}  <**p**>No student available.</**p**> {% endif %}  {% if user.is\_authenticated %}  <**br**>  <**input** type='button' value='Add Student'     onclick='location.href="{% url "add\_edit\_student" 0 %}"'>   <**br**><**br**>  You are logged in as {{ user.profile.fullname }} ({{ user.username }})  <**a** href="{% url 'logout' %}">logout</**a**>  {% else %}  <**br**>  <**a** href="{% url 'login' %}">login</**a**> / <**a** href="{% url 'signup' %}">signup</**a**> {% endif %} |

File list\_student.html sau khi tích hợp chức năng đăng nhập

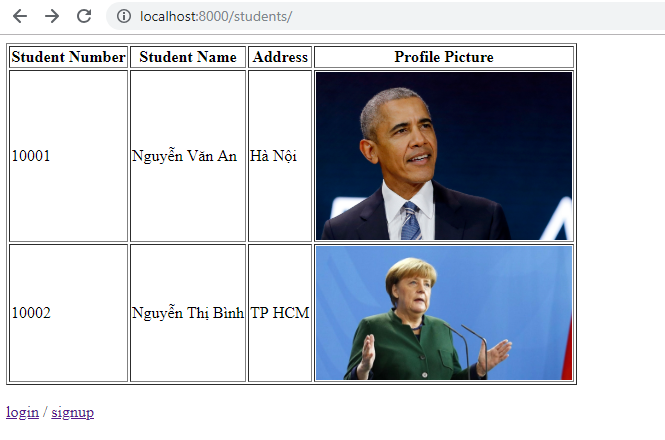
Với phần controller:

Sử dụng kí hiệu @login\_required  với các chức năng mà chỉ người dùng đăng nhập mới được phép thao tác. Cụ thể, file students/views.py được sửa như sau:

|  |
| --- |
| **import** traceback **from** django.http **import** HttpResponse **from** django.shortcuts **import** render, redirect, get\_object\_or\_404 **from** django.contrib.auth.decorators **import** login\_required  **from** .models **import** Student **from** .forms **import** StudentForm  **def** **listStudent**(request):    studentList = Student.objects.all()        context = {'studentList': studentList}        **return** render(request, 'students/list\_student.html', context)  @login\_required     **def** **addEditStudent**(request, studentId):    errorList = []    form = StudentForm()        **if** request.method == 'POST':        **try**:            form = StudentForm(request)            errorList = form.validate()                        **if** **not** errorList:                form.save()                **return** redirect('list\_student')                        **except** Exception **as** e:            traceback.print\_exc()            errorList.append(str(e))                **elif** studentId > 0:        student = get\_object\_or\_404(Student, pk=studentId)                    form = StudentForm(dbModel=student)                context = {'form' : form, 'errorList' : errorList}        **return** render(request, 'students/add\_edit\_student.html', context)     @login\_required     **def** **deleteStudent**(request, studentId):    student = get\_object\_or\_404(Student, pk=studentId)    student.delete()    **return** redirect('list\_student') |

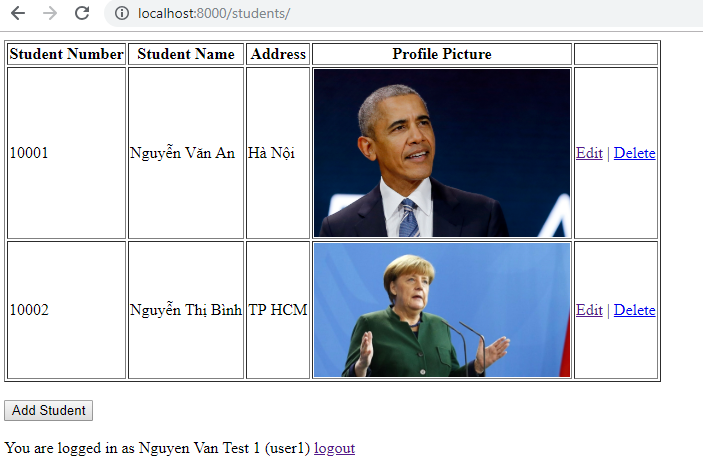
File students/views.py sau khi tích hợp chức năng đăng nhập

Sau khi tích hợp xong, khởi động lại server, truy nhập vào ứng dụng quản lý học sinh ([http://localhost:8000/students](https://www.google.com/url?q=http://localhost:8000/students&sa=D&ust=1550420032772000)), khi chưa đăng nhập, màn hình ứng dụng sẽ có dạng sau:



Màn hình ứng dụng khi chưa đăng nhập.Chức năng thêm mới/chỉnh sửa/xóa đã bị ẩn đi

Click vào link login ở dưới cùng, sau khi đăng nhập, màn hình ứng dụng sẽ như sau:



Màn hình ứng dụng sau khi đăng nhập

Xây dựng webservice với django-rest-framework (DRF)

Hiện nay, xu hướng sử dụng các ứng dụng Single Page Application (SPA) càng nhiều, do đó phần này chúng ta tìm hiểu cách sử dụng django để làm backend cho các framework SPA (Angular/React/Vue)

Để tạo webservice với django, chúng ta sử dụng django-rest-framework:

pip install djangorestframework

Cách tạo một web service đơn giản

* Tạo mới project

django-admin startproject mysite

* Tạo mới một ứng dụng trong project

cd mysite

python manage.py startapp app

Thêm app vừa tạo vào danh sách ứng dụng trong file mysite/settings.py, đồng thời thêm luôn rest\_framework vào danh sách ứng dụng này :

INSTALLED\_APPS = [

    'app',                  # new

    'rest\_framework',       # new

    'django.contrib.admin',

    ….

Thêm app vừa tạo và rest\_framework  vào danh sách ứng dụng trong mysite/settings.py

* Tạo một endpoint trong file app/views.py:

|  |
| --- |
| **from** rest\_framework.decorators **import** api\_view **from** rest\_framework.response **import** Response  @api\_view(['GET', 'POST']) **def** **hello**(request):        **return** Response({"message" : "Hello world!"}) |

Xây dựng endpoint trong file app/views.py

* Cấu hình url:

Tạo mới file app/urls.py với nội dung sau:

|  |
| --- |
| **from** django.urls **import** path **from** .views **import** hello  urlpatterns = [    path('hello', hello), ] |

File app/urls.py

Include file vừa tạo vào file mysite/urls.py :

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path, include  urlpatterns = [    path('app/', include('app.urls')),    path('admin/', admin.site.urls),     ] |

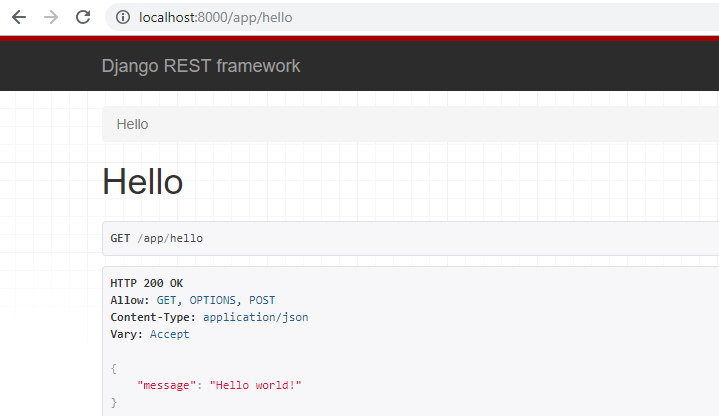
File mysite/urls.py

* Khởi tạo database:

python manage.py makemigrations app

python manage.py migrate

* Khởi động ứng dụng và truy cập api tại địa chỉ [http://localhost:8000/app/hello](https://www.google.com/url?q=http://localhost:8000/app/hello&sa=D&ust=1550420032784000), chúng ta sẽ thấy nội dung như sau:



Test thử api từ trình duyệt

* Thêm phần xử lý nhận dữ liệu từ request :
* Với GET method : dữ liệu được lấy từ request.query\_params
* Với POST method : dữ liệu được lấy từ request.data

Ví dụ:

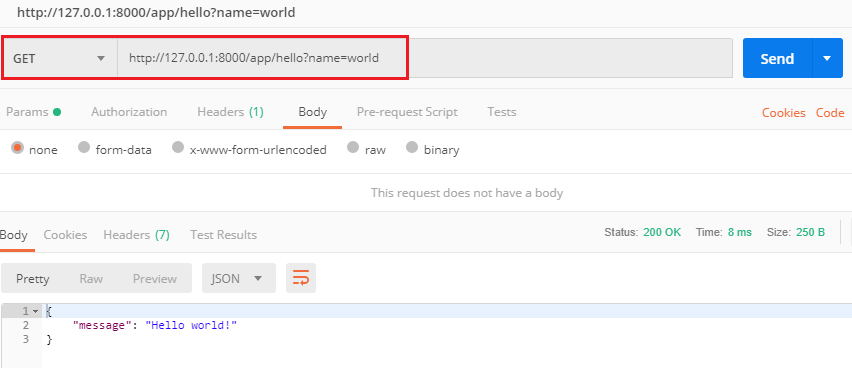
|  |
| --- |
| **from** rest\_framework.decorators **import** api\_view **from** rest\_framework.response **import** Response  @api\_view(['GET', 'POST']) **def** **hello**(request):        **if** request.method == 'POST':        name = request.data.get('name', 'world')        **return** Response({"message" : f"Hello {name}!"})            **else**:        name = request.query\_params.get('name', 'world')        **return** Response({"message" : f"Hello {name}!"}) |

Cách lấy dữ liệu từ request với POST và GET method trong file app/views.py

* Sử dụng Postman để test API

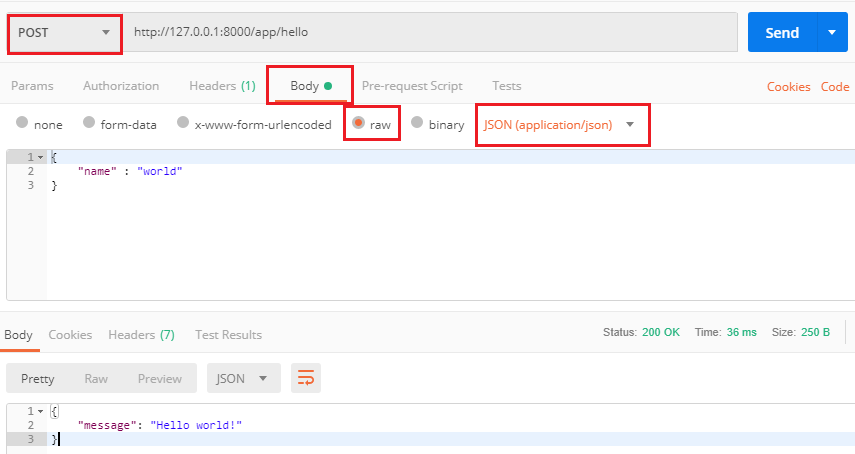
Mặc dù django cung cấp giao diện test API từ trình duyệt, nhưng để khách quan, chúng ta nên sử dụng một tool REST client để test API. Một trong những tool phổ biến là Postman, có thể download tại [https://www.getpostman.com/](https://www.google.com/url?q=https://www.getpostman.com/&sa=D&ust=1550420032790000)

Test GET method:



Sử dụng postman để test GET method

Test POST method:



Sử dụng postman để test POST method

* Sử dụng View Class

Ngoài việc sử dụng kí hiệu @api\_view để tạo endpoint, có thể sử dụng View Class để tạo endpoint. View Class cần kế thừa class rest\_framework.views.APIView và cần cung cấp các hàm get, post, … tương ứng với các endpoint xử lý cho các method.

|  |
| --- |
| **from** rest\_framework.views **import** APIView **from** rest\_framework.response **import** Response  **class** **HelloView**(APIView):        **def** **get**(self, request, format=None):        name = request.query\_params.get('name', 'world')        **return** Response({"message" : f"Hello {name}!"})            **def** **post**(self, request, format=None):        name = request.data.get('name', 'world')        **return** Response({"message" : f"Hello {name}!"}) |

Sử dụng View Class trong file app/views.py

Việc mapping url trong file app/urls.py được điều chỉnh như sau:

|  |
| --- |
| **from** django.urls **import** path **from** .views **import** HelloView  urlpatterns = [    path('hello', HelloView.as\_view()), ] |

Thực hiện mapping url cho View Class trong file app/urls.py

Truy xuất dữ liệu với Serializer

Mặc dù django cung cấp ORM để truy xuất database nhưng DRF cung cấp thêm Serializer để chuyển đổi giữa JSON object và database model một cách thuận tiện.

Trong phần này, chúng ta xây dựng một ví dụ thao tác trên database sử dụng Serializer. Để dễ hình dung, chúng ta dùng mô hình database của ứng dụng quản lý học sinh ở phần trước.

* Tạo mới project và ứng dụng

django-admin startproject mysite

cd mysite

python manage.py startapp students

* Thêm ứng dụng vào setting.py của project :

INSTALLED\_APPS = [

    'students',             # new

    'rest\_framework',       # new

    'django.contrib.admin',

    ….

Thêm ứng dụng vào file mysite/settings.py

* Tạo database model trong file students/models.py:

|  |
| --- |
| **from** django.db **import** models  **class** **Student**(models.Model):    studentNo = models.CharField(db\_column='student\_no', max\_length=20, unique=**True**)                                  studentName = models.CharField(db\_column='student\_name', max\_length=50)                                  address = models.CharField(db\_column='address', max\_length=100) |

File students/models.py

* Khởi tạo database:

python manage.py makemigrations students

python manage.py migrate

* Tạo mới file students/serializers.py với nội dung sau:

|  |
| --- |
| **from** rest\_framework **import** serializers **from** .models **import** Student   **class** **StudentSerializer**(serializers.ModelSerializer):    **class** **Meta**:        model = Student        fields = ('studentNo', 'studentName', 'address') |

File students/serializers.py

Class StudentSerializer kế thừa class ModelSerializer của DRF, có nhiệm vụ thực hiện chuyển đổi từ database model sang JSON object và ngược lại.

* Tạo các api xử lý dữ liệu trong file students/views.py :

|  |
| --- |
| **from** rest\_framework.views **import** status **from** rest\_framework.decorators **import** api\_view **from** rest\_framework.response **import** Response **from** django.shortcuts **import** get\_object\_or\_404  **from** .models **import** Student **from** .serializers **import** StudentSerializer   @api\_view(['GET']) **def** **getAllStudents**(request, format=None):    students = [StudentSerializer(student).data **for** student **in** Student.objects.all()]    **return** Response(students)  @api\_view(['GET']) **def** **getStudent**(request, pk):    student = get\_object\_or\_404(Student, pk=pk)    **return** Response(StudentSerializer(student).data)     @api\_view(['POST']) **def** **createStudent**(request):    serializer = StudentSerializer(data=request.data)            **if** **not** serializer.is\_valid():        **return** Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)            student = serializer.save()    **return** Response(StudentSerializer(student).data) |

File students/views.py

Việc sử dụng Serializer giúp cho việc chuyển đổi giữa database model sang JSON thực hiện nhanh chóng, ví dụ :

StudentSerializer(student).data

Sẽ tương đương với :

{

        'studentNo' : student.studentNo,

        'studentName' : student.studentName,

        'address' : student.address

}

* Cuối cùng, chúng ta thêm các mapping url cho các API vừa tạo:

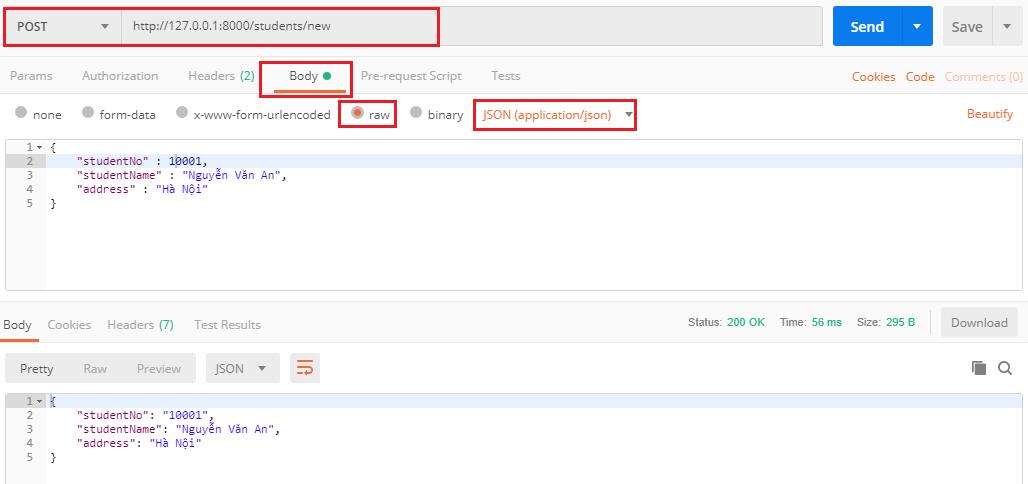
|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path, include  urlpatterns = [    path('students/', include('students.urls')),    path('admin/', admin.site.urls), ] |

File mysite/urls.py

|  |
| --- |
| **from** django.urls **import** path **from** .views **import** getAllStudents, getStudent, createStudent  urlpatterns = [    path('all', getAllStudents),            path('<int:pk>', getStudent),    path('new', createStudent),     ] |

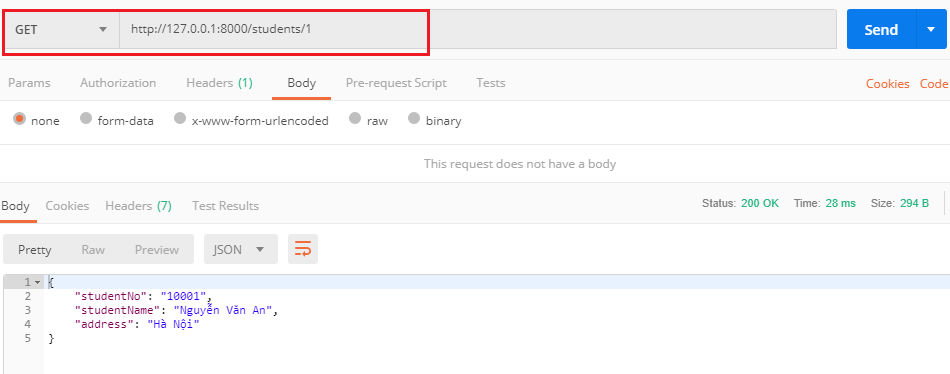
File students/urls.py

* Khởi động ứng dụng và sử dụng postman để test các API:
* Tạo mới học sinh:



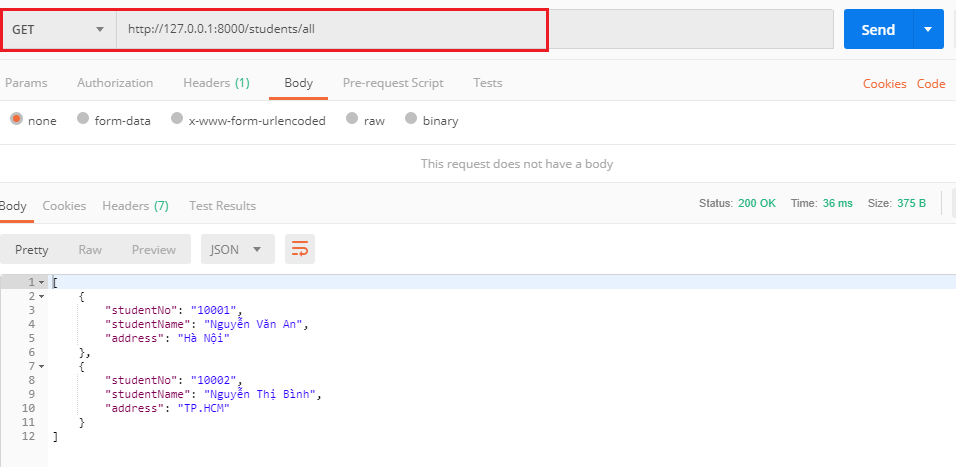
Sử dụng postman để tạo mới học sinh

* Xem thông tin một học sinh theo id :



Sử dụng postman để xem thông tin học sinh theo id

* Xem toàn bộ danh sách học sinh :



Sử dụng postman để xem toàn bộ danh sách học sinh

Authentication với JWT

Khi client gọi service ở backend thì cần có cơ chế authentication để chỉ tránh truy nhập từ các nguồn lạ không đăng ký. Việc này tương tự như cơ chế đăng nhập trong các ứng dụng web truyền thống.

Với các ứng dụng SPA, việc authentication thường sử dụng giao thức JWT (Json web tokens), theo đó các bước đăng nhập như sau:

* Ban đầu client gửi thông tin đăng nhập (username/password) lên server, server xác thực, nếu đúng sẽ trả về token
* Sau khi client nhận token sẽ lưu giữ cho các lần gọi API sau. Mỗi lần gọi API, client phải gửi token (đặt trong trường Authorization của request header), khi server nhận request sẽ kiểm tra token có hợp lệ không trước khi cho phép truy nhập đến API
* Sau một thời gian, token sẽ hết hiệu lực, client phải refresh lại để lấy token mới

Với DRF, để thêm chức năng authentication với JWT, chúng ta sử dụng package djangorestframework\_simplejwt :

pip install djangorestframework\_simplejwt

Để thêm chức năng authentication cho API, chúng ta gắn permission cho endpoint:

|  |
| --- |
| **from** rest\_framework.views **import** status **from** rest\_framework.decorators **import** api\_view, permission\_classes **from** rest\_framework.response **import** Response **from** django.shortcuts **import** get\_object\_or\_404 **from** rest\_framework.permissions **import** IsAuthenticated  **from** .models **import** Student **from** .serializers **import** StudentSerializer  @api\_view(['GET']) @permission\_classes((IsAuthenticated,)) **def** **getAllStudents**(request, format=None):    students = [StudentSerializer(student).data **for** student **in** Student.objects.all()]    **return** Response(students)  @api\_view(['GET']) @permission\_classes((IsAuthenticated,)) **def** **getStudent**(request, pk):    student = get\_object\_or\_404(Student, pk=pk)    **return** Response(StudentSerializer(student).data)     @api\_view(['POST']) @permission\_classes((IsAuthenticated,)) **def** **createStudent**(request):    serializer = StudentSerializer(data=request.data)            **if** **not** serializer.is\_valid():        **return** Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)            student = serializer.save()    **return** Response(StudentSerializer(student).data) |

Thêm permission cho endpoint trong file students/views.py

Nếu sử dụng View Class, việc thêm permission như sau:

|  |
| --- |
| **class** **StudentsView**(APIView):    permission\_classes = (IsAuthenticated,)        **def** **get**(request, format=None):        students = [StudentSerializer(student).data **for** student **in** Student.objects.all()]        **return** Response(students) |
|  |

Thêm permission trong trường hợp dùng View Class

Trong file urls.py project, bổ sung thêm khai báo sau url như sau:

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path, include **from** rest\_framework\_simplejwt **import** views **as** jwt\_views              # new  urlpatterns = [    path('students/', include('students.urls')),    path('api/token', jwt\_views.TokenObtainPairView.as\_view()),      # new    path('api/token/refresh', jwt\_views.TokenRefreshView.as\_view()), # new    path('admin/', admin.site.urls), ] |

Thêm khai bảo url trong file urls.py của project

Hai url được khai báo thêm là:

* api/token : Lấy token lần đầu.
* api/token/refresh : Refresh token khi hết hạn

Ngoài ra, bố sung vào file settings.py của project thông tin sau:

STATIC\_URL = '/static/'

# new

REST\_FRAMEWORK = {

    'DEFAULT\_AUTHENTICATION\_CLASSES': [

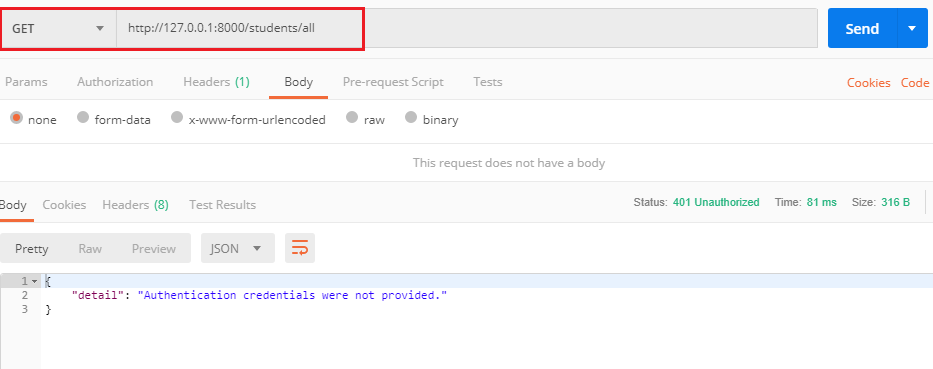
        'rest\_framework\_simplejwt.authentication.JWTAuthentication',

    ],

}

Bổ sung cấu hình cho file settings.py của project

Khởi động ứng dụng, dùng postman để test API, chúng ta thấy sẽ có kết quả như sau:



Kết quả gọi API khi chưa có thông tin authentication

Do chưa có thông tin authentication nên server không trả về dữ liệu. Để thực hiện authentication, trước hết chúng ta lấy token bằng cách POST đến endpoint api/token với nội dung (dạng application/json) chứa thông tin đăng nhập như sau:

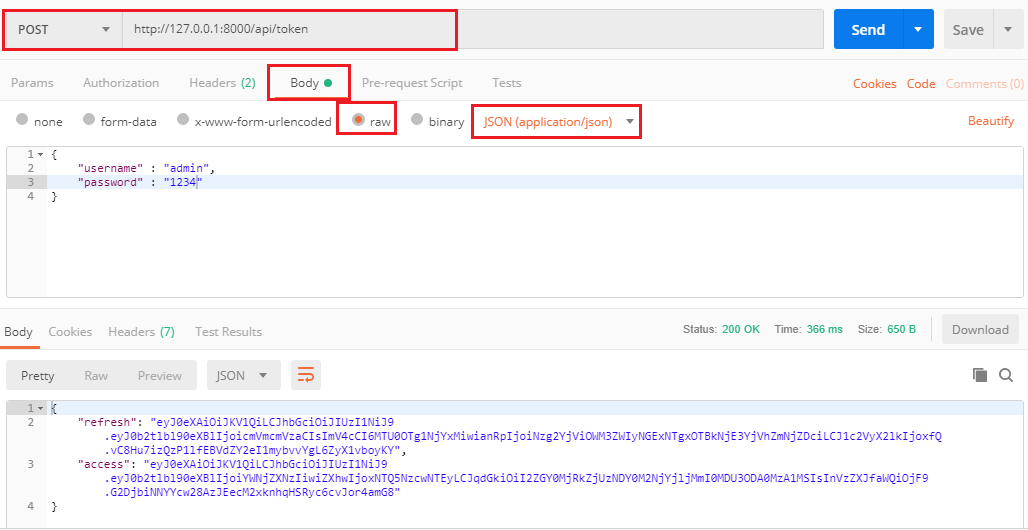
{

    "username" : "<username>",

    "password" : "<password>"

}

Trong đó <username>, <password> là thông tin tài khoản của django (có thể tạo ra từ lệnh **python manage.py createsuperuser**)



Sử dụng postman để lấy token

Kết quả trả về sẽ chứa 2 trường:

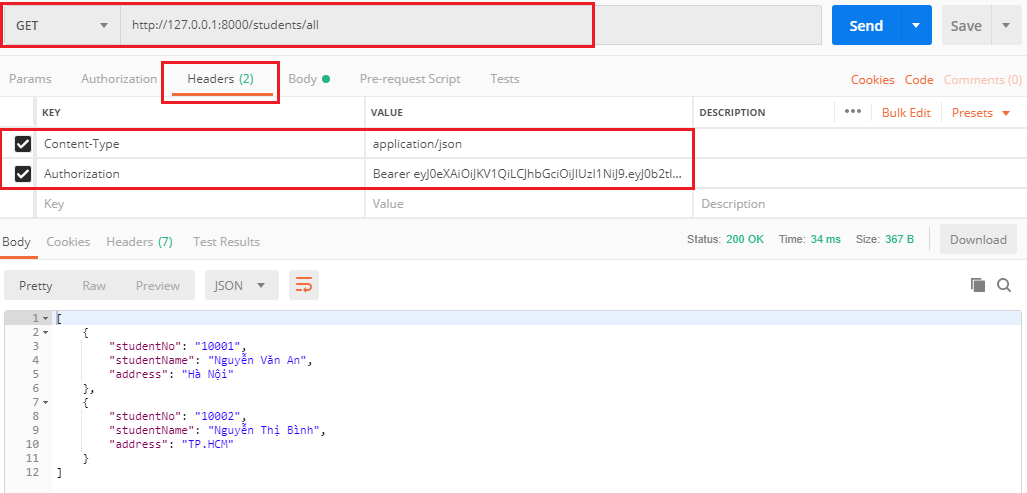
* access : dùng làm token cho các lần truy nhập API tiếp theo
* refresh : dùng để lấy token mới khi token cũ hết hạn

Để truy nhập đến các API của ứng dụng, chúng ta cần lưu lại giá trị access, ở ví dụ trên là:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNTQ5NzcwNTEyLCJqdGkiOiI2ZGY0MjRkZjUzNDY0M2NjYjljMmI0MDU3ODA0MzA1MSIsInVzZXJfaWQiOjF9.G2DjbiNNYYcw28AzJEecM2xknhqHSRyc6cvJor4amG8

Mỗi lần gọi API của ứng dụng, chúng ta cần thêm vào header của request trường Authorization với nội dung bằng "Bearer " + access, cụ thể  ở ví dụ trên là:

Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNTQ5NzcwNTEyLCJqdGkiOiI2ZGY0MjRkZjUzNDY0M2NjYjljMmI0MDU3ODA0MzA1MSIsInVzZXJfaWQiOjF9.G2DjbiNNYYcw28AzJEecM2xknhqHSRyc6cvJor4amG8



Thêm trường Authorization trong request header để truy nhập API ứng dụng

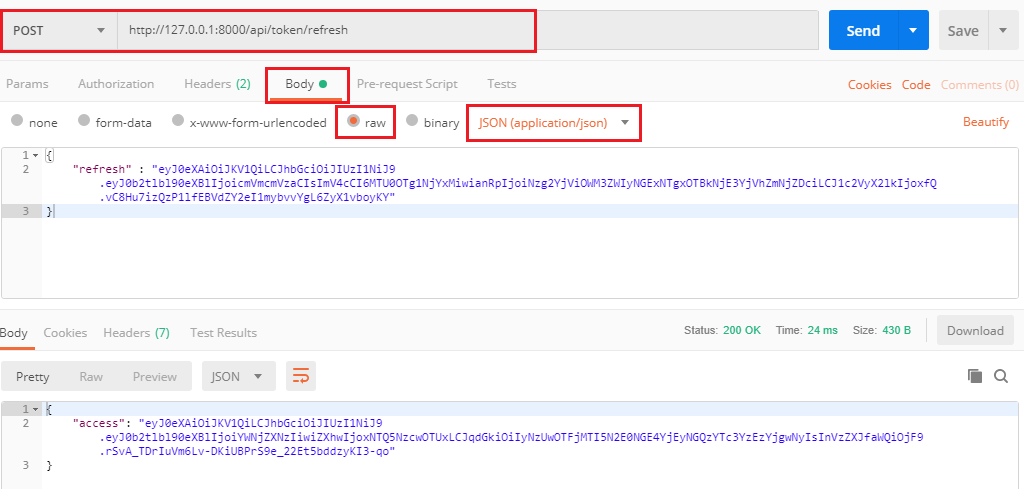
Trong trường hợp token hết hạn, chúng ta sử dụng giá trị refresh (đã nhận được ở phía trên) để lấy lại token bằng cách post đến endpoint /api/token/refresh với nội dung (dạng application/json) như sau:

{

    "refresh" : "<refresh>"

}

Trong đó <refresh> là giá trị đã nhận được khi gọi hàm /api/token ở phía trên



Lấy token mới khi token cũ hết hạn