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

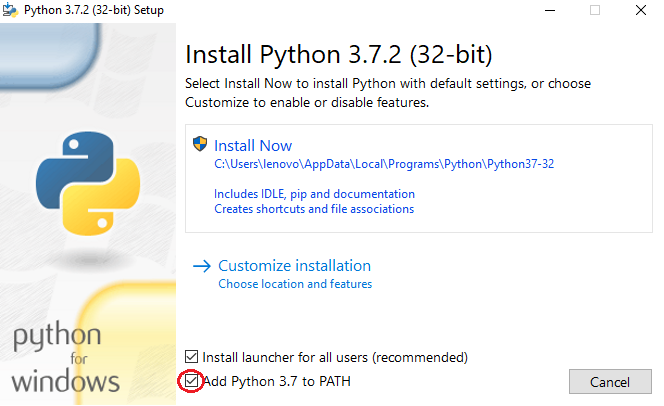
Giới thiệu

[Flask](https://www.google.com/url?q=http://flask.pocoo.org/&sa=D&ust=1550421914823000) là một microframework cho phép phát triển webservice trên Python một cách đơn giản và nhanh chóng.

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=1550421914825000)
* 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

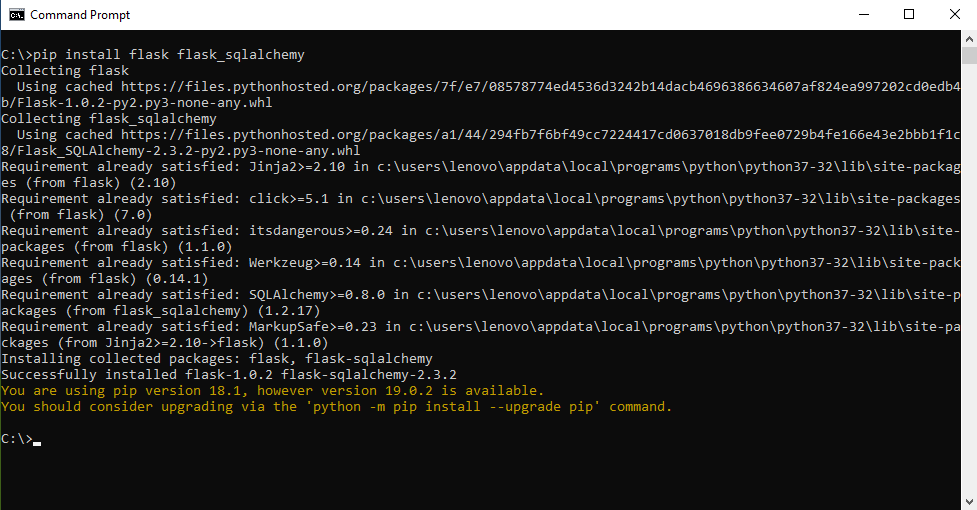


Thêm lựa chọn “Add Python to PATH” khi cài đặt Python

Cài đặt thư viện Flask

Sử dụng pip để cài đặt thư viện Flask và các thư viện liên quan. Từ cửa sổ cmd, gõ lệnh sau để cài đặt Flask:

pip install flask flask\_sqlalchemy



Cài đặt flask bằng pip từ cửa sổ cmd

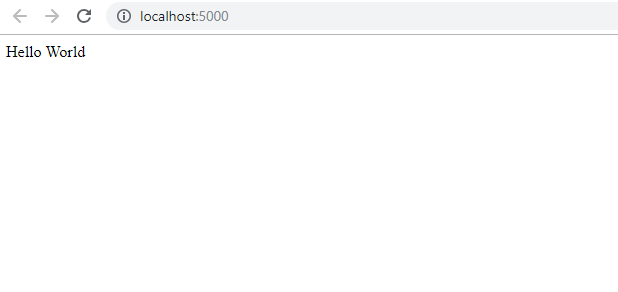
Phát triển ứng dụng web với flask

Xây dựng một ứng dụng đơn giản

Tạo một file với tên app.py với nội dung sau và chạy chương trình:

|  |
| --- |
| **from** flask **import** Flask app = Flask(\_\_name\_\_)  @app.route('/') **def** **hello\_world**():    **return** 'Hello World'  **if** \_\_name\_\_ == '\_\_main\_\_':    app.run() |

Khi ứng dụng chạy, server sẽ lắng nghe ở cổng mặc định là 5000. Truy nhập vào địa chỉ [http://localhost:5000](https://www.google.com/url?q=http://localhost:5000&sa=D&ust=1550421914830000), chúng ta sẽ thấy kết quả ứng dụng trả về:



Kết quả trả về khi truy nhập trang chủ ứng dụng tại http://localhost:5000

Ở chương trình ví dụ trên, lệnh khởi động server trong hàm main là:

app.run()

Với lệnh mặc định này, server sẽ lắng nghe ở địa chỉ 127.0.0.1 , cổng 5000 (cổng mặc định của flask). Để thay đổi cấu hình server, có thể truyền vào lệnh run này các tham số sau:

* host : Thay đổi ip mà server sẽ lắng nghe
* port : Thay đổi cổng mà server sẽ lắng nghe
* debug : True/False. Nếu giá trị này bằng True thì mỗi khi sửa file app.py và lưu lại, server sẽ tự khởi động lại

Ví dụ:

app.run(host='0.0.0.0', port=8080, debug=True)

Khai báo địa chỉ url và phương thức cho endpoint

Khai báo url cho endpoint

@app.route('/hello')

def hello\_world():

    return 'Hello World'

Khi chạy ứng dụng, endpoint sẽ có url là http://<ip:port>/hello

Khai báo phương thức cho endpoint:

Theo mặc định, phương thức của endpoint sẽ là GET. Để chỉ định các phương thức mà endpoint sẽ hỗ trợ, cần khai báo danh sách các phương thức này vào cuối annotation

@app.route như sau:

|  |
| --- |
| **from** flask **import** Flask, request app = Flask(\_\_name\_\_)  @app.route('/hello', methods = ['POST', 'GET']) **def** **hello\_world**():    **if** request.method == 'POST':        **return** 'Hello from POST method'    **else**:        **return** 'Hello from GET method'  **if** \_\_name\_\_ == '\_\_main\_\_':    app.run() |

Lấy tham số từ request

Với GET method, các tham số do client gửi lên server được lấy qua request.args, còn với POST method, các tham số lấy từ request.form nếu dữ liệu gửi lên ở dạng form-data, hoặc request.json nếu dữ liệu gửi lên ở dạng JSON. Ngoài ra nếu form ở dạng multi-part thì còn có thêm request.files chứa các file upload lên từ client.

|  |
| --- |
| **from** flask **import** Flask, request app = Flask(\_\_name\_\_)  @app.route('/hello', methods = ['POST', 'GET']) **def** **hello\_world**():    **if** request.method == 'POST':        **if** 'name' **in** request.form:            name = request.form['name']            **return** f'Hello {name} from POST method (form-data)'                                 **elif** 'name' **in** request.json:            name = request.json['name']            **return** f'Hello {name} from POST method (JSON)'                                         **else**:            **return** 'Hello from POST method'    **else**:        name = request.args.get('name', '')        **return** f'Hello {name} from GET method'  **if** \_\_name\_\_ == '\_\_main\_\_':    app.run() |

Cách lấy tham số trong GET và POST method

Mapping tham số trong url

Ngoài cách lấy tham số từ request, có thể mapping tham số trong khai báo url:

|  |
| --- |
| **from** flask **import** Flask app = Flask(\_\_name\_\_)  @app.route('/hello/<name>') **def** **hello\_name**(name):   **return** f'Hello {name}'  @app.route('/student/<int:studentId>') **def** **getStudent**(studentId):   **return** f'Student {studentId}'    **if** \_\_name\_\_ == '\_\_main\_\_':   app.run(debug = **True**) |

Truy nhập static files

Các file static (css, js, jpg, png …) có thể được truy nhập qua một folder có tên là static đặc ngang hàng với file app.py. Ví dụ:

app.py

static/

    test.jpg

Trong trường hợp này, link tời file test.jpg là http://localhost:5000/static/test.jpg

Sử dụng template

Mỗi endpoint phải trả về một giá trị string chứa nội dung của trang web cần hiển thị. Tuy nhiên thực tế , hầu như không ứng dụng nào trả về giá trị string này trong file app.py, mà sẽ sử dụng thông qua các template. Các file template này thường có tên mở rộng là html và đặt trong thư mục templates ngang hàng với file app.py:

app.py

templates/

    index.html

Chương trình sau minh họa một ví dụ sử dụng template đơn giản:

|  |
| --- |
| **from** flask **import** Flask, render\_template app = Flask(\_\_name\_\_)  @app.route('/') **def** **hello**():   **return** render\_template('index.html')  **if** \_\_name\_\_ == '\_\_main\_\_':   app.run() |

File app.py

|  |
| --- |
| <**html**>    <**body**>        <**b**>Hello, world!</**b**>    </**body**> </**html**> |

File templates/index.html

Truyền biến xuống template

Thông thường với các ứng dụng web, các biến dữ liệu sẽ được tính toán trong hàm xử lý (Controller), sau đó sẽ truyền xuống cho template. Ví dụ sau minh họa cách flask truyền biến xuống template:

|  |
| --- |
| **from** flask **import** Flask, render\_template app = Flask(\_\_name\_\_)  @app.route('/hello/<user>') **def** **hello\_name**(user):   **return** render\_template('index.html', user=user)  **if** \_\_name\_\_ == '\_\_main\_\_':   app.run() |

File app.py

|  |
| --- |
| <**html**>    <**body**>        <**b**>Hello, {{ user }}!</**b**>    </**body**> </**html**> |

File templates/index.html

Ở template, các biến từ hàm controller gửi sang sẽ được đặt trong kí hiệu {{ }}

Redirect và error

Trong trường hợp đang ở hàm xử lý của một endpoint mà chúng ta muốn chuyển sang một endpoint khác thì có thể dùng hàm **redirect**. Hàm này nhận tham số là url cần chuyển hướng tới. Tuy nhiên, có thể dùng hàm **urlfor** và truyền vào tên hàm xử lý của endpoint cần chuyển hướng tới. Trong trường hợp xảy ra lỗi và chúng ta muốn thoát ngay khỏi hàm xử lý và trả về lỗi luôn thì có thể dùng hàm **abort**.

Ví dụ sau minh họa cách dùng các hàm trên.

|  |
| --- |
| **from** flask **import** Flask, request, render\_template, redirect, url\_for, abort app = Flask(\_\_name\_\_)  @app.route('/hello/<user>') **def** **hello\_name**(user):    **return** render\_template('index.html', user=user)  @app.route('/login', methods=['GET', 'POST']) **def** **login**():    **if** request.method == 'POST':        user = request.form.get('user', '')        password = request.form.get('password', '')                         **if** user == 'admin' **and** password == 'password':            **return** redirect(url\_for('hello\_name',user=user))        **else**:            abort(401)                **return** render\_template('login.html')  **if** \_\_name\_\_ == '\_\_main\_\_':   app.run() |

File app.py

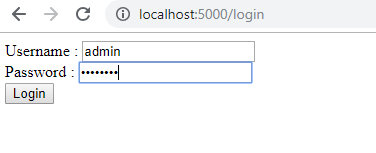
|  |
| --- |
| <**form** method='post'>    Username : <**input** name='user'> <**br**>    Password : <**input** name='password' type='password'> <**br**>    <**input** type='submit' value='Login'> </**form**> |

File templates/login.html

|  |
| --- |
| <**html**>    <**body**>        <**b**>Hello, {{ user }}!</**b**>    </**body**> </**html**> |

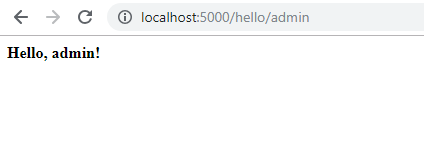
File templates/index.html

Chạy ứng dụng, truy nhập vào link [http://localhost:5000/login](https://www.google.com/url?q=http://localhost:5000/login&sa=D&ust=1550421914861000) sẽ xuất hiện màn hình đăng nhập như sau:



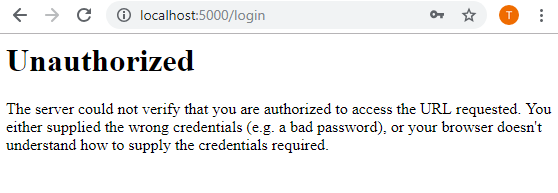
Màn hình đăng nhập http://localhost:5000/login

Nếu username/pasword nhập vào là "admin"/"password" thì chương trình sẽ redirect sang trang thành công



Redirect sang trang mới sau khi đăng nhập thành công

Nếu nhập username/password khác giá trị ở trên thì sẽ xuất hiện màn hình lỗi 401



Màn hình lỗi 401 khi nhập sai username/password

Truy nhập database với flask\_sqlalchemy

Flask\_sqlalchemy là thư viện ORM cho phép làm việc với nhiều database khác nhau một cách đơn giản và thuận tiện.

Khai báo model với flask\_sqlalchemy

|  |
| --- |
| **from** flask **import** Flask **from** flask\_sqlalchemy **import** SQLAlchemy  app = Flask(\_\_name\_\_) app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///students.db' db = SQLAlchemy(app)   **class** **Student**(db.Model):    id = db.Column(db.Integer, primary\_key = **True**)    studentNo = db.Column('student\_no', db.String(20), unique=**True**)    studentName = db.Column('student\_name', db.String(50))    address = db.Column('address', db.String(50))       **def** **\_\_init\_\_**(self, studentNo, studentName, address):        self.studentNo = studentNo        self.studentName = studentName        self.address = address |

File app.py

Ở file ví dụ trên, chúng ta khai báo một model Student với các trường:

* Id : INT, PRIMARY KEY
* studentNo : VARCHAR(20), UNIQUE
* studentName : VARCHAR(50)
* address : VARCHAR(100)

Cơ sở dữ liệu sử dụng được khai báo trong biến app.config['SQLALCHEMY\_DATABASE\_URI'], ở ví dụ này là sqlite (một ví dụ về MySQL sẽ được trình bày ở phần sau)

Sau khi đã có file app.py trên, chúng ta có thể chạy ứng dụng python shell trong thư mục chứa file app.py này và thực hiện một số thao tác sau:

Tạo mới bảng

Sử dụng lệnh sau để tạo mới các bảng tương ứng với các model đã khai báo:

>>> from app import db

>>> db.create\_all()

Sau lệnh này, file students.db sẽ được tạo ra trong cùng thư mục với file app.py. Nếu sử dụng các tool database client để xem nội dung file này, sẽ thấy bảng student đã được tạo ra.

Tạo mới một số bản ghi

Từ cửa sổ python shell, chúng ta có thể tạo một số bản ghi để test thử

>>> from app import Student

>>> student1 = Student('101', 'Nguyễn Văn An', 'Hà Nội')

>>> student2 = Student('102', 'Nguyễn Thị Bình', 'TP.HCM')

>>> db.session.add(student1)

>>> db.session.add(student2)

>>> db.session.commit()

Xem danh sách các bản ghi trong database:

>>> students = Student.query.all()

>>> for student in students:

>>>     print(student.id, student.studentNo, student.studentName, student.address)

Tìm kiếm bản ghi theo id:

>>> student1 = Student.query.get(1)

>>> print(student1.id, student1.studentNo, student1.studentName, student1.address)

Sửa bản ghi:

>>> student1 = Student.query.get(1)

>>> student1.address = 'TP. Hà Nội'

>>> db.session.commit()

Xóa bản ghi:

>>> student1 = Student.query.get(1)

>>> db.session.delete(student1)

>>> db.session.commit()

Xây dựng ứng dụng web với các chức năng truy nhập cơ sở dữ liệu cơ bản

Để hiểu rõ hơn các thành phần chức năng trong flask như đã mô tả ở các phần trên, trong phần này chúng ta sẽ xây dựng một ứng dụng web demo thực hiện quản lý học sinh trong một lớp học. Ứng dụng có 2 màn hình chính:

* Màn hình danh sách học sinh : hiển thị toàn bộ học sinh trong lớp học
* Màn hình thêm mới, chỉnh sửa thông tin học sinh : cho phép chỉnh sửa, thêm mới thông tin của từng học sinh.

Các bước thực hiện như sau :

Cài đặt các thư viện cần thiết

Trong ứng dụng này, chúng ta sẽ sử dụng cơ sở dữ liệu là MySQL, do đó cần cài đặt thư việc MySQL client cho Python.  Cách thực hiện như sau:

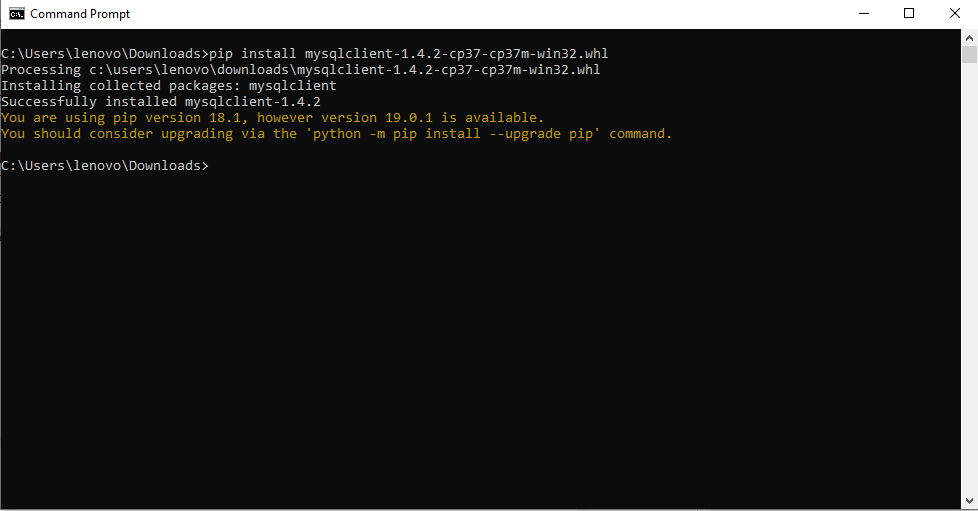
* Download thư viện tại : [https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient](https://www.google.com/url?q=https://www.lfd.uci.edu/~gohlke/pythonlibs/%23mysqlclient&sa=D&ust=1550421914874000)

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

Tạo khung ứng dụng

Mặc dù flask không quy định cấu trúc thư mục và file của các thành phần xử lý nhưng để tiện phát triển, chúng ta nên định nghĩa cấu trúc của các module thành phần ngay từ đầu. Trong ví dụ này, cấu trúc thư mục của ứng dụng sẽ gồm các thành phần như sau:

static/

    images

templates/

    list\_student.html

    add\_edit\_student.html

config.py

app.py

forms.py

db\_serice.py

Trong đó :

* static : thư mục chứa các file tĩnh, thư mục con images chứa các ảnh profile của các học sinh
* templates : thư mục chứa các template html
* config.py : File chứa các thông tin cấu hình
* app.py : File ứng dụng chính của flask
* forms.py : File định nghĩa các form để nhận tham số từ request
* db\_service.py : File chứa các chức năng truy xuất dữ liệu

Khai báo các thông tin cấu hình trong config.py

Các thông tin cấu hình trong file config.py như sau:

|  |
| --- |
| **from** flask **import** Flask  app = Flask(\_\_name\_\_) app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://admin:1234@localhost/testdb' app.config['IMG\_DIR'] = 'static/images' app.config['SECRET\_KEY'] = "abcxyztuv" |

File config.py

Tạo database model và các hàm truy xuất trong db\_serice.py :

|  |
| --- |
| **from** flask\_sqlalchemy **import** SQLAlchemy **from** config **import** app  db = SQLAlchemy(app)  **class** **Student**(db.Model):    id = db.Column(db.Integer, primary\_key = **True**)    studentNo = db.Column('student\_no', db.String(20), unique=**True**)    studentName = db.Column('student\_name', db.String(50))    address = db.Column('address', db.String(50))       **def** **\_\_init\_\_**(self, studentNo, studentName, address):        self.studentNo = studentNo        self.studentName = studentName        self.address = address  **def** **getAllStudents**():    **return** Student.query.all()  **def** **getStudentById**(id):    **return** Student.query.get(id)  **def** **createStudent**(studentNo, studentName, address):    student = Student(studentNo, studentName, address)    db.session.add(student)    db.session.commit()    **return** student  **def** **updateStudent**(id, studentNo, studentName, address):    student = Student.query.get(id)    student.studentNo = studentNo    student.studentName = studentName    student.address = address    db.session.commit()    **return** student         **def** **deleteStudent**(id):    student = Student.query.get(id)    db.session.delete(student)    db.session.commit()  db.create\_all() |

File db\_service.py

Các chức năng trong file db\_service.py đã được mô tả trong phần làm việc với flask\_sqlalchemy.

Xây dựng màn hình danh sách học sinh

Màn hình danh sách học sinh hiển thị toàn bộ các học sinh có trong cơ sở dữ liệu. Do chưa có chức năng thêm mới thông tin học sinh trên giao diện, chúng ta dùng python shell để thêm mới bằng tay một số bản ghi trước để làm dữ liệu test:

>>> import db\_service

>>> db\_service.createStudent('101', 'Nguyễn Văn An', 'Hà Nội')

>>> db\_service.createStudent('102', 'Nguyễn Thị Bình', 'TP.HCM')

Hàm xử lý endpoint trong file app.py như sau:

|  |
| --- |
| **from** flask **import** Flask, render\_template **from** config **import** app **import** db\_service  @app.route('/') **def** **list\_student**():    studentList = db\_service.getAllStudents()    **return** render\_template("list\_student.html", studentList=studentList)    **if** \_\_name\_\_ == '\_\_main\_\_':   app.run(debug = **True**) |

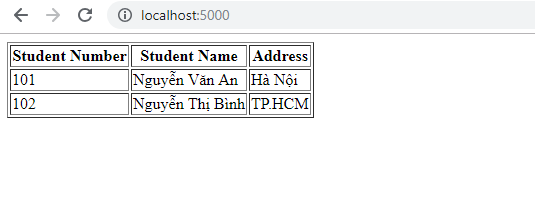
File app.py với chức năng hiển thị danh sách học sinh

File template cho màn hình danh sách được đặt ở đường dẫn templates/list\_student.html với nội dung như sau:

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

File templates/list\_student.html

Khởi động ứng dụng (python app.py), truy nhập vào địa chỉ [http://localhost:5000](https://www.google.com/url?q=http://localhost:5000&sa=D&ust=1550421914892000) chúng ta sẽ thấy màn hình danh sách học sinh như sau:



Màn hình danh sách học sinh (draft)

Màn hình thêm mới chỉnh sửa thông tin học sinh

Trước hết , chúng ta tạo một form thông tin học sinh trong file templates/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'> <**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**>  <**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="/"'> </**form**> |

File templates/add\_edit\_student.html

Trong form nhập thông tin, có các trường : mã học sinh, tên, địa chỉ và ảnh đại diện. Sau khi có form html này, chúng ta thêm vào file forms.py một class để chứa các thông tin khi form submit lên server:

|  |
| --- |
| **import** os **import** db\_service **from** config **import** app  **class** **StudentForm**():     **def** **\_\_init\_\_**(self, request=None, dbModel=None):                    **if** request:            self.studentId = request.form['studentId']            self.studentNo = request.form['studentNo']            self.studentName = request.form['studentName']            self.address = request.form['address']                        file = request.files.get('profilePicture')            **if** file **and** file.filename != '':                self.profilePictureFile = file            **else**:                self.profilePictureFile = **None**                    **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):                **if** self.studentId:            student = db\_service.updateStudent(                   self.studentId,                   self.studentNo,                   self.studentName,                   self.address)        **else**:            student = db\_service.createStudent(                   self.studentNo,                   self.studentName,                   self.address)                **if** student **and** self.profilePictureFile:            self.profilePictureFile.save(os.path.join(app.config['IMG\_DIR'], str(student.id) + '.jpg')) |

File forms.py

Tiếp theo, chúng ta bổ sung các hàm xử lý request trong app.py :

|  |
| --- |
| **from** flask **import** Flask, render\_template, request, redirect, url\_for, jsonify **import** traceback **import** time  **from** forms **import** StudentForm **from** config **import** app **import** db\_service  @app.route('/student/add\_edit/<int:studentId>', methods = ['GET', 'POST']) **def** **add\_edit\_student**(studentId):    errorList = []    form = StudentForm()        **if** request.method == 'POST':        **try**:            form = StudentForm(request)            errorList = form.validate()                        **if** **not** errorList:                student = form.save()                                    **return** redirect(url\_for('list\_student'))                        **except** Exception **as** e:            traceback.print\_exc()            errorList.append(str(e))                **elif** studentId > 0:        student = db\_service.getStudentById(studentId)        form = StudentForm(dbModel=student)        **return** render\_template('add\_edit\_student.html', form=form, errorList=errorList)     @app.route('/student/delete/<int:studentId>') **def** **delete\_student**(studentId):    **try**:        db\_service.deleteStudent(studentId)    **except** Exception:        traceback.print\_exc()             **return** redirect(url\_for('list\_student'))  @app.route('/') **def** **list\_student**():    studentList = db\_service.getAllStudents()    **return** render\_template("list\_student.html", studentList=studentList, time = int(time.time()))    **if** \_\_name\_\_ == '\_\_main\_\_':   app.run(debug = **True**) |

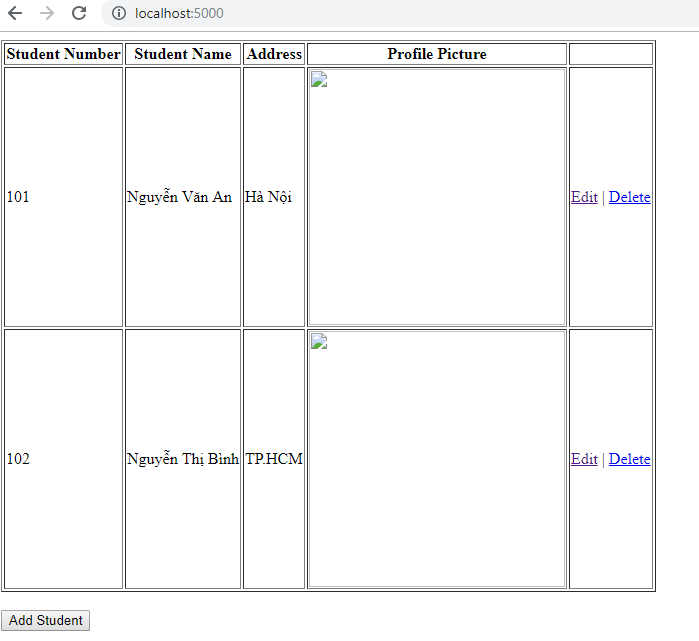
File app.py sau khi bổ sung các hàm xử lý request thêm mới/chỉnh sửa

Cuối cùng update file template/list\_student.html để link màn hình danh sách sang màn hình thêm mới/chỉnh sửa:

|  |
| --- |
| {% 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\_for("add\_edit\_student", studentId=student.id) }}'>Edit</**a**> |          <**a** href='javascript:confirmDelete("{{ url\_for("delete\_student", studentId=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\_for("add\_edit\_student", studentId=0) }}"'> |

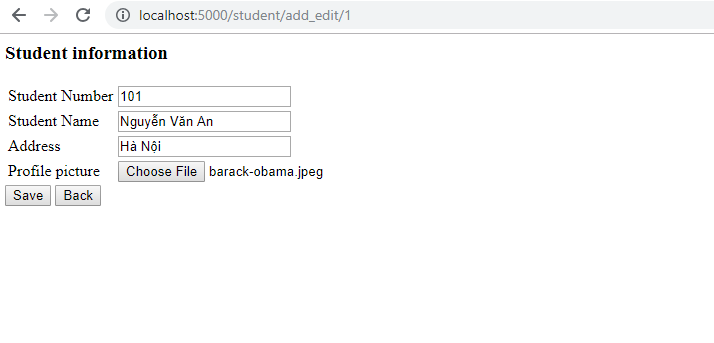
File templates/list\_student.html sau khi update

Khởi động lại ứng dụng, vào màn hình danh sách chúng ta sẽ thấy như sau:



Màn hình danh sách học sinh sau khi update

Click vào các link thêm mới hoặc chỉnh sửa, chúng ta sẽ sang màn hình thông tin học sinh  như sau:



Màn hình thêm mới/chỉnh sửa thông tin học sinh

Sau khi update ảnh profile cho các học sinh, quay lại màn hình danh sách chúng ta sẽ thấy có dạng như sau:



Màn hình danh sách học sinh sau khi update ảnh profile

Xây dựng webservice với flask

Flask cho phép xây dựng các webservice một cách nhanh chóng. Để tạo một webservice với Flask, sử dụng cách khai báo các endpoint theo GET/POST như trong phần trước, chỉ khác là khi trả kết quả về, thì dùng hàm **jsonify** để chuyển dữ liệu sang dạng JSON. Ngoài ra hàm jsonify tự động thêm Content-Type cho response trả về là application/json.

Ví dụ:

|  |
| --- |
| **from** flask **import** Flask, jsonify app = Flask(\_\_name\_\_)   app = Flask(\_\_name\_\_)  @app.route('/list\_student') **def** **list\_student\_api**():    studentList = [        {'id' : 1, 'name' : 'Nguyễn Văn An'},        {'id' : 2, 'name' : 'Nguyễn Thị Binh'},            ]    **return** jsonify({'students' : studentList})   **if** \_\_name\_\_ == '\_\_main\_\_':   app.run(debug = **True**) |

Tạo một webservice đơn giản với flask

Authentication với flask-jwt

JWT (Json web tokens) là phương pháp authentication thường được sử dụng trong các webservice. Trình tự các bước authentication với JWT 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

Để sử dụng JWT với flask, cần cài đặt Flask-JWT:

pip install Flask-JWT

Để thực hiện authentication, cần có danh sách username/password tương ứng với các user được phép gọi API. Thông thường danh sách này lưu trong database, ở ví dụ minh họa dưới đây, chúng ta lưu thông user trong một mảng của chương trình:

class User(object):

    def \_\_init\_\_(self, id, username, password):

        self.id = id

        self.username = username

        self.password = password

users = [

    User(1, 'admin', 'abc@123'),

    User(2, 'guest', '1234'),

]

username\_table = {u.username: u for u in users}

userid\_table = {u.id : u for u in users}

Để thực hiện authentication với flask-jwt, cần cung cấp 2 hàm:

* Hàm authenticate : kiểm tra username/password do client gửi đến (khi lấy token) có hợp lệ không. Hàm này có cấu trúc như sau:

def authenticate(username, password):

    if isCorrect(username, password):

        return User(username=username)

Trong trường hợp username/password hợp lệ, hàm authenticate cần trả về đối tượng User tương ứng với username/password đó. Đối tượng User này có thể có các trường thông tin tùy ý, nhưng bắt buộc phải có trường id (sẽ được dùng ở dưới)

* Hàm identity : lấy ra thông tin của một user dựa trên id, hàm này có cấu trúc như sau:

def identity(payload):

    userid = payload['identity']

    return User(id=userid)

Khi một user đăng nhập thành công và nhận token, Flask-jwt sẽ lưu lại thông tin (userid, token) trong bộ nhớ để authentication cho các lần sau. Khi client gọi API ở các lần tiếp theo, server dựa vào token mà client gửi lên để truy ngược ra user đang thực hiện request (thông tin này được nằm trong một biến được đặt tên là current\_identity và có thể được khai thác trong hàm xử lý của API)

Sau khi đã cung cấp 2 hàm authenticate và identity ở trên, các bước cần thực hiện để thêm chức năng authentication cho một API như sau:

* Tạo mới một đối tượng jwt:

from flask\_jwt import JWT

jwt = JWT(app, authenticate, identity)

* Thêm annotation @jwt\_required() vào phía trước các hàm xử lý của các API cần authentication:

from flask\_jwt import jwt\_required, current\_identity

@app.route('/path')

@jwt\_required()

def api():

    print('Request from user : ', current\_identity.id)

        ....

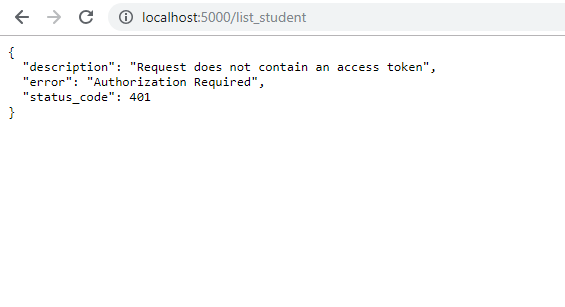
Bên trong hàm xử lý của API, biến current\_identity có thể được dùng để lấy thông tin về user đang thực hiện gọi API.

Chương trình đầy đủ minh họa cách sử dụng Flask-JWT để authentication như sau:

|  |
| --- |
| **from** flask **import** Flask, jsonify **from** flask\_jwt **import** JWT, jwt\_required, current\_identity  **class** **User**(object):    **def** **\_\_init\_\_**(self, id, username, password):        self.id = id        self.username = username        self.password = password  users = [    User(1, 'admin', 'abc@123'),    User(2, 'guest', '1234'), ]  username\_table = {u.username: u **for** u **in** users} userid\_table = {u.id : u **for** u **in** users}  **def** **authenticate**(username, password):    user = username\_table.get(username, **None**)    **if** user **and** user.password == password:        **return** user  **def** **identity**(payload):    userid = payload['identity']    **return** userid\_table[userid]  app = Flask(\_\_name\_\_) app.debug = **True** app.config['SECRET\_KEY'] = 'abcdxyztuv'  jwt = JWT(app, authenticate, identity)  @app.route('/list\_student') @jwt\_required() **def** **list\_student\_api**():    studentList = [        {'id' : 1, 'name' : 'Nguyễn Văn An'},        {'id' : 2, 'name' : 'Nguyễn Thị Binh'},            ]    **return** jsonify({'username' : current\_identity.username, 'students' : studentList})  **if** \_\_name\_\_ == '\_\_main\_\_':    app.run() |

Chương trình minh họa cách sử dụng authentication với Flask-JWT

Khởi động server, dùng trình duyệt để truy nhập API ở địa chỉ [http://localhost:5000/list\_student](https://www.google.com/url?q=http://localhost:5000/list_student&sa=D&ust=1550421914931000) chúng ta sẽ thấy kết quả như sau:



Lỗi khi truy nhập API từ trình duyệt

Do không có thông tin đăng nhập nên API không trả về kết quả mà sẽ hiện thông báo lỗi 401.

Để thực hiện test API, cần sử dụng một tool REST client, ví dụ [Postman](https://www.google.com/url?q=https://www.getpostman.com/&sa=D&ust=1550421914932000).

Quá trình gọi API gồm các bước:

* Thực hiện POST đến địa chỉ [http://localhost:5000/auth](https://www.google.com/url?q=http://localhost:5000/auth&sa=D&ust=1550421914933000) với nội dung (dạng contentType là application/json) với nội dung như sau:

{

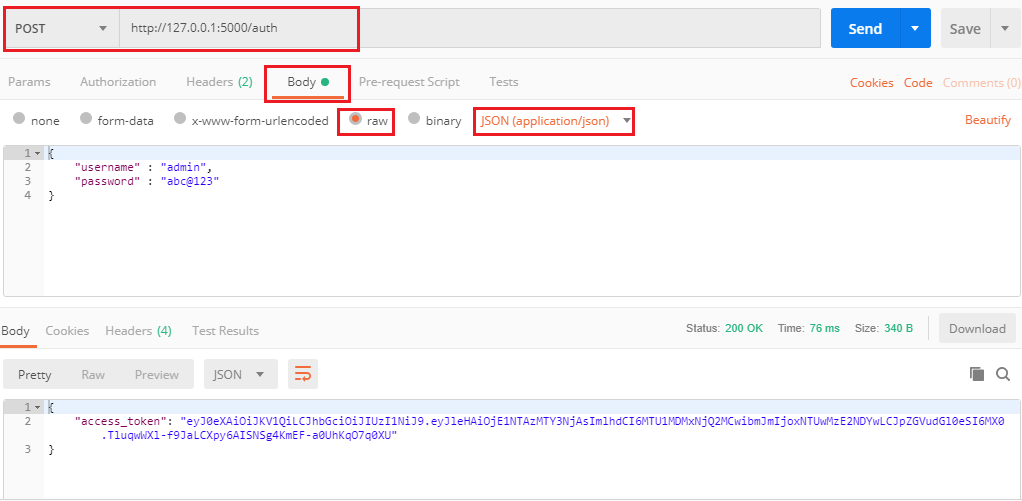
    "username" : "<username>",

    "password" : "<password>"

}

Nếu thông tin username/password đúng, server sẽ trả về token, ví dụ:

{"access\_token" : "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1NTAzMTY3NjAsImlhdCI6MTU1MDMxNjQ2MCwibmJmIjoxNTUwMzE2NDYwLCJpZGVudGl0eSI6MX0.TluqwWXl-f9JaLCXpy6AISNSg4KmEF-a0UhKqO7q0XU"}

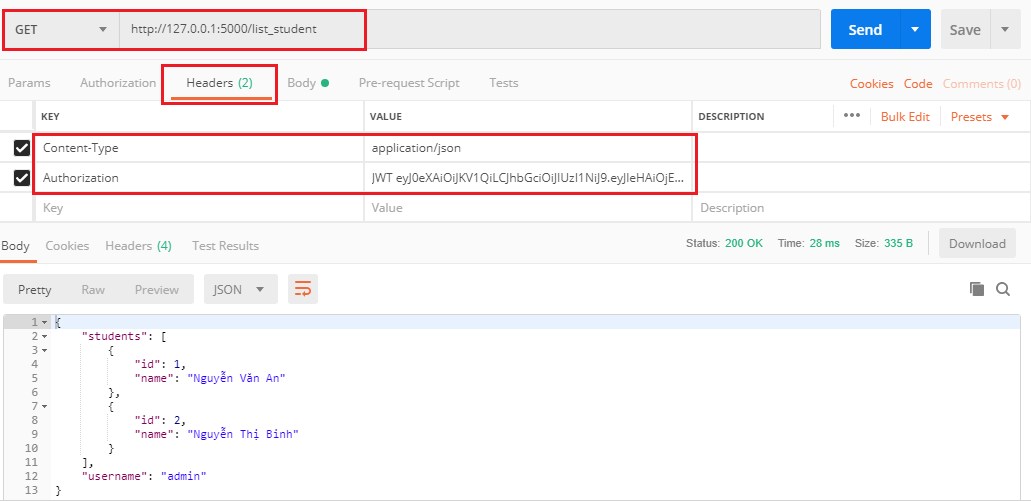


Lấy token từ server

* Sử dụng token đã nhận được để gọi API

Sau khi đã có token, để gọi API, ngoài các tham số thông thường do API quy định, cần thêm vào header của request trường Authorization với nội dung có giá trị bằng "JWT " + token, cụ thể ở ví dụ trên là :

JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1NTAzMTY3NjAsImlhdCI6MTU1MDMxNjQ2MCwibmJmIjoxNTUwMzE2NDYwLCJpZGVudGl0eSI6MX0.TluqwWXl-f9JaLCXpy6AISNSg4KmEF-a0UhKqO7q0XU



Thêm thông tin Authorization vào header request để gọi được API