Python APIs with flask

Επικοινωνία: kbitsak@cslab.ece.ntua.gr Κωνσταντίνος Μπιτσάκος, nchalv@cslab.ece.ntua.gr Νίκος Χαλβαντζής

Τι είναι ένα API;

Ένα API (Application Programming Interface) είναι η διεπαφή ή ένα πρωτόκολλο που σου προσφέρει ένα λειτουργικό σύστημα, μια βιβλιοθήκη, ένα backend μιας βάσης δεδομένων, ώστε να μπορούμε να επικοινωνούμε μαζί του για ανταλλαγή δεδομένων.
Μέσω APIs μπορούν να επικοινωνήσουν δύο εφαρμογές λογισμικού.

Τι είναι η REST αρχιτεκτονική;

Η Representational State Transfer (REST) είναι μια αρχιτεκτονική λογισμικού που καθορίζει κάποιους κανόνες τους οποίους πρέπει να ακολουθούν οι WEB εφαμογές.
Πριν τη REST δημοφιλές μοντέλο που ακολουθούσε το WEB ήταν το SOAP.
Η REST προσφέρει μεγάλη ευκολία καθώς χρησιμοποιεί βασικές HTTP και CRUD εντολές.

Τα πιο συχνά HTTP verbs είναι

- Get: Η μέθοδος με την οποία μπορούμε να ζητήσουμε δεδομένα από έναν server.
- Post: Μέθοδος με την οποία μπορούμε να στείλουμε δεδομένα σε έναν server χρησιμοποιώντας html φόρμες.
- Put: Χρησιμοποιείται για αντικατάσταση των δεδομένων που υπάρχουν στον server, με τα δεδομένα που ανεβάζουμε εμείς.
- Delete: Χρησιμοποιείται για να διαγράψουμε δεδομένα-πόρους στον server.

Όταν στέλνουμε ένα HTTP request, αυτό περιέχει

1. Έναν header
2. Ένα κενό που διαχωρίζει το header από το body του request σου
3. Ένα body προεραιτικά

Ο header περιέχει ένα HTTP verb από τα παραπάνω, ένα URI και την έκδοση του HTTP.
Πχ, GET /home.html HTTP/1.1.

Όταν ένας server λαμβάνει ένα request απαντάει με ένα μήνυμα, είτε επιτυχές επιστρέφοντας δεδομένα είτε με error μήνυμα.

Συνήθως ένα HTTP response περιλαμβάνει

1. Έναν header
2. Ένα κενό που διαχωρίζει το header από το body του request σου
3. Ένα body προεραιτικά

Σε αυτή την περίπτωση ο header περιέχει την έκδοση του HTTP, ένα code για το status της απάντησης και μια φράση που εξηγεί το stadus της απάντησης.

Μερικά συνήθη codes που συναντάμε κατά τη διεπαφή ενός server και ενός client, είναι τα ακόλουθα:

1. 200 Ok: Σημαίνει πως το request μας ήταν επιτυχές.
2. 201 Created: Σημαίνει ότι το resource που θέλαμε να δημιουργήσουμε δημιουργήθηκε
3. 400 Bad Request: Το request που κάναμε δεν μπορούσε να επεξεργαστεί από τον server
4. 404 Not Found: Σημαίνει πως ο server δεν μπορούσε να βρει τη σελίδα που ζητήσαμε

Εισαγωγή σε Flask APIs

Το flask είναι ένα web framework μέσω του οποίου μπορούμε να φτιάξουμε APIs, backend applications σε python.
Άλλα γνωστά τέτοια frameworks είναι το Django για python, η Ruby on Rails για Ruby , το Spring για Java.

Για κάποιον που πρωτοεισάγεται σε web frameworks το Flask προσφέρει τη μεγαλύτερη ευκολία στη χρήση του.

Αρχικά ας φτάξουμε ένα Hello World σε Flask. Έχουμε εγκατεστημένα στο σύστημά μας python και pip.

Τεχουμε

pip install flask

Ανοίγουμε ένα αρχείο πχ, helloworld.py και αντιγράφουμε

```
import flask

app = flask.Flask(__name__)
app.config["DEBUG"] = True


@app.route('/', methods=['GET'])
def home():
    return "<h1>Hello world</h1>"
```

```
app.run()
```

Το flask χρησιμοποιεί την port 5000, ανοίγουμε τον browser μας στο 127.0.0.1:5000 και βλέπουμε πως εκεί ο κώδικάς μας κάνει expose το Hello World.

Εφαρμογή ανταλλαγής μηνυμάτων μέσω Flask

Για να καταλάβουμε καλύτερα τη λογική του Flask θα φτιάξουμε μια πολύ απλή εφαρμογή ανταλλαγής μηνυμάτων.
Ο τρόπος που θα το κάνουμε δεν ενδείκνυται για μια actual εφαρμογή ανταλλαγής μηνυμάτων.
Σε μια τέτοια περίπτωση θα έπρεπε να χρησιμοποιήσουμε jQuery (βιβλιοθήκη της Javascript που βοηθάει στην ασύγχρονη επικοινωνία μεταξύ server και client).

Θα σηκώσουμε δύο διαφορετικές διεργασίες.

Φτιάχουμε δύο αρχεία, user1.py και user2.py.

User1.py

```
from flask import Flask, render_template, request
app = Flask(__name__)
messages={}


@app.route('/',methods = ['POST', 'GET'])
def student():
   if request.method == 'GET':
      return render_template('user1.html')
   if request.method == 'POST':
      currentLen=len(messages)
      currentLen=str(currentLen)
      key="Message"+currentLen
      result = request.form.to_dict()
      messages[key]={"Name": result["Name"], "Message": result["Message"]}
      return render_template('user1.html')

@app.route('/history',methods = ['POST', 'GET'])
def result():
   if request.method == 'GET':

      return render_template("history1.html", messages = messages)

if __name__ == '__main__':
   app.run(port=5000,debug = True)

```

User2.py

```
from flask import Flask, render_template, request
app = Flask(__name__)
messages={}


@app.route('/',methods = ['POST', 'GET'])
def student():
   if request.method == 'GET':
      return render_template('user2.html')
   if request.method == 'POST':
      currentLen=len(messages)
      currentLen=str(currentLen)
      key="Message"+currentLen
      result = request.form.to_dict()
      messages[key]={"Name": result["Name"], "Message": result["Message"]}
      return render_template('user2.html')

@app.route('/history',methods = ['GET'])
def result():
   if request.method == 'GET':

   return render_template("history2.html", messages=messages)

if __name__ == '__main__':
   app.run(port=5001,debug = True)
```

Όπως βλέπουμε η μία διεργασία κάνει expose στο port 5000 και η άλλη στο 5001.

Δίνουμε δύο routes μία στο “/” και μία στο “/history”

Στο history θα δείχνουμε το ιστορικό των μηνυμάτων που έχουν ανταλλαχθεί μεταξύ των δύο χρηστών.

Στο / έχουμε δύο μεθόδους. Μία get και μία post.
Αν το αίτημα του cilent είναι get τότε απλά rendάρεται η κεντρική σελίδα.

Αν το αίτημα είναι post τότε ο χρήστης έχει στείλει ένα μήνυμα το οποίο η εφαρμογή μας διαχειρίζεται με το να το αποθηκεύει στο ιστορικό μηνυμάτων (στο dict messages που έχουμε φτιάξει).

Προκειμένου να δείξουμε στο forntend τη λειτουργικότητα του backend φτιάχνουμε έναν φάκελο tempates κα βάζουμε τα htlm αρχεία που καλούμε από τον κώδικα του backend μας.

Φτιάχουμε τα history1.html, history2.html

history1.html

```
<!doctype html>
<html>
  <body>
    <table border = 1>
      {% for key, value in messages.items() %}
        <tr>
          <th> {{ key }} </th>
          {%for key1, value1 in value.items()%}
          <td> {{ key1 }} </td>
          <td> {{ value1 }} </td>
                {% endfor %}

        </tr>
      {% endfor %}
    </table>
  </body>
</html>
```

history2.html

```
<!doctype html>
<html>
  <body>
    <table border = 1>
      {% for key, value in messages.items() %}
        <tr>
          <th> {{ key }} </th>
          {%for key1, value1 in value.items()%}
          <td> {{ key1 }} </td>
          <td> {{ value1 }} </td>
                {% endfor %}

        </tr>
      {% endfor %}
    </table>
  </body>
</html>
```

Τα history templates έχουν πολύ απλή λειτουργικότητα, διαβάζουν το περιεχόμενο του ιστορικού μηνημάτων και τα παρουσιάζουν.

Φτιάχνουμε τα user1.html και user2.html

user1.html

```
<html>
  <body>
    <form action = "http://localhost:5001/" method = "POST" target="_self">
      <p>Name <input type = "text" name = "Name" /></p>
      <p>Message <input type = "text" name = "Message" /></p>

      <p><input type = "submit" value = "submit" /></p>
    </form>
  </body>
</html>
```

user2.html

```
<html>
  <body>
    <form action = "http://localhost:5000/" method = "POST" target="hiddenFrame">
      <p>Name <input type = "text" name = "Name" /></p>
      <p>Message <input type = "text" name = "Message" /></p>

      <p><input type = "submit" value = "submit" /></p>
    </form>
  </body>
</html>
```

Αυτά περιέχουν μια φόρμα για να γράψει ο χρήστης το μήνυμα που θέλει να στείλει στον άλλο χρήστη.

Παρατηρούμε πως το post request το κάνουμε μέσα από την hmtl φόρμα.
(Στην εξαμηνιαία τα post request θα τα κάνουμε μέσα από το bakcend του κάθε node χρησιμοποιώντας το request module της python.)

Τρέχουμε python user1.py, python user2.py ανοίγουμε τους browser μςα στα 127.0.0.1:5000, 127.0.0.1:5001 και παίζουμε με την εφαρμογή μας.