Τμήμα Ψηφιακών Συστημάτων
Πανεπιστήμιο Πειραιώς

Πληροφοριακά Συστήματα - Εργαστήριο 4ο

Χρυσόστομος Συμβουλίδης, simvoul@unipi.gr
Jean-Didier Totow, totow@unipi.gr

Πίνακας περιεχομένων

MongoDB Docker container (Run & Start)

Κατεβάζουμε το Docker image της MongoDB και τρέχουμε το container του στη port 27012 του host με την εντολή:

```
(sudo) docker run -d -p 27017:27017 --name mongodb mongo
```

Ξεκινάμε το Docker container με την εντολή:

```
(sudo) docker start ID / name
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Chrissimvou> docker start mongodb
mongodb
```

Mongo Import για εισαγωγή object στη βάση δεδομένων από αρχείο

Στο terminal γράφουμε τη παρακάτω εντολή για να εισάγουμε το `students.json` στη collection `Students` της βάσης `InfoSys`:

1. Κάνουμε copy τα δεδομένα από τον host στο container:
   ```
   docker cp students.json mongodb:/students.json
   ```
2. Εκτελούμε την εντολή:
   ```
   docker exec -it mongodb mongoimport --db=InfoSys --collection=Students --file=students.json
   ```

```
2020-04-24T10:29:53.511+0000    connected to: localhost
2020-04-24T10:29:53.542+0000    imported 2 documents
```

Αν είναι αρχείο διαφορετικού τύπου γράφουμε το ανάλογο type στο argument `--type`:

- `json`, για json
- `csv`, για csv (Comma Separated Values)
- `tsv` , για tsv (Tab Separated Values)

PyMongo (1/2)

Δημιουργούμε ένα αντικείμενο mongoClient για να μπορούμε να επικοινωνούμε με τη MongoDB βάση:

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017')

db = client['DB_Name']
coll = db['Collection_name']
```

Ενημέρωση αντικειμένου:

- Update One / Update many: Βρίσκει ένα ή περισσότερα αντικείμενα και τα ενημερώνει:

```
coll.update_one({'email': 'name@mail.com'},
          {'$set': {'year_of_birth': 1993}})
coll.update_many({'email': 'name@mail.com'},
                 {'$set': {'year_of_birth': 1993}})
```

PyMongo (2/2)

Δημιουργούμε ένα αντικείμενο mongoClient για να μπορούμε να επικοινωνούμε με τη MongoDB βάση:

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017')

db = client['DB_Name']
coll = db['Collection_name']
```

Διαγραφή αντικειμένου:

- Delete One / Delete Many: Βρίσκει ένα ή περισσότερα αντικείμενα και τα διαγράφει:

```
coll.delete_one({'email': 'name@mail.com'})

coll.delete_many({'email': 'name@mail.com'})
```

HTTP methods (1/2)

POST:

- Χρησιμοποιείται για τη δημιουργία ή/και την ενημέρωση των δεδομένων.
- Μετά την ενημέρωση πληροφορίας με τη χρήση POST, πρέπει να καλείται η GET ώστε να επιβεβαιώνεται η σωστή ενημέρωση.
- Υλοποιεί τις λειτουργίες «C» (Create) και «U» (Update) του «CRUD».

PUT:

- Χρησιμοποιείται για τη μετατροπή ή/και εισαγωγή πληροφορίας στο server.
- Η διαφορά με το POST είναι ότι το PUT είναι σταθερό, δηλαδή ότι όσες φορές και να το καλέσουμε, το αποτέλεσμα πρέπει να παραμένει ίδιο.
- Μετά την ενημέρωση πληροφορίας με τη χρήση PUT, πρέπει να καλείται η GET και να επιστρέφει τα νέα δεδομένα.
- Υλοποιεί τη λειτουργία «U» (Update) του «CRUD».

HTTP methods (2/2)

PATCH:

- Χρησιμοποιείται μόνο για μερική μετατροπή δεδομένων στο server.
- Υλοποιεί τη λειτουργία «U» (Update) του «CRUD».

DELETE:

- Χρησιμοποιείται για τη διαγραφή δεδομένων από το server.
- Υλοποιεί την λειτουργία «D» (Delete) του «CRUD».

POST (CREATE)

```
from flask import Flask, request
import json

app = Flask(__name__)
people = [{'name':'Alice', 'mail':'alice@mail.com'},{'name':'Bob','mail':'bob@mail.com'}]
@app.route('/post-create', methods=['POST'])
def post_create(): # add new person to people with POST method
      if request.data:
            data = json.loads(request.data)
            people.append(data)
            return Response('data was added to people, status=200)
      else:
            return Response('data was not added', status=500)



if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

POST (UPDATE)

```
from flask import Flask, request, Response
import json

app = Flask(__name__)
people = [{'name':'Alice', 'mail':'alice@mail.com'},{'name':'Bob','mail':'bob@mail.com'}]
@app.route('/post-update/<string:mail>', methods=['POST'])
def post_update(mail): # find person by mail and update with POST method
      if request.data:
            data = json.loads(request.data)
            for i, person in enumerate(people):
                  if person['mail'] == mail:
                        people[i] = data
                        return Response('people was updated', status=200)
            return Response('people was not updated', status=200)
      else:
            return Response('people was not updated', status=500)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

PUT (UPDATE)

```
from flask import Flask, request, Response
import json

app = Flask(__name__)
people = [{'name':'Alice', 'mail':'alice@mail.com'},{'name':'Bob','mail':'bob@mail.com'}]
@app.route('/put-update/<string:mail>', methods=['PUT'])
def put_update(mail): # find person by mail and update with PUT method
      if request.data:
            data = json.loads(request.data)
            for i, person in enumerate(people):
                  if person['mail'] == mail:
                        people[i] = data
                        return Response('people was updated', status=200)
            return Response('people was not updated', status=200)
      else:
            return Response('people was not updated', status=500)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Άσκηση 1

Να υλοποιηθεί το παρακάτω Flask service:

1. Να γίνεται σύνδεση με μία MongoDB η οποία θα λειτουργεί σε ένα container τοπικά στον υπολογιστή σας
2. Στο collection Courses της βάσης δεδομένων InfoSys να υλοποιηθούν τα παρακάτω endpoint:
   a. [POST] `/insert-course`: το οποίο θα χρησιμοποιείται για να γίνεται εισαγωγή μαθημάτων. Στο body του μηνύματος από το client θα υπάρχει json με τα εξής fields: `_id` (string), `name` (string), `ects` (int)
   b. [GET] `/get-course/<string:id>`: το οποίο θα χρησιμοποιείται για να επιστρέφει τα στοιχεία του συγκεκριμένου μαθήματος
   c. [PUT] `/add-course/<string:student_email>`: το οποίο θα παίρνει ως παράμετρο το `student_email` (Students:email) και θα εισάγει στο key (array) courses του φοιτητή το μάθημα με το id που θα δίνεται στο body του μηνύματος στο πεδίο `course_id`.

PATCH (UPDATE)

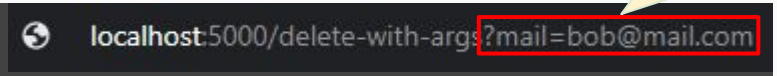
```
from flask import Flask, request, Response
import json

app = Flask(__name__)
people = [{'name':'Alice', 'mail':'alice@mail.com'},{'name':'Bob','mail':'bob@mail.com'}]
@app.route('/patch-update/<string:mail>', methods=['PATCH'])
def patch_update(mail): # find person by mail and partially update with PATCH method
      if request.data:
            data = json.loads(request.data)
            for i, person in enumerate(people):
                  if person['mail'] == mail:
                        people[i]['mail'] = data
                        return Response('people was partially updated', status=200)
            return Response('people was not updated', status=200)
      else:
            return Response('people was not updated', status=500)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

DELETE (DELETE)

```
from flask import Flask, request, Response
import json

app = Flask(__name__)
people = [{'name':'Alice', 'mail':'alice@mail.com'},{'name':'Bob','mail':'bob@mail.com'}]
@app.route('/delete/<string:mail>', methods=['DELETE'])
def delete(mail): # find person by mail and remove from people with DELETE method
      for i, person in enumerate(people):
            if person['mail'] == mail:
                  del people[i]
                  return Response('the person with that mail was removed', status=200)

      # status=404 because the resource was not found
      return Response('people was not removed', status=404)



if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Flask Arguments

Τα arguments περνάνε με τη χρήση του συμβόλου «?» στο URL




```
from flask import Flask, request, Response
import json

app = Flask(__name__)
people = [{'name':'Alice', 'mail':'alice@mail.com'},{'name':'Bob','mail':'bob@mail.com'}]
@app.route('/delete-with-args', methods=['DELETE'])
def delete_with_args(): # find person by mail and remove from people with DELETE method
      mail = request.args.get('mail')
      for i, person in enumerate(people):
            if person['mail'] == mail:
                  del people[i]
                  return Response('the person with that mail was removed', status=200)

      # status=404 because the resource was not found
      return Response('people was not removed', status=404)


if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Άσκηση 2

Να υλοποιηθεί το παρακάτω Flask service:

1. Να γίνεται σύνδεση με μία MongoDB η οποία θα λειτουργεί σε ένα container τοπικά στον υπολογιστή σας
2. Στο collection Courses της βάσης δεδομένων InfoSys να υλοποιηθούν τα παρακάτω endpoint:
   a. [DELETE] `/delete-student`: το οποίο θα παίρνει ως argument το email ενός φοιτητή και θα τον διαγράφει από το σύστημα
   b. [POST] `/insert-course-description/<string:course_id>`: το οποίο θα χρησιμοποιείται για να εισάγει τη περιγραφή ενός μαθήματος σε ένα νέο field που θα ονομάζεται `description`
   c. [PUT] `/update-course/<string:course_id>`: το οποίο θα παίρνει ως παράμετρο το `course_id` και θα ενημερώνει όλα τα πεδία (`name`, `ects`, `description`) ενός μαθήματος (εκτός του `_id`)

Docker compose (1/3)

- Το Docker compose είναι ένα εργαλείο για να ορίσουμε και τρέξουμε πολλα containers μαζί.
- Οι ορισμοί των containers γίνονται σε ένα αρχείο .yaml
- Λειτουργικότητες
  - Πολλαπλά απομονωμένα περιβάλλοντα σε έναν Host
  - Διατηρήση volumes
  - Δημιουργία μόνο των containers που έχουν αλλάξει
  - Μεταβλητές και μετακίνηση μιας σύνθεσης μεταξύ περιβαλλόντων

Docker compose (2/3)

- Το docker-compose υπάρχει ήδη στα Docker για Windows και Mac.
- Εγκατάσταση σε Linux:

Η εντολή σε μία γραμμή!

1. `sudo curl -L https://github.com/docker/compose/releases/download/1.21.2/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose`
2. `sudo chmod +x /usr/local/bin/docker-compose`

Docker compose (3/3)

Παράδειγμα ενός Docker-compose.yml:

```
version: '2'
services:
 mongodb:
   image: mongo
   restart: always
   container_name: mongodb
   ports:
   - 27017:27017
   volumes:
   - /home/user/mongodb/data:/data/db
 flask-service:
   image: my_flash_image
   restart: always
   container_name: flash
   ports:
   - 5000:5000
```

Εντολες

- `docker-compose up -d` // Εκτέλεση
- `docker-compose down` // Τερματισμός όλων υπηρεσιών και διαγραφή των container

Προαιρετική Εργασία

Να υλοποιηθεί το παρακάτω Flask service:

1. Να γίνεται σύνδεση με μία MongoDB η οποία θα λειτουργεί σε ένα container τοπικά στον υπολογιστή σας
2. Θα δημιουργηθεί ένα νέο collection που θα ονομαστεί Courses και θα περιέχει τα παρακάτω keys: `name`, `course_id`, `ects`
3. Να υλοποιηθούν τα παρακάτω endpoint:
    1. [POST] `/insert-course` : το οποίο θα περιμένει από το χρήστη ένα json με τα παραπάνω key και θα το κάνει εισαγωγή στο collection Courses. Εδώ να υπάρχει έλεγχος ώστε να μην μπορεί να προστεθεί δεύτερο course στο collection με το ίδιο `course_id`
    2. [GET] `/get-course` : το οποίο θα δέχεται ως argument το course_id και θα επιστρέφει τα στοιχεία του μαθήματος αυτού
    3. [PUT] `/add-course/<string:email>` : το οποίο θα εισάγει στο key (array) courses του φοιτητή το μάθημα με το id που θα δίνεται στο body του μηνύματος στο πεδίο `course_id`
    4. [DELETE] `/delete-student` : το οποίο θα παίρνει ως argument το email ενός φοιτητή και θα τον διαγράφει από το σύστημα
    5. [POST] `/insert-course-description` : το οποίο θα με χρήση argument το course_id και θα χρησιμοποιείται για να εισάγει τη περιγραφή ενός μαθήματος σε ένα νέο field που θα ονομάζεται description
    6. [PUT] `/update-course` : το οποίο θα παίρνει ως argument το `course_id` και θα ενημερώνει όλα τα πεδία (`course_id`, `name`, `ects`, `description`) ενός μαθήματος