МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

Лабораторна робота № 1

з курсу «Основи хмарних обчислень»

Тема: «Реєстрація та налаштування сервісу хмарних обчислень»

Виконав:

студент I курсу

групи ДА-71мн

Колінько Анжела

Київ – 2017

Варіант 2

Мета: ознайомитися на практиці з засобами реєстрації та налаштування сервісу хмарних обчислень Google Application Engine.

1. Описати роботу з механізмом реєстрації та налаштуванням хмарного середовища.

Якщо натискати “TRY IT FREE”, то отримаємо запит на введення карти. Аби не вводити ці дані можна просо пройти в розділ Documentation, обрати мову програмування та слідувати згідно документації.

2. Описати процес встановлення App Engine SDK.

Згідно документаціЇ:

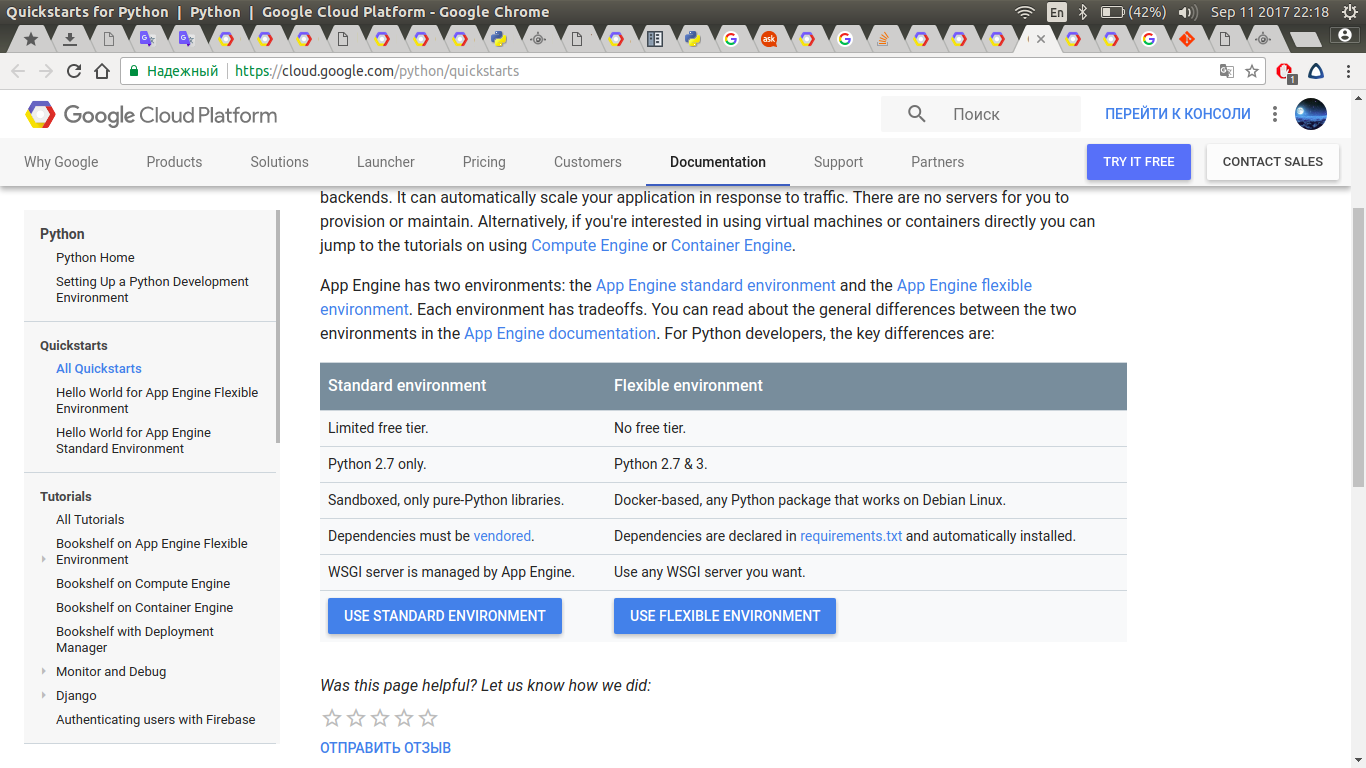
Install Python version 2.7:

Install and initialize Google Cloud SDK

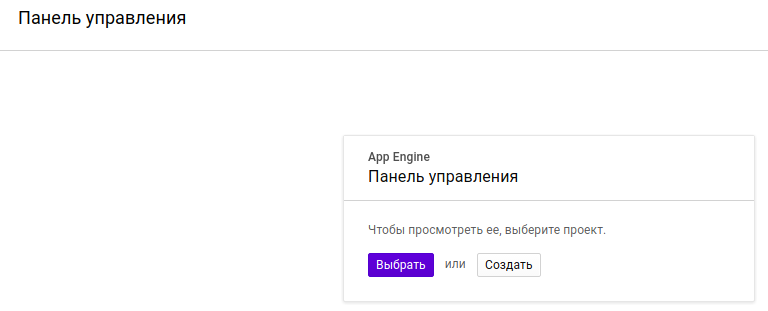
Run the following command to install the gcloud component that includes the App Engine extension for Python: gcloud components install app-engine-python

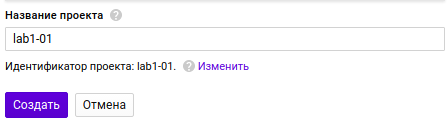
Download and install Git for access to code, samples, libraries, and tools in GitHub.

Проте на багатьох дистрибутивах Ubuntu вже встановлено Python 2.\* (важливо аби була встановлена саме ця версія для standart environment (при виборі flexible environment отримуємо запит на введення даних карти))

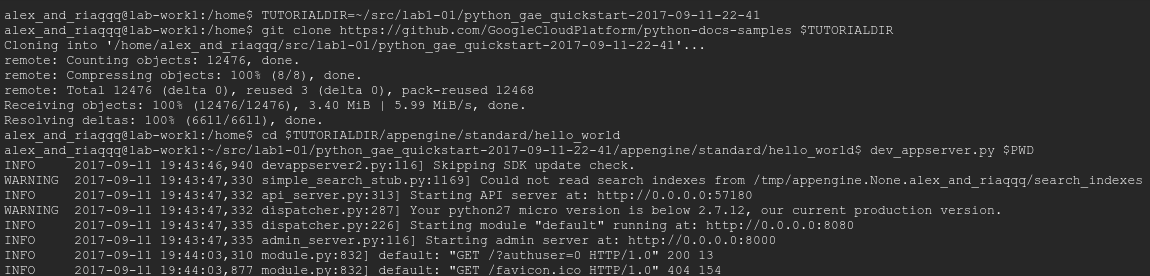


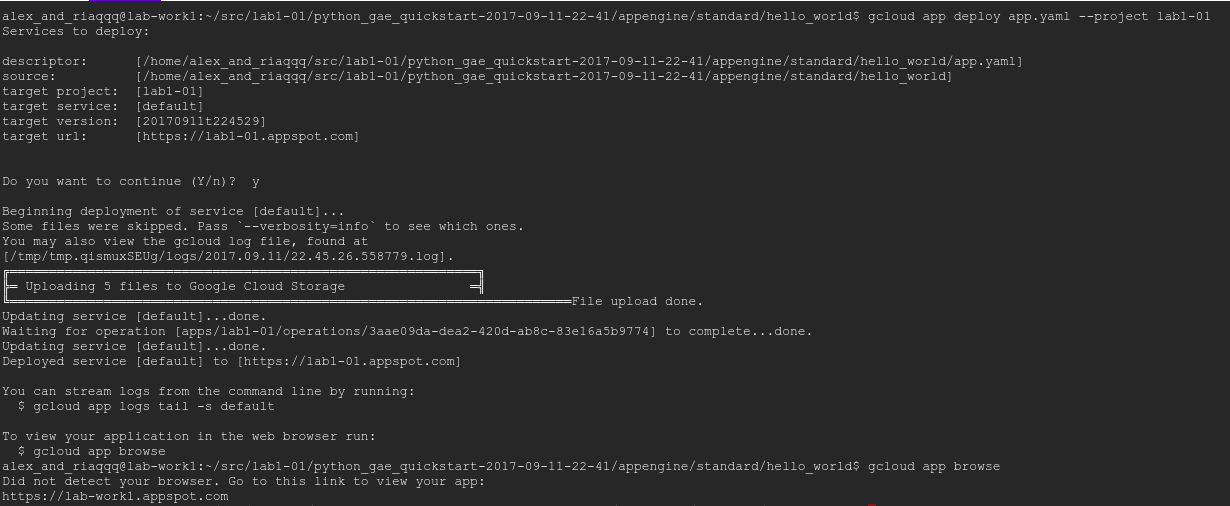
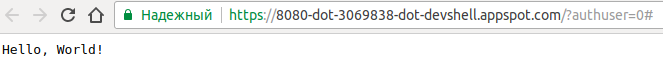
3. Ознайомитися з роботою консолі адміністрування хмарного додатку.



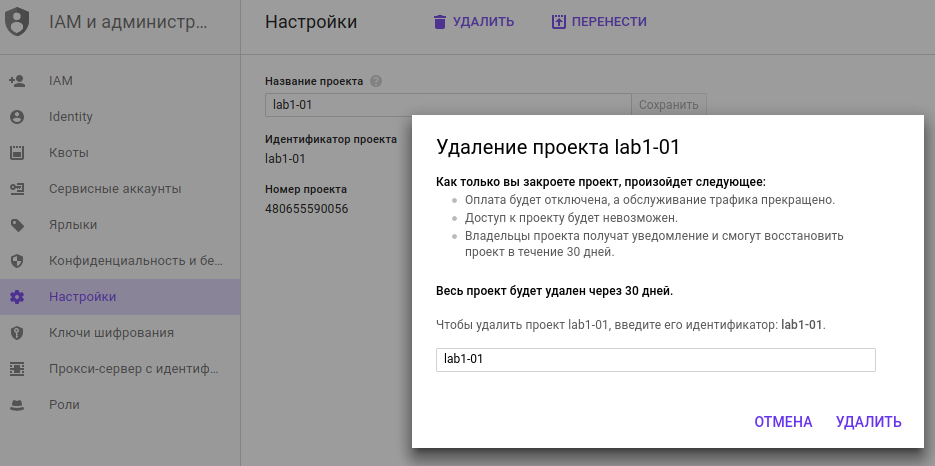












Висновки: на лабораторній роботі було розглянуто засоби реєстрації та налаштування сервісу хмарних обчислень Google Application Engine, підготовлено середовище розробки для виконання наступних лабораторних робіт.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

Лабораторна робота № 2

з курсу «Основи хмарних обчислень»

Тема: «Розробка коду програми для роботи у хмарному середовищі»

Виконав:

студент I курсу

групи ДА-71мн

Колінько Анжела

Київ – 2017

Варіант 2

Мета: набути практичних навичок розробки архітектури та коду програми в хмарному середовищі.

1. Хмарна інформаційна система обліку книжок в домашній бібліотеці.

2. Додаток складається з наступних файлів

1) index.html – html файл зрозміткою. В даному файлі також використовуються Django template language

2) stylesheets/main.css – відповідний .css файл

3) lab2.py – .py файл з логікою додатку

4) app.yaml – файл кофігурації

4) index.yaml – автоматично згенерований файл datastore indexes.

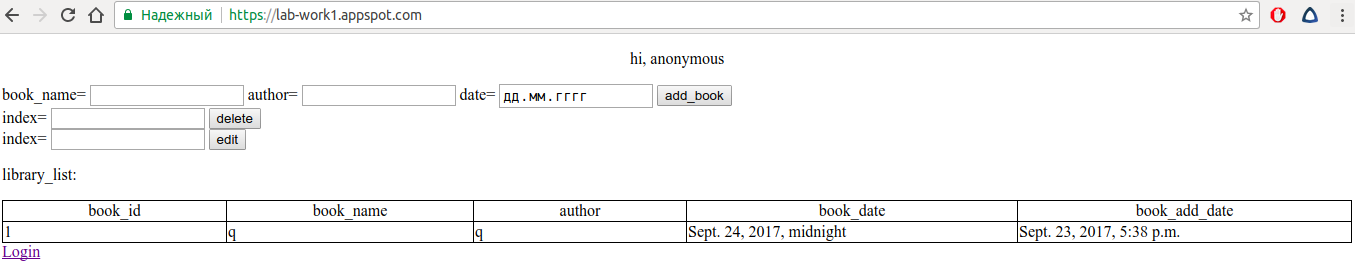
3. Для деплойменту були використані наступні команди:

1) gcloud init – для (ре)ініціалізації gcloud в даній директорії

2) gcloud app deploy app.yaml index.yaml – деплоймент

3) gcloud datastore create-indexes index.yaml – для завантаження індексів запитів до бази даних.

4.Зовнішній вигляд додатку:



Висновки: на лабораторній роботі розглянуто процес розробки архітектури та коду програми в хмарному середовищі.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

Лабораторна робота № 3

з курсу «Основи хмарних обчислень»

Тема: «Дослідження і практична робота з квотами хмарних обчислень»

Виконав:

студент I курсу

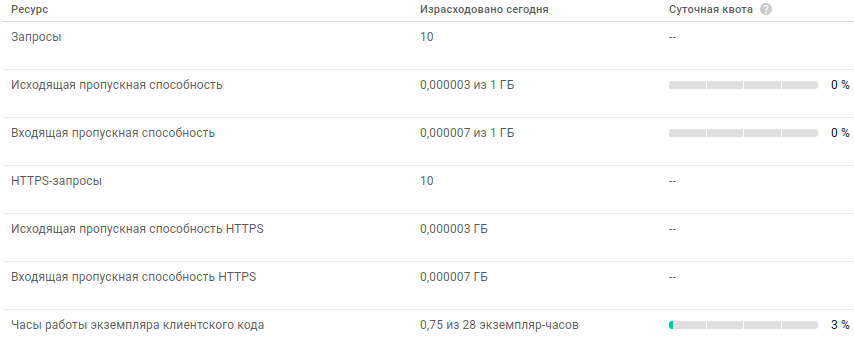
групи ДА-71мн

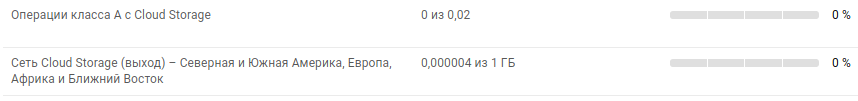
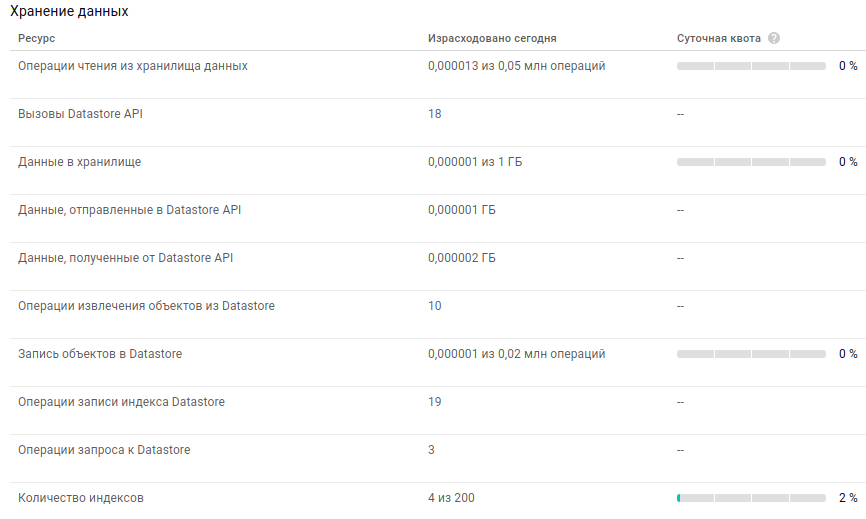
Колінько Анжела

Київ – 2017

Мета роботи: вивчити систему квот хмарних обчислень та виконати конфігурацію додатків у хмарному сервісі Google Application Engine.

1. Були використані наступні ресурси:





Висновки: на лабораторній роботі було розглянуто систему квот хмарних обчислень.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

Лабораторна робота № 4

з курсу «Основи хмарних обчислень»

Тема: «Дослідження і практична робота з квотами хмарних обчислень»

Виконав:

студент I курсу

групи ДА-71мн

Колінько Анжела

Київ – 2017

Мета роботи: набути практичних навичок використання та налаштування служби збереження даних у хмарному середовищі.

1. Для збереження даних використовувалась база даних google.appengine – HRD (High Replication datastore):

from google.appengine.ext import db – підключення

class Model(db.Model):

"""Models an individual entry with fields"""

id\_b=db.IntegerProperty()

name\_b=db.StringProperty(multiline=True)

author\_b = db.StringProperty(multiline=True)

date\_b=db.DateTimeProperty()

date\_b\_add=db.DateTimeProperty(auto\_now\_add=True)

клас моделі бази даних. id\_b – числовий ідентифікатор книги; name\_b – назва книги; (multiline=True – дозволяє зберігати дані з символом ‘\n’) author\_b – автор книги; date\_b – дата випуску книги; date\_b\_add – дата додання/оновлення запису (книги).

В якості parent key для entity group використовується user\_name:

def lib\_key(user\_name=None):

"""Constructs a datastore key for a Lib entity with user\_name"""

return db.Key.from\_path('Lib', user\_name)

що встановлюється в users.get\_current\_user().nickname(), якщо коритувач зареєстрований в своєму аккаунті google, чи, інакше, в 'anonymous':

from google.appengine.api import users

if users.get\_current\_user():

url = users.create\_logout\_url(self.request.uri)

url\_linktext = 'Logout'

user=users.get\_current\_user().nickname()

username=user

else:

url = users.create\_login\_url(self.request.uri)

url\_linktext = 'Login'

user='anonymous'

username=user

2. Були здійснені наступні запити:

books\_query = Model.all().ancestor(

lib\_key(username)).order('id\_b')

books = books\_query.fetch(10)

отримання перших 10 книг відсортованих по id\_b.

books\_query.filter('id\_b =',edit\_i)

book\_u\_e=books\_query.get()

'''for edited\_book\_ent in books\_query:

print edited\_book\_ent.name\_b'''

if book\_u\_e.name\_b!=edited\_name\_b:

book\_u\_e.name\_b=edited\_name\_b

if book\_u\_e.author\_b!=edited\_author\_b:

book\_u\_e.author\_b=edited\_author\_b

if edited\_date\_b!=u'':

print 'edited\_date\_b={}'.format(edited\_date\_b)

book\_u\_e.date\_b=datetime.strptime(edited\_date\_b, "%Y-%m-%d")

book\_u\_e.put()

редагуваннязапису в базі даних

books\_query.filter('id\_b =',delete\_i)

books\_query.get().delete()

видалення елементу з бази даних

Висновки: на лабораторній роботі було розглянуто використання та налаштування служби збереження даних у хмарному середовищі.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

Лабораторна робота № 5

з курсу «Основи хмарних обчислень»

Тема: «Дослідження і практична робота з квотами хмарних обчислень»

Виконав:

студент I курсу

групи ДА-71мн

Колінько Анжела

Київ – 2017

Мета: створити простий HTTP API для хмарного додатку.

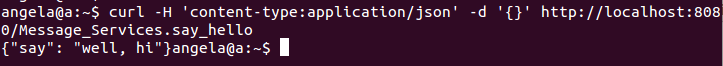
1. Опис методів API:

Реалізовано 4 методи:

1) @remote.method(message\_types.VoidMessage,Say)

def say\_hello(self,request)

Це тестовий метод. В якості вхідних даних він отримує пусте повідомлення, повертає в якості повідомлення об’єкт класу Say. В даному випадку Say(say="well, hi"). Маємо:

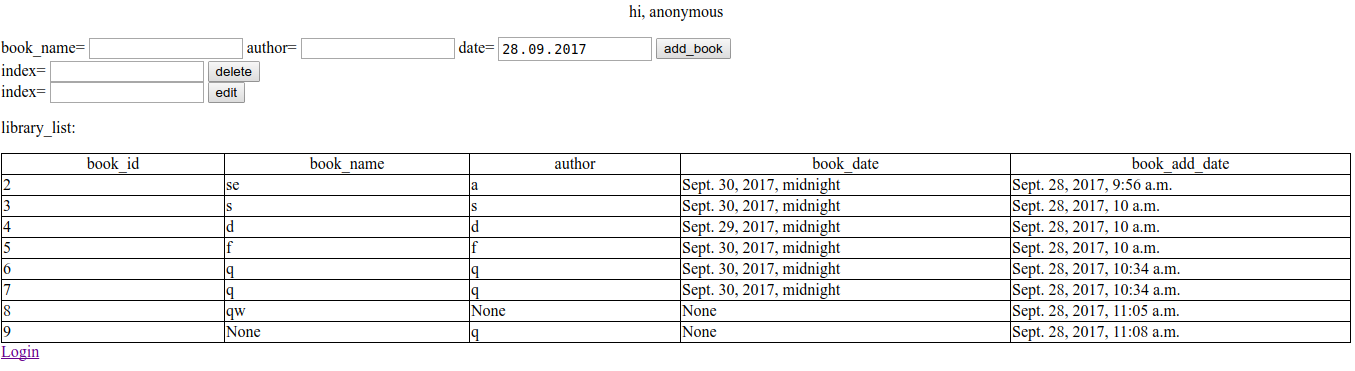


2) @remote.method(List\_books,Books)

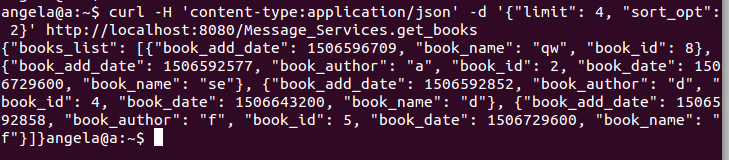
def get\_books(self,request)

Даний метод відображає вміст бази даних (в даному випадку – бібліотеки). Можна задати кількість записів через limit (за замовчуванням limit==default\_limit==10) та/або спосіб сортування через sort\_opt (за замовчуванням – сортування по id).

Нехай база даних має наступний вміст:



Виберемо перші 4 рядки з відсортованої по авторам таблиці:



3) @remote.method(Book\_ent,message\_types.VoidMessage)

def add\_book(self,request)

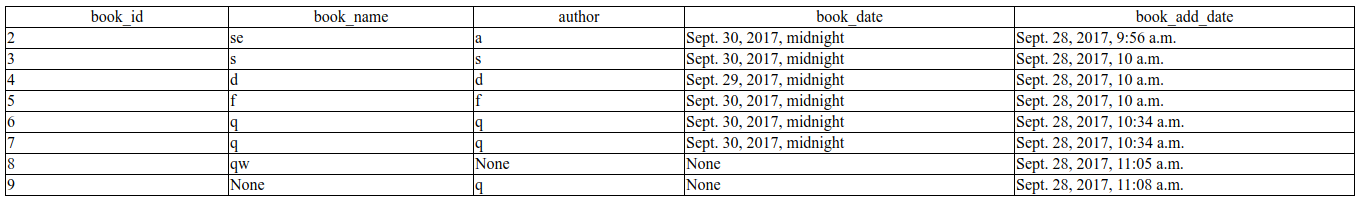
Даний метод додає книгу до бази даних. Хоча в описі класу Book\_ent (клас, об’єкт якого виступає в якісті вхідних даних) не вказано жодного обов’язкового поля, книга буде додана лише за наявності автора або назви:

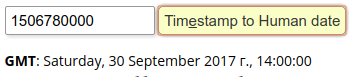
if n\_book.name\_b or n\_book.author\_b:

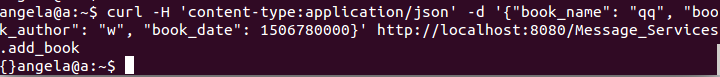
//…

n\_book.put()

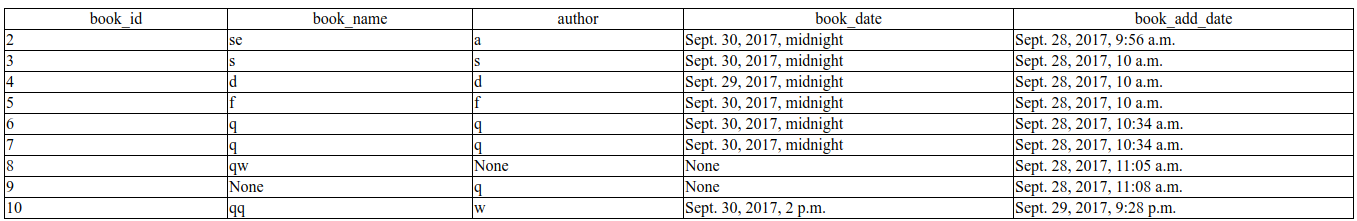
Нехай база даних має наступний вміст:



Додамо книгу з назвою qq, автором w та датою випуску 30.09.2017 ():



База даних змінилася:



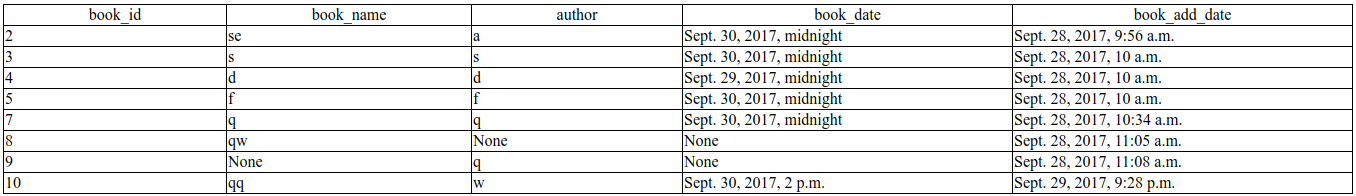
4) @remote.method(Book\_id,message\_types.VoidMessage)

def del\_book(self,request)

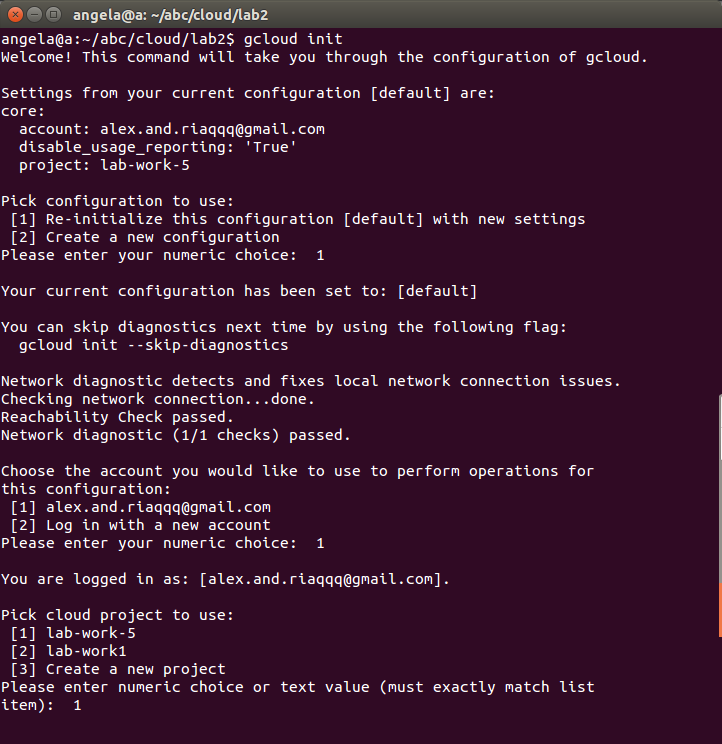
Даний метод видаляє книгу за її ідентифікатором – єдиним і обов’язковим полем класу Book\_id.

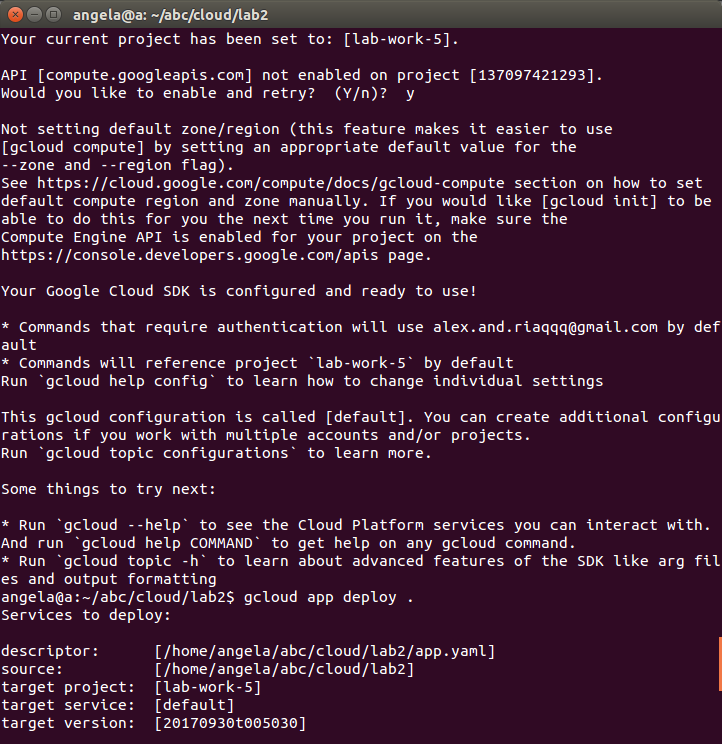
Видалимо книгу з 6 ідентифікатором:

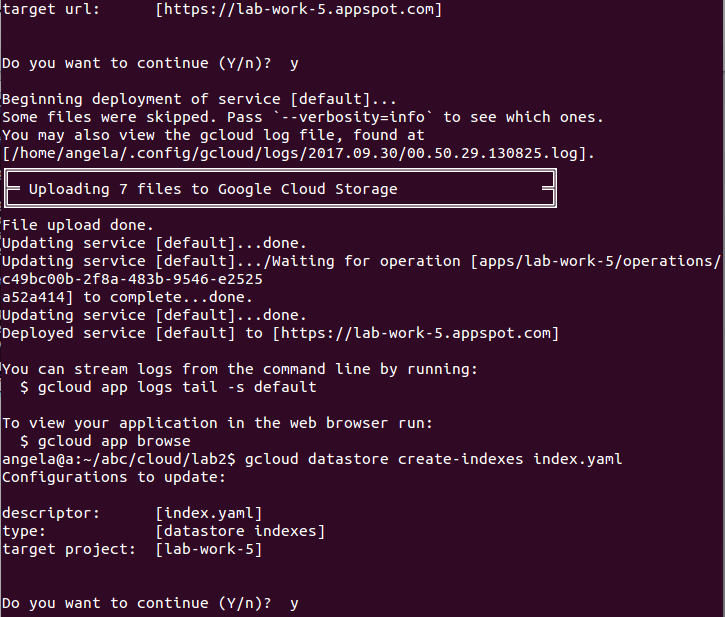


****

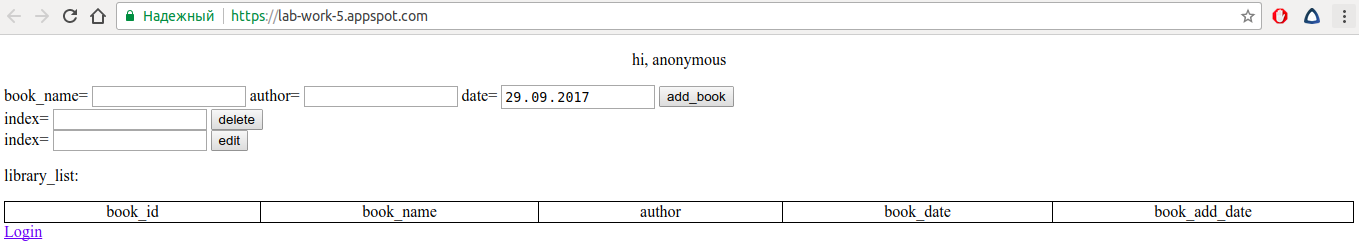
2. Процес депломенту:





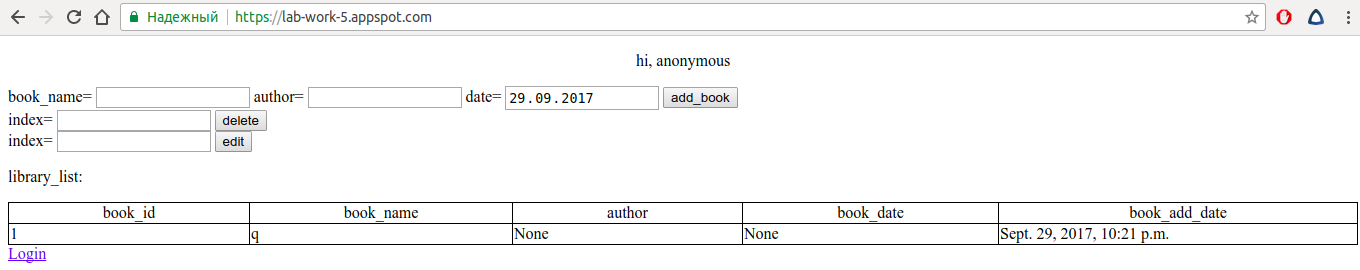


Маємо:



Додамо книгу з назвою q:





Висновки: на лабораторній роботі було створено API для хмарного додатку.

Додаток 1. lab2.py

import cgi

import datetime

import urllib

import wsgiref.handlers

from datetime import datetime

from google.appengine.ext import db

from google.appengine.api import users

import webapp2

import os

from google.appengine.ext.webapp import template

books\_num=0

username='anonymous'

edit\_i=-1

delete\_i=-1

edited=False

editing=False

deleted=False

edited\_name\_b=''

edited\_author\_b=''

edited\_date\_b=u''

def inc\_globvar():

global books\_num # Needed to modify global copy of globvar

books\_num +=1

class Model(db.Model):

"""Models an individual entry with fields"""

id\_b=db.IntegerProperty()

name\_b=db.StringProperty(multiline=True)

author\_b = db.StringProperty(multiline=True)

date\_b=db.DateTimeProperty()

date\_b\_add=db.DateTimeProperty(auto\_now\_add=True)

def lib\_key(user\_name=None):

"""Constructs a datastore key for a Lib entity with user\_name"""

return db.Key.from\_path('Lib', user\_name)

class MainPage(webapp2.RequestHandler):

def get(self):

global username

global books\_num

global edit\_i

global delete\_i

global edited

global editing

global deleted

global edited\_name\_b

global edited\_author\_b

global edited\_date\_b

if users.get\_current\_user():

url = users.create\_logout\_url(self.request.uri)

url\_linktext = 'Logout'

user=users.get\_current\_user().nickname()

username=user

else:

url = users.create\_login\_url(self.request.uri)

url\_linktext = 'Login'

user='anonymous'

username=user

books\_query = Model.all().ancestor(

lib\_key(username)).order('id\_b')

books = books\_query.fetch(10)

tmp\_id=0

for c\_book in books:

if tmp\_id<c\_book.id\_b:

tmp\_id=c\_book.id\_b

books\_num=tmp\_id

auto\_redirect=False

if edited==True:

books\_query.filter('id\_b =',edit\_i)

book\_u\_e=books\_query.get()

'''for edited\_book\_ent in books\_query:

print edited\_book\_ent.name\_b'''

if book\_u\_e.name\_b!=edited\_name\_b:

book\_u\_e.name\_b=edited\_name\_b

if book\_u\_e.author\_b!=edited\_author\_b:

book\_u\_e.author\_b=edited\_author\_b

if edited\_date\_b!=u'':

print 'edited\_date\_b={}'.format(edited\_date\_b)

book\_u\_e.date\_b=datetime.strptime(edited\_date\_b, "%Y-%m-%d")

print '1edited\_name\_b={}'.format(edited\_date\_b)

edited\_name\_b=''

edited\_author\_b=''

edited\_date\_b=u''

edited=False

auto\_redirect=True

edit\_i=-1

book\_u\_e.put()

if deleted==True:

books\_query.filter('id\_b =',delete\_i)

books\_query.get().delete()

deleted=False

delete\_i=-1

auto\_redirect=True

unicode\_date=u''

if editing==True:

books\_query.filter('id\_b =',edit\_i)

unicode\_date=unicode(books\_query.get().date\_b)

print 'unicode\_date={} (type()={})'.format(unicode\_date,type(unicode\_date))

editing=False

template\_values = {

'user': user,

'url': url,

'url\_linktext': url\_linktext,

'books': books,

'unicode\_date': unicode\_date[0:10],

'edit\_i': edit\_i

}

path = os.path.join(os.path.dirname(\_\_file\_\_), 'index.html')

self.response.out.write(template.render(path, template\_values))

if auto\_redirect:

self.redirect('/')

class Add\_book(webapp2.RequestHandler):

def post(self):

global books\_num

inc\_globvar()

n\_book = Model(parent=lib\_key(username))

n\_book.id\_b=books\_num

n\_book.name\_b=self.request.get('n\_book\_name')

n\_book.author\_b=self.request.get('n\_book\_author')

if self.request.get('n\_book\_date'):

n\_book.date\_b=datetime.strptime(self.request.get('n\_book\_date'), "%Y-%m-%d")

print 'type(n\_book\_date={}), type(n\_book.date\_b)={}'.format(type(self.request.get('n\_book\_date')),type(n\_book.date\_b))

if n\_book.name\_b or n\_book.author\_b:

n\_book.put()

else:

books\_num-=1

self.redirect('/')

class Edit\_book(webapp2.RequestHandler):

def post(self):

global books\_num

global edit\_i

global editing

'''print 'edit\_i={},type={}'.format(edit\_i,type(edit\_i))'''

edit\_i=int(self.request.get('e\_index'))

'''print 'edit\_i={},type={}'.format(edit\_i,type(edit\_i))'''

if edit\_i>=1 and edit\_i<=books\_num:

editing=True

else:

editing=False

edit\_i=-1

self.redirect('/')

class Edit\_ok(webapp2.RequestHandler):

def post(self):

global edited

global edited\_name\_b

global edited\_author\_b

global edited\_date\_b

edited\_name\_b=self.request.get('u\_book\_name')

edited\_author\_b=self.request.get('u\_book\_author')

edited\_date\_b=self.request.get('u\_book\_date')

edited=True

print 'q:{} {} {}'.format(edited\_name\_b,edited\_author\_b,edited\_date\_b)

if edited\_date\_b==u'':

print 'unicode '

self.redirect('/')

class Delete\_book(webapp2.RequestHandler):

def post(self):

global books\_num

global delete\_i

global deleted

delete\_i=int(self.request.get('d\_index'))

if delete\_i>=1 and delete\_i<=books\_num:

deleted=True

else:

deleted=False

delete\_i=-1

self.redirect('/')

app = webapp2.WSGIApplication([

('/', MainPage),

('/add\_book', Add\_book),

('/edit\_book',Edit\_book),

('/edit\_ok',Edit\_ok),

('/delete\_book',Delete\_book)

], debug=True)

Додаток 2. index.html

<!DOCTYPE html>

<html>

<head>

<title>lab2</title>

<link type="text/css" rel="stylesheet" href="/stylesheets/main.css" />

</head>

<body>

<!--

{% for greeting in greetings %}

{% if greeting.author %}

<b>{{ greeting.author.nickname }}</b> wrote:

{% else %}

An anonymous person wrote:

{% endif %}

<blockquote>{{ greeting.content|escape }}</blockquote>

{% endfor %}

-->

<p id='hi\_m'>hi, {{user}}</p>

<form action="/add\_book" method="post">

book\_name= <input type="text" name="n\_book\_name">

author= <input type="text" name="n\_book\_author">

date= <input type="date" name="n\_book\_date">

<input type="submit" value="add\_book">

</form>

<form action="/delete\_book" method="post">

index= <input type="number" name=d\_index>

<input type="submit" value="delete">

</form>

<form action="/edit\_book" method="post">

index= <input type="number" name=e\_index>

<input type="submit" value="edit">

</form>

<!--<p>{{unicode\_date}}={{edit\_i}}</p>-->

<p>library\_list:</p>

<table id='full\_t'>

<tr id='tible\_t'><td>book\_id</td><td>book\_name</td><td>author</td><td>book\_date</td><td>book\_add\_date</td></tr>

{%for book in books %}

{%if book.id\_b == edit\_i%}

<form action="/edit\_ok" method="post">

<tr><td>{{book.id\_b}}</td>

<td>book\_name= <input type="text" name="u\_book\_name" value="{{book.name\_b}}"></td>

<td>author= <input type="text" name="u\_book\_author" value="{{book.author\_b}}"></td>

<td>date= <input type="date" name="u\_book\_date" value="{{unicode\_date}}"></td>

<td><input type="submit" value="ok"></td>

</tr>

</form>

{%else%}

<tr><td>{{book.id\_b}}</td><td>{{book.name\_b}}</td><td>{{book.author\_b}}</td><td>{{book.date\_b}}</td><td>{{book.date\_b\_add}}</td></tr>

{%endif%}

{% endfor %}

</table>

<a href="{{ url }}">{{ url\_linktext }}</a>

</body>

</html>

Додаток 3. app.yaml

runtime: python27

api\_version: 1

threadsafe: true

handlers:

- url: /stylesheets

static\_dir: stylesheets

- url: /.\*

script: lab2.app

Додаток 4. stylesheets/main.css

#hi\_m{

text-align: center;

}

#full\_t{

max-width:100%;

white-space:nowrap;

}

#tible\_t{

text-align: center;

}

table {

border-collapse: collapse;

}

table, th, td {

border: 1px solid black;

}

td{

border:1px solid #000;

width:1%;

white-space:nowrap;

}

Додаток 5. index.yaml

indexes:

# AUTOGENERATED

# This index.yaml is automatically updated whenever the dev\_appserver

# detects that a new type of query is run. If you want to manage the

# index.yaml file manually, remove the above marker line (the line

# saying "# AUTOGENERATED"). If you want to manage some indexes

# manually, move them above the marker line. The index.yaml file is

# automatically uploaded to the admin console when you next deploy

# your application using appcfg.py.

- kind: Model

ancestor: yes

properties:

- name: date\_b

- kind: Model

ancestor: yes

properties:

- name: date\_b

direction: desc

- kind: Model

ancestor: yes

properties:

- name: id\_b

- kind: Model

ancestor: yes

properties:

- name: id\_b

direction: desc

Додаток 6. lab5.py

import time

import datetime

from protorpc import messages

from protorpc import message\_types

from protorpc import remote

import lab2

default\_limit=10

class Say(messages.Message):

say=messages.StringField(1)

class Book\_view(messages.Message):

book\_id=messages.IntegerField(1)

book\_name=messages.StringField(2)

book\_author=messages.StringField(3)

book\_date=messages.IntegerField(4)

book\_add\_date=messages.IntegerField(5)

class Book\_ent(messages.Message):

book\_name=messages.StringField(1)

book\_author=messages.StringField(2)

book\_date=messages.IntegerField(3)

class Book\_id(messages.Message):

id\_b=messages.IntegerField(1,required=True)

class List\_books(messages.Message):

global default\_limit

limit=messages.IntegerField(1, default=default\_limit)

class Sort\_by(messages.Enum):

ID=0

NAME=1

AUTHOR=2

DATE\_B=3

DATE\_B\_ADD=4

sort\_opt=messages.EnumField(Sort\_by,2,default=Sort\_by.ID)

class Books(messages.Message):

books\_list=messages.MessageField(Book\_view,1,repeated=True)

class Message\_Services(remote.Service):

@remote.method(message\_types.VoidMessage,Say)

def say\_hello(self,request):

if request:

return Say(say="well, hi")

'''else:

return Say(say="hello");'''

@remote.method(List\_books,Books)

def get\_books(self,request):

query=lab2.Model.all().ancestor(lab2.lib\_key(lab2.username))

'''query.order('id\_b')

tst\_books=query.fetch(10)

for tst\_book in tst\_books:

print 'book.id\_b={}'.format(tst\_book.id\_b)'''

global default\_limit

limit=default\_limit

if request.limit:

limit=request.limit

if request.sort\_opt:

if request.sort\_opt==List\_books.Sort\_by.ID:

query.order('id\_b')

elif request.sort\_opt==List\_books.Sort\_by.NAME:

query.order('name\_b')

elif request.sort\_opt==List\_books.Sort\_by.AUTHOR:

query.order('author\_b')

elif request.sort\_opt==List\_books.Sort\_by.DATE\_B:

query.order('date\_b')

else: #request.sort\_opt==List\_books.Sort\_by.DATE\_B\_ADD;

query.order('date\_b\_add')

books\_list=[]

for fnd\_book in query.fetch(limit):

if fnd\_book.date\_b:

tmp\_date=int(time.mktime(fnd\_book.date\_b.utctimetuple()))

else:

tmp\_date=None

if fnd\_book.date\_b\_add:

tmp\_date\_add=int(time.mktime(fnd\_book.date\_b\_add.utctimetuple()))

else:

tmp\_date\_add=None

book\_ent=Book\_view(

book\_id=fnd\_book.id\_b, book\_name=fnd\_book.name\_b,book\_author=fnd\_book.author\_b,book\_date=tmp\_date,book\_add\_date=tmp\_date\_add)

books\_list.append(book\_ent)

return Books(books\_list=books\_list)

@remote.method(Book\_ent,message\_types.VoidMessage)

def add\_book(self,request):

'''global lab2.books\_num'''

print 'blab2.books\_num={}'.format(lab2.books\_num)

print 'lab2.username={}'.format(lab2.username)

n\_book = lab2.Model(parent=lab2.lib\_key(lab2.username))

n\_book.id\_b=lab2.books\_num

if request.book\_name is not None:

n\_book.name\_b=request.book\_name

if request.book\_author is not None:

n\_book.author\_b=request.book\_author

if request.book\_date is not None:

n\_book.date\_b=datetime.datetime.utcfromtimestamp(request.book\_date)

if n\_book.name\_b or n\_book.author\_b:

global default\_limit

query=lab2.Model.all().ancestor(lab2.lib\_key(lab2.username))

tmp\_books = query.fetch(default\_limit)

tmp\_id=0

for c\_book in tmp\_books:

if tmp\_id<c\_book.id\_b:

tmp\_id=c\_book.id\_b

lab2.books\_num=tmp\_id

print 'alab2.books\_num={}'.format(lab2.books\_num)

lab2.books\_num+=1

n\_book.id\_b=lab2.books\_num

n\_book.date\_b\_add=datetime.datetime.now()

n\_book.put()

else:

print 'q\nq\n'

return message\_types.VoidMessage()

@remote.method(Book\_id,message\_types.VoidMessage)

def del\_book(self,request):

global default\_limit

books\_query = lab2.Model.all().ancestor(lab2.lib\_key(lab2.username)).order('id\_b')

tmp\_books = books\_query.fetch(default\_limit)

tmp\_id=0

for c\_book in tmp\_books:

if tmp\_id<c\_book.id\_b:

tmp\_id=c\_book.id\_b

lab2.books\_num=tmp\_id

if request.id\_b>=1 and request.id\_b<=lab2.books\_num:

books\_query.filter('id\_b =',request.id\_b)

books\_query.get().delete()

return message\_types.VoidMessage()

Додаток 7. services.py

from protorpc.wsgi import service

import lab5

# Map the RPC service and path (/PostService)

app = service.service\_mappings([('/Message\_Services', lab5.Message\_Services)])

Додаток 8. Модифікований app.yaml

runtime: python27

api\_version: 1

threadsafe: true

handlers:

- url: /stylesheets

static\_dir: stylesheets

- url: /Message\_Services.\*

script: services.app

- url: .\*

script: lab2.app