Rapid Application Development

Forms and Users

Member of VTC Group
VTC 機構成員

Coventry University

Rapid Application Development

Forms and Users

Member of VTC Group

VTC 機構成員

Outcomes

Be able to build and process HTML forms

Use the Python Logging library to debug applications

Understand how to send email using the Mail library

Integrate user authentication to allow users to sign in using their Google account

Interactivity

So far we have only shown how to display information (read-only)

The only interactivity has been navigating between pages using hyperlinks

In this session we will integrate web forms

Supply a form asking the user for information which is then processed

HTTP GET vs POST

GET

Parameters in URL

Used for fetching documents

Maximum URL length

OK to cache

Shouldn't change the server

POST

Parameters in body

Used for updating data

No maximum length

Not OK to cache

OK to change the server

Capturing Form Data

Create an HTML form

- Action attribute tells browser where to send form data
- Method attribute defines which HTTP method to use
- Each form field is given a unique name

```
<form action="/" method="post">
    <div>
        <textarea name="content" rows="3" cols="60"></textarea>
    </div>
    <div>
        <input type="submit" value="Send Feedback">
    </div>
</form>
```

Processing Form Data

Remember the default method:

```
def get(self):
```

This gets called when the GET method is used

Because we are using the POST method to send the form

We need a new method to handle this:

```
def post(self):
```

Simple example on next slide...

```
import webapp2

class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.response.write('Displaying form...')

    def post(self):
        self.response.write('Processing form data...')

app = webapp2.WSGIApplication([
    ('/', MainHandler)
], debug=True)
```

The Request Object

Any form data can be accessed using the response object

Call the get method and pass a string representing the name of the form field we want to read

In the previous form the textarea is called content

So to retrieve the text entered by the user we use this code:

```
content = self.request.get('content')
```

Logging

System message appear in the Log Console

We can use the Python logging framework to send our own messages

Allows us to debug our code without sending text to the web browser

In our example it would be useful to capture the content of the textarea

So we store it in a variable then send the variable to the log

```
import webapp2
import logging

class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.response.write('Displaying form...')

    def post(self):
        self.response.write('Processing form data...')
        content = self.request.get('content')
        logging.info(content)

app = webapp2.WSGIApplication([
    ('/', MainHandler)
], debug=True)
```

User Authentication

App Engine integrated with Google Accounts

Enable app's account system – no need to build your own

Users that already have Google accounts can sign in your app using their existing accounts – no need to create new accounts for your app

OpenID and OAuth

No obligation to use Google accounts – you can use OpenID

Oauth support - OAuth is a protocol that allows a user to grant a third party limited permission to access a web application on her behalf, without sharing her credentials (username and password) with the third party

Testing for a Logged-In User

We need to import the users class

This is part of the google.appengine.api framework

We can retrieve the currently logged in user

```
user = users.get_current_user()
```

If this is not null there is a user logged in

```
import webapp2
import logging
from google.appengine.api import users

class MainHandler(webapp2.RequestHandler):
    def get(self):
        user = users.get_current_user()
        if user:
            self.response.write('Displaying form...')
            logging.info('Logged-in user')
            logging.info(user.user_id())
            logging.info(user.nickname())
            logging.info(user.email())
        else:
            logging.info('No user currently logged in')
```

Redirecting to a Login Page

If the user is not logged in

We should send the person to a page where they can choose to log in using their google account

When we test our application locally we are provided with a dummy login

Creating a Logout Link

If we are allowing a user to log in

We should also provide a way for them to log out again

The users object provides a method to generate this

```
logout_url = users.create_logout_url(self.request.path)
```

We can then use this to display a link for the user to click on

If they click on this they will be logged out.

```
class MainHandler(webapp2.RequestHandler):
    def get(self):
        user = users.get_current_user()
        if user:
            self.response.write('Displaying form...')
            logging.info('Logged-in user')
            logging.info(user.user_id())
            logging.info(user.nickname())
            logging.info(user.email())
            logout_url = users.create_logout_url(self.request.path)
            self.response.write('<p><a href="'+logout_url+'">Log Out</a></p>')
        else:
            logging.info('No user currently logged in')
            self.redirect(users.create_login_url(self.request.uri))
```

A Quick Way to Force Login

A quick way to force a user to be logged in is to add a login parameter to the handler in the yaml file

```
- url: .*
  script: main.app
  login: required
```

This simplifies the Python controller logic

```
import webapp2
import logging
from google.appengine.api import users

class MainHandler(webapp2.RequestHandler):
    def get(self):
        user = users.get_current_user()
        self.response.write('Displaying form...')
        logging.info('Logged-in user')
        logging.info(user.user_id())
        logging.info(user.nickname())
        logging.info(user.email())
        logout_url = users.create_logout_url(self.request.path)
        self.response.write('<p><a href="'+logout_url+'">Log Out</a></p>')
```

Mail

Applications can send email messages on behalf of the app's administrators, and on behalf of users with Google Accounts.

Apps can receive email at various addresses.

Apps send messages using the Mail service and receive messages in the form of HTTP requests initiated by App Engine and posted to the app.

Sending Mail

Need to import the mail class

This is part of the google.appengine.api library

Create a new message first:

```
message = mail.EmailMessage()
```

Then we set properties for the sender, to, body, etc

Finally we call the `send()` method to send the email.

```
user = users.get_current_user()
message = mail.EmailMessage()
message.sender = user.email()
message.body = content
message.to = 'johndoe@csi.com'
message.send()
```

References

Forms

https://developers.google.com/appengine/docs/python/gettingstartedpython27/handlingforms

Logging

https://developers.google.com/appengine/articles/logging

Mail

https://developers.google.com/appengine/docs/python/mail/

Users

https://developers.google.com/appengine/docs/python/gettingstartedpython27/usingusers

https://developers.google.com/appengine/docs/python/users/userclass

https://developers.google.com/appengine/docs/python/config/appconfig#Python_app_yaml_Requiring_login_or_administrator_status

https://developers.google.com/appengine/docs/python/users/adminusers


SHAPE