

Rapid Application Development

Contact Form

The outcome for this second lab is to develop a fully featured feedback form that could be incorporated into a public-facing website. The information entered will be formatted and sent to a predetermined email address. Note that this email functionality will only work once the project has been deployed to the cloud.

This lab is designed to take you through many of the key features of Google App Engine (GAE) development. Don't worry if you don't understand many of the terms. Spend time finding your way around the the tools and ensure you understand the purpose behind the steps you carry out. Any skill covered in a previous worksheet will be indicated with the :-) characters.

To complete this lab you will need to have Google App Engine Python SDK and the Brackets code editor installed.

As you work through the activities you will be practicing the essential skills you will need as you develop apps using the App Engine Framework.

Software Requirements

In order to complete this lab you will need access to a computer running the Google App Engine SDK and the Brackets code editor. These are both free downloads.

Regular Testing

The lab is broken down in a series of tasks. At the end of each task you should run your project in the web browser to ensure there are no build errors. The steps it takes you through are typical for all GAE projects.

Extension Activities

Once you have successfully completed the main lab tasks you are encouraged to have a go at the extension activities. These are slightly more challenging and will help your understanding of writing GAE apps.

Create a New Project :-)

Use the App Engine Launcher to create a new project called feedback. Save the project in a suitable folder.

Create a Simple Feedback Form


You should now modify the get method in the request handler so that the page displays a simple HTML form that includes a textarea with the name attribute set to 'content' (for the user to enter their feedback) and a submit button. Remember to use the self.response.write method.

The action attribute of the form should send the form data to the web server root (/) and you should use the POST method.

Your code might look something like this:

```
class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.response.write('<form action="/" method="post"><div>')
        self.response.write('<textarea name="content" rows="3" cols="60"></textarea>')
        self.response.write('</div><div><input type="submit" value="Send Feedback">')
        self.response.write('</div></form>')
        self.response.write('</body></html>')
```

If you run your project and browse to the local web server you should see something like this:



405 Method Not Allowed

The method POST is not allowed for this resource.

Try entering a comment and pressing the button. Notice that we get an error message! We have not implemented the POST method. So this should be our next step.

Implement the Form Handler

The current method in your request handler is called `get(self)`. This gets called when the page is accessed using the GET method. To respond to a POST method we need to add another method called `post(self)`.

```
class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.response.write('<form action="/" method="post"><div>')
        self.response.write('<textarea name="content" rows="3" cols="60"></textarea>')
        self.response.write('</div><div><input type="submit" value="Send Feedback">')
        self.response.write('</div></form>')
        self.response.write('</body></html>')

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

Try returning to the form page and clicking on the button again. The error has now gone and we see the expected message but we need to access the feedback comment and, eventually, email this.

Accessing Form Data

We can now add some code to our new method to access the user feedback and store it in a local variable however to see if this is

working

```
def post(self):
    self.response.write('Processing form data...')
    feedback = self.request.get('content')
    self.response.write('<p>' + feedback + '</p>')
```

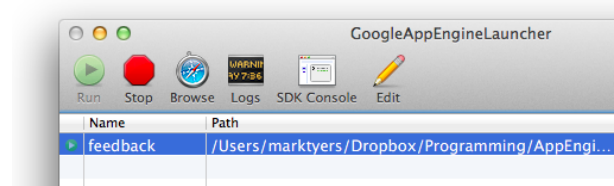
`Processing form data...` we need to display it. One option is to write it to the web browser.

Cool!

Sending debugging data to the web browser is not ideal. as it may upset the layout of the web page and in many instances might be a security risk. In the next section we will learn how to send messages to the Log Console which is a far more robust approach.

Using the Log Console

When you run your GAE application, any system messages are sent to the Log Console which can be accessed using the Logs button in the App Engine Launcher.



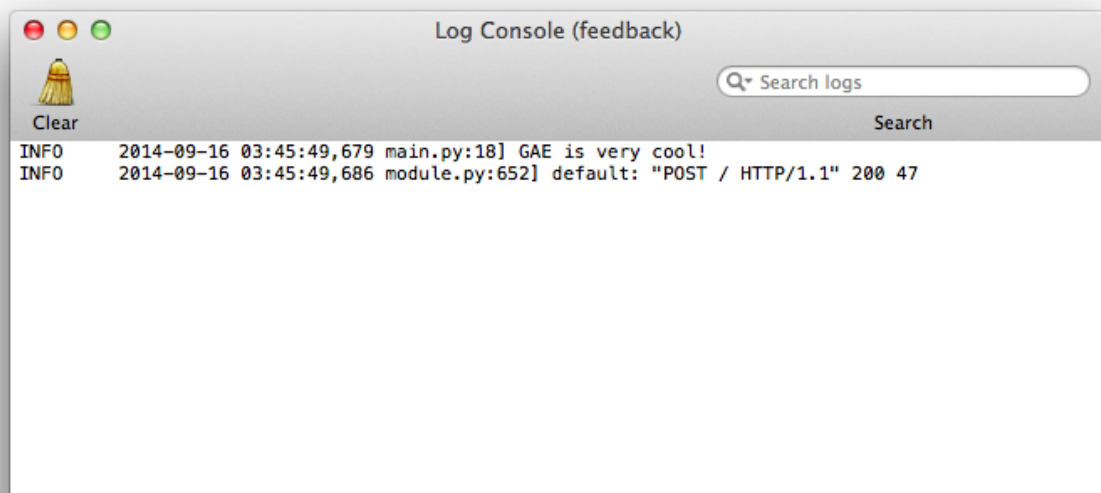
This opens the logs window. In the top corner there is a button to clear the log messages. Click this then try sending a message using your form. Notice that every HTTP request is shown.

This is a much better place to display our debugging messages.

We can write to this using the standard Python logging framework (remember to import this!).

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

Once this has been done we can use the `logging.info()` method to send text to the log console



Redirecting the User

Once the user has submitted their comment we should send them to a suitable page. For example you could send them back to the site homepage or back to the form itself. The simplest way to do this would be to create a hyperlink that they could click on however in this worksheet we will automatically redirect the user back to the form and supply a short message so they know their message has been received (once we create that functionality of course).

Creating a redirect is easy and is achieved using the `self.redirect()` method, passing the URL we want to be redirected to like this:

```
def post(self):
    feedback = self.request.get('content')
    logging.info(feedback)
    self.redirect('/')
```

Unfortunately the user won't now see the message so they won't know if their message has been received. It would be a good idea to display a short message under the form to reassure them. This means we need to pass a 'flag' back to the homepage. This flag will tell the page to display the message so the message won't appear when the form first loads.

Lets start by adding a query string to the redirect URL to act as a flag

```
def post(self):
    feedback = self.request.get('content')
    logging.info(feedback)
    self.redirect('/?message=true')
```

```
def get(self):
    message = self.request.get('message')
    if len(message) > 0:
        self.response.write('<p>Thanks for your feedback.</p>')
```

When we load the form page we can see if this string exists and display a message above the form.

Thanks for your feedback.

Sending Email

So far our users can type in a message and get confirmation that their message has been received however apart from logging this message we are doing nothing with it! The next stage is to send this message to our email address so we can keep track of the comments on the website.

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

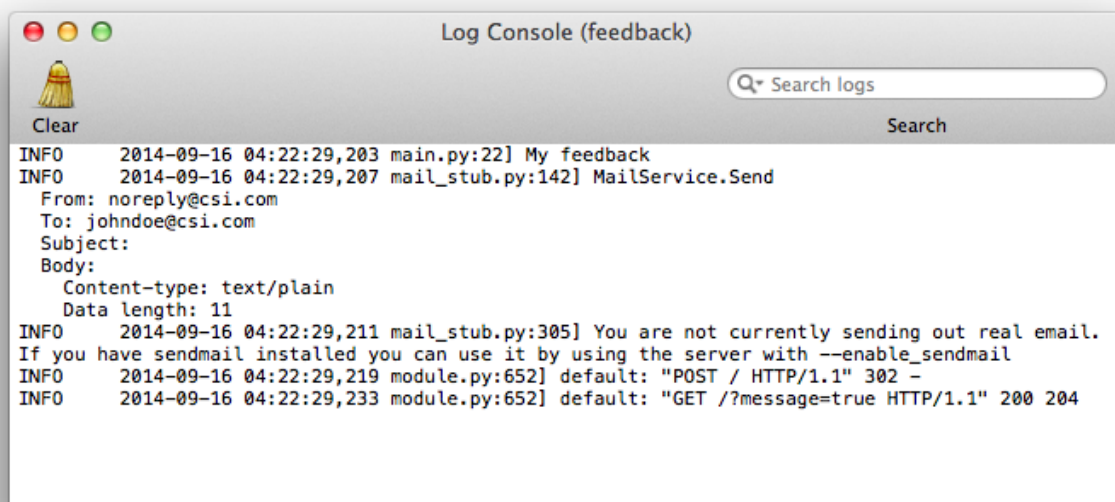
Email functionality is provided using the mail class which is part of the app engine api library so before we can create any messages we will need to import this class as shown

```
def post(self):
    feedback = self.request.get('content')
    logging.info(feedback)
    message = mail.EmailMessage()
    message.sender = 'noreply@csi.com'
    message.body = feedback
    message.to = 'johndoe@csi.com'
    message.send()
    self.redirect('/?message=true')
```

Once this has been imported we can create an EmailMessage object, set its various properties and call its send() method to send the completed email.

If you are testing locally the message won't get sent but you will get detailed information in the Console Log to confirm everything is correct. To fully test this you would have to upload your

project to the Google Cloud.



Logging Users In (and out)

Rather than building our own user account system which would require us to build registration forms, confirmation links and store personal data we can allow users to log in using their existing Google accounts. This makes a lot of sense and is really easy to achieve.

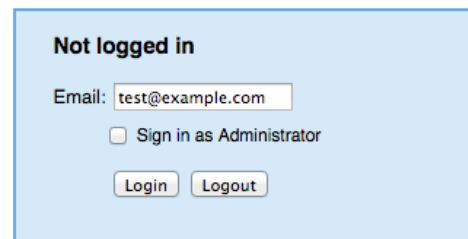
All this functionality is available through the users class which is part of the app engine api so the first step is to import this so it is available to your program.

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

```
def get(self):
    user = users.get_current_user()
    if user:
        logging.info('User is logged in')
        # display the form
    else:
        logging.info('No user currently logged in')
        uri = users.create_login_url(self.request.uri)
        self.redirect(uri)
```

Once this is done we can add some code to load the currently logged in user. If this does not exist we use the users class to generate a unique login URL and send the user there. Note that to save space the form has been omitted in the screenshot.

If we test this locally we will be redirected to a 'fake' login screen. You can enter any email address you like for testing purposes.

A screenshot of a web form titled "Not logged in". It has a light blue background and a thin blue border. The form contains an "Email:" label followed by a text input field containing "test@example.com". Below the input field is a checkbox labeled "Sign in as Administrator". At the bottom of the form are two buttons: "Login" and "Logout".

The final step is to add a link that will allow users to log back out again. The URL is again generated using the users class.

```
def get(self):
    user = users.get_current_user()
    if user:
        logging.info('User is logged in')
        # display the form
        url = users.create_logout_url(self.request.path)
        self.response.write('<a href="'+url+'">Log Out</a>')
    else:
        logging.info('No user currently logged in')
        uri = users.create_login_url(self.request.uri)
        self.redirect(uri)
```

Challenge

NOTE: You should not attempt this until you have finished the worksheet and understood the theory behind this. Your lecturer may talk through this or you may be required to watch a short video.

You have just built a simple feedback form that lets users log in using their Google accounts. Once logged in they can type in some feedback and this gets sent to you together with their Google details.

For your challenge you are required to build a more sophisticated survey that asks logged-in users for their opinions of this module. You should make use of a wide variety of HTML form elements such as check boxes, radio buttons and dropdown lists. This information should be carefully and clearly formatted in the emails that are sent.

After testing this locally you must upload it to the cloud, send the link to your classmates and ask them to complete it.