Rapid Application Development

Book Catalogue

In this worksheet you will be developing an application to allow users to browse a book catalogue. It will contain both a list of books and a detail page to view individual books.

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.

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.

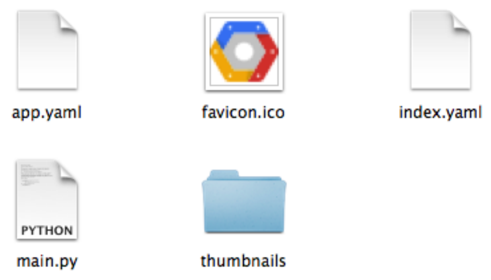
Extension Activities

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

Start by creating a new Google App Engine project called bookshop. Open the project folder and create a subfolder called thumbnails.

Download around six book covers from the Internet and make sure they are around the same size (small), put these in the thumbnails folder. Open the project in the editor, you should be able to see the new folders and the thumbnails in the sidebar.



Adding a Handler

Now we have a folder to contain our book covers we need to set up a handler to direct appropriate URLs to the folder. This needs to be added to the app.yaml file like so:

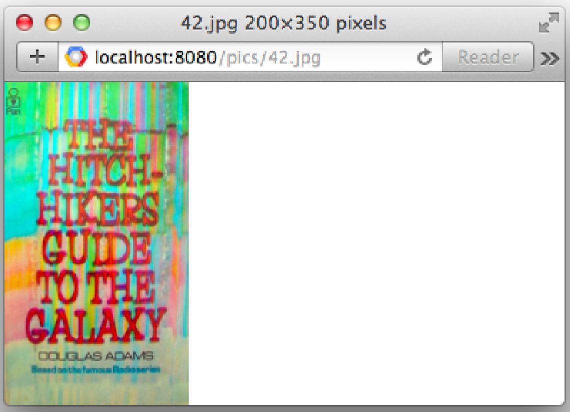
```
handlers:
- url: /favicon\.ico
  static_files: favicon.ico
  upload: favicon\.ico

- url: /pics
  static_dir: thumbnails

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

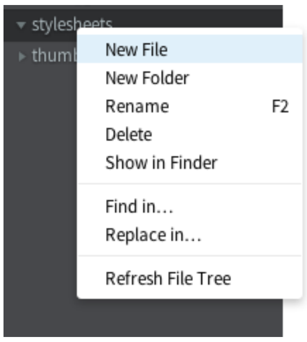
When a URL is requested the handlers are checked from top to bottom. If a match is detected that handler is immediately processed and any later ones ignored.
Thats why the most specific ones are at the top and the last one matches any URL.
The one we just added (middle) detects any URL / pics and responds with the file requested, looking in the thumbnails folder.

For example if there was a file called 42.jpg in the thumbnails folder it could be accessed like this:


42.jpg 200×350 pixels
localhost:8080/pics/42.jpg
Reader
THE HITCH-HIKERS GUIDE TO THE GALAXY
DOUGLAS ADAMS

Check you can access each file in your thumbnails folder.

Linking a Stylesheet



The next task builds on the previous one. Start by creating a folder called stylesheets in your project folder (same place as the thumbnails folder). Refresh the Brackets editor (same key combination as your web browser) to see the new folder. Right-click on the stylesheets folder in the sidebar and choose New File, name it main.css

Add some CSS to turn the paragraph tags red (so we can see when the stylesheet is linked correctly).

```
p {
   color: red;
}
```

Next we need to add another handler to the app.yaml file since we have created a static css file. Position it just below the previous handler.

```
- url: /pics
  static_dir: thumbnails

- url: /css
  static_dir: stylesheets
```

Finally we need to send the stylesheet link to the web browser by editing the main.py file.

```
def get(self):
    self.response.write('<link type="text/css" rel="stylesheet" href="/css/main.css" />')
    self.response.write('<p>Hello world!</p>')
```

Notice how we link to the stylesheet and also wrap our message in a paragraph tag.

Adding More Pages

All the applications produced so far have only had a single page. If we are to produce more than trivial applications we need to be able to add more pages. In the previous sections we created routes to add static directories, now we will add another route, this time to load a different python file.

Start by adding a new Python file called page2.py in the same folder as your original main.py file. Copy the contents of the main.py file into it.

Change the message displayed so you know which file you are viewing. You will also need to make a small change to the line of code that creates a new app object so it responds to URLs that contain additional characters after the root.

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

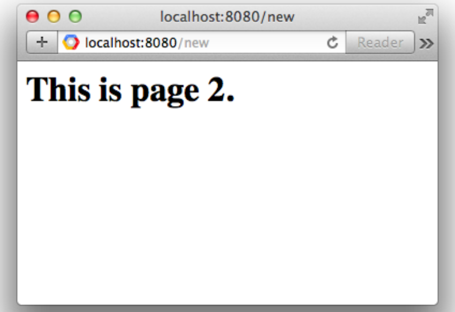
Now we can modify the app.yaml file and add another handler.

```
- url: /css
  static_dir: stylesheets

- url: /new
  script: page2.app

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

We can now save the edited files and test this new URL.



This is page 2.

Handling URL Segments

A URL segment is a way to pass a variable to a page by modifying part of the URL. Lets look at an example. Consider the following two similar URLs:

```
http://localhost:8080/book/978-0596800697
http://localhost:8080/book/978-1449398262
```

The numerical value at the end represents the ISBN of a specific book. Our application needs to be able to read this to determine which book to display needs to be read by the page handler and be stored in a variable.

Start by creating a new script called book.py, copying the contents of main.py in as a template. Now modify the app.yaml file by adding a new handler.

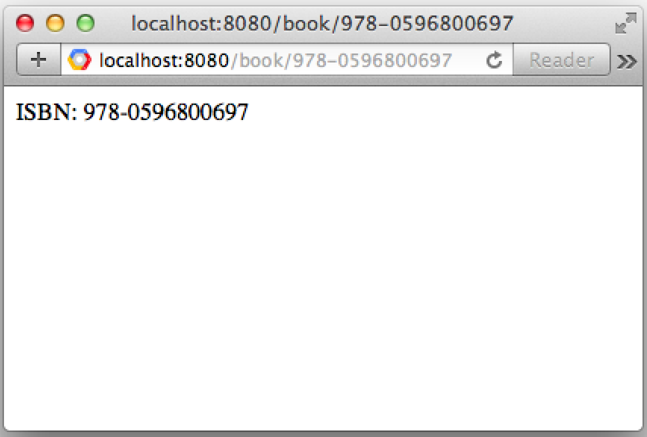
```
- url: /book.*
  script: book.app
```

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

Now edit the script. We need to modify the pattern match when we create the WSGIApplication by replacing the existing matcher with a simple regular expression. We denote the regular expression by preceding the pattern with a lower case r. Anything in brackets gets captured and assigned to a variable in our code.

Finally we modify the get method by adding a second parameter. This will receive any characters captured by the brackets in the regular expression, in this example this will be the ISBN number.

```
def get(self, isbn):
    self.response.write('<p>ISBN: '+isbn+'</p>')
```


localhost:8080/book/978-0596800697
localhost:8080/book/978-0596800697
Reader

ISBN: 978-0596800697

Using the Jinja2 Templating Language

The final step is to separate the page layout from the data to make your web pages easier to understand and maintain. We will be modifying the /book page.
Start by opening your project folder and creating a folder called templates. Inside this create a simple HTML template file suitable for the book page, and call it book.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Book Details</title>
  </head>
  <body>
    <h1>Book Details</h1>
    <p>ISBN: {{isbn}}</p>
  </body>
</html>
```

```
libraries:
- name: webapp2
  version: "2.5.2"
- name: jinja2
  version: "2.6"
```

Now modify the app.yaml file and import the jinja2 library. The version at the time of writing was 2.6 but you should check online to see if there has been an update.

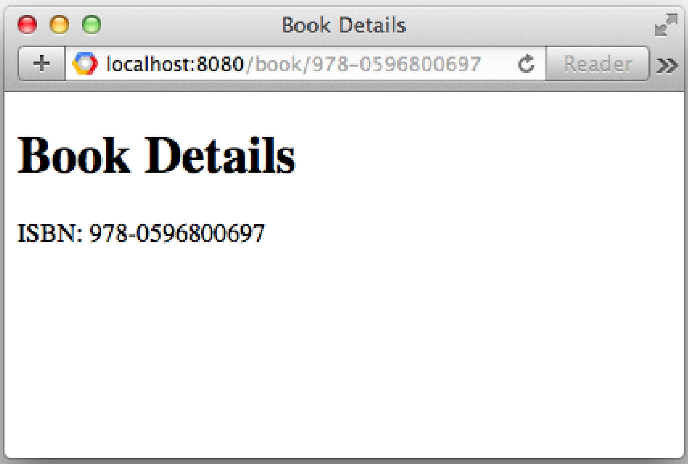
Open the book.py file and, before the MainHandler class, add the following:

```
ENVIRONMENT = jinja2.Environment(
  loader=jinja2.FileSystemLoader(os.path.join(os.path.dirname(__file__), 'templates')),
  extensions=['jinja2.ext.autoescape']
)
```

This creates a Jinja2 environment variable, notice we pass the name of the folder containing our html template.

Finally we change the get method to load the html file into a template object which is then called and passed the data we want to display.

```
def get(self, isbn):
    template = ENVIRONMENT.get_template('book.html')
    data = {
      'isbn': isbn
    }
    self.response.write(template.render(data))
```



Book Details

ISBN: 978-0596800697

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 implemented a number of important features however your challenge is to combine all of these into a single useful application.

You are required to build a two-page application. The first page should contain thumbnails. The table will need to be hand-coded.

When the user clicks on one of the books they should be taken to a separate page that displays the book isbn, title and thumbnail. You will need to pass several pieces of information to this second page. Both pages should use the jinja2 templating system.