Python and Python Web Applications

ICT Enterprise Project

Supervisor: Catherine Woods

Submitted By: David Murphy (TL\_KCOMP\_GY6)

<<Submitted on: 28th April 2014>>

Table of Contents

[1 Introduction 1](#_Toc379479797)

[2 Requirements Specification 2](#_Toc379479798)

[2.1 Project goals 2](#_Toc379479799)

[3 Project Plan 3](#_Toc379479800)

[4 Python Research 5](#_Toc379479801)

[5 Comparison of Python with Java 6](#_Toc379479802)

[6 Python Web Application Research 7](#_Toc379479803)

[7 Domain Application of Python and Python Frameworks 8](#_Toc379479804)

[8 Python/Flask Web Application 9](#_Toc379479805)

[8.1 Introduction 9](#_Toc379479806)

[8.2 Analysis 9](#_Toc379479807)

[8.3 Design 9](#_Toc379479808)

[8.4 Implementation Increments 9](#_Toc379479809)

[8.5 Testing Strategy 9](#_Toc379479810)

[8.6 Conclusion 9](#_Toc379479811)

[9 Project Conclusions 10](#_Toc379479812)

REFERENCING – for citing

BIBLIOGRAPHY – other resources not cited

# Introduction

Introduction goes here.

# Requirements Specification

## Project goals

The goals of this project are as follows:

* Acquire programming skills in the Python programming language
* Acquire programming skills in a Python web framework (Flask)
* Compare and contrast Python with another high-level programming language (Java) in terms of programming paradigms, syntax, standard libraries, domain applications and web development
* Perform analysis, design, implementation and testing of a web application programmed in a Python/Flask environment

# Project Plan

The following is a detailed breakdown of the tasks required to achieve the project goals outlined above:

|  |  |
| --- | --- |
| **PROJECT GOALS** | **TIMELINE - Wk1 – Wk12 (college term)** |
| **Acquire Python programming skills**  Tasks to complete:   * study official Python.org documentation * complete official Python.org tutorial * complete ‘Dive Into Python’ online course   All research conducted in this phase will be documented in this section accordingly. | This research will be carried out continuously over Wks1 – Wks6 |
| **Acquire skills in a Python web framework (Flask)**  Tasks to complete:   * read Flask official documentation * complete Flask online tutorial * complete other online tutorials   All research conducted in this phase will be documented in this section accordingly. | This research will be carried out continuously over Wks3 – Wks7 |
| **Comparison of Python with another high-level programming language (Java)**  A critical comparison of Python with Java in terms of the following headings will be presented:   * programming paradigms * features of the language * syntax and semantics * standard libraries * domain applications * web development | Work in this section will be carried out over Wk5 – Wk8 |
| **Analysis, design, implementation & testing of a web application programmed in a Python/Flask environment**  Summary of implementation  The application to be developed will allow a client-side user to log on to a web application, select from a list of currencies and input a monetary amount to be converted from one denomination to another.  When instructed, the web application will connect RESTfully to a remotely hosted web service using the required input data. The web service will perform the necessary business logic calculation and RESTfully return data to the web application for presentation to the user.  The web service could be enhanced by connecting to a database to retrieve currency data.  Technologies to be employed   * The web application and web service will be developed in Python using the Flask web micro-framework for web development. * The web service will be remotely hosted on a suitable hosting service, such as Heroku * Analysis and design of the application will include analysis of the problem and a description of the solution design, and will also include UML diagrams as required.   Product delivery  Due to the nature of the research into Python and Flask being carried out the application will be developed incrementally, with each increment adding further functionality and/or improvements to the application until a satisfactory solution is implemented.  Testing strategy  A testing strategy will be designed and the application tested accordingly in conjunction with each increment. | Work in this section will be carried out over Wk3 – Wk11  Analysis & design to be completed by Wk4  First increment delivered Wk4  Weekly increments thereafter until satisfactory implementation is achieved  Testing strategy to be completed by Wk4, and carried out in conjunction with the various increments  Final submission 28/04/2014 |

# Python Research

# Python Web Framework Research

## Introduction

## Python Web Frameworks

# Comparison of Python with Java

# Python/Flask Web Application

## Introduction

This section describes the analysis, design, implementation and testing of a web application developed in the Python programming language using the Flask “microframework” for web development.

The implementation was designed such that it would make use of the technologies studied as part of this project.

The web application to be developed is a currency conversion application. The application will allow a client-side user to securely log in through a browser to a web application, hosted locally. The web application will present a selection of currencies to the user, and allow user-input of a monetary amount, with the user selecting the base currency and the converted currency. The user submits the information, and a currency conversion is performed. The converted amount is displayed to the user.

The application will connect to a remotely hosted RESTful web service, posting currency and amount data. The web service will carry out the necessary business logic and return the resulting data to the web application for presentation to the user.

The design for this project has been kept deliberately simple in order to create a suitable working product commensurate with the new technologies explored earlier. It is considered that the proposed implementation will demonstrate a significant knowledge of Python application programming, Python/Flask web application development and remote API/web service deployment to a suitable host in the cloud. The focus is not on what the system does, but rather *how* it does it.

## Technologies

The web application will employ the following technologies in development:

* client-side web application: to be developed in Python employing the Flask web framework
* server-side web service: to be developed in Python employing the Flask web framework. The web service is to comply with the requirements of REST (REpresentational State Transfer) architectural principles
* remote hosting: the web service will be hosted on the Heroku Cloud Application Platform. Heroku provides support for Python applications developed with the Flask web framework
* Git/GitHub: a GitHub remote repository will be used for the duration of the project. Source code and documentation will be pushed to the repository such that a full history of incremental development will be recorded. Heroku also requires the use of Git and GitHub for the deployment of applications
* Notepad++ has been chosen as a simple and lightweight IDE for the development of Python source code for this project
* The application will be developed on a Windows 7 platform, with Python scripts executed using the Python interpreter from Windows Powershell

## Product Delivery

The web application will be developed incrementally. It is considered that this is an appropriate method of delivering the final product, given the concurrent research into the various technologies that is taking place.

The first implementation will be a very simple version of the application, locally hosted, performing minimal calculation. It is intended that increments will be delivered approximately weekly, with each increment adding further functionality/usability/style until a satisfactory solution is implemented which achieves all of the requirements detailed in the following section.

## Analysis

### Requirements Hierarchy Chart

### User Requirements

The requirements of the web application are identified as follows:

* the web app will allow a new user to create a user account for logging into the application
* the web app will allow a previously registered user to securely log in to the application
* the web app will carry out a currency conversion on a specified amount between currencies of the user’s choice

### System Requirements

There will be two modules in the proposed system as follows:

* the client-side web application
* remotely-hosted web service

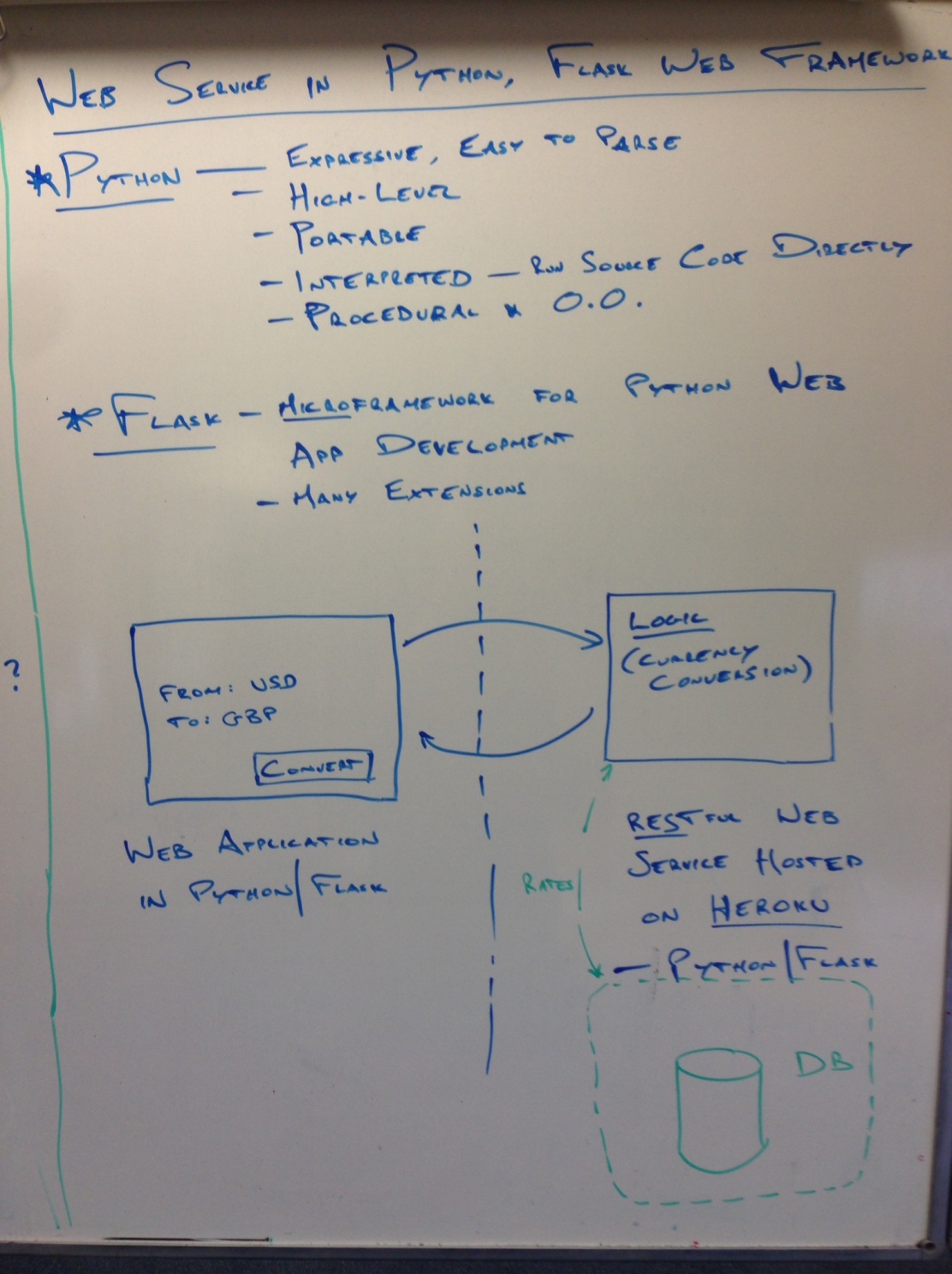
<<System requirement format to be agreed with placement>>

## Design

### Initial Design Development

The initial design of the system was developed to include minimum functionality, and demonstrates the basic web application/web service architecture to be employed in the system. Functionality is initially limited to user input of currencies, with the web service performing the necessary calculation and returning data as required.

Figure 7.xx following is a snapshot of the original design developed in the author’s work placement enterprise.



**Figure 7.xx – initial design**

The design shows the separation of the client-side web application and the remotely hosted web service. The design is explained as follows:

* A web application, developed in Python using the Flask web framework, will be accessed by a user through a browser. The application allows the user to select a base currency from a list, select a target currency from a list and enter a monetary amount to be converted. The text fields presented to the user must only accept a valid monetary amount (i.e. cannot be negative, ust be numeric)
* The inputted data are validated, and posted RESTfully over HTTP to the remotely hosted web service.
* The RESTful web service will be hosted on the Heroku Cloud Application Platform. The service will accept posted data from the web application and perform the required calculation to perform the currency conversion. Currency exchange rates for use in the calculation will be hard-coded into the web service. The converted amount will be posted RESTfully over HTTP to the web application.
* The web application will display the converted amount to the user.
* A database could be used to store currency exchange rates, and may be explored in later increments, depending on implementation progress.

It was also decided to include a user login element to the client-side web application, as this will demonstrate further interesting usage of the capabilities of the Flask web framework.

### Implementation

The application will be developed incrementally. The outline plan for incremental development is as follows:

* Initially develop a basic self-contained web application in Python/Flask that possesses minimal functionality and user input and performs some basic calculation. The application should maintain separation between presentation (HTML using Jinja2 template engine) and logic. The application will be self-contained in that the web service will be intrinsically part of the web application. Little or no styling may be included.
* Further iterations will aim to add the following functionalities to the application, in no particular order:
* enhanced user input (selection of currencies from drop-down lists)
* inclusion of user account control and login/logout functions
* separation of web service/API element from client-side application, and deployment of this to the Heroku Cloud Application Platform
* enhanced styling using cascading style sheets
* any other enhancements which may extend the learning opportunities on this project, such as adding database connectivity

Version control is to be used during the development process. Git and GitHub will be used for maintaining version control. Incremental code and documentation development will be regularly ‘pushed’ to a remote repository on GitHub, allowing full historical version control for the application development. The GitHub repository history will demonstrate a continuous work output for the duration of the project and application development.

## Testing Strategy

The testing plan for the project will be developed incrementally as functionality is added to the application. However, as an overall testing strategy the following is to be adopted:

* Unit testing – client-side and server-side modules are to be fully tested during development. Unit testing and results to be appropriately documented.
* Black-box testing – all user input fields are to be fully tested and results compared with expected behavior as follows:
* Numerical fields should be tested for in-range values, boundary values, just out-of-bounds values, out-of-range values, negative values, alphabetical values, non-alpha-numeric values, etc.
* String fields should be tested for empty strings, maximum string length, etc.
* Email fields should be tested for correct email format

In all cases on this project, the agile approach of ‘test-early, test-often’ is assumed.

## Implementation Increments

### Increment #01

The first increment of the application was pushed to GitHub on 07th February 2014. The application runs as a web application hosted on localhost:5000 (or 127.0.0.1:5000), and can be accessed locally by browsing to this address in a web browser.

#### Increment #01 Functionality

The home-page of the application requests the user to input a monetary amount. A nominal currency conversion is carried out and the result displayed to the user. If the user enters an invalid (non-numeric) value, an error message is displayed and the calculation is not carried out. The user can click a button to clear the current conversion if they so wish.

#### Increment #01 Code

The following is the main Python script developed for Increment #01.

PyConvert\_V0.1.py

1 # ICT Project 2014

2 # D Murphy

3 #

4 # PyConvert\_V0.1

5

6 # imports

7 **from** flask **import** Flask**,** request**,** redirect**,** url\_for**,** render\_template

8

9 # configuration

10 DEBUG **=** **True**

11 SECRET\_KEY **=** 'development key'

12

13 # hard-coded currency conversion rates

14 GBP\_TO\_EURO **=** 1.21

15

16 # create the application

17 app **=** Flask**(**\_\_name\_\_**)**

18 app**.**config**.**from\_object**(**\_\_name\_\_**)**

19

20 @app.route**(**'/'**)**

21 **def** render\_home**():**

22 **return** render\_template**(**'base.html'**,** rate**=**GBP\_TO\_EURO**)**

23

24 @app.route**(**'/calculate'**,** methods**=[**'POST'**])**

25 **def** calculate\_values**():**

26 **try:**

27 amount\_as\_string **=** request**.**form**[**'currency'**]**

28 amount **=** float**(**amount\_as\_string**)**

29 converted\_amount **=** GBP\_TO\_EURO **\*** amount

30 **return** render\_template**(**'conversion.html'**,** input**=**amount**,** output**=**converted\_amount**,** rate**=**GBP\_TO\_EURO**)**

31 **except** ValueError**:**

32 error **=** 'You must enter a numerical value! Try again.'

33 **return** render\_template**(**'conversion.html'**,** error**=**error**,** rate**=**GBP\_TO\_EURO**)**

34

35 @app.route**(**'/clear'**,** methods**=[**'POST'**])**

36 **def** clear\_conversion**():**

37 **return** redirect**(**url\_for**(**'render\_home'**))**

38

39 **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

40 app**.**run**()**

This script represents a simple, self-contained web application. The import statement on line 7 imports some functions from the Flask framework, including the Flask object, of which the application (app) is an instance. The code is self-documenting and so a detailed explanation of each line is not presented. However, some interesting items of note are considered as follows:

* configuration data is included on lines 10/11. The SECRET\_KEY is used for secure sessions, and will be discussed later. DEBUG presents a stack-trace to assist in debugging when an error occurs
* the web application is an instance of the Flask object (line 17)
* ‘decorators’ are used in the script to associate web URL’s with Python functions. This is achieved using the @app.route decorator. When a user browses to the URL in the parameter list of the decorator, the function defined immediately underneath is executed.
* the @app.route decorator is associated with HTTP ‘GET’ methods by default. If the associated web template is to be used for user input, the ‘POST’ method must be included in the parameter list.
* the render\_template function renders a HTML template (Flask uses the Jinja2 template engine). Additional parameters are included as required, depending on the variables used in the template itself.
* the concept of template inheritance was explored in this increment. The base template (base.html) is routed to the URL ‘/’. The template conversion.html extends the base template. These templates are presented below.
* the actual currency conversion is carried out in the calculate\_values function, using a hard-coded exchange rate, provided on line 14. A try-except is used to ensure that valid numerical data is entered by the user, otherwise an error message is returned to the template.

The following are the HTML templates (base.html and conversion.html) for this increment:

base.html

**1** <!-- base.html

**2**  ICT Project

**3** D Murphy -->

**4**

**5** <!doctype html>

**6** <head>

**7** <title>**PyConvert**</title>

**8** </head>

**9** <body>

**10** <div class=page>

**11** <h1>**PyConvert(V0.1)**</h1>

**12** <p>**Enter amount in GBP in form below. PyConvert will return Euro value (1GBP = {{ rate }} Euro)**</p>

**13**

**14** <form action=**"/calculate"** method=post>

**15** <dl>

**16** <dt>**Amount in GBP:**

**17** <dd><input type=text name=currency>

**18** <dd><input type=submit value=**"Calculate"**>

**19** </dl>

**20** </form>

**21**

**22 {% block calc %}{% endblock %}**

**23**

**24** </div>

**25** </body>

conversion.html

**1** <!-- conversion.html

**2**  ICT Project

**3** D Murphy -->

**4**

**5 {% extends "base.html" %}**

**6 {% block calc %}**

**7 {% if output %}**

**8** <p><strong>**{{ input }}(GPB) in Euro is: {{ output }}** </strong></p>

**9 {% elif error %}**

**10** <p><strong>**{{ error }}**</strong></p>

**11 {% endif %}**

**12**

**13** <hr>

**14** <form action=**"/clear"** method=post>

**15** <input type=submit value=**"Clear conversion"**>

**16** </form>

**17 {% endblock %}**

The templates demonstrate template inheritance (conversion.html extends base.html) and the use of Python-like code, such as **if-elif-endif**, within the templates (provided for by the Jinja2 template engine).

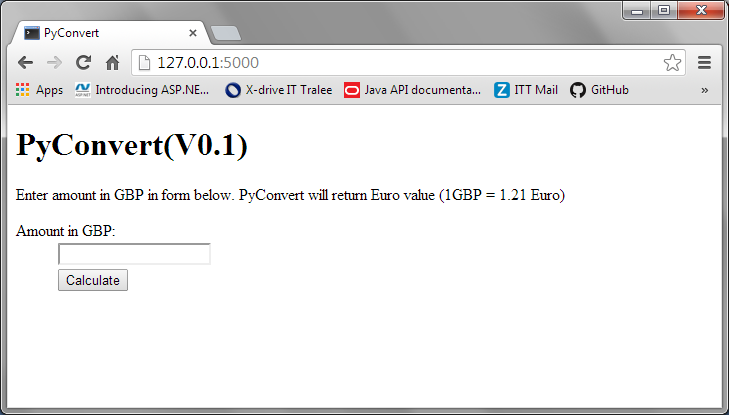
#### Increment #01 Testing

The following user input values are tested and compared with expected behavior:

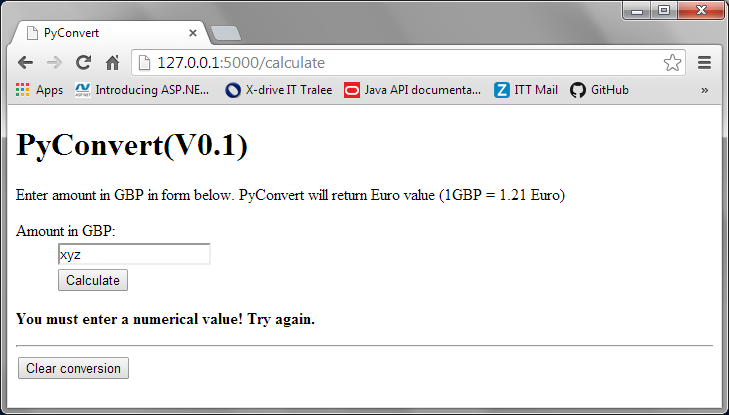
|  |  |  |  |
| --- | --- | --- | --- |
| Test Ref | User Input | Test Value | Expected Result |
| 1.01 | Conversion amount | Non-numerical value (‘xyz’) | Error message |
| 1.02 | Conversion amount | Empty string | Error message |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Increment #01 Output

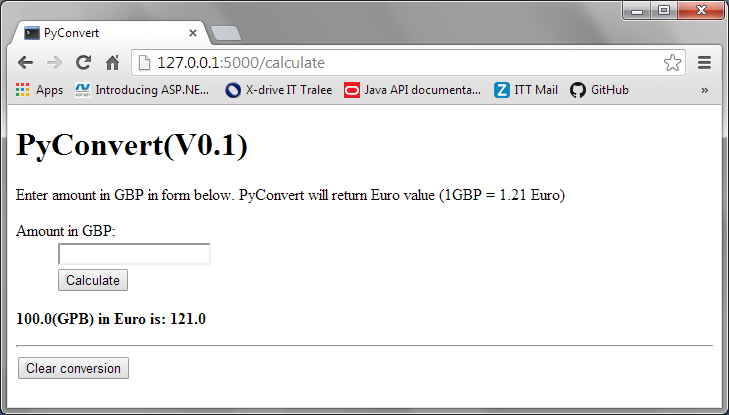
The following figures show screen output from a typical session using the web application.



**Figure 7.xx – Increment #01 web application**



**Figure 7.xx – Increment #01 with non-valid test data**



**Figure 7.xx – Increment #01 with valid test data**

### Increment #02

The second main increment of the application was pushed to GitHub on 13th February 2014. Again the application runs as a web application hosted on localhost:5000 (or 127.0.0.1:5000), and can be accessed locally by browsing to this address in a web browser.

#### Increment #02 Functionality

The code from the first increment was re-written in order to move towards separating presentation from the business logic in an effort to move towards the *Model-View-Controller* software pattern.

The project directory was reorganised to mirror that recommended in the Flask documentation [[1]](#footnote-2)(which is that of a Python module), and this structure is as follows:

/WebApp

/runserver.py

/\_documentation/

/[various project documentation]

/\_source\_code

/[python .py scripts]

/static/

/css/

[.css files]

/templates

[html templates]

When the application is started, browsing to the home URL ‘/’ presents a home page for the application, with a link and a submit button to direct the user to the conversion application. It is noted on the home page that this may be a login page in a future increment.

The currency conversion application allows the user to:

* select a base currency from a dropdown list of pre-loaded currencies
* select a currency for the currency conversion
* input a monetary amount to convert

If the user submits a valid numerical amount, the conversion is displayed on a new page. The user can click a link to carry out another conversion, or alternatively to return to the home page.

If invalid input is entered, helpful messages are ‘flashed’ on the webpage to direct the user accordingly.

The application has also benefitted from css styling.

#### Increment #02 Code

New Python scripts routes.py, forms.py and rates.py were created. Routes.py contains the URL mapping to functions which control data passing to the templates.

Forms.py contains form objects for form input using the Flask-WTF extension. It is useful to be able to define forms in this manner as the form can be defined in the Python code, with Flask-WTF generating the associated HTML. In addition, Flask-WTF provides useful methods for form validation.[[2]](#footnote-3)

Rates.py separates the business logic from the presentation layer. A dictionary of hard-coded exchange rates (relative to Euro = 1.0) is provided for currency conversion. Various functions are defined, including functions for returning an exchange rate between two given currencies and for calculating the currency conversion and returning the result. The separation of this part of the code from the presentation layer represents the first move towards the design goal of implementing the business logic as a RESTful API. For now, all the code required in order for the application to function is self-contained.

Two further Python scripts are included; \_\_init\_\_.py, which creates the Flask instance and contains application configuration data (in accordance with the Python module structure discussed earlier), and runserver.py, located in the project root directory, which kickstarts the application.

runserver.py

1 **from** \_source\_code **import** app

2

3 app**.**run**(**debug**=True)**

\_\_init\_\_.py

1 **from** flask **import** Flask

2

3 # config

4 app **=** Flask**(**\_\_name\_\_**)**

5 app**.**secret\_key **=** 'development key'

6

7 # import routes

8 **import** routes

routes.py

1 # ICT Project 2014

2 # D Murphy

3 #

4 # routes.py

5

6 # imports

7 **from** \_source\_code **import** app

8 **from** flask **import** Flask**,** request**,** redirect**,** url\_for**,** render\_template**,** flash

9 **from** forms **import** ConversionForm**,** ResultForm**,** EnterForm

10 **from** rates **import** getRate**,** getSpecificRate**,** exchange

11

12 @app.route**(**'/'**,** methods**=[**'GET'**,** 'POST'**])**

13 **def** home**():**

14 form **=** EnterForm**()**

15

16 **if** request**.**method **==** 'POST'**:**

17 **return** redirect**(**url\_for**(**'conversion'**))**

18

19 **elif** request**.**method **==** 'GET'**:**

20 **return** render\_template**(**'home.html'**,** form**=**form**)**

21

22 @app.route**(**'/convert'**,** methods**=[**'GET'**,** 'POST'**])**

23 **def** conversion**():**

24 form **=** ConversionForm**()**

25

26 **if** request**.**method **==** 'POST'**:**

27 **if** form**.**validate**()** **==** **False:**

28 form**.**conversionAmount**.**data **=** ''

29 **return** render\_template**(**'conversion.html'**,** form**=**form**)**

30 **else:**

31 unformatted\_input\_amount **=** float**(**form**.**conversionAmount**.**data**)**

32 from\_currency **=** form**.**fromCurrency**.**data

33 to\_currency **=** form**.**toCurrency**.**data

34

35 converted\_amount **=** exchange**(**from\_currency**,** to\_currency**,** unformatted\_input\_amount**)**

36 specific\_rate **=** getSpecificRate**(**from\_currency**,** to\_currency**)**

37

38 input\_amount **=** '%.2f' **%** **(**unformatted\_input\_amount**)**

39

40 **return** render\_template**(**'conversion.html'**,** form**=**form**,** input\_amount**=**input\_amount**,**

41 from\_currency**=**from\_currency**,** to\_currency**=**to\_currency**,**

42 converted\_amount**=**converted\_amount**,** specific\_rate**=**specific\_rate**)**

43

44 **elif** request**.**method **==** 'GET'**:**

45 **return** render\_template**(**'conversion.html'**,** form**=**form**)**

forms.py

1 **from** flask**.**ext**.**wtf **import** Form

2 **from** wtforms **import** TextField**,** SubmitField**,** SelectField**,** validators**,** ValidationError

3 **from** rates **import** populateCurrencyList

4

5

6 **class** **ConversionForm(**Form**):**

7 fromCurrency **=** SelectField**(**'From currency:'**,** choices**=**populateCurrencyList**())**

8 toCurrency **=** SelectField**(**'To currency:'**,** choices**=**populateCurrencyList**())**

9

10 conversionAmount **=** TextField**(**"Enter amount to convert:"**,** **[**validators**.**Required**(**"Please enter a conversion amount"**)])**

11 convertButton **=** SubmitField**(**"Convert"**)**

12

13 **def** \_\_init\_\_**(**self**,** **\***args**,** **\*\***kwargs**):**

14 Form**.**\_\_init\_\_**(**self**,** **\***args**,** **\*\***kwargs**)**

15

16 **def** validate**(**self**):**

17 **if** **not** Form**.**validate**(**self**):**

18 **return** **False**

19

20 **try:**

21 amount **=** float**(**self**.**conversionAmount**.**data**)**

22 **return** **True**

23 **except** ValueError**:**

24 self**.**conversionAmount**.**errors**.**append**(**'You must enter a numerical value! Try again.'**)**

25 **return** **False**

26

27 **class** **ResultForm(**Form**):**

28 clearButton **=** SubmitField**(**"Another conversion"**)**

29

30 **class** **EnterForm(**Form**):**

rates.py

1 # ICT Project 2014

2 # D Murphy

3 #

4 # rates.py

5 # exchange rates and conversion functions

6 # getSpecificRate() and exchange() return Strings

7 # getRate() returns a float

8

9 # populateCurrencyList() returns a list of currencies from

10 # dicts in exchange\_rates to populate WTForms SelectFields

11

12

13 # exchange rates are relative to 1.0 \* Euro

14 exchange\_rates **=** **{**'EUR'**:** 1.0**,**

15 'USD'**:** 1.36379**,**

16 'GBP'**:** 0.82910**,**

17 'AUD'**:** 1.51202**,**

18 'CAD'**:** 1.50140**,**

19 'NZD'**:** 1.64072**}**

20

21 **def** populateCurrencyList**():**

22 currencyList **=** **[]**

23 **for** currency **in** exchange\_rates**:**

24 currencyList**.**append**((**currency**,** currency**))**

25

26 **return** currencyList

27

28 **def** getRate**(**currency**):**

29 **if** currency **in** exchange\_rates**:**

30 **return** exchange\_rates**[**currency**]**

31

32 **def** getSpecificRate**(**from\_currency**,** to\_currency**):**

33 from\_rate **=** exchange\_rates**[**from\_currency**]**

34 to\_rate **=** exchange\_rates**[**to\_currency**]**

35

36 **return** '%.5f' **%** **(**to\_rate **/** from\_rate**)**

37

38 **def** exchange**(**from\_currency**,** to\_currency**,** amount**):**

39 from\_rate **=** exchange\_rates**[**from\_currency**]**

40 to\_rate **=** exchange\_rates**[**to\_currency**]**

41

42 **return** '%.2f' **%** **(**amount **\*** to\_rate **/** from\_rate**)**

The HTML templates used in the project were also rewritten, and now include of a ‘baseLayout.html’, which serves as a foundation layout for all other web templates (home.html and conversion.html).

baseLayout.html

**1** <!-- baseLayout.html

**2**  ICT Project

**3** D Murphy -->

**4**

**5** <!doctype html>

**6** <html>

**7** <head>

**8** <title>**PyConvert**</title>

**9** <link rel=**"stylesheet"** href=**"{{ url\_for('static', filename='css/webapp.css') }}"**>

**10** </head>

**11** <body>

**12** <header>

**13** <div class=**"container"**>

**14** <h1 class=**"logo"**>**PyConvert(V0.1)**</h1>

**15** <nav>

**16** <ul class=**"menu"**>

**17** <li><a href=**"{{ url\_for('home') }}"**> **Home**</a></li>

**18** <li><a href=**"{{ url\_for('conversion') }}"**> **Convert!**</a></li>

**19** </ul>

**20** </nav>

**21** </div>

**22** </header>

**23**

**24** <div class=**"container"**>

**25 {% block content %}**

**26 {% endblock %}**

**27** </div>

**28** </body>

home.html

**1 {% extends "baseLayout.html" %}**

**2**

**3 {% block content %}**

**4** <div class=**"jumbo"**>

**5** <h2>**Welcome to PyConvert Web App**</h2>

**6** <h3 class=**"home"**>**[[This might be a future login page for secure login]]**</h3>

**7** <hr>

**8** </div>

**9**

**10** <form action=**"{{ url\_for('home') }}"** method=**"post"**>

**11 {{ form.hidden\_tag() }}**

**12**

**13 {{ form.enterButton() }}**

**14** </form>

**15 {% endblock %}**

conversion.html

**1** <!-- conversion.html

**2**  ICT Project

**3**  D Murphy -->

**4**

**5 {% extends "baseLayout.html" %}**

**6**

**7 {% block content %}**

**8 {% if converted\_amount %}**

**9** <h3><strong>**{{ input\_amount }} {{ from\_currency}} = {{ converted\_amount }} {{ to\_currency }}** </strong></h3>

**10** <p id=**"conversion"**>**[1.00 {{ from\_currency }} = {{ specific\_rate }} {{ to\_currency }}]**</p>

**11** <br/>

**12** <div class=**"convertPanel"**>

**13** <a href=**"{{ url\_for('conversion') }}"** class=**"convertLink"**>**Another Conversion**</a>

**14** </div>

**15 {% else %}**

**16**

**17** <h3>**Select from currencies and enter conversion amount**</h3>

**18** <hr>

**19**

**20 {% for message in form.conversionAmount.errors %}**

**21** <div class=**"flash"**>**{{ message }}**</div>

**22 {% endfor %}**

**23**

**24** <form action=**"{{ url\_for('conversion') }}"** method=**"post"**>

**25 {{ form.hidden\_tag() }}**

**26**

**27 {{ form.fromCurrency.label }}**

**28 {{ form.fromCurrency }}**

**29**

**30 {{ form.toCurrency.label }}**

**31 {{ form.toCurrency }}**

**32**

**33 {{ form.conversionAmount.label }}**

**34 {{ form.conversionAmount(autocomplete="off") }}**

**35**

**36 {{ form.convertButton }}**

**37 </form>**

**38**

**39 {% endif %}**

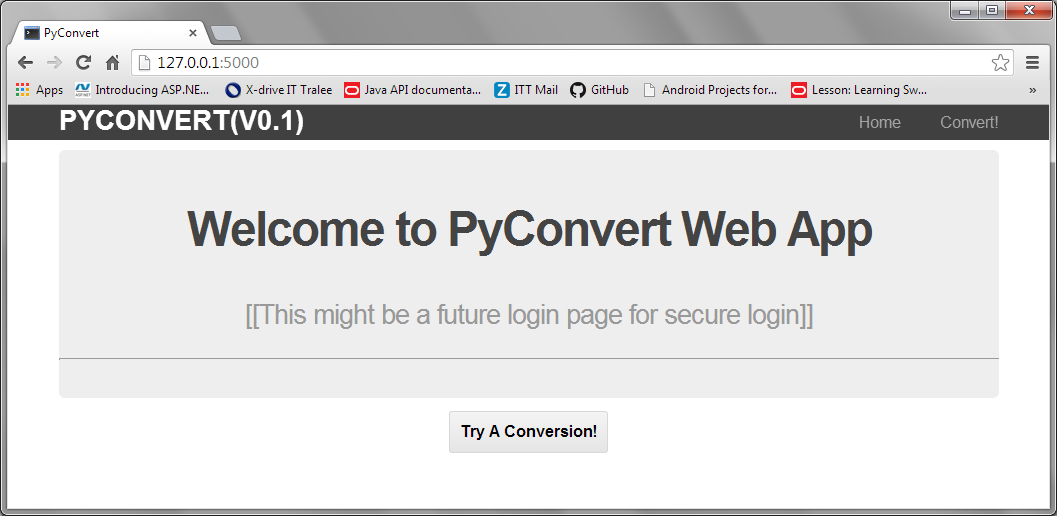
#### Increment #02 Testing

The following user input values are tested and compared with expected behavior:

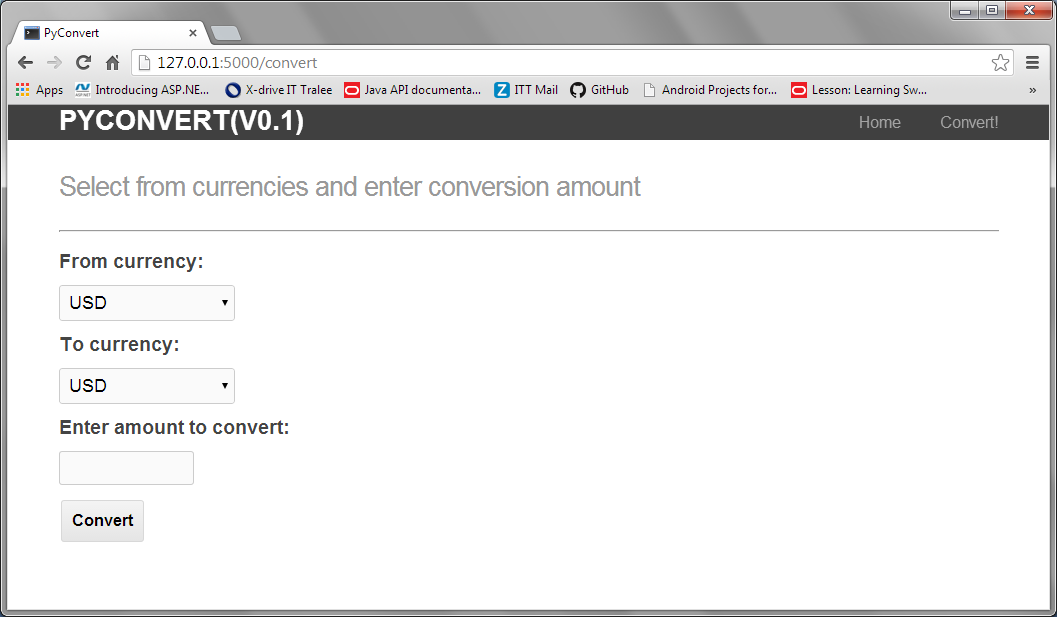
|  |  |  |  |
| --- | --- | --- | --- |
| Test Ref | User Input | Test Value | Expected Result |
| 2.01 | Conversion amount | Non-numerical value (‘xyz’) | Error message |
| 2.02 | Conversion amount | Empty string | Error message |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Increment #02 Output

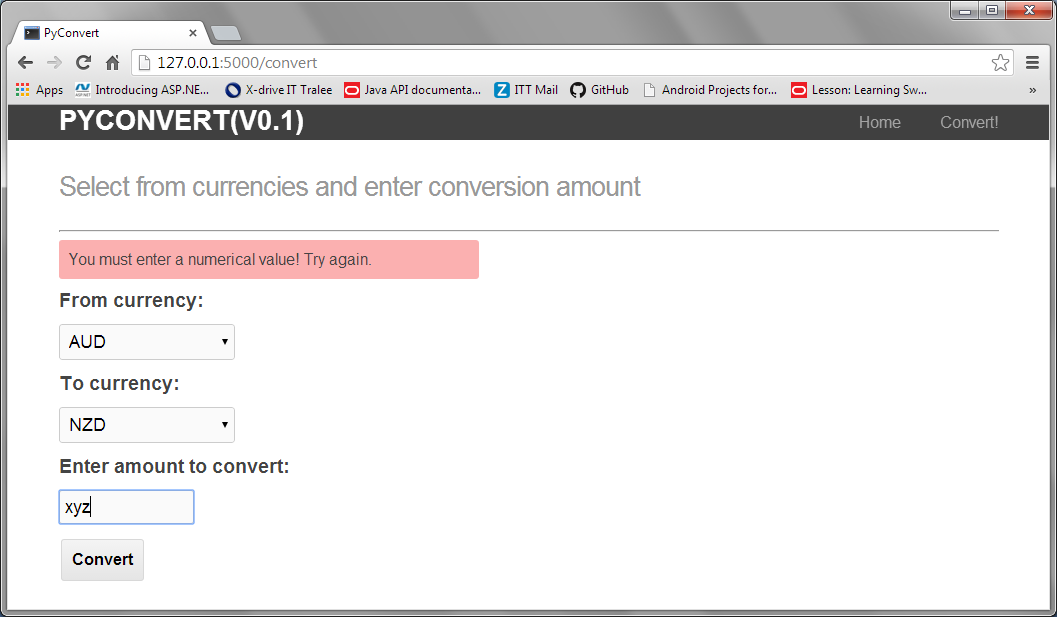
The following figures show screen output from a typical session using the web application.



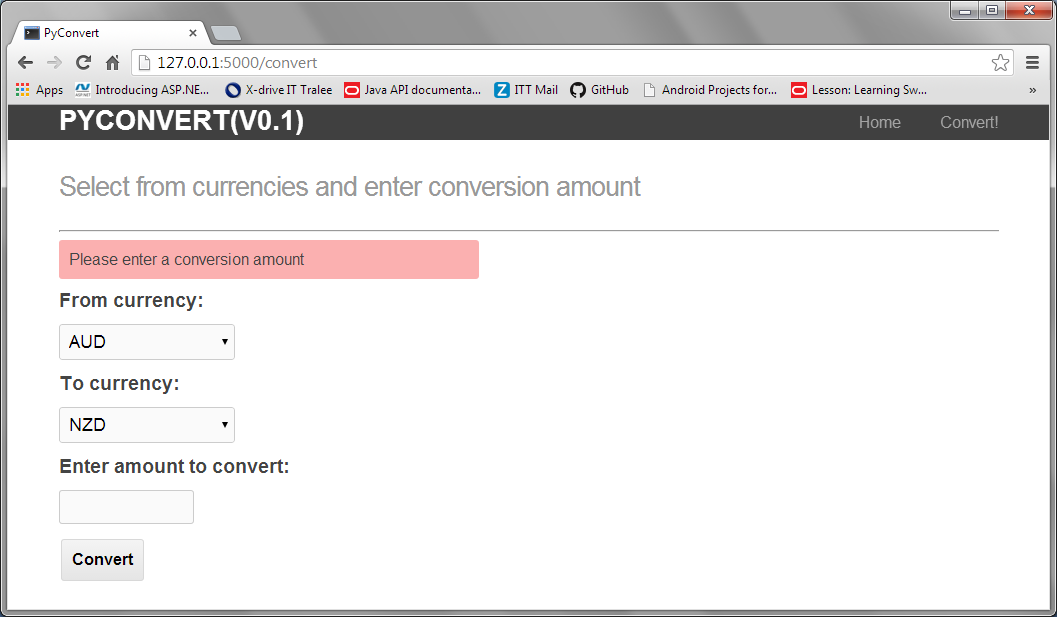
**Figure 7.xx – Increment #02 home page**

****

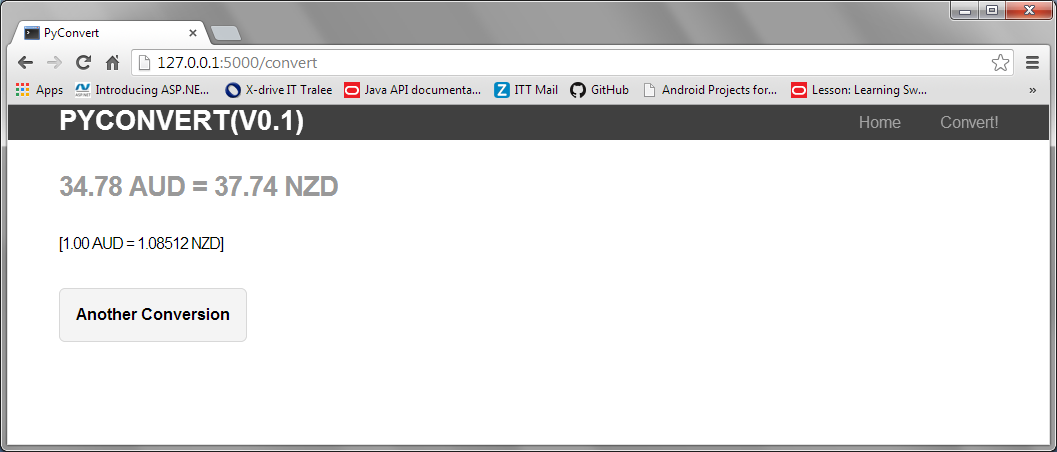
**Figure 7.xx – Increment #02 conversion application**

****

**Figure 7.xx – Increment #02 with invalid test data**

****

**Figure 7.xx – Increment #02 with invalid test data**



**Figure 7.xx – Increment #02 currency conversion output**

## Final Implementation

## Conclusion

# Project Conclusions

1. http://flask.pocoo.org/docs/patterns/packages/ [↑](#footnote-ref-2)
2. http://wtforms.readthedocs.org/en/latest/validators.html [↑](#footnote-ref-3)