College of Arts, Technology and Environment
ACADEMIC YEAR 2023/24

Assessment Brief

Submission and feedback dates

Submission deadline: Before 14:00 on 16/01/2025
This is an individual assessment component eligible for a 48 hour late submission window.

Marks and Feedback due on: 13/02/2025
N.B. all times are 24-hour clock, current local time (at time of submission) in the UK

Submission details:
Module title and code: UFCFX3-15-3 Advanced Topics in Web Development 1

Assessment type: Microservice Design and Build Task

Assessment title: Restful Currency Converter Microservice

Assessment weighting: 100% of total module mark

Size or length of assessment: N/A

Module learning outcomes assessed by this task:

Main Learning Goals & Outcomes (from the Module Specification)

- Build a REST based API (web service) using PHP, JavaScript, XML & XPath and critically reflect on and evaluate this type of Service Oriented Architecture (SOA) based Microservice.
- Build a "responsive" client form using HTML5 + the JavaScript DOM + AJAX to build a test interface to the POST, PUT and DELETE API.
- Use a range of web-oriented software architecture and design principles and make use of the principles of declarative & functional programming in your implementation.

Assignment background & context

Restful Currency Conversion Microservice

The task in this assignment is to build a Restful API (web service) based microservice for currency conversion with full CRUD functionality (using PHP) as well as a client component (HTML + JavaScript) to demonstrate & test the application.

A critical reflection on the development process and learning outcomes achieved are also required.

Completing your assessment

What am I required to do on this assessment?

This is an individual assessment task requiring you to design, build and implement a RESTful currency conversion microservice using PHP. Additionally, you are required a form based interface to demonstrate the service at work.

Additionally, you are required to produce a report (in markdown format) describing (<800 words) explaining the overall process undertaken, any issues and resolutions and the learning outcomes you have achieved.

Your submission should consist of a single ZIP file `atwd1-assign.zip` containing all files and the two reports as specified in this brief.

Where should I start?

This assignment consists of four major tasks with several embedded sub-tasks. This is the task breakdown:

Part 1: Design, Build & Implement a Conversion Microservice (40 marks)

a) RESTful Currency Conversion API

You are required to design, build, implement & test a REST based Web Service that provides a currency conversion function to & from the following (24) ISO 4217 currencies.

ISO Code	Currency
AUD	Australian Dollar
BRL	Brazilian Real
CAD	Canada Dollar
CHF	Swiss Franc
CNY	Chinese Yuan Renminbi
DKK	Danish Krone
EUR	Euro
GBP	Pound Sterling (reference currency)
HKD	Hong Kong Dollar
HUF	Hungarian Forint
INR	Indian Rupee
JPY	Japanese Yen
MXN	Mexican Peso
MYR	Malaysian Ringgit
NOK	Norwegian Krone
NZD	New Zealand Dollar
PHP	Philippine Peso
RUB	Russian Ruble
SEK	Swedish Krona
SGD	Singapore Dollar
THB	Thai Baht
TRY	Turkish Lira
USD	US Dollar
ZAR	South African Rand

The service will receive (GET) requests encoded as follows (example):

http://localhost/atwd1/assignment/?from=GBP&to=JPY&amnt=10.35&format=xml

and return an XML response with the following structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<conv>
    <at>12 Feb 2016 23:44</at>
    <rate>164.24</rate>
    <from>
        <code>GBP</code>
        <curr>Pound Sterling</curr>
        <loc>
            United Kingdom, Isle of Man, Jersey, Guernsey,
            South Georgia and the South Sandwich Island,
            British Indian Ocean Territory, Tristan da Cunha,
            British Antarctic Territory
        </loc>
        <amnt>10.35</amnt>
    </from>
    <to>
        <code>JPY</code>
        <curr>Japanese Yen</curr>
        <loc>Japan</loc>
        <amnt>1699.85</amnt>
    </to>
</conv>
```

Or if the request specifies a JSON format - for example:

http://localhost/atwd1/assignment/?from=GBP&to=JPY&amnt=10.35&format=json

The response should be encoded as follows:


```
{"conv": {
    "at": "12 Feb 2016 23:44",
    "rate": 164.24,
    "from": {
        "code": "GBP",
        "curr": "Pound Sterling",
        "loc": "United Kingdom, Isle of Man, Jersey, Guernsey,
               South Georgia and the South Sandwich Island,
               British Indian Ocean Territory, Tristan da Cunha,
               British Antarctic Territory",
        "amnt": 10.35
    },
    "to": {
        "code": "JPY",
        "curr": "Japanese Yen",
        "loc": "Japan",
        "amnt": 1699.85
    }
}}
```

Rates

You must ensure that your exchange rates are up to date. These should be updated periodically and must not be more than 8 hours old. The "at" date should show the date and time in the given format of when the currency rates were last updated.

Error Handling

The service will also be required to handle errors in a graceful manner and when required, return an error message in the following format:

XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<conv>
    <error>
        <code>nnnn</code>
        <msg>error message</msg>
    </error>
</conv>
```

JSON:

```json
{"conv": {"error": {
    "code": "nnnn",
    "msg": "error message"
}}}
```

Error codes and messages are as follows:

Code	Message
1000	Required parameter is missing
1100	Parameter not recognized
1200	Currency type not recognized
1300	Currency amount must be a decimal number
1400	Format must be xml or json
1500	Error in service

Note that if the "format" parameter is missing - the service should default to XML.
Also, if there is more than one error in the request (e.g. parameter is missing and parameter not recognized) - the error message should report on the first error encountered.

You can view a working version of this service by using the following URL's -

Service called with no parameters - generates a 1000 error: https://fetstudy.uwe.ac.uk/~p-chatterjee/atwd1/assignment

Convert 525.00 American Dollars (USD) to Japanese Yen (JPY) [returns default XML]: https://fetstudy.uwe.ac.uk/~p-chatterjee/atwd1/assignment/?from=USD&to=JPY&amnt=525.00

Same as above but returning JSON: https://fetstudy.uwe.ac.uk/~p-chatterjee/atwd1/assignment/?from=USD&to=JPY&amnt=525.00&format=json

---

Part 2: PUT, POST & DELETE implementation. (20 marks)

Refactor and extend the above GET service to full CRUD functionality by implementing the processes behind the following three (example) URL's.

```
PUT: http://localhost/atwd1/assignment/update/?cur=USD&action=put

POST: http://localhost/atwd1/assignment/update/?cur=USD&action-post

DEL: http://localhost/atwd1/assignment/update/?cur=GBP&action=del
```

Message Formats for PUT, POST & DELETE

The response XML messages should have the following formats:

PUT - using USD as the example currency and updating the rate from 1.22724 to 1.22961:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<action type="put">
    <at>30 Sep 2019 23:44</at>
    <rate>1.22961</rate>
    <old_rate>1.22724</old_rate>
    <curr>
        <code>USD</code>
        <name>US Dollar</name>
        <loc>American Samoa, Bonaire, Sint Eustatius And Saba, British Indian Ocean Territory,
            Ecuador, El Salvador, Guam, Haiti, Marshall Islands, Micronesia (Federated States Of),
            Northern Mariana Islands, Palau, Panama, Puerto Rico, Timor-Leste,
            Turks And Caicos Islands, United States Minor Outlying Islands, United States Of America,
            Virgin Islands (British), Virgin Islands (U.S.)
        </loc>
    </curr>
</action>
```

POST - using XCD as the example currency:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<action type="post">
    <at>30 Sep 2019 23:45</at>
    <rate>3.31245</rate>
    <curr>
        <code>XCD</code>
        <name>East Caribbean Dollar</name>
        <loc>Anguilla, Antigua And Barbuda, Dominica, Grenada, Montserrat, Saint Kitts And Nevis,
            Saint Lucia, Saint Vincent And The Grenadines</loc>
    </curr>
</action>
```

DELETE - using XCD as the example currency:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<action type="del">
    <at>30 Sep 2019 23:45</at>
    <code>XCD</code>
</action>
```

POST, PUT & DELETE Error Handling

Again, the post/put/delete service will be required to handle errors in a graceful manner and when required, return an error message in the following format:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<action type="tttt">
    <error>
        <code>nnnn</code>
        <msg>error message</msg>
    </error>
</action>
```

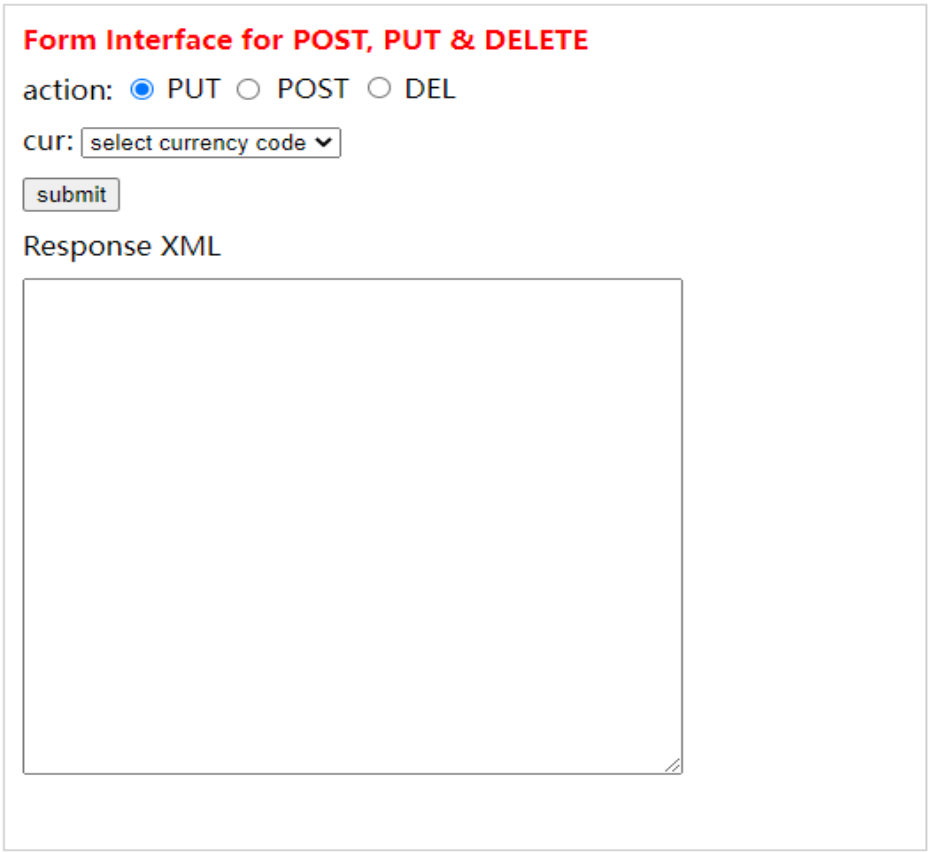
Error codes and messages are as follows:

Code	Message
2000	Action not recognized or is missing
2100	Currency code in wrong format or is missing
2200	Currency code not found for update
2300	No rate listed for this currency
2400	Cannot update base currency
2500	Error in service

Note - if the action (method) is missing then the action type attribute will have the value of "NULL".

Part 3: Build a form interface to test the service. (20 marks)

Design and implement a HTML5/JavaScript form interface like the one illustrated below to call & test the services above.



You should use AJAX to show the response on the same form when a PUT, POST or DELETE action is initiated through submitting the form.

Part 4: Critical Evaluation (10 marks)

Explain and discuss the following 3 areas in respect to the work attempted above (around 800 words.):

1. What were your learning outcomes in respect to this project?
2. How might the application you have built be extended and improved?
3. How could the application and/or its components promote re-use?

Part 5: ZIP file submission (10%)

You must submit a **valid** zip file which must contain all code, markup and reports unzipping to a directory called atwd1/assignment/.

** **Note** - this file is the only file acceptable as submission and is essential for marking. If this file is missing or not properly structured, your assignment cannot be marked.

Resources

The International Standards Organisation (ISO) has a complete list of currencies (ISO 4217) in XML format here (as of October 1, 2021): https://www.six-group.com/dam/download/financial-information/data-center/iso-currrency/amendments/lists/list_one.xml. This will be the standard and only recognized currency list required for this project.
Several free API's are available for getting current currency rates. Both Fixer https://fixer.io/ &
CurrencyLayer https://currencylayer.com/ provide free API's allowing up to 100 requests per month. Since the assignment requires our rates to be updated every 8 hours - you will not need to make more than about (3*31) 93 requests a month. You will need to sign up and acquire an API key.

What do I need to do to pass?
The pass mark is 50%.

How do I achieve high marks in this assessment?
We are looking for a well-constructed design transformed into a complete and valid implementation. The service should work as required for all GET, POST, PUT and DELETE requests with the URL starting with:

`http://localhost/atwd1/assignment/`

All tests will be run using this URL at the start and then adding further parameters.

The FORM interface should make use HTML5 and AJAX to call and display all returned messages.

Well structured, commented code making good use of user defined functions will earn extra marks.

The final report should reflect on the tasks undertaken, the problems encountered, and the solutions found. There should be reflective section that considers how the application could be extended. You will make use UWE/Harvard referencing if any external resources are referenced.

How does the learning and teaching relate to the assessment?
The lectures and particularly the workshops will guide you on each of design and implementation tasks. All teaching will be completed before the assignment is due for submission.

What additional resources may help me complete this assessment?
You will find relevant material in the lectures and worksheets. You can also make use of LinkedIn Learning for hands on lessons and practice.

What do I do if I am concerned about completing this assessment?
UWE Bristol offer a range of Assessment Support Options that you can explore through this link, and both Academic Support and Wellbeing Support are available.
For further information, please see the Academic Survival Guide.

How do I avoid an Assessment Offence on this module? [2]

Use the support above if you feel unable to submit your own work for this module.
Avoid collusion and explain things in your own words (not those of a machine).

Marks and Feedback

Your assessment will be marked according to the following marking criteria.

You can use these to evaluate your own work before you submit.

Criterion	<39	40-49%	50-59%	60-69%	≧70%
Task 1: Design, Build & Implement a Conversion Microservice (40 marks)	Service is not functional and does not return the expected XML or JSON. Little evidence of overall effort.	Service is not fully functional but works to an extent. Some effort is apparent.	Service works as expected returning XML and/or JSON. Error messages are properly encoded. May contain some minor errors.	Service works fully with all messages, both error and currency conversion returning as expected. Code might have been more optimal or is inadequately structured/commented in places.	Service is fully implemented with all error and conversion messages 5returning the correct format and content. Code is optimal (quick) and is well structured and commented. Good use is made of user defined functions.
Task 2: PUT, POST & DELETE implementation (20 marks)	URLs for GET, PUT, POST & DELETE not implemented or dot work without major errors.	All URLs for GET, PUT, POST & DELETE work reasonably well and return expected messages. Some minor errors in places.	All URLs for GET, PUT, POST & DELETE work as expected returning the correct messages.	All URLs for GET, PUT, POST & DELETE work as required with no errors. Reasonable quick with well-structured code.	All URLs for GET, PUT, POST & DELETE work as required with no errors. Optimal with well-structured and commented code. Good use of user defined functions.
Task 3: Build a form interface to test the service.	Not working or barely working. Lacking good use	Works reasonably well although some errors apparent.	A good implementation making use of HTML, JavaScript and AJAX.	A complete implementation making good use of HTML, JavaScript and	An excellent implementation that is functionally complete.

(20 marks)	of HTML and Javascript. AJAX not implemented. Perhaps a PHP rather than a HTML/JavaScript implementation.	Reasonable use of HTML and JavaScript and an attempt has been made to implement AJAX. May not be fully functional.	May have some minor errors in functionality and/or message formats.	AJAX. Messages return as expected overall. Code may lack sufficient comments or good structure.	Good use of HTML, JavaScript and AJAX. Messages are accurately formatted and return as expected. Code is well structured and commented.
Task 4: Critical Evaluation (10 marks)	Inadequate report that is short and which does not meet the requirement. Shows a lack of effort.	Reasonable report although lacking detail in places. Discussion of extending the application is limited in places. Learning outcomes not obvious.	An adequate report with some discussion of the problems encountered and solutions implemented. Incudes some discussion of extending the application.	A good report with adequate discussion of problems and solutions. Some discussion of learning outcomes. A discussion of extending the application is included.	An excellent and complete report with detailed discussion of problems, solutions and the learning outcomes achieved. A detailed discussion and reflection on how the application could be extended and re-used..
Task 5: ZIP file submission (10 marks)	ZIP file present but does not contain required files when unzipped.	ZIP file is present and unzips to the required files. Naming convention for file names not adhered to.	ZIP file is present and unzips to the required files. Naming convention adhered to.	ZIP file is present and unzips to the required files. Naming convention adhered to.	ZIP file is present and unzips to the required files. Naming convention is fully and accurately adhered to.

1. In line with UWE Bristol's Assessment Content Limit Policy (formerly the Word Count Policy), word count includes all text, including (but not limited to): the main body of text (including headings), all citations (both in and out of brackets), text boxes, tables and graphs, figures and diagrams, quotes, lists.
2. UWE Bristol's UWE's Assessment Offences Policy requires that you submit work that is entirely your own and reflects your own learning, so it is important to:
    - Ensure you reference all sources used, using the UWE Harvard system and the guidance available on UWE's Study Skills referencing pages.
    - Avoid copying and pasting any work into this assessment, including your own previous assessments, work from other students or internet sources
    - Develop your own style, arguments and wording, so avoid copying sources and changing individual words but keeping, essentially, the same sentences and/or structures from other sources
    - Never give your work to others who may copy it
    - If an individual assessment, develop your own work and preparation, and do not allow anyone to amend your work (including proof-readers, who may highlight issues but not edit the work).

When submitting your work, you will be required to confirm that the work is your own, and text-matching software and other methods are routinely used to check submissions against other submissions to the university and internet sources. Details of what constitutes plagiarism and how to avoid it can be found on UWE's Study Skills pages about avoiding plagiarism.