COMP S362F Final Individual Project

Question paper

In this project, you build an online shopping application with web service access. In addition to the server program, you also develop associated tests and documentation.

Requirements

The functional requirements (R1 to R4) and non-functional requirements (R5 to R8) of the project are described in the following.

R1. Application and data setup

The server program manages a catalog of at least 10 products. Each product has these attributes:

- Product ID, an integer of zero or a positive value
- Product description, a string (please use short and meaningful descriptions like “apple” and “orange”)
- Unit price, an integer of zero or a positive value
- Quantity in stock, an integer of zero or a positive value

Product ID, product description, and unit price are fixed throughout the execution of the program; thus, these attributes can be hard-coded or defined as constants in code or stored in a database or file (discussed later). The quantity in stock attribute varies in the program execution and should be stored in a database or file.

R2. Web service APIs

The server program provides web services for managing the product data. The web services are REST-like and use JSON formatted data. There are three required operations:

- Query a product: This operation takes the input of a product ID, and returns all attributes of the product.
- Buy a product: This operation takes the inputs of a product ID, a quantity (integer) to buy, and a credit-card number (a string of 16 digits), updates the quantity in stock if appropriate, and returns either:
    - a success status and the amount deducted from the credit card, when the quantity in stock is sufficient, i.e. greater than or equal to the quantity to buy; or
    - a failure status and the failure reason, when the quantity in stock is insufficient, i.e. smaller than the quantity to buy.
- Replenish a product: This operation takes the input of a product ID and a quantity (integer) to replenish the stock of the product, updates the quantity in stock if appropriate, and returns a status.

In addition to the results of the operations stated above, all server responses include a server execution ID with the key "exe_id" (described below).

The server is required to validate the input data of an operation. If an input product ID does not exist in the server, return the HTTP status code 404 and a reason in the response data. If some required input data are missing or invalid, return the HTTP status code 400 and a reason in the response data.

Design the JSON formatted requests and responses of the operations, and document the formats in the README file (described below).

R3. Data persistence and concurrency access

The product quantities in stock are persisted, i.e. they are saved in storage and remain when the server program restarts. These quantities can be saved in a database or a file in plain text format or Python pickle format etc. In case you choose to use a database, use the built-in SQLite only. (It is optional whether to persist the other product attributes – ID, description, price – or keep them fixed in code.)

Implement mutual exclusion to avoid race conditions when updating the product quantities in stock. Do this for quantities kept both in any program variables and in storage. You are required to implement mutual exclusion in your code, and assume the database system and file system provide no concurrency facility.

R4. Server execution ID

The server uses an execution ID that is unique for each execution of the server program. This execution ID is included in all web service responses using the key "exe_id".

When the server program starts, obtain the execution ID as follows:

1. Obtain a time string using the TCP daytime service from `time-a-g.nist.gov` (or another server listed in https://tf.nist.gov/tf-cgi/servers.cgi).
2. Convert the time string to bytes, and find the SHA256 checksum of the bytes using `hashlib.sha256()`. A string of the checksum's hex digits is the server execution ID.

This is some sample code of obtaining a SHA256 checksum and its hex digits:

```
import hashlib
a_string = "a string"
exe_id = hashlib.sha256(a_string.encode("utf-8")).hexdigest()
print(exe_id)
# "c0dc86efda0060d4084098a90ec92b3d4aa89d7f7e0fba5424561d21451e1758"
```

R5. Unit tests

Use automated testing to show that the server program fulfills the functional requirements.

Some testing items are described below; write one or more unit test cases for each of these items. In addition, add other suitable testing items of your own.

- A query about a product returns the correct product attributes.
- Buying a product with sufficient stock in the server succeeds and the quantity in stock is updated.
- Buying a product with insufficient stock in the server fails and the quantity in stock remains unchanged.
- Replenishing a product updates the server's quantity in stock.
- When the product ID does not exist, the server returns the 404 status code.
- When some required input data are missing or invalid, the server returns the 400 status code.
- If two requests for buying the same product arrive almost simultaneously and the quantity in stock is insufficient for the second request, the server must not mistakenly fulfill the second request.

R6. Code style, comments, and organization

Maintain good code style in your programs. For example, use proper and meaningful names for variables and functions, set a proper line length e.g. 78, adopt tidy code format and indentation, remove unnecessary code, and so on.

Write appropriate comments. Remember that lengthy unfocused trivial outdated comments are as bad as the lack of comments.

Organize the code within a program file or into multiple program files as appropriate.

R7. Documentation and submission format

Create a README file that documents the application beyond code comments. It may be either `.txt` file or a `.docx` file. The contents of the README file include the following:

- Your full name and 8-digit student number
- A list of file names and brief descriptions of the submitted files
- Instructions for setting up and executing the application
- Instructions for setting up and executing the unit tests
- JSON formats of the web service requests and responses
- Discussion of adopting advanced technologies (discussed below)
- (Any other relevant topics you like to include in the README file)

For the instructions for setting up and executing the application and unit tests, briefly describe what third-party libraries are used and how to install them, what data files are required and how to initialize them, the command to execute the application or unit tests, and so on.

Put all your files into a ZIP file called `s12345678final.zip` where `12345678` is your student number. Submit only that one ZIP file to OLE.

R8. Discussion of adopting advanced technologies

Select two technologies (or techniques) covered in units 11 to 13 (message queues, async programming, high-performance computing) and discuss their adoption in the application. Specifically, describe the benefits, new or improved functionality, and high-level implementation of adopting the technologies to the application. You may consider a scaled-up version of the application in the discussion, such as having a larger catalog of products, more web service operations, higher transaction volumes, and the like.

Try to provide your own unique opinions and discussion that pinpoints the specifics of the application. Common and general views would not be sufficient for a high score in this discussion.

Write the discussion in the README file. You may include code segments in the discussion in the README file. Do not actually modify the application code in the Python program files.

Scoring and submission

It is easy to get a passing score but not so easy to get a high score. The 8 items (R1 to R8) in the requirements are worth roughly the same, about 10% to 15% each.

Re-submission in OLE before the deadline, within the limitations of the OLE multiple submission function, is allowed.

Late submission will be penalized by 20% reduction of score per day. Less than a day is counted as a whole day. For example, a submission that is 24 hours and 1 second late after the deadline will have the score reduced by 40%. All submitted files and submission times are considered as recorded in OLE.

You are encouraged to submit at least one day before the deadline, so that you have sufficient time to deal with submission problems and other unexpected issues. After submission, please re-enter the submission page to verify the submission status again.