COMP S362F Final Individual Project

Question paper

In this project, you build a JSON web service server, called the "project server" in this paper. In addition to the server program, you also develop associated tests and documentation. There is no standalone client program to develop, but your test cases would make client requests to the project server.

Compulsory rules

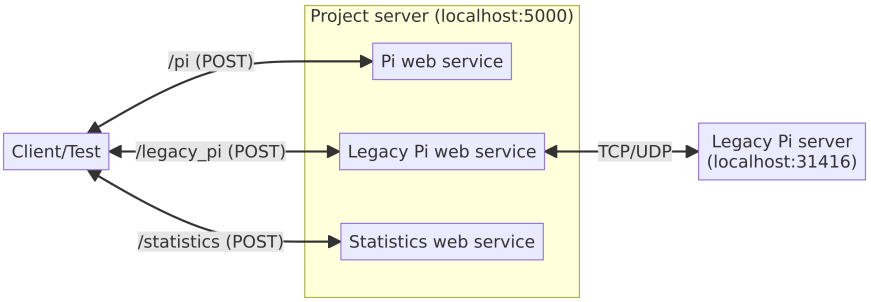
You are required to observe and obey the following rules. Violation of any of the rules would result in zero mark or severe penalty.

- You must develop the project, including all programs and documents, solely by yourself. Work and contents generated by AI or tools, or from others, must not be incorporated in part or in whole into your project submission.
- Python 3.11.x/3.12.x and Flask 3.0.x must be used to develop the project server. Other third-party libraries may be used but must be documented clearly (see R8 below).
- When the project server is executed according to your documented instructions (see R8), it must be started successfully and there must be no errors such as invalid syntax, undefined names, missing libraries, and so on.
- Submission must be made in OLE. In particular, submission, re-submission, or supplementary submission by email is not accepted. Incomplete submission, such as saved but not submitted work in OLE, or a corrupted ZIP file, is not accepted.
- Re-submission in OLE before the deadline, within the limitations of the OLE multiple submission function, is allowed.
- Late submission is 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.

Overview

The figure below shows an overview of the project server. Detailed requirements are described subsequently.



The project server supplies three JSON web services to its clients, and works with a legacy server. The legacy server is provided to you in the `legacy_pi_server.py` program. The project server runs on localhost TCP port 5000, while the legacy server runs on localhost TCP and UDP ports 31416.

Requirements

The main program of your project server must be called `s12345678_server.py`, and your test program must be called `s12345678_test.py`, where `12345678` is your student number.

The functional requirements (R1 to R5) and non-functional requirements (R6 to R9) of the project are described in the following.

R1. The *Pi* web service

Implement the *Pi* web service. The project server supplies the *Pi* JSON web service at the URL pattern `"/pi"` with the HTTP POST method. This web service performs Monte Carlo simulations to calculate the value of π. See Lab 1 solution for an explanation of calculating π using Monte Carlo simulations.

The request JSON data of the *Pi* web service contains the following fields:

Data field	Key	Value data type and format	Error handling
User name (required)	"username"	(See R4)	(See R4)
Password (required)	"password"	(See R4)	(See R4)
Number of simulations (required)	`"simulations"`	Integer from 100 to 100,000,000, both inclusive	If the key is missing or the value is invalid, return response code 400 and an error message of `"missing field simulations"` or `"invalid field simulations"`, respectively.
Level of concurrency (optional, default to 1)	`"concurrency"`	Integer from 1 to 8, both inclusive	If the value is invalid, return response code 400 and an error message of `"invalid field concurrency"`.

The level of concurrency specifies the client's desired number of threads or processes used for fulfilling the request. You are required to design, implement, and document a concurrency solution suitable for the task, such as whether to use threads or processes, which Python API to use, and so on.

If the user name and password in the request are valid, the web service replies a response of JSON data with the following fields:

Data field	Key	Value data type
Number of simulations, same value from request	`"simulations"`	Integer
Level of concurrency, same value from request	`"concurrency"`	Integer
Calculated value of π	"pi"	Floating-point number
Measured time for processing the request, in ms	`"execution_time"`	Floating-point number

See R4 for how to check the user name and password and respond when they are invalid.

R2. The *Legacy Pi* web service

Implement the *Legacy Pi* web service. The project server supplies the *Legacy Pi* JSON web service at the URL pattern `"/legacy_pi"` with the HTTP POST method. This web service obtains multiple π values from a legacy server, the provided `legacy_pi_server.py` program, and returns a average π value to the client.

The legacy server works like a daytime server, except that it returns text of π values instead of server times. It serves both TCP and UDP clients: when a TCP client connects, it returns a text line of an estimated π value and closes the connection; when a UDP packet arrives where the packet source address is the client address, it sends a packet to the client containing a text line of an estimated π value. Do not modify the `legacy_pi_server.py` program.

The legacy server sometimes fails to work and returns corrupted results that contain some non-digit and non-period characters, e.g. "3.111o35708114255", "r.093063841105263". The web service response contains the number of valid results from the legacy server. Besides, when there are 1 or more valid results from the legacy server, the response contains the average of these valid results as the returned π value; when there are no valid result from the legacy server, the returned π value is 0.

The request JSON data of the *Legacy Pi* web service contains the following fields:

Data field	Key	Value data type and format	Error handling
User name (required)	"username"	(See R4)	(See R4)
Password (required)	"password"	(See R4)	(See R4)
Protocol (required)	`"protocol"`	String, either `"tcp"` or `"udp"`	If the key is missing or the value is invalid, return response code 400 and an error message of `"missing field protocol"` or `"invalid field protocol"`, respectively.
Level of concurrency (optional, default to 1)	`"concurrency"`	Integer from 1 to 8, both inclusive	If the value is invalid, return response code 400 and an error message of `"invalid field concurrency"`.

The protocol specifies whether TCP or UDP is used by the project server to communicate to the legacy server.

The level of concurrency specifies the client's desired number of concurrent TCP/UDP operations to the legacy server for fulfilling the request. You are required to design, implement, and document a concurrency solution suitable for the task, such as whether to use threads or processes, which Python API to use, and so on.

As an example, to serve a client request with protocol TCP and level of concurrency 4, the project server makes 4 concurrent TCP connections to the legacy server to obtain 4 results, combines the 4 π values, and sends the average of those valid values in the response to the client or 0 if all values are invalid.

If the user name and password in the request are valid, the web service replies a response of JSON data with the following fields:

Data field	Key	Value data type
Protocol, same data from requset	"protocol"	String
Level of concurrency, same value from request	`"concurrency"`	Integer
Number of valid results from legacy server	`"num_valid_results"`	Integer
Average value of π or 0	`"pi"`	Floating-point number
Measured time for processing the request, in ms	`"execution_time"`	Floating-point number

See R4 for how to check the user name and password and respond when they are invalid.

R3. The *Statistics* web service

Design and implement the *Statistics* web service. The project server supplies the *Statistics* JSON web service at the URL pattern `"/statistics"` with the HTTP POST method. This web service returns the current statistics of the project server. The statistics are described in R5.

The request JSON data of the *Statistics* web service contains the following fields:

Data field	Key	Value data type and format	Error handling
User name (required)	"username"	(See R4)	(See R4)
Password (required)	"password"	(See R4)	(See R4)

You are required to design and docment the response JSON format containing the statistics for requests with valid user name and password.

See R4 for how to check the user name and password and respond when they are invalid.

R4. User information in web service requests

Check user information on all web service requests.

All web service requests to the project server are required to contain user name and password in the request JSON data. The requirement and error handling on these data are described in the following table.

Data field	Key	Value data type and format	Error handling
User name (required)	`"username"`	String of 4 digits, e.g. `"0000"`, `"0123"`, `"3721"` (Some invalid examples are `1352`, `"a123"`, `"12345"`.)	If the key is missing or the value is invalid, return response code 401 and an error message of `"user info error"`.
Password (required)	`"password"`	String containing the user name followed by "-pw", e.g. `"0123-pw"` for the user name `"0123"`	If the key is missing or the value is invalid, return response code 401 and an error message of `"user info error"`.

When a request arrives, the project server first checks the user information. If the user information is invalid, the project server returns a response of JSON data with an `"error"` key and an error message value:

Data field	Key	Value data type
`"user info error"`	`"error"`	String

If the user information is valid, the project server updates the request statistics (see R5), and proceeds to process the request (see R1, R2, R3).

Note that there is no need to store the user names and passwords in a file or database. Their processing and validation are implemented in code and logic in the program.

R5. Statistics persistence and concurrency access

Keep track of and persist web service request statistics. The statistics include the users who have made requests and the numbers of requests made; users who haven't made a request are not included.

The project server maintains and persists these request statistics, which are saved in storage and remain when the project server restarts. These statistics can be saved in a database, or in 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. Design the format of a record/line, which contains the user name (a string of 4 digits) and the number of requests made (an integer greater than 0).

Implement mutual exclusion to avoid race conditions when updating the request statistics. Do this for data 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.

R6. Testing the project server

Implement automated integration/system testing, in a test program, to show that the project server fulfills the functional requirements.

Some testing items are described below. You are required to write one or more test cases for each of these items, and to add other suitable testing items of your own.

- When the user information is missing or invalid in a request, the server returns the 401 status code with a proper error message.
- The *Pi* web service returns an approximate value of π.

- The *Legacy Pi* web service returns an approximate value of π or 0.
- The *Statistics* web service returns the current statistics of the project server.
- For the *Pi* web service, a higher level of concurrency generally has a smaller processing time than a lower level of concurrency.
- For each web service, when a field is missing or invalid in a request, the server returns the 400 status code with a proper error message.

R7. Code style, comments, and organization

Maintain good programming style in your code. For example, use proper and meaningful names for variables and functions, set a proper line length e.g. 78, adopt tidy consistent 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.

R8. Documentation and submission format

Create a README file that documents the project beyond code comments. It may be a `.txt`, `.md` or `.docx` file, with the file name `s12345678readme.txt` or `s12345678readme.md` or `s12345678readme.docx`, where `12345678` is your student number. 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 project server program
- Instructions for setting up and executing the test program
- JSON format of the *Statistics* web service (see R3)
- Description and justification of the concurrency solutions in the *Pi* and *Legacy Pi* web services (see R1, R2)
- Discussion of adopting advanced technologies (see R9)
- (Any other relevant topics you like to include)

For the instructions for setting up and executing the project server and test programs, describe what third-party libraries and their versions are used, how to install them, what data files are required, how to initialize the data files, the command to execute the programs, and so on.

Put your document, program, and data files into a ZIP file called `s12345678final.zip` where `12345678` is your student number. Do not put third-party libraries in the ZIP file. Submit only that one ZIP file to OLE.

R9. 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 project server. Specifically, describe the benefits, new or improved functionality, and high-level implementation of adopting the technologies to

the project server. You may consider a scaled-up version of the project server in the discussion, such as having more web service operations, higher level of concurrency, higher transaction volumes, and the like.

Try to provide your own unique opinions and discussion that pinpoints the specifics of the project. 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 code of the project server program files for this discussion.

❦❦❦