Enrollment process 2024-10-C_shp-149UNG

Software Division

Qualification test

NUMETO

October 2024

PRACTICAL CASE

Case name: **Transactions optimizer for better performance.**

Numeto company has an online transactional system, which collects money deposits, and allocates those to diverse sources to process them.

Some sources must be processing gateways like Authorize.Net and ApplePay. The way in which the third parties communicate with our system is by its own API.

Issue:

The system was defined to process hundreds of transactions per minute; nevertheless currently, we are processing thousands. This fact brings timeouts to the customer's transactions due to the assumption of overloaded hardware and halted resources.

Due to that, IT infrastructure decided to shift the servers to better hardware in the Cloud and implement Network Load Balancers (NLB).

Nevertheless, the marketing department is expecting to increase the volume ten times more in the next two quarters. Then the snowball effect situation is knocking on our door.

The main technical architect realized that rather than a hardware issue, this is a problem in the program managing the synchronization of the processes. There are no threads managing the requests in the queue and if one transaction is being updated twice (or more) simultaneously, the system is entering into a deadlock situation.

Solution to present:

Taking into consideration that the basis of transactions scheme has: customer full name, PAN (personal account number, such as Credit Card number or unique identifier in wallets systems or Crypto Wallets) and extra information to process (expiration date, CVV, etc.), present a simple solution [1] to handle timeouts in the application from an architectural level.

The focus of the work must be just on the algorithm to manage the data structures on by the network problem, since this hypothetical case must be surrounded by external issues that are not part of the evaluation. There is not a single solution and been creative or *thinking out of the box* is what is pursued.

Output (expected deliverable):

1) Define an architectural diagram of the solution. You can use any editor you want, but lucidchart.com and miro.com are two good online options; as well as a standalone solution is Visual Paradigm and free of cost as well.
2) Create the associated .NET project presenting the solution. Add comprehensive comments and use a clean code. Code in such a way that QA (unit testing, integration testing, so on) and continuous integration (CI) concepts are incorporated.
3) At least you need to use one design pattern in your solution, try to avoid singleton.
4) Do a small presentation of the project solution, explaining two levels: high-level architectural solution and technical details to achieve it. If you like to record your presentation using video or audio recorder software is also a plus. If you do so, then append the media files into your solution package.
5) Zip all and send it to us by email. If your solution is too heavy on size, you can let us know and we will find an alternative solution for deployment.

[1] Simple solution means on this context an algorithm / code which elegantly handle the issue, not necessarily means that solution is simplistic. Also, it aims that solution must be focused on the matter of discussion and for sure not a full fledge working from A-to-Z business solution.

6) Deadline Friday 25th October 2024. After the deadline, no posts will be taken.

Additional points to take into consideration (bonus points):

1. If API for end-user interface is involved, then use a BackendLess API for deploying the interface (this is not mandatory, but it yields extra points).
2. Extra points if you describe the API using a Swagger or homologous solution (advise use editor.swagger.io).
3. Extra points if you use Micro-service architecture (if you can justify another design pattern, it also can yield extra points).
4. Multimedia usage on presentation.

Exclusions of the solution:

1) Include only .NET code in the solution project (if end-user API is done, remember to do only the server part, front-end is not required).
2) Database parts are not necessary to include on those code snippets where the database connection is needed add some comments on this way:

   /* Here is where database update will come in this particular transaction */

3) API integration with any third party is not required, in such cases do the same comment way that done in DB updates.

Code clean and good!

Appendix α.

Evaluation matrix.

Item	**Yield Points**
Core algorithm / solution	20
Data structures handling	10
Simplicity of solution	10
Diagram	5
Pure code evaluation	30
QA standards applied	10
Notes applied	5
Presentation	10
Total	**100**
Bonus	
Using multimedia on presentation	5
APIs usage	5
Swagger or homologous description usage	5
Micro-service implementation	10