IUGO

Server Developer Assessment

You will need to set up your own LAMP stack at a minimum. If you already have one, feel free to use that as well. Hint: AWS has a wide variety of free tier services that are sufficient for this. With that stack, you are tasked to create a variety of API endpoints given some basic direction of the purpose, expected input and output formats.

Your code will be reviewed, and should be designed with extensibility in mind.

If you are unable to complete all tasks, do submit the progress you reached for consideration along with details on what works and what doesn't.

Please read the follow Criterias **CAREFULLY**.

**General Criteria**

* You have **3 days** to complete this assessment. A due date will be communicated to you on delivery of this assessment.
* Use an Object-oriented programming approach with PHP.
* No PHP frameworks allowed (Zend/etc).
* All endpoints must always give a JSON response.
* Any error response should contain in the JSON response:
  + {"Error":true,"ErrorMessage":"Reason for failure"}
* Unless otherwise specified, error messages are at your discretion.
* Your endpoints will be tested electronically, so please keep your server up for at least **7 days** after you submit your assessment.
* Please provide the **URL** of your server and submit a zip or tarball containing **all** of your code.
* Any User IDs passed as data can be assumed to be valid for the purposes of these tests.
* Write down any assumptions that you made for directions that were not specific enough.
* Is there anything you would have done differently, given more time?

**Endpoints**

**1. Timestamp**

Endpoint: http://{hostname}/Timestamp

The simplest of endpoints which should return the current time in Unix Timestamp format.

Input: A GET request to /Timestamp with no parameters.

Output: Returns the current unix timestamp in the format:

{"Timestamp":<integer>}

**2. Transaction Recording**

Endpoint: http://{hostname}/Transaction

An endpoint to record simple user transactions involving game currency with hash verification. Duplicate transactions should respond with an error, regardless of any other data passed in.

Secret Key:

NwvprhfBkGuPJnjJp77UPJWJUpgC7mLz

Input: A POST request to /Transaction with JSON input in the format:

{

"TransactionId": <integer>,

"UserId": <integer>,

"CurrencyAmount": <integer>,

"Verifier": <string>

}

Verifier is valid if the SHA-1 hash of the following values concatenated together in the order given matches the Verifier string passed in:

-> Secret Key, Transaction ID, UserId, CurrencyAmount

Sample Input (with correct Verifier):

{"TransactionId":1,"UserId":2,"CurrencyAmount":3,"Verifier":"fd6b91387c2853ac8467bb4d90eac30897777fc6"}

Output: Returns success if TransactionId has NOT already been logged by this endpoint, and if the Verifier string is valid. Format:

{"Success":true}

**3. Transaction Data Querying**

Endpoint: http://{hostname}/TransactionStats

An endpoint to query basic details about the transactions recorded for a specific UserId from the Transaction endpoint. Endpoint must calculate the number of unique transactions and the total sum of currency logged for a specific UserId.

Input: A POST request to /TransactionStats with JSON input in the format:

{

"UserId": <integer>

}

Output: Returns the UserId passed in along with their unique transaction count and currency sum:

{

"UserId": <integer>,

"TransactionCount": <integer>,

"CurrencySum": <integer>

}

**4. Leaderboard Score Posting**

Endpoint: http://{hostname}/ScorePost

An endpoint which accepts and records scores posted for a user and calculates their rank in the leaderboard. A UserId's score should only update if the score posted for that UserId and LeaderboardId was HIGHER than all previous scores posted for that UserId and LeaderboardId combination (higher scores are better, and ranks should start at 1).

Input: A POST request to /ScorePost with JSON input in the format:

{

"UserId": <integer>,

"LeaderboardId": <integer>,

"Score": <integer>

}

Output: Returns the UserId and LeaderboardId passed in, along with the highest score this UserId has ever achieved for the given LeaderboardId (including the one just posted), and the rank of that highest score compared to others. Format:

{

"UserId": <integer>,

"LeaderboardId": <integer>,

"Score": <integer>,

"Rank": <integer>

}

**5. Leaderboard Get**

Endpoint: http://{hostname}/LeaderboardGet

An endpoint which lists a range of entries from a specified LeaderboardId in order of their rank, along with the score and rank of a specific UserId. A higher score is better, and ranks should start at 1.

Input: A POST request to /LeaderboardGet with JSON input in the format:

{

"UserId": <integer>,

"LeaderboardId": <integer>,

"Offset": <integer>,

"Limit": <integer>

}

Output: Returns an array of entries from the LeaderboardId passed in, offset by the number of entries passed in Offset, and limited to a maximum number of entries passed in Limit. Also returns the current Score and Rank of the passed in UserId for the passed in LeaderboardId. Format:

{

"UserId": <integer>,

"LeaderboardId": <integer>,

"Score": <integer>,

"Rank": <integer>,

"Entries": [

{

"UserId": <integer>,

"Score": <integer>,

"Rank": <integer>

},

{

"UserId": <integer>,

"Score": <integer>,

"Rank": <integer>

},

...

]

}

**6. User Save**

Endpoint: http://{hostname}/UserSave

An endpoint which accepts arbitrary JSON data to persist for a UserId. If any key inside Data matches one already sent up by this request previously, the value for this key should be overridden by the passed up value. If a key previously passed up was not subsequently passed up, its data is assumed to be unchanged. Hint: Each key should be persisted separately (ie. Primary key of (user\_id, data\_key))

Input: A POST request to /UserSave with JSON input in the format:

{

"UserId": <integer>,

"Data": { <JSON> }

}

Sample 1:

{

"UserId": 1,

"Data": {

"Piece1": {

"SubData": 1234,

"SubData2": "abcd"

},

"Piece2": {

"SubData": {

"SubSubData": 5678

}

}

}

}

Sample 2:

{

"UserId": 1,

"Data": {

"Piece2": {

"SubData": {

"SubSubData": 9999

}

}

}

}

Output: If data was successfully persisted:

{"Success":true}

**7. User Load**

Endpoint: http://{hostname}/UserLoad

An endpoint which will return an aggregation of all previously persisted keys and their latest data for a UserId.

Input: A POST request to /UserLoad with JSON input in the format:

{

"UserId": <integer>

}

Output: If the pass ed in UserId does not have any records, must return an empty JSON object:

{}

Sample Output for {“UserId”:1} passed up, given the two requests in UserSave above have run:

{

"Piece1": {

"SubData": 1234,

"SubData2": "abcd"

},

"Piece2": {

"SubData": {

"SubSubData": 9999

}

}

}

**Bonus**

If you did not already do so, use a NoSQL solution for persisting User data sent through UserSave.