IUGO

Server Developer Assessment

Author:

Rémi Valentie / [remi@valentie.fr](mailto:remi@valentie.fr) / 7788741472

Objectves :

To build a simplified REST API (only GET/POST Request) including a set of endpoints. The business logic is described in the specification document.

Sources :

* Sources :
  + Github : <https://github.com/lyrad/iugo-assessment>
* Demo :
  + API url : <http://iugo-assessment.valentie.fr>
  + API test scenario : http://iugo-assessment.valentie.fr/test.html
* Database access :
  + <http://phpmyadmin.rebelote.org>
  + User : iugo-assessment, passwd : NdcuPWA89HtXSJfM

Software architecture

/web/test.html :

* API test scenario, sending a set of request to the endpoints.
* Needs an human analysis (no automatic check of API Response validity)

/web/index.php :

* Autoloader : Include required php files;
* Bootstrap : application configuration (should be in ini files, should depend on environment with a real project)
  + API secret key
  + Database object
  + Endpoints list and configuration (expected parameters)
* Request object creation
  + Very simple here, only 2 request type possible (GET/POST), no advanced HTTP request functionality (cache, authorization…)
  + Should consider using an existing component, like Symfony HTTP Foundation component (<https://github.com/symfony/http-foundation>)
* Api object creation :
  + Create API object with configuration previously establish (database, inputs expected, Request…)
* API handle client request
  + API will execute routine depending on request, and return a HTTP Response object.
* Sending HTTP response to the client.

/src/Api.php :

* Act as front controller, should only contain business logic (endpoint is a method, no SQL…).
* Also filter client request inputs (security).
* Handle repositories & entities

/src/Model/ :

* Contains Entities (business objects used by business logic to perform operations). Entities allow to deal with objects in endpoint code, which is not really necessary here (could deal with data array coming from database…) but could be great if business logic gets harder.
* Contains Repositories (factories responsive for persisting/merging/deleting objects in the database)
* Contains Exceptions : Extending \Exception php built in class to get customized business exception based on business objects.

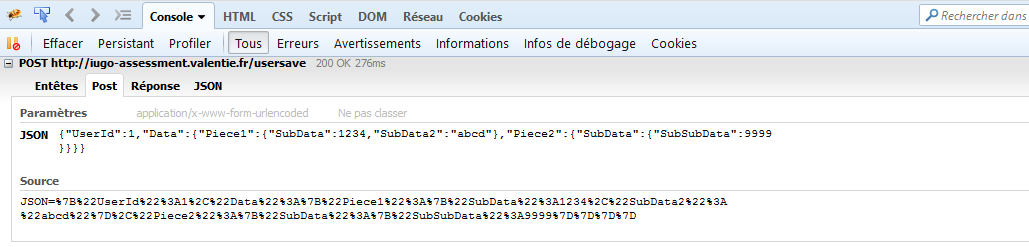
API inputs

According to the spec, for all endpoint requiring input data :

“*Input: A POST request to /<endpoint> with JSON input in the format: {<json>}*“

* Assuming that POST request must include a “JSON” index parameter.
* Parameter’s value must be a JSON formatted string, conform to the format provided

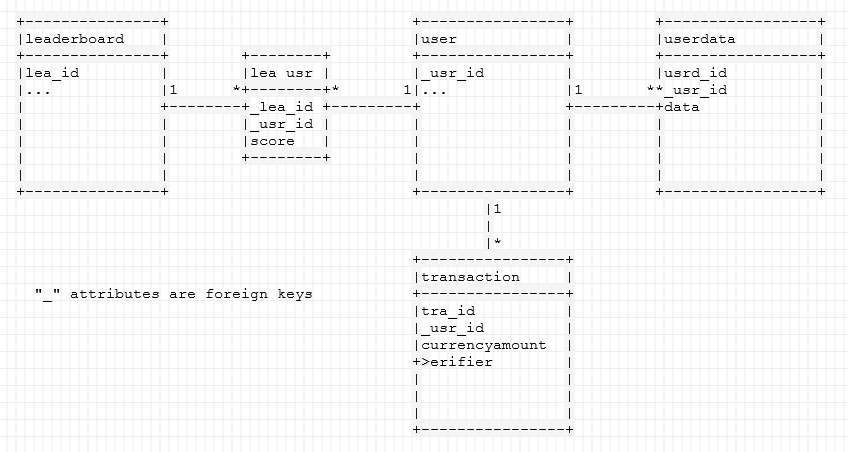
Firebug view exemple :



API endpoints

A “reset” endpoint was added, to reset database (in order to be able to launch a test senario).

Data model & business objects



|  |  |  |
| --- | --- | --- |
| Table | Business object associated | Description |
| leaderboard | Leaderboard | Leaderboard storage Table.  Only one attribute (id), could add title…  According to specs, this table may not be necessary.  But according to a relational database schema, it’s not logic to reference an object (lea\_usr => \_lea\_id) if it doesn’t exist…  So since there are no endpoints for leaderboard management (adding…), exemple database already include 12 leaderboard (id: 1->12).  Those rows are not deleted by database reset endpoint. |
| user | User | User storage table.  One one attribute (id), could add name… |
| transaction | Transaction | Transaction storage table.  Many to one (One user can have Many Transactions |
| lea\_usr | None (handled by Leaderboard object/reposirory) | Many to Many User/leaderboard join table.  A relation is characterized by to id (uers/leaderboard, foreign keys) and a score attribute.  \_lea\_id is a foreign key. Leaderboard MUST exist in leaderboard table when trying to add a score using scorePost endpoint |
| userdata | None (handled by User object/reposirory) | Data attribute for a user (ManyToOne, One user can have Many data)  Primary key : (data\_id, user\_id). ! data\_id is string ! performance issue on joined request if a lot of userdata rows in table?  Data attribute : a serialized php array (according to the spec, data is arbitrary). |

What is missing ?

* PHPDocumentor formatted comments for class/methods/attributes and PHPDOC associated.
* Improved test scenario, with automatic API response validity checking.
* Some specific error management cases have to be improved (cf. “do nothing” in some catch blocs, In repositories).
* TODO tags
* Leaderboard add endpoint
* Some repositories method are conform with business model : method getTransactionStats, in transcationReposiroty should not exists. Transactions statistics should be loaded and accessed through a user object (getuserById), since they depend on an user. If we want to keep this method for performance issues, it should at least take an user object (not a user id) as a parameter.