**Documentation for PROJECT2**

PART C - Code Comments

1. **Denote where and explain why you instantiated the database and session objects in that location**

To make sure every information will be stored in the database is correctly formatted and not duplicated, my session object codes are placed after implementing the database syntaxes.

**POST**

* Login part (api.php line63~)
* Register part: creating user account (api.php line96~)
* Register part: creating category lists (api.php line130~)
* Update category list (api.php line157~)
* Add/Remove product in cart (api.php line182~/ api.php line208~)

**GET**

* Order information for user (api.php line240~)
* Show product information (api.php line259~)
* Call cat list (api.php line281~)
* Logged in part (api.php line300~)
* Logout part (api.php line321~)

1. **Explain the mathematics of either of your two-rate limiting code.**

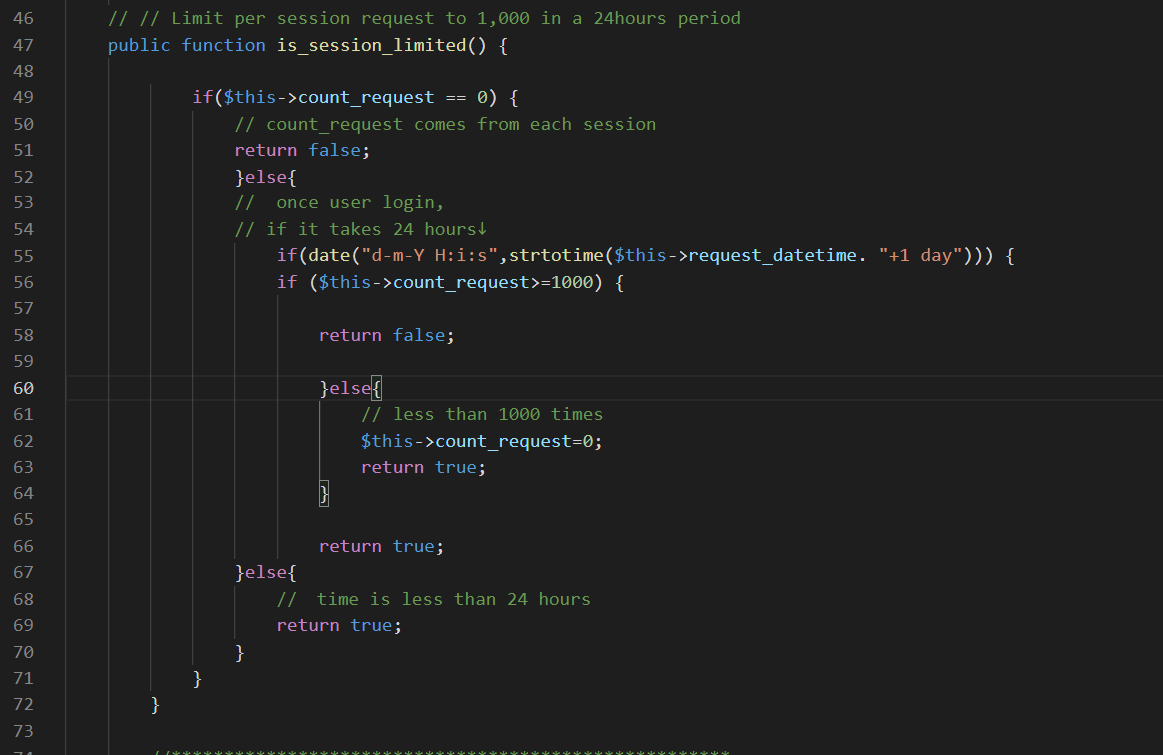
As the way to Limit per session request to 1,000 in a 24hour period, I declared the variables **request\_datetime** and **count\_request** at the top of the code in se.php. Both of variables are assigned integer 0.

When user access the app, **request\_datetime** stores the time and **count\_request** stores the number of sessions.The value of **count\_request** increases by one, once user access.

Normally, function **is\_session\_limited()** returns true, unless **count\_request** is 0 or the value in **count\_request** reaches more than 1000 times in 24 hours.

Line 54, I use strtotime() function to calculate 24 hours since the first session.

And if **count\_request** less than 1000 times in 24 hours since first session, **count\_request** will be reset and will count againfrom 1.



1. Note where you are checking if a session pre-exists, what are you doing if it does.

* Find out “Symfony\Component\HttpFoundation\Session\Session” in your php file, it should be on the top area of the file
* Write the code $session->getId(), if your session exists, you would get new session id.

1. **Explain the code structure that checks all of the GET/POST structures**

In api.php, I wrote the code that includes if statement to identify which method to transfer the information from html such as $request->getMethod() == 'POST' and $request->getMethod() == 'GET'

1. **Write a README file that explains how to setup and configure Web Service**

Readme file is in readme folder. Please find it.

1. **Write a test script that interacts with the web service to test all the known GET and POST requests as a part of Unit testing**

**POST**

* curl -d "login\_username=user1&login\_password=123123" -X POST http://localhost/match/api/api.php?action=loginmatch
* curl -d "username\_register=user1&password\_register=123123" -X POST <http://localhost/match/api/api.php?action=checkaccount>
* curl -d "categories= [7,8]" -X POST <http://localhost/match/api/api.php?action=createcate>
* curl -d "ud\_categories= [7,8]" -X POST <http://localhost/match/api/api.php?action=updatecat>
* $ curl " http://localhost/match/api/api.php?action=loginmatch " -o /dev/null -w '%{http\_code}\n' -s
* $ curl "http://localhost/match/api/api.php?action=callcatlist" -o /dev/null -w '%{http\_code}\n' -s
* $ curl " <http://localhost/match/api/api.php?action=createcate> " -o /dev/null -w '%{http\_code}\n' -s
* $ curl " <http://localhost/match/api/api.php?action=updatecat>" -o /dev/null -w '%{http\_code}\n' -s

**GET**

* $ curl "<http://localhost/match/api/api.php?action=showproduct>" -o /dev/null -w '%{http\_code}\n' -s
* $ curl "http://localhost/match/api/api.php?action=callcatlist" -o /dev/null -w '%{http\_code}\n' -s
* $ curl " http://localhost/match/api/api.php?action=isLoggedin " -o /dev/null -w '%{http\_code}\n' -s
* $ curl " http://localhost/match/api/api.php?action=logout " -o /dev/null -w '%{http\_code}\n' -s