**Backend Overall Model**

The system backend must run on a Linux operating system. The application server and the database system will work concurrently in the same machine to satisfy the proposed prototype product. The different implementation solutions are still in analysis and requires more research.

The backend will be developed using PHP. For this purpose, Apache application server will be used to serve the Webpages accordingly. The backend will be responsible to serve data from the database to the client as it is demanded.

The backend system should avoid doing any heavy computation or rendering to the client side such that most of the processing and rendering is done on client’s side. It should minimize the use of the server.

More information can be find in the backend domain folder.

**Other Factors**

Learn about PostgreeSQL:

<http://www.postgresqltutorial.com/>

Documentation:

<https://www.postgresql.org/docs/8.0/static/tutorial.html>

**Handling the number of database connections:**

<https://wiki.postgresql.org/wiki/Number_Of_Database_Connections>

**Connection Pooling and Acceleration**

<https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling#Connection_Pooling_and_Acceleration>

PgBouncer:

<https://wiki.postgresql.org/wiki/PgBouncer>

**youtube** - https://www.youtube.com/watch?v=x\_XpPbfomso

Pgpool-2:

<http://www.pgpool.net/mediawiki/index.php/Main_Page>

Apache – dynamic pool allocation code

<http://www.apachetutor.org/dev/reslist>

**Persistent connection functionality and postgresSQL**

<http://php.net/manual/en/features.persistent-connections.php>

**External Pooling**

The amount of connection to the database may be very high at a time. For each connection, several transactions may happen. It is possible to manage the number of connections to the database giving it a maximum number of connections and queueing the rest.

The postgreeSQL server does not have a connection pooler system, therefore, the polling must be performed by an intermediate software. There are 2 recommendations on the postgreeSQL documentation pgbouncer and pgpool-2. Further information about this subject please find it on the links above.

**Connection pooling and PHP**

PHP **does not** support connection pooling, instead it supports what is called persistent connection. There are several issues when using persistent connection, one of them is when the number of the limit of simultaneous connection is reached, the other connections attempting to connect should then be shut down. In our case, we would be like that those extra connections to be served too when a connection is freed. Please follow the link for persistent connection above for more detailed information.

More can be found here:

<https://stackoverflow.com/questions/39753/connection-pooling-in-php>

For PHP tuning:

<http://www.oracle.com/technetwork/articles/dsl/white-php-part1-355135.html>

Apache reverse proxy. Can be used for load balancing and or caching too.

# mod\_proxy - <http://httpd.apache.org/docs/current/mod/mod_proxy.html>

# mod\_cache - <http://httpd.apache.org/docs/current/mod/mod_cache.html>

# mod\_dbd - http://www.apache2.es/httpd.apache.org/docs/2.4/es/mod/mod\_dbd.html

Caching is a great approach for some pages of php serves. For example, it works very well for static pages that never change or for pages that does not change very often. One idea is to cache the PHP page that serves the learder-board data such that the same data is served to all requests. This page\data would then be refreshed after sometime, let’s say every 1 min, that would significantly reduce processing in the server.

Using extensions such as pgBouncer seems to resolve the issue of the implantation of a connection pool using when using PHP as our backend language. It is because pgBouncer is seen as the postgreeSQL database and all we must do is to point to the pgBouncer port instead.

**Drop connections with database after time out**

It refers to our discussion: “Dropping idle users seems to be something postgre has handled”.

It is of my understanding that after a set of transaction is performed by a connection, it should then be closed programmatically. That means that the code should explicitly call the close statement of that connection with the database. That been said, a connection to the database should not be idle at any time unless it is a connection available in the connection pool waiting to be used. Therefore, there is no need to drop the connection with the database.

If using connection pooling & queueing, some clients will be waiting to get a connection from the connection pool. It is the case that a client attempt to get a connection is timed out, this can be configured. Thus, it is important that some kind of monitoring is performed to ensure all clients are given the change to connect to the DB.

**Drop idle connection in the Application server**

By what is described in the PHP documentation, connection handling is possible and it is performed by the PHP. The default behaviour for a PHP script is to be aborted when the remote client disconnects, also when the it times out, the default timeout is 30 secs and can be changed. Please see more information at:

http://php.net/manual/en/features.connection-handling.php