Final Design Document for Practicum Chat Program

Prepared by:

Team Members:
Raymond Chamberglain
William Mason
Jerrad Allen
Elh Barry

Franklin University CS Practicum
April 12, 2008

Purpose:

This document represents a written outline of the practicum chat development. It is used to articulate how the whole system is developed. Indeed, it also gives an overall guidance of the architecture of the chat application.

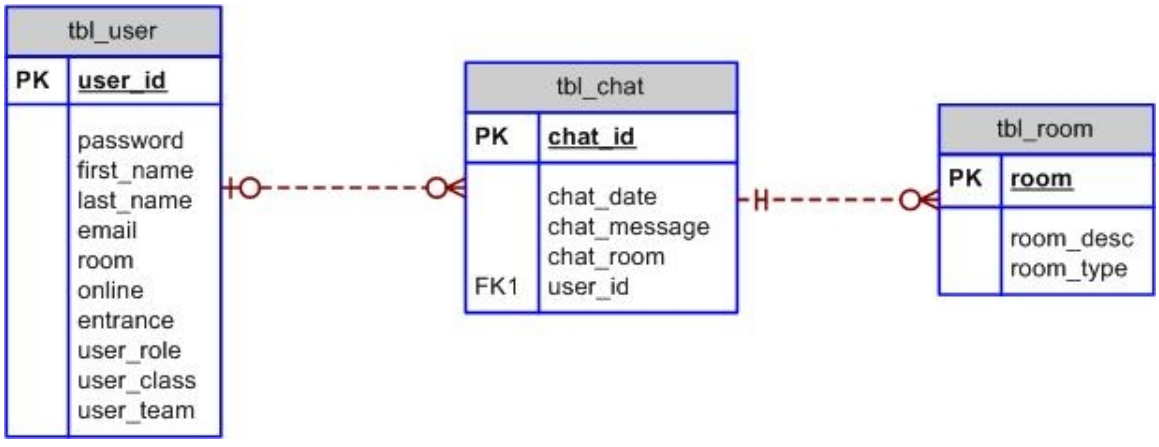
This document details all the information regarding how the chat system is put together using the object-oriented design approach and a UML showing the operation flowchart as well.

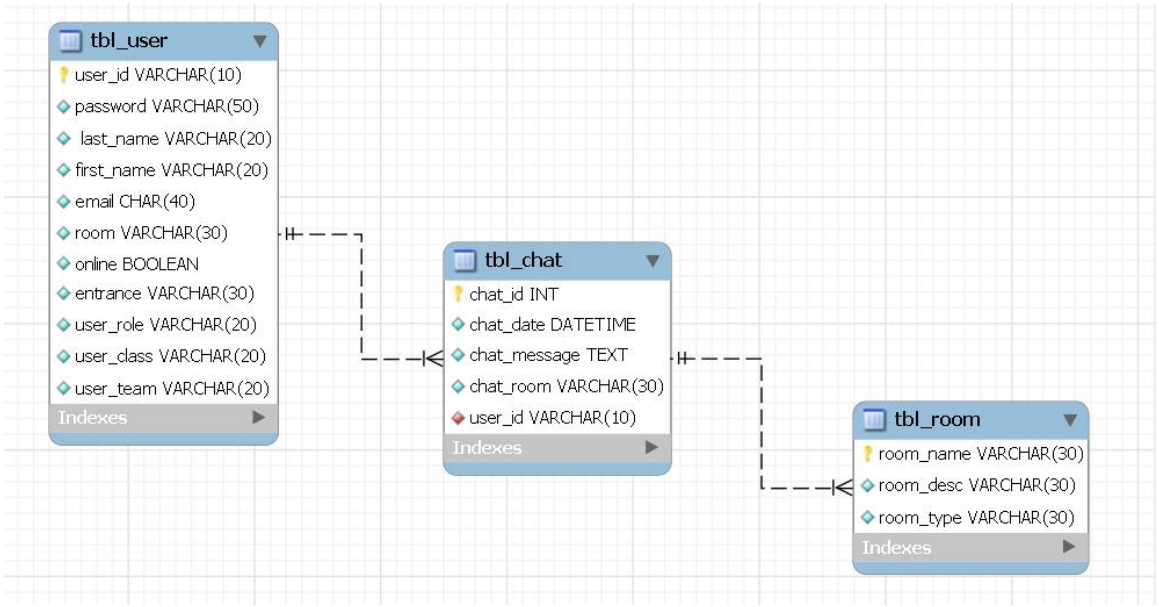
Index

Database Design Structure:

One of the first stages in the process of creating the Practicum Chat is to define and create the database for holding our tables and data.

/** **ERD Schema of the Database**:*/





/** **Description of each of the tables**:*/

1. User table: tbl_user

This table is used to hold the current settings for each student in the chat system.

Field names include:

- **user_id**: This is the primary key. This key field identifies a person in the chat system. Data validation is switched on ensuring that the field is unique, i.e. no two records have the same **user_id** filed.

- **password**: key field to hold the password of the user.

- **first_name** and **last_name:** These two fields hold the user's name. These fields are essential and could be useful in tracking someone down who abuses the system.

- **email**: The email field is not essential and is just there so that if students have any problems, then the administrator can contact them.

- **room**: The room field holds the current room in which the user is chatting. When the user logs in, the user's room is set to "none". When the user selects a chat room, this value is set to coincide with the chosen room. When the user leaves the chat room, this key is set back to "none".

- **online**: The online field tracks whether the user is currently online. This field is set by the login and logout scripts.

- **user_role, user_class,** and **user_team**: these are the fields that respectively define the user's role, the user's class, and the user's team.

2. Chat table: tbl_chat

This table is used to hold the actual messages that users send during a chat session. It uses five fields and one relationship.

Field names include:

- **chat_id**: This field is used as the primary key for the Chat table.
- **chat_date:** This field holds the date and time the message was created.
- **chat_message**: This field holds the actual text of the message.
- **chat_room**: This field identifies the room that the user was in when the message was created.
- **user_id**: (foreign key, reference to tbl_user table)

3. Room table: tbl_room

The need for this table is essential, as the system should give privilege to the administrator to create, edit, and delete chat rooms.

Field names are:

- **room** (primary key)
- **room_desc**: holds the description of the room.
- **room_type**: holds the type of the room.

UML Class Diagram:

The following class diagram represents a type of static structure diagram that describes the structure of our chat system by showing the system's classes, their attributes, and the relationships between the classes.

/ Description of the Classes */**

Class:

_database

Manages all connections to the MySQL database

Functions:

__construct

sets server configuration specific variables necessary for connecting to the database.

db_connect

creates a connection (link) to the MySQL database.

db_create

creates the database entry on the server
(usage depends on server configuration and permissions)

db_select

selects the database

db_disconnect

closes the connection (link) to the MySQL database

db_query

processes a query and returns the result

db_row_count

returns the number of rows in the result of a query

db_result

returns a specific result value based on the given query result, row number, and column name

db_fetch_array

returns an array from a query result

db_fetch_assoc
returns an associative array from the query result
(essentially the same as *db_fetch_array*)

db_create_table
creates a new table
(either from a given file or an sql string)

db_update_table
calls the function *db_create_table* passing it the same data it received.

Class:
_users
Manages user data

Functions:
__construct
sets global variables

users
queries the *tbl_user* table and returns a xml string which contains a list of users currently logged into the given chat room.

login
verifies username and password are correct
returns warning if unsuccessful
sets *$_SESSION* super global values and updates the *tbl_user* table and redirects the user to the room_change page if successful.

room_check
queries the *tbl_user* table and returns an array containing the user's role, class, and team

room_change
sets *$_SESSION* super global values and updates the *tbl_user* table with selected chat room

logout
updates the *tbl_user* table and destroys *$_SESSION* super global values

Class:

_chat

Manages chat sessions

Functions:

__construct

sets global variables

chat_send

updates the *tbl_chat* table with user submitted chat message

chat_load

calls the *chat_session_check* function to determine if a new session is being started.

If true – the system message: "__SESSION__START__" is entered to the *tbl_chat* table, and then the welcome message is returned as an XML string

If false – queries the *tbl_chat* table for the last occurrence of the system message: "__SESSION__START__" and sets the "last" variable to the resulting *chat_id* value + 1. Then the welcome message and all subsequent chat messages for the given room are returned as an XML string.

chat_get

queries the *tbl_chat* table for new chat entries for the given room then loops through the result and returns all new messages as an XML string.

chat_session_check

queries the *tbl_user* table to determine if the requesting user is the first to enter the chat room.

Returns true or false.

exit_chat

queries the *tbl_user* table to determine if the requesting user is the last to leave the chat room.

If true – the system message: "__SESSION__END__" is added to the *tbl_chat* table for the given room, then *$_SESSION* super globals and *tbl_user* table values are updated. Returns true.

If false - the *$_SESSION* super globals and *tbl_user* table values are updated. Returns false.

Class:

_room

Manages requests for available static chat rooms

Functions:

__construct

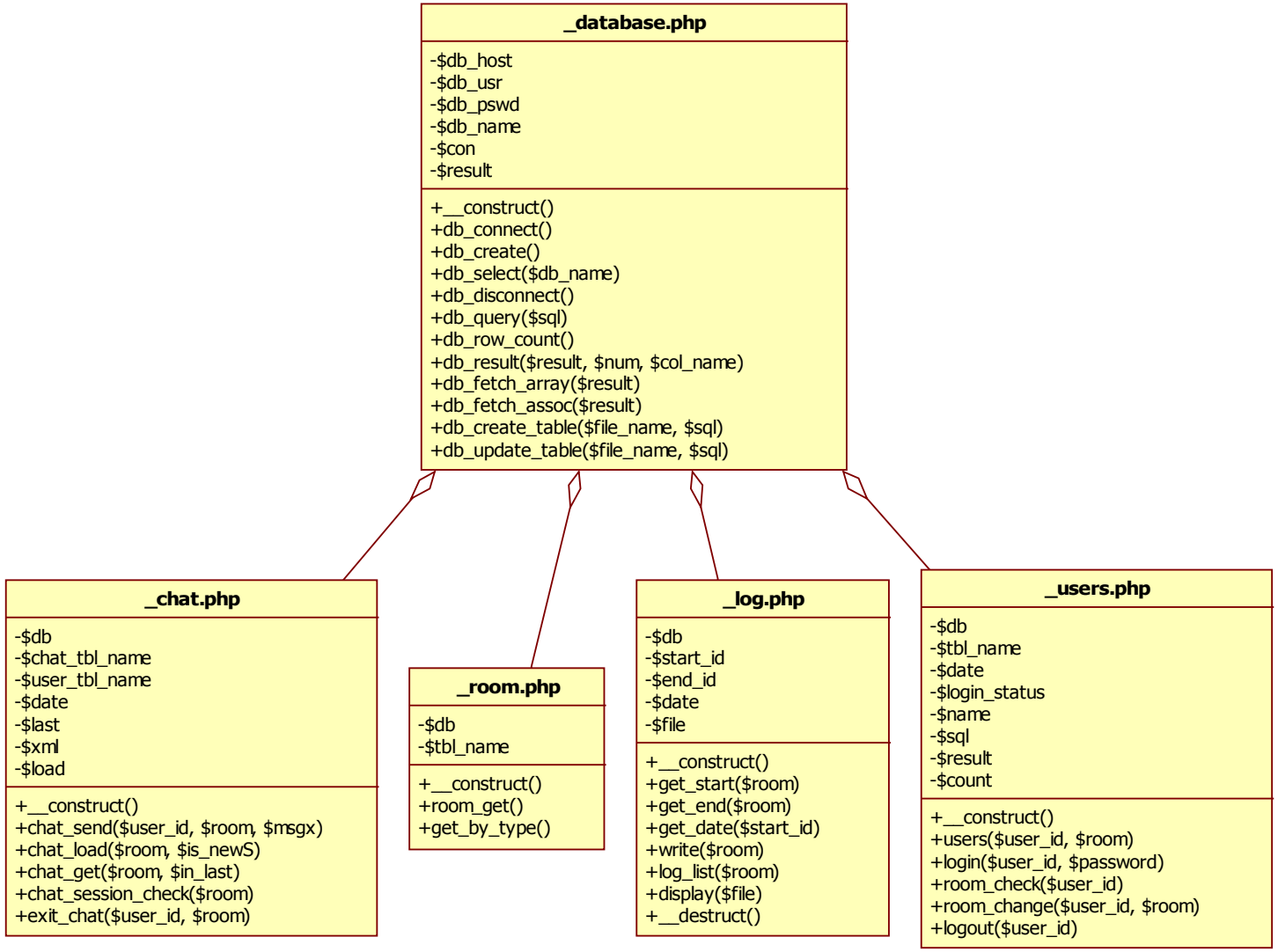
sets global variables

room_get

queries the *tbl_room* table and returns an array of all available rooms

get_by_type

queries the *tbl_room* table and returns an associative array made up of three associative arrays, one for each type of room: global, class, and team.


_database.php
-$db_host
-$db_usr
-$db_pswd
-$db_name
-$con
-$result
+__construct()
+db_connect()
+db_create()
+db_select($db_name)
+db_disconnect()
+db_query($sql)
+db_row_count()
+db_result($result, $num, $col_name)
+db_fetch_array($result)
+db_fetch_assoc($result)
+db_create_table($file_name, $sql)
+db_update_table($file_name, $sql)
_chat.php
-$db
-$chat_tbl_name
-$user_tbl_name
-$date
-$last
-$xml
-$load
+__construct()
+chat_send($user_id, $room, $msgx)
+chat_load($room, $is_newS)
+chat_get($room, $in_last)
+chat_session_check($room)
+exit_chat($user_id, $room)
_room.php
-$db
-$tbl_name
+__construct()
+room_get()
+get_by_type()
_log.php
-$db
-$start_id
-$end_id
-$date
-$file
+__construct()
+get_start($room)
+get_end($room)
+get_date($start_id)
+write($room)
+log_list($room)
+display($file)
+__destruct()
_users.php
-$db
-$tbl_name
-$date
-$login_status
-$name
-$sql
-$result
-$count
+__construct()
+users($user_id, $room)
+login($user_id, $password)
+room_check($user_id)
+room_change($user_id, $room)
+logout($user_id)

MVC / SMARTY Framework:

System Description:

This project was developed with an MVC framework. The MVC term means "Model-View-Controller". The MVC framework is the basic structure in which the role of each file is processed for this project.

For this project, the MVC is based on two important files. The first is a file named ".htaccess" and the second is named "index.php". The .htaccess is a script for the Apache web server, which is put in the root directory and processed before any other file. Within this file, there is a command to re-enter the path to the file names. The ".htaccess" tells the server to capture any file that ends with the ".php" extension and will pass the filename to the "index.php" file which initiates the MVC framework. The name of the requested file is processed by "index.php" file, which calls the default Controller.php, Model.php, and View.php containing the respective classes: Controller, Model, and View. The index.php file also checks if the requested file has a specific Controller sub-class. If it does, then that sub-class is called. The index.php creates a Controller object and pass it to the $page variable -- which contains the requested filename (with the .php extension dropped) – through the Controller's execute function.

The default Controller class creates a Model object and a View object, and then checks if the requested file name has a specific Model or View sub-class. If it does, then that sub-class is called. The default Controller also calls

the display function from the View object, passing to it both the $page variable and the created model object.

The Model class is responsible for creating and passing along the array ($arg) containing vital data for the construction of the Menus that are uniformly shown throughout the site. This class is necessary only if the page being requested is a page to be displayed on the Web Browser.

The View class assumes that any filename request ($page) passed to it is actually a template file – with and extension of ".tpl". In addition, the View class will import the SMARTY class and create a SMARTY object. All data is assigned to the SMARTY object. The View class is responsible for allocating the arrays passed from the Model class, and any other data that will be interpreted with the SMARTY tags contained in the template files. Next, the requested page is processed through the SMARTY object, and then displayed in the browser.

In case there is a sub-class for any of the three (Controller, Model, and View) classes, then these extensions have an override priority. This is useful when processing a true php script which will not be displayed.

SMARTY is an open source template engine. It is a PHP class that will process HTML code with or without SMARTY tags. A SMARTY tag is a special tag that has a similar function to PHP tag. It allows the use of loops, conditional statements, and variables within the HTML code. HTML and SMARTY tags must be contained within a SMARTY template file, which ends with the ".tpl" extension. The SMARTY template file is usually stored

in a folder named “templates”. These templates are processed by the SMARTY class, which is created as an object in the View class. Once processed, these templates, then, are stored on the server as compiled templates, which are usually placed in the “templates_c” directory folder.

The following flowchart shows the diagram for pages requested within the MVC / SMARTY framework.

Basic MVC flowchart
This chart shows the flow for Pages requested within the MVC / SMARTY framework