CO539 Coursework 1

Microblog Site

Introduction

For this piece of Coursework, you are going to use the CodeIgniter MVC framework introduced in lectures to create a simple microblog (forum) website, covering some of the basic functionality offered by twitter/facebook et al. To help you, we have created a database with all of the tables you will require, along with some seed data to get you started. The coursework is split into three parts.

- In the first part, you will create functionality allowing the messages sent by a particular user to be viewed, and create the site's search function.
- In the second part, you will add the functionality required to allow users to login, and to post messages.
- In the third part, you will add the functionality required to allow users to follow other users, and to view a combined feed of messages posted by everybody they follow.

10% of the marks are available for your site's design and code quality. This covers both the visual appearance of the site, and how concise, well laid out and well commented your code is.

To start, you will need to install the CodeIgniter framework, which is available at http://www.codeigniter.com/ . You will also need a copy of the initial database, which is available via a link on Moodle. Import the sample database into your personal MySQL database on dragon (using the mysql command) and unzip CodeIgniter into its destination folder (strongly recommended to be your submission directory), configuring it following its install instructions.

A demonstration of this coursework once completed is available at http://www.cs.kent.ac.uk/~iau/LOCAL-ONLY/messager to give you a feel for how the finished site should function.

Logins are: username: kris, ben, paul, sam or hannah
password: password

The Database

The database consists of three tables: *Users*, *Messages*, and *User_Follows*. The *Users* table contains the login details of each of the site's users, including an SHA1-hashed password. The *Messages* table contains all of the messages users have posted, using the poster's username as a foreign key. The *User_Follows* table stores "who follows who", using the username of both the following and followed user as foreign keys. A database diagram illustrating the tables, their fields and the relationships between them is pictured below.

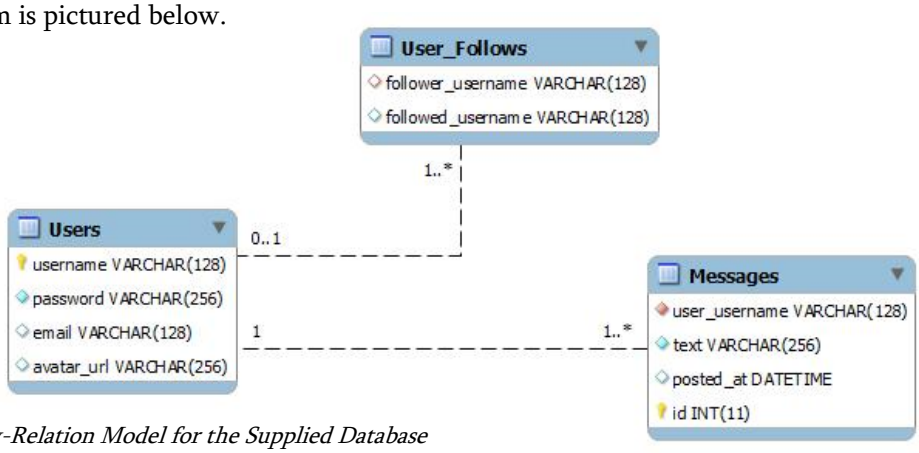

Figure 1: Entity-Relation Model for the Supplied Database

The database has been pre-populated with some users, messages and followers. The password for each user is “password”, which will be useful for testing purposes, and deliciously insecure.

Part I – Basic Functionality (30%)

Visiting your site with /user/view/kris URL segments should display all of the messages posted by user “kris” whereas visiting /user/view/ben should do likewise for “ben”

Visiting your site with a /search URL segment should display a search form. Entering a search string into the form should retrieve all the messages containing that term. The search form should submit to the search/dosearch action, passing the search string via GET.

Design

Models

Model Name	Function name	Description
Messages_model	getMessagesByPoster($name)	Returns all messages posted by the user with the specified username, most recent first
Messages_model	searchMessages($string)	Returns all messages containing the specified search string, most recent first

Controllers

Controller Name	Action Name	Description
User	view($name)	Loads Messages_model, runs getMessagesByPoster() passing the specified name, and displays output in the ViewMessages view
Search	index()	Displays the Search view
Search	dosearch()	Loads Messages_model, retrieves search string from GET parameters, runs searchMessages() and displays the output in the ViewMessages view

Views

View Name	Description
ViewMessages	Displays a list of messages, with details of poster, content and time of each message. Poster’s name should be linked to the user/view/{poster} for that user
Search	Displays a form with a search box, in which the user can enter search terms. The form should submit to /search/dosearch via GET

Part II - Logging in & Posting (40%)

Users should be able to login and logout from the site using their credentials. The login page should be at user/login and the logout page at user/logout. N.B. Users do not need to login to view messages or search.

Once logged in, a user should be able to post a message by visiting the /message page

Design

Models

Model Name	Function name	Description
Users_model	checkLogin($username,$pass)	Returns TRUE if a user exists in the database with the specified username and password, and FALSE if not. Note: passwords in the database are SHA1 hashed, so hash supplied passwords before searching the database
Messages_model	insertMessage($poster, $string)	Adds the supplied message to the messages table in the database

Controllers

Controller Name	Action Name	Description
User	login()	Displays the Login view
User	doLogin()	Loads the Users_model, calls checkLogin() passing POSTed user/pass & either re-displays Login view with error message, or redirects to the user/view/{username} controller to view their messages. If login is successful, a
		session variable containing the username is set.
User	logout()	Logs the user out, clearing their session variable, and redirects to /user/login
Message	index()	Redirects to /user/login if not logged in. Displays the Post view
Message	doPost()	Redirects to /user/login if not logged in. Loads the Messages_model, runs the insertMessage function passing the currently logged in user from your session variable, along with the posted message. Redirects to /user/view/{username} when done, which should show the user's new post

Views

View Name	Description
Login	Displays a login form for a user to supply their username and password. This form should submit via POST to /user/dologin
Post	Displays a form in which the user can write a new post. This form should submit via POST to the /message/doPost action

Part III – Following Others (20%)

When viewing another users /user/view/{username} page, and if currently logged in, a "Follow" button should appear, allowing the currently logged in user to follow this user

There should be a /user/feed/{username} page that views all of the messages, from all of the users followed by the specified user

Design

Models

Model Name	Function name	Description
Messages_model	getFollowedMessages($name)	Returns all of the messages from all of those followed by the specified user, most recent first
Users_model	isFollowing($follower,$followed)	Returns TRUE if $follower is following $followed, FALSE otherwise
Users_model	follow($followed)	Inserts a row into the Following table indicating that the logged-in user follows $followed

Controllers

Controller Name	Action Name	Description
User	view($name)	This action will need modifying to add a "follow" button or link if, after loading Users_model, the isFollowing() function indicates the currently logged in user (from your login session variable) isn't following the viewed user. This link should point to user/follow/
User	follow($followed)	Loads the Users_model, and invokes the follow() function to add the entry into the database table. Should redirect back to the /user/view/{followed} page when done
User	feed($name)	Loads the Messages_model, invokes the getFollowedMessages() function, and puts the output into the ViewMessages view.

Design & Code Quality (10%)

Although the suggested design in each of the previous sections will, if implemented correctly, be enough to get you a working micro-blogging site, it won't be particularly user friendly. For example, there won't be links atop each page to the login/logout pages, your own page, and your feed. To score well in this section, give some thought to both usability and visual design, and turn your coursework into a site people could actually use. Your code should be well laid out, commented, and readily maintainable.