**Myblog Documentation**

**Project description:**

This is a blog website. There are two parts in this application – front end and back end.

Front end will consist of blog posts. On the left side of the body, there will be a post list. On the right side of the body, there will be the post body. Post body will have post title, post content, post date, last updated, number of likes, comments.

Back end will consist of two menus – “create a post” and “view old posts”. In “create a post”, user will be able to create a new post and publish it in the front end. In “view old posts”, user will be able to view all his/her posts up to the date.

**Database Features:**

The database for this project is named “myblog”. The database contains 5 tables:

* **Users**
* **Posts**
* **Likes**
* **Tags**
* **Posts\_has\_tags**

**Users** table has 5 attributes:

* User\_id [int(11),NN,AI]
* Username [varchar(30),NN]
* Email [varchar(50),NULL]
* Password [varchar(255),NN]
* Create\_time [timestamp, CURRENT\_TIMESTAMP]

**Posts** table has 6 attributes:

* Post\_id [int(11),NN,AI]
* Post\_content [text,utf32\_unicode\_ci,NULL]
* Post\_title [varchar(100), utf32\_unicode\_ci,NN]
* Post\_time [varchar(255),NN]
* Users\_user\_id [int(11),NN]
* Post\_last\_update\_time [varchar(255),NN]

**Likes** table has 3 attributes:

* Like\_id [int(11),NN,AI]
* Like\_ip\_address [varchar(45),NULL]
* Posts\_post\_id [int(11),NN]

**Posts** table has 2 attributes:

* Tags\_id [int(11),NN,AI]
* Tag\_name [varchar(45),utf32\_unicode\_ci,NN]

**Posts\_has\_tags** has 3 attributes:

* Posts\_post\_id [int(11),NN]
* Tags\_tags\_id [[int(11),NN]

**Front End Features:**

Here are the features of front end:

**Header**: This will have a heading

**Post List**: This will show the post titles of all the blog posts of the user. If a user clicks on a title, the post body will display the corresponding post.

**Post Body**: This will show the body of the post the user loaded by clicking on a post title of the post list. This will consist of post title, post content, post date, last updated, number of likes, like button, tags and disqus comments.

**Disqus Comments**: Comments will be dynamically loaded using disqus comments. The configuration variables for disqus are:

* disqus\_shortname = 'faheemsblog'
* disqus\_url = 'http://faheemsblog.info'
* disqus\_identifier = id of post
* disqus\_title = title of post

**Back End Features**:

Here are the features of back end:

**New Post**: This tab will allow user to create a new post. User will be able to input post title, post content and post tags and then publish it. Nicedit is used for creating post content.

**Old Posts**: This tab will list all the posts the user has created. It will display a table which will have the following rows: Post id, Post Name, Published on, Likes, Edit and Delete. The Edit button will allow the user to edit a post – a modal will open with input for post title, post content (Nicedit) and post tags. The Delete button will delete a post – a modal will open for confirmation of deleting.

**MVC Features:**

This application has the following controllers:

* **Blog**
* **Login**
* **Backend**

This application has the following models:

* **Blogdb**

This application has the following views:

* **Blog**
* **Blogcontent**
* **Verifico**
* **Backend**
* **Backendc**

**Controllers:**

Description of the controllers are as follows:

**Blog**:

The blog controller controls the blog view. The blogdb model is loaded at first.

* Index: Gets post names from blogdb. Loads blog view, sends post names to the view.
* Getid: Sends post id to the view. Gets id of post from view, explodes it with delimiter “\_” and sends the post id using flash cookie.
* Insertlike: Gets post id from blogcontent view. Sends post id and ip address to insertlike function of blogdb which returns count of likes. Echo the count.

**Login**:

The login controller controls the login view. Form, URL, form validation helpers and blogdb model are loaded at first.

* Index: Sets form validation rules. Username is required, must have at least 6 characters, xss cleaning. Password is required, must have at least 6 characters, md5 hashing. If form data was not valid, view will be loaded again. Else, gets username, password from view, sends user data to verifylogin function in blogdb. If user exists, unset password, get user id using verifylogin function in blogdb, set user data in session and redirect to backend.

**Backend**:

The backend controller controls the backend and backendc views. The blogdb model and the date helper is loaded at first.

* Index: If username is set in session data, then get post list from blogdb. Load backend view and send post list.
* Postsubmit: This function has “pid” parameter with default as null. If backend view sends post title and post content as POST data, then the following data will be put in an array –post content, post title, tag names (the value will be exploded with delimiter “,” and sent as an array), post time (using now() function) and user id (from session data). If post id is not empty then post id will also be put in the array. Then the data array will be send to the pubish\_post function in blogdb.
* Editpost: If backend view sends only “pid” as POST data, it means that it is an AJAX request requesting for post title and post content. The “pid” or post id will be sent to the getpostcontent function of blogdb, which will return the post row from posts table. Then the “pid” or post id will be sent to the gettags function of blogdb, which will return the tags array from tags table. Then each tag name will be pushed onto tags array, which will be imploded in tags name variable. Then the post title, post content and tags name received from blogdb will be set as flash data in the session cookie. Else if backend view sends post id, post title, post content and tags name as POST data, it means that the post is to be updated in the database. The following data will be put in an array – post id, post content, post title, tag names (the value will be exploded with delimiter “,” and sent as an array), user id (from session data) and last updated post time. Then the data array will be send to the pubish\_post function in blogdb. Then the new row of post list will be returned to the backendc view.
* Deletepost: If backend view sends post id as POST data, then the post id is sent to delete\_post function of blogdb. If the function returns true, then echo “success”.

**Models:**

Description of the models are as follows:

**Blogdb**:

The Blogdb model is connected with the “myblog” database. It is the core model of the application.

* Publish\_post: This function takes a data array as parameter. If the post id value of the array is not empty, then do the following – run a query on posts table where the post id matches post id value of data array. If the query returns a row, then set post last update time as post time value of data array. Unset post time value of data array. Update posts table with data array where post id matches post id value of data array. For each tag name value of tag names array in the data array, do the following – run a query on tags table where tag name matches tag name value of tag names array and save the result on tag id variable. Run a query on posts\_has\_tags table where posts\_post\_id matches post id value of data array and tags\_tags\_id matches tag id. If the query returns no row, then insert post id and tag id in posts\_has\_tags table. Else do the following – set last update time as current time. Insert data array into posts table. For each tag name value of tag names array in the data array, do the following - run a query on tags table where tag name matches tag name value of tag names array and save the result on tag id variable. Insert post id and tag id in posts\_has\_tags table. Return last update time of post.
* Delete\_post: This function takes post id as parameter. Delete the row of posts table in which post\_id matches post id. Return true.
* Verifylogin: This function takes a data array as a parameter. Run a query on users table where username matches username value of data array and password matches password (MD5) value of data array. If the query returns a row, then return user\_id of that row. Else return 0.
* Getpostlist: Run a query on posts table selecting post\_id, post\_title and post\_time where users\_user\_id matches user id of current session data. Return the query result.
* Getpostnames: This function takes post id as parameter having default value NULL. If post id is empty then do the following – run a query on posts table selecting post\_id and post\_title. Return query result. Else do the following - run a query on posts table selecting post\_title where post\_id matches post id. Return query result.
* Getpostcontent: This function takes post id as parameter. Run a query on posts table where post\_id matches post id. Return the query row.
* Gettags: This function takes post id as parameter. Run a query on tags and posts\_has\_tags table selecting tag\_name where posts\_post\_id matches post id and tags\_tags\_id mathes tags\_id. Return query result.
* Getlikecount: This function takes post id as parameter. Run a count query on posts and likes table selecting like\_id where post id matches posts\_post\_id. Return the count of number of rows.
* Isliked: This function takes post id and ip address as parameter. Run a query on likes table where like\_ip\_address matches ip address and posts\_post\_id matches post id. If the query returns a row then return true, else return false.
* Insertlike: This function takes post id and ip address as parameter. Insert post id and ip address on likes table. Return the count of likes in that particular post.

**Views:**

Description of the views are as follows:

**Blog**:

The blog view is the main front end view. It displays post titles and blogcontent view. Mentionable parts are as follows:

* $postdata is an array containing post id and post title. For each value of $postdata, the view displays the post title and sets post id as the id of anchor tag. If current\_id flash session data is set and the post id of the $postdata matches current\_id, then the class will be set as active.
* If user clicks on any one of the post titles, the id of that post title is sent to getid function of blog controller via AJAX, which sets the post id in a flash session data named current\_id. If the current\_id is set, it is set in a post\_id variable inside a content variable, which is then loaded in blogcontent view. Else 1 is set in a post\_id variable inside a content variable, which is then loaded in blogcontent view.

**Blogcontent**:

The blogcontent view contains the post body. It displays the post corresponding to the post title clicked by the user. Mentionable parts are as follows:

* The blogdb model is loaded in the view. The post\_id of the content variable is sent to the getpostcontent function of blogdb, which returns post row.
* The post\_id of the content variable is sent to the gettags function of blogdb, which returns tag names.
* The function isliked of blogdb takes ip address and post id as input and returns Boolean. If the function returns true, it means the post is already liked by the user, and the view outputs “Liked”. Else the user is given the option to like the post.
* If user clicks on like button, then post id is sent to insertlike function of blog controller as AJAX POST data, which returns current like count, which is then added to the view.

**Verifico**:

The verifico view is used to log in to the backend.

**Backend**:

The backend view is the main view of backend controller. It has two tabs – “new post” and “old posts”. Each of these tabs load the backend view.