USER TUMBLR

A TUMBLR LIKE APP IN RAILS

INTRODUCTION

This web app written in Ruby on Rails framework provides interface to the registered users to upload their images, which are shown on the user profile as a wall of images. The app basically serve these functions:-

Sign Up

The design is to quickly enable the users to signup to have maximum conversions. So we have opted for minimum number of required fields to register a user. The user can sign up by providing only following personal information:-

1. Username: A Username to uniquely identify the user across domain.
2. Email Id: User's email ID. Each email can have only one account.
3. Password: Password user needs to login in the site.

Sign up page has following validations and behaviour in place:-

1. Username should be between 3 and 20 letters.
2. Email Id should be a valid one

3. Both User ID and Email ID need to be unique across site. A user can't register twice
4. Password is between 6 to 20 characters
5. Any of the above validation is failed will led to failure to sign up with correct message being displayed to the user.
6. User password is not saved in text anywhere in the system. We save a salted copy in the DB, along with salt. Same is used to decrypt the password later.

The signup page will immediately redirect the user to login page to move forward.

Log In

Login page will ask user for username/email and password to login to the site. Correct information will lead the user to his homepage, incorrect information will display and error. Login page has following process in place:-

1. User can login with either of the username and email id. Again the focus is to make the use of app easy for the user.
2. User password is decrypted using salt present in DB to authenticate.
3. A session is created which is killed when user is logged out.

Profile Page

This is the main and only home page of the user. It allows users to upload and view images. It has following characteristics:-

1. User can upload single image at the time by clicking on “choose image” and then “upload” button.
2. All the uploaded images are saved in reduced form to save the disk space at the servers while maintaining the quality of image.
3. There is validation in place at both server and client to check if it is a valid image file.
4. Immediately after the image upload user would be able to see the image at the home page below the upload buttons.
5. User can click on the image to see the actual image.

TECHNICAL INFORMATION

Languages: Ruby, Javascript
Scripting: erb, css, html5
Framework: Rails, Reactjs
Databases: Mysql

SOFTWARE DESIGN

The server is written in Ruby on Rails framework with 4 controllers, 2 models and 3 views. We will discuss design and features here.

Application Controller:

It is the base controller for the app which provides three important methods. All other controllers extend this controller class.

1. **authenticate_user:** Check if user is authenticated, if yes do nothing, else redirect to the login page.
2. **save_login_state:** if user is already logged in, take him to the home page, else do nothing.
3. **set_cache_buster:** Tell the browser to not cache, in order to prevent user to access cached copy from browser after creating/deleting session

Session Controller:

As the name suggest this controller provides login and logout functions for the user.

1. **Login:** Authenticate user using *authenticate* method in the User model, if success creates the session, else send 422 HTTP status
2. **logout:** Deletes the user session and redirect to the login page.

Post Controller:

This controller handles the actions related to user homepage which are:-

1. **create:** Create new post taking image from post request and take the user back to the homepage.
2. **show:** Shows the image in a new tab if user clicks the image in the thumbnails.
3. **index:** Take the user to the home page. while setting the model objects to be used by view later.

Model

Model component models the communication to the two and only tables of the mysql database.

User

1. Validates input for the User table, i.e. the data coming from the signup page.
2. **encrypt_password**: it encrypts the user password with a random salt. *BCrypt ruby gem is used.*
3. **authenticate:** Authenticate user with username/ email and the password provided. It encrypt the password with the same salt and compare the salted passwords.

Post

Post model validates the DB fields, like username, image type etc.

Views

We have three views here:-

1. **posts/index.erb:** erb file which renders the user home page by taking two post model object created by the user actions in post controller. *post_create* object creates form for image upload and *post_show* object is used to show all the user images.
2. **posts/show.erb:** Another erb file simply shows image associated to the tumbnail clicked by the user.
3. **user/new.erb:** This erb page basically imports single *React* module called "HomePage".

React Components

React components in the page are used only in the user signup/login page. There are following components:-

1. **HomePage:** Contains the current action state, i.e. if users wants to login or sign up. depending on the state if renders one of the corresponding component.
2. **Signup**: Handles the rendering of the signup page and submit the ajax call to user controller for the signup. It calls *Error* component to show any error if occurred.

3. **Login:** Similar to signup, renders the page, sends request to user controller and handles the errors using Error component.
4. **Error**: Simply created a div with showing error message and passed in the props.

DATABASE DESIGN

We have two tables with following schemas in Mysql.

```
users | CREATE TABLE `users` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `salt` varchar(255) DEFAULT NULL,
  `created_at` datetime NOT NULL,
  `updated_at` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
```

```
posts | CREATE TABLE `posts` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `caption` varchar(255) DEFAULT NULL,
  `created_at` datetime NOT NULL,
  `updated_at` datetime NOT NULL,
  `image_file_name` varchar(255) DEFAULT NULL,
  `image_content_type` varchar(255) DEFAULT NULL,
  `image_file_size` int(11) DEFAULT NULL,
  `image_updated_at` datetime DEFAULT NULL,
  `username` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
```

DEMO INSTRUCTION

The app has been hosted on Heroku at: https://protected-depths-36556.herokuapp.com/

1. Click on the signup and give information to signup
2. Or use following information to login, username:rhythmwalia, email: rhythm.walia@gmail.com, password:12345678
3. Upload any valid image.
4. See the recently and other images in the homepage.

KNOWN ISSUES

1. **Issue #1:** Browser fast back caching.

When user press back button *It loads page from cache*. *no-cache* in response is already set, but Browsers seems to have another layer of cache on top to provide fast back, as the network tab in the inspect element doesn't even show *HTTP 304 status*. It affects user homepage.
Solution is to use cache buster tools in the market which will keep changing tag field.

FUTURE IMPROVEMENTS

This is a project developed as a demo in limited time and resources, it is not meant to be a serious production code. There are lot of things which could have been better. Some of them are:-

1. Have better and stronger validations for user fields like password etc.
2. Don't load whole images for the homepage as it will hamper the performance, generate thumbnails and show them.
3. CSS could be better in the user homepage
4. Email user a link to validate the account. This will prevent users from adding fraud emails, which looks good.
5. Use cache buster tool to solve the above issue