Preliminary Summary Document

Bassim Beshry, Jordan Pohr, Rhythm Trivedi

Contents

Notes

This document is subject to change as it is not the final version. Although we are handing this in as a Preliminary Summary Document, it will most likely be repurposed as a Technical User Guide. The admin section is unfinished and does not explain how anything works. Although the features themselves are essentially complete, I will not take the time to expand on such a big portion of the website until the Technical User Guide is requested on the day of the final milestone.

Overview

Our website Onyx is a replacement for websites like Reddit and Twitter. Designed with the idea of public discussions that are always live. Posts can be viewed and then comments on these posts work almost like group chats or message boards for people who are viewing the posts. We hope to change the social media landscape with our idea.

Our design philosophy when creating Onyx was to make an application where there are no limits to what you can post. We also wanted to make everything have a 'live' feeling. Comments on posts are essentially chat rooms for this reason. It is important to be careful what you post on Onyx because you cannot delete or edit it. This is a conscious decision to emulate an environment that is living. For example, if you and two other people were in a room together, you wouldn't be able to delete what you say, correct? Onyx is meant to recreate that feeling.

Web Pages

Login and Signup

Greetingpage.php, home.php, login.php, loginhandle.php, signup.php, signuphandle.php, logout.php

To use Onyx, it is important to have an account to experience it in its entirety. When you first navigate to the Onyx website, you will be directed to the *greetingpage.php*. Here, you have the option to either browse the website without an account (limited functionality), login with an existing account or create a new account.

1. Clicking the browse button will direct you to *home.php*. Here you can view posts and view a user's accounts. Any actions that you try to perform that require an account will redirect you to the *signup.php* page.
2. Clicking the login button will direct you to the *login.php*. Here you can login to your account. It will check your password via a SELECT query from the *loginhandle.php* and redirect you to *home.php*.
3. Clicking the signup page will direct you to *signup.php* where you can fill out a form with your username, email, and password. You will need to also upload a profile picture here as well as re-enter your password a second time to ensure that they match. *Signuphandle.php* will ensure that the username and email address are unique and then add you into the database. Congratulations! You have just made an account for our website!

Account

Account_page.php, account_settings.php, updatesettings.php

Setting up an account for Onyx is essential to receiving the full experience. Once you do, you will have access to a multitude of different features. To view the account that you just created, find the navbar and click on the account button. Doing this will take you to *account_page.php*. This page displays your profile and includes your username, profile picture and account description, while also keeping track of your followers count, following count and joined communities count.

Hovering over the account button will bring up the following three buttons: Manage Account, Friends and Saved Posts. The only one relevant to this section is *account_settings.php.* This page has 3 buttons: Edit Profile Picture, Edit Username and Edit Password. Pressing either 3 of these buttons will trigger a popup modal to display on the page. Each popup contains a form that will then send a POST request to *updatesettings.php* where a series of queries will be executed eventually resulting in an UPDATE query being ran to change the user's information.

Searching

Searchpage.php

Now that your Onyx account is set up properly, it is time to start searching for communities to join and users to follow! Our search page looks for tweets and communities by taking a keyword in the search bar and displaying all tweets or communities with that keyword in either the title or description. It is a quick and simple search bar powered by a SELECT statement. After a match is found, it displays the posts or communities in the respective columns. For this page, we opted for a 2 column layout that displays the tweets or posts in the left box while displaying all relevant communities in the right.

Follow System

Deletefollow.php, deletefollower.php, insertfollow.php, manage_friends.php, randomuserpage.php

After searching for a user, you can view their profile in the *randomuserpage.php*. This page is very similar to the account page where you can view your profile, however it works by using a GET request to retrieve the profile's name. From there, we can display all the account information that you would find in *account_page.php*.

When looking at a random user's profile, you will notice the follow button. You can click this button to follow the user. This will display all new posts by this user in your home feed. We used AJAX to change that button to an unfollow button upon click. When you follow a user, *insertfollow.php* is called. Imagine a scenario where user 1 adds user 2. User 1's account id will be added alongside user 2's id to the Follows table on the database as a follower_id and followed_id which is a foreign key to their respective account id's. Unfollowing a user removes your follower_id and the associated followed_id from the database so you are no longer following them. This action is done through the *deletefollower.php* file.

Manage_friends.php allows you to view all your followers and users you follow on one page. The page is formatted on a two-column layout with your followers list on the left column and following list on the right. Each user is displayed from left to right starting with their profile picture, then their username, then a button to either remove them as a follower or unfollow them. Removing a user as a follower removes your followed_id and the associated follower_id from the database so that they are no longer following you.

Communities

Comcreationhandle.php, Communitiespage.php, createcommunity.php, joinablecommunitypage.php, joincom.php, leavecom.php

Communities are the main aspect of viewing and interacting with others on Onyx. To create a community, click on the create community button while hovering over the community button in the navbar. Clicking on the create community button will bring you to the *createcommunity.php* page. On this page, a form will ask you for the community's name and the community's description. Filling this out and submitting it will create a new community on the database. *Comcreationhandle.php* creates a community table in the database and the community is officially created.

Because you are the creator of this community, you will join upon creation. However, if you found this community via the search bar, you can also press the join button when viewing the community via *joinablecommunitypage.php* to join the community. Doing this will add your account_id to the community membership table in the database alongside the communityid associated with the joined community. This is done via *joincom.php*. The file *leavecom.php* does a very similar action by removing your account_id from that same table.

Posting

Createpost.php, home.php, postcreationhandle.php, postpage.php, saved_posts.php, savepost.php

Onyx wouldn't be a thing unless users were to post and share their thoughts on our website. For this reason, we tried our best to implement posting and make it as robust as possible. To create a post, hit the create post button when on either the home page (*home.php*) or the page of the community that you are making the post in.

Creating a post will put it on the feeds of all those who are following you as well as the feeds of anyone who is a part of the community that your post was posted to. It will also display in the community's page according to date relevance. Postcreationhandle.php is what handles all this through the use of a couple of SELECT queries and one INSERT query.

Unless you are an administrator, there is no current way to delete or edit a post. Onyx is meant to be live and eternal. It is very important to be careful what you post for this reason. Perhaps we will implement a way to delete or edit posts in the future, however for now this goes against our design philosophy. See the overview on page 2 for a better understanding of this.

Comments

Fetchcomments.php, postpage.php, submitcomment.php

Comments on Onyx are designed to be live. When viewing a post, all the comments are displayed below. These comments are live as if you joined a chatroom. To do this, every instance of the page in the background is a function running every 1 second to fetch new incoming comments. These comments are written to the database so upon refresh, they are still there. Users can see incoming comments in real time.

Admin

Admin.php, deletepost.php, deleteuser.php, deletecom.php, updatepost.php, updateposthandle.php, viewcommunities.php, viewpost.php, viewusers.php

Admin features were designed with simplicity in mind. From the admin dashboard you can

1. Manage all users
2. Delete users
3. Search for specific users
4. Manage all communities
5. Delete communities
6. Search for specific communities
7. Manage all posts
8. Delete posts
9. Edit posts
10. Search for specific posts

All of these features and the explanations for how they work will be fleshed out in the final document.

Security

Throughout our development process, we added a multitude of security features in order to ensure that users can safely use Onyx without any concerns.

Client-Side Security

Password Hashing

Password hashing is when passwords are encoded before being sent to the database for verification. This was done through use of a php command named md5(). This simple hash enhanced our client-side security significantly.

Form Validation

Form validation is used to ensure that the information being sent to the database is valid and not malicious. To do this, we used javascript and created a password checking script. This ensured that the password was the same locally as well as on the database.

Server-Side Security

Prepared Statements

Prepared statements are used to prevent SQL injection attacks. Before we queried anything from the database, we made sure to use prepared statements before binding the parameters. This was done by assigning a variable (usually named $query) to the query we wanted to run on the database. Followed by executing that variable and setting its result to another variable (usually named $result. This basic security check served as a strong part of our server-side security.