

CSC667/867 (FALL 2019)

Internet Application Design and Development Project Report

Students Name:

Poorva Rathi (918182790)
Phuong Tran
Sunny Srijan (917105649)
Chris Eckhardt (915736372)
Jimmy Chen (916260831)
Sushil Plassar (918818893)
Nayan Pandey (920065390)

Project Title:

Switter

Project Description:

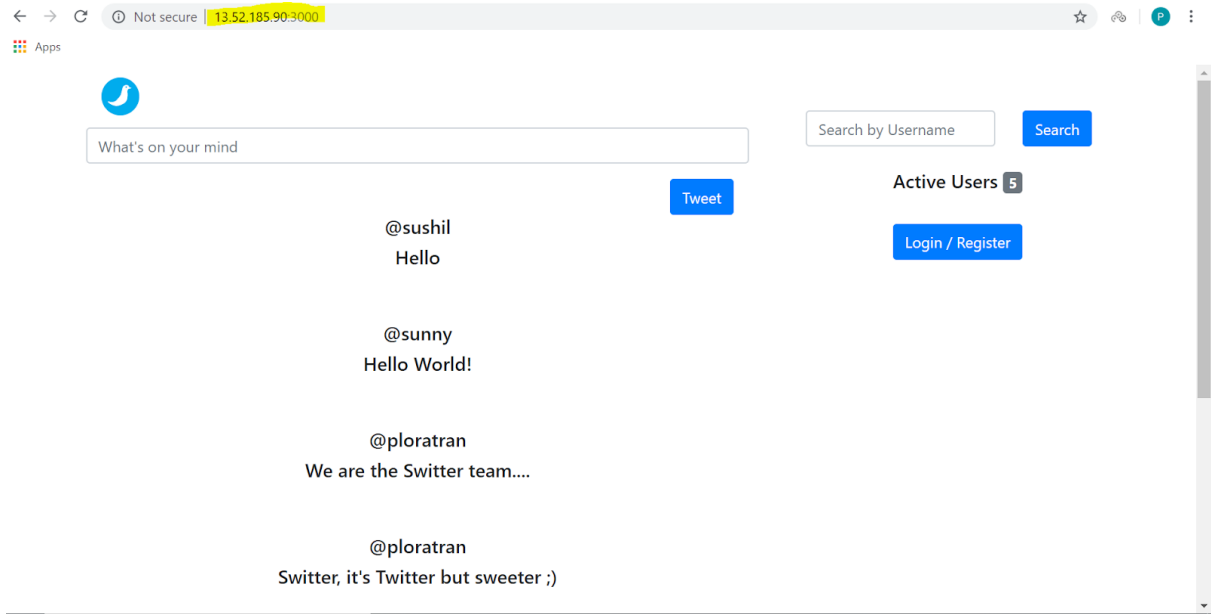
This program website will be similar to Twitter where users can post messages. However, only registered users can post and search any user also show how many active users are currently on the website. Users can also search up other users and see what they have posted recently, or first posted. If a user has already registered, they will click the signup button, or if the user is returning and wants to log in, they will click the login button. If the registered user wants to leave, they will log out.

Technology used:

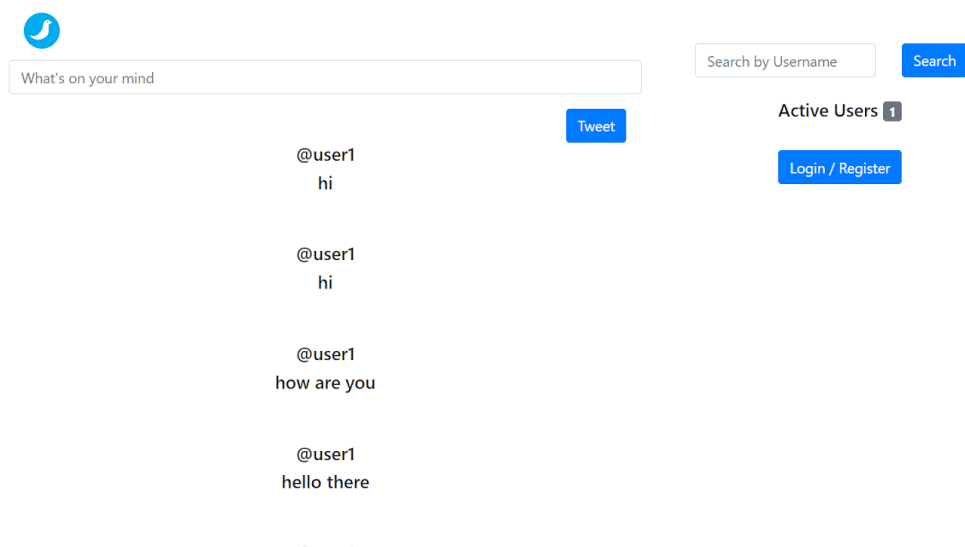
- FrontEnd: React, Redux, JavaScript, Bootstrap, CSS
- BackEnd: Node.Js, Express, Websocket
- Database: MongoDB with Mongoose

Screenshots:

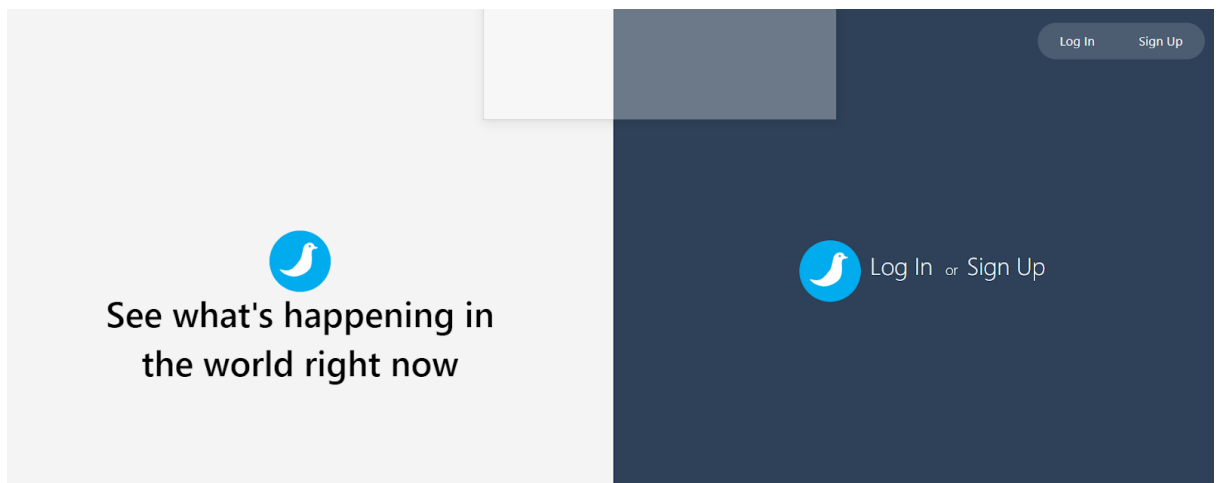
1) The application is working on the AWS server:



2) Home page of the unregistered users. We have implemented Lazy Registration for the users. It means that unregistered users can view all the tweets posted by other users but to post tweets or search any particular user, they need to be logged in.



3) Welcome Page: to navigate to sign up or login page.



- 4) Sign up: Allows unregistered users to register into the app. We made a condition to pick a username that is not already registered in the system.

Create your account




- 5) Login Page: Registered users can login into the app with unique username.

Login with your username



Login

6) Home page: For registered users, the users can tweet the post, search any user and view their own profile.



@user1
hi

@user1
hi

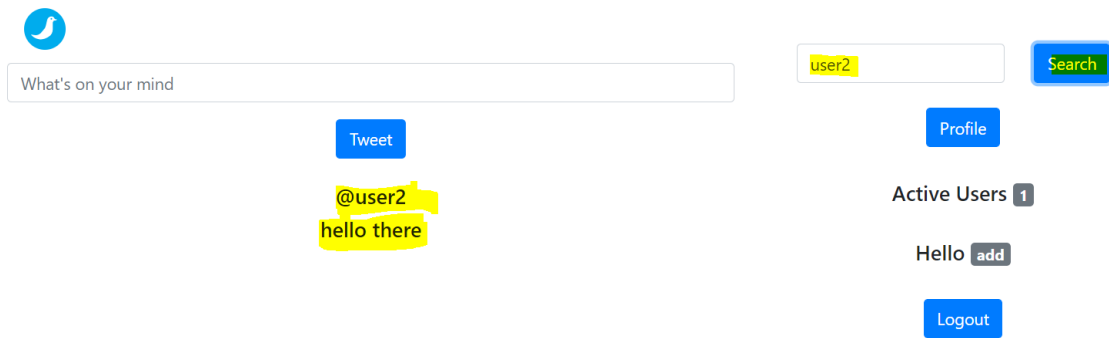
@user1
how are you

@user1
hello there

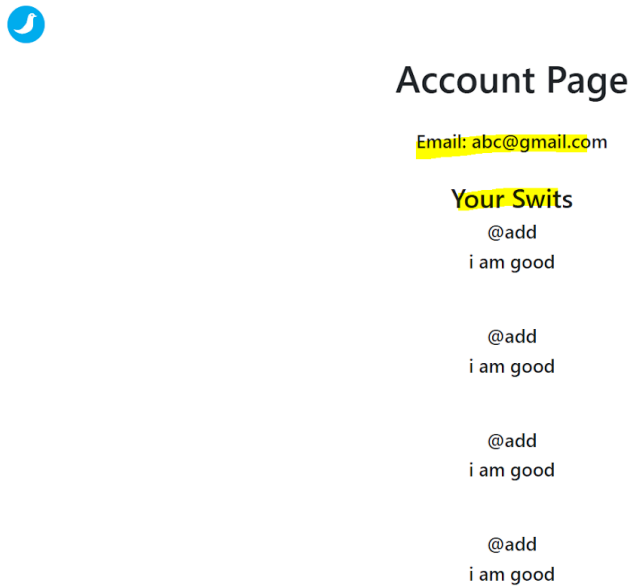
Active Users **1**

Hello

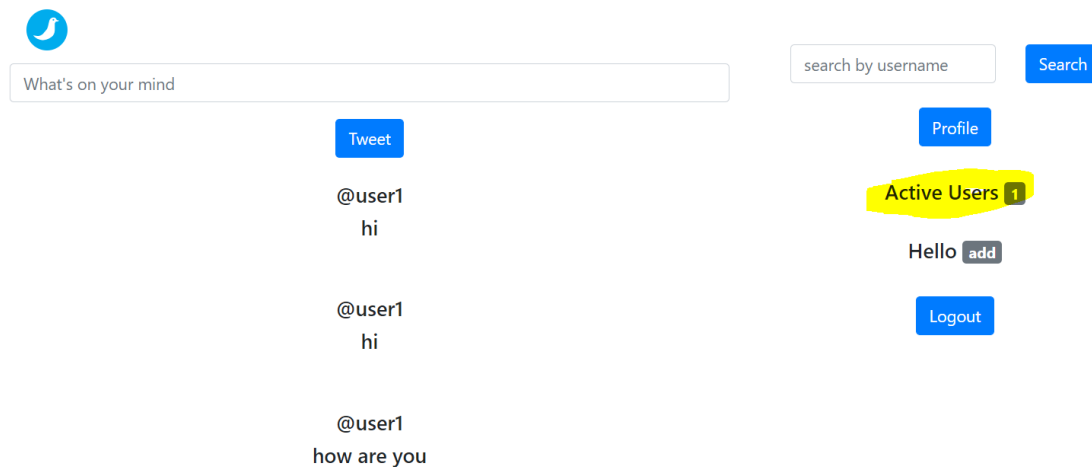
7) Search through username: User can search any user in the profile and look what all tweets they have posted.



8) Profile: The users can view their own profile.



9) Active users: To know how many number of users are viewing the app at the same time.



Project Report

In this project, we use React and Redux JS to build user interfaces of all the pages. ReactJS is used to create page components such as the header or the search form in the main page. Redux JS is used to manage states change when there are new update of states from one page to another page. To create a uniform template for the buttons, columns, and forms, we use React Bootstrap and customize CSS for a better UI looks.

First we create pages/ folder in order to create all the UI pages in the project. Then, we create a components/ folder to split up the working code logic to sub-components based on their functionality. For example, the ProfileTweetDashboard.jsx is the page used for registered users to display all the tweet messages; the ProfileSearch.jsx is for handling the "search by username" functionality. These sub-components are used in the ProfilePage.jsx. In our App.js file, we make six front-end React routes for six components. For example, '/profile' for ProfilePage component, the '/' is for Logged_in_main component. Then, we use React Router Dom and Redux state to do authentication for the

front-end. We check if the state of “isLoggedIn” is true, users will be redirected to /profile and /account pages where the user has been authenticated with the backend. If the state of “isLoggedIn” is false, user will stay in the main “Logged_in_main” page where they can only see the tweet messages but cannot do any functionality such as search by username or posting tweets.

In Switter application, we have two databases using Mongoose DB: Users and Tweets databases. The authentication will be done with Mongoose DB. In Users database, we ask users to enter their username, password, and email (with the correct format of email with @) in SignUp page. Once users finish sign up, a POST request will be sent to the database to store users' information. When users have successfully signed up, they will be automatically redirected to Login Page where they are prompted to put their username and password. A GET request will be sent to the database to verify if the username and password match with that particular user. If there is a mismatch, an error message will display on Login Page saying incorrect username and/or password.

In this application, we enforce lazy-registration in order to get the username from Users database. After users have successfully logged in, the username retrieved from the Users database will be used to make tweets into the Tweets database. Whenever users make a new tweet, a POST request with the username retrieved from Login Page from the Users database, along with the tweet message will be saved in Tweet database. The TweetDashboard component will display all the tweet messages from the Tweet database, including the newly inserted tweet message. By doing this, a GET request will be sent to the Tweet

database to find all tweets.

The Account Page will display all the tweets and email of the username who is currently logged in. This is done by sending a GET request to the Users database to retrieve the email of the user when they first sign up. Another GET request is sent to Tweet database to retrieve all the tweet message inserted by that username.

The search by username functionality at the top left corner is done by sending a GET request to the Tweet database to find the tweet message by the specified username input in the search bar. What our team has not done is handling the error when there are no username found by what the user input in the “search by username”. To improve this feature, we should display a message saying “No username found!”.

We make a proxy server at port 3004 so all front-end requests will first hit gateway at port 3004. Our application has three server files: websocket, fileServer to serve front-end, and mongodb with router at api.js. The websocket server is at port 4005. When websocket server is on connection, it will update the Redux state of the active users. So that when a new tab is open on Google Chrome, the number of active users is incremented and decremented when tab is closed.

All of our backends are contained using Docker.

The hardship we have encountered is that our team has not fully implemented the real-time update using Kafka and Websocket. Because of that, the real-time tweeting feature is not accomplished. We also have not used redis for caching tweets. Because of that, when displaying tweets from Tweet database, the speed is slow because the tweet messages are fetching using Mongoose DB instead of the in-memory database using Redis.