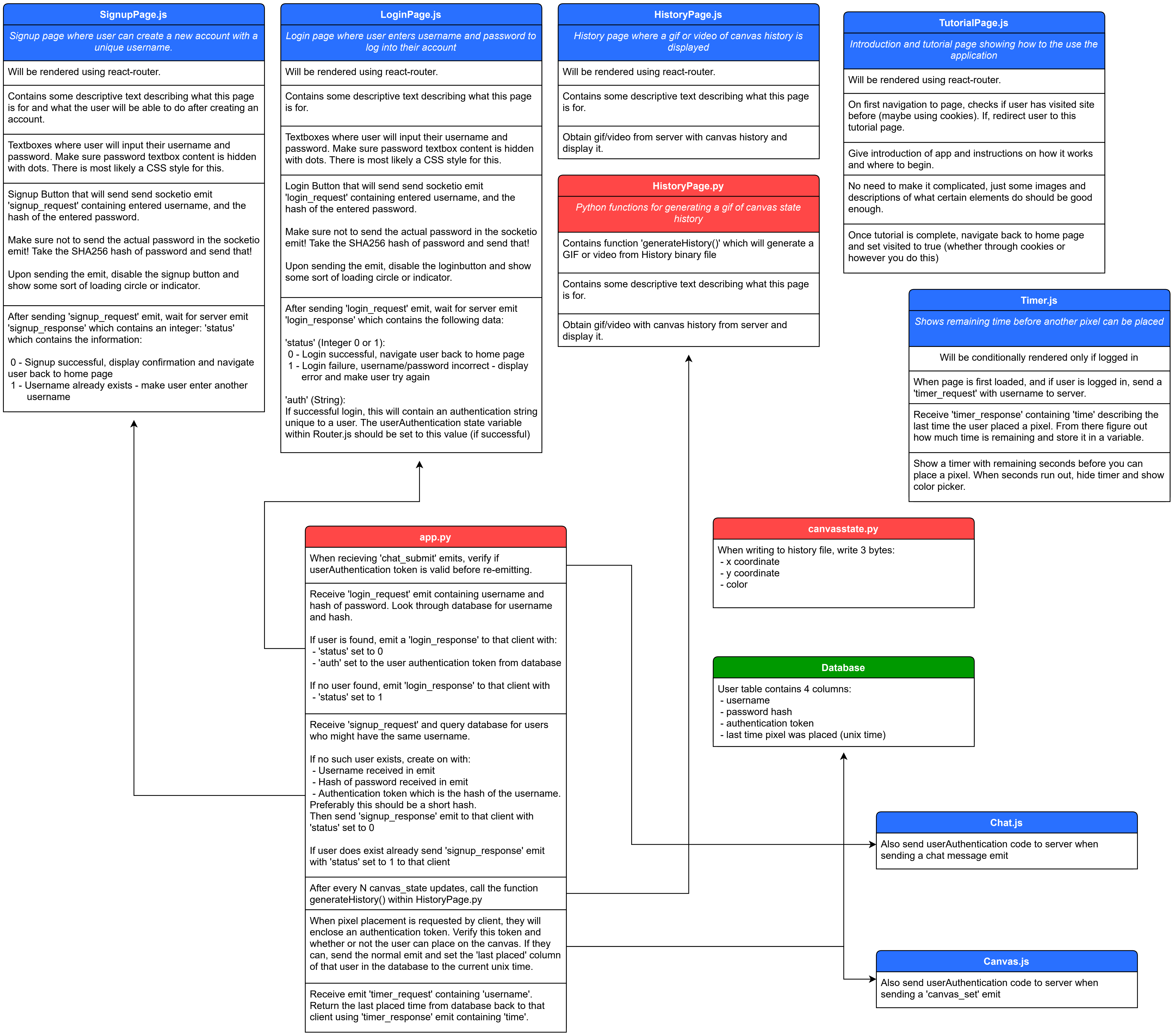

SignupPage.js

Signup page where user can create a new account with a unique username.

Will be rendered using react-router.

Contains some descriptive text describing what this page is for and what the user will be able to do after creating an account.

Textboxes where user will input their username and password. Make sure password textbox content is hidden with dots. There is most likely a CSS style for this.

Signup Button that will send send socketio emit 'signup_request' containing entered username, and the hash of the entered password.

Make sure not to send the actual password in the socketio emit! Take the SHA256 hash of password and send that!

Upon sending the emit, disable the signup button and show some sort of loading circle or indicator.

After sending 'signup_request' emit, wait for server emit 'signup_response' which contains an integer: 'status' which contains the information:

0 - Signup successful, display confirmation and navigate user back to home page
1 - Username already exists - make user enter another username

LoginPage.js

Login page where user enters username and password to log into their account

Will be rendered using react-router.

Contains some descriptive text describing what this page is for.

Textboxes where user will input their username and password. Make sure password textbox content is hidden with dots. There is most likely a CSS style for this.

Login Button that will send send socketio emit 'login_request' containing entered username, and the hash of the entered password.

Make sure not to send the actual password in the socketio emit! Take the SHA256 hash of password and send that!

Upon sending the emit, disable the loginbutton and show some sort of loading circle or indicator.

After sending 'login_request' emit, wait for server emit 'login_response' which contains the following data:

'status' (Integer 0 or 1):
0 - Login successful, navigate user back to home page
1 - Login failure, username/password incorrect - display error and make user try again

'auth' (String):
If successful login, this will contain an authentication string unique to a user. The userAuthentication state variable within Router.js should be set to this value (if successful)

HistoryPage.js

History page where a gif or video of canvas history is displayed

Will be rendered using react-router.

Contains some descriptive text describing what this page is for.

Obtain gif/video from server with canvas history and display it.

HistoryPage.py

Python functions for generating a gif of canvas state history

Contains function 'generateHistory()' which will generate a GIF or video from History binary file

Contains some descriptive text describing what this page is for.

Obtain gif/video with canvas history from server and display it.

TutorialPage.js

Introduction and tutorial page showing how to the use the application

Will be rendered using react-router.

On first navigation to page, checks if user has visited site before (maybe using cookies). If, redirect user to this tutorial page.

Give introduction of app and instructions on how it works and where to begin.

No need to make it complicated, just some images and descriptions of what certain elements do should be good enough.

Once tutorial is complete, navigate back to home page and set visited to true (whether through cookies or however you do this)

Timer.js

Shows remaining time before another pixel can be placed

Will be conditionally rendered only if logged in

When page is first loaded, and if user is logged in, send a 'timer_request' with username to server.

Receive 'timer_response' containing 'time' describing the last time the user placed a pixel. From there figure out how much time is remaining and store it in a variable.

Show a timer with remaining seconds before you can place a pixel. When seconds run out, hide timer and show color picker.

app.py

When recieving 'chat_submit' emits, verify if userAuthentication token is valid before re-emitting.

Receive 'login_request' emit containing username and hash of password. Look through database for username and hash.

If user is found, emit a 'login_response' to that client with:
- 'status' set to 0
- 'auth' set to the user authentication token from database

If no user found, emit 'login_response' to that client with
- 'status' set to 1

Receive 'signup_request' and query database for users who might have the same username.

If no such user exists, create on with:
- Username received in emit
- Hash of password received in emit
- Authentication token which is the hash of the username.

Preferably this should be a short hash.
Then send 'signup_response' emit to that client with 'status' set to 0

If user does exist already send 'signup_response' emit with 'status' set to 1 to that client

After every N canvas_state updates, call the function generateHistory() within HistoryPage.py

When pixel placement is requested by client, they will enclose an authentication token. Verify this token and whether or not the user can place on the canvas. If they can, send the normal emit and set the 'last placed' column of that user in the database to the current unix time.

Receive emit 'timer_request' containing 'username'. Return the last placed time from database back to that client using 'timer_response' emit containing 'time'.

canvasstate.py

When writing to history file, write 3 bytes:
- x coordinate
- y coordinate
- color

Database

User table contains 4 columns:
- username
- password hash
- authentication token
- last time pixel was placed (unix time)

Chat.js

Also send userAuthentication code to server when sending a chat message emit

Canvas.js

Also send userAuthentication code to server when sending a 'canvas_set' emit