***Documentation Packet [ 10 01 11 24 ] Nov 1st 2024***

|  |  |
| --- | --- |
| Student Name: |  |
| Goals:  1. Identify risks of hosting user data 2. Use OAuth2 | Events:  1. DocPac Due Monday 2. Two-hour Delay Wednesday |
| Included Documentation  1. Formbar OAuth2 2. User Login Handling 3. CIW Lesson 4 4. Nodejs Submission Rules 5. CIW Submission Rules | Required Documentation:  1. Formbar OAuth2 2. User Login Handling 3. CIW Chapter 4 4. Reflections |
| Changes/Notes:  * None | |

# Websockets Chat

## NEE Setup

1. Create an HTTP Listen server with NodeJS, ExpressJS, and EJS as usual
   1. There will be two GET endpoints. One at ‘/’ and one at ‘/chat’
2. The ‘/’ template will have a text-box for putting a username, and a submit button for redirecting the user.
   1. The text box must have content for the Submit button to work
   2. The submit box does not POST to ‘/chat’. Instead, it directs the user to ‘/chat?name=<text box content>’ by a GET request, where ‘<text box content>’ is the value of the text box. You will need to Google how this is done with Javascript.
3. The ‘/chat’ template will have the following HTML format:
   1. Javascript to save the passed name EJS value to a variable on the page
      1. For example: *var localName = ‘<%- name ->’;*
   2. A div layer that floats left or right for holding a vertical list of user names (other elements will wrap around the side of this box)
   3. A large div layer to hole a vertical list of messages sent in the server
   4. A small, but wide text box for typing messages
   5. A button to send the message to the server
4. The ‘/chat’ endpoint will check for a query parameter called “name”.
   1. If it does not exist, redirect the user back to ‘/’
   2. If it does exist, render the ‘/chat’ template, passing the name query parameter value along to the template

## Websockets Server Setup

1. Install and Import the ‘ws’ module
2. Use the ‘http’ module to create an HTTP server from the ExpressJS ‘app’ object
3. Use that server to create a WS server using the ‘ws’ module
   1. Example:

const app = express();

const http = require('http').Server(app);

const wss = new WebSocket.Server({ server: http });

http.listen(3000, () => { console.log(`Server started on [http://localhost:3000`](http://localhost:3000%60)); });

1. Create a ‘broadcast’ function with the parameters ‘wss’ and ‘message’
   1. For each client in the ‘wss‘s ‘client’ property, use the WS client’s send() method to send the ‘message’
2. Create a ‘userList’ function with the parameter ‘wss’
   1. Create an empty user list
   2. Use a for loop to loop through each element in the ‘wss’s ‘client’ property.
      1. If that client has a ‘name’ property, append it to the user list
   3. return an object with the property ‘list’ whose value is the user list
3. Create an event listener ( *wss.on(‘event name’)* )for the WS server object for the ‘connection’ event. This will create a ws client connection in the callback of the event listener function. Inside of this callback:
   1. Create an event listener for the ‘message’ event. This will create a ‘message’ in the callback of the event listener. In this callback:
      1. Parse the ‘message’ string into JSON
      2. If ‘message.text’ exists, use ‘broadcast’ with this WS server object and the stringified ‘message’ object
      3. If the ‘message.name’ exists, save the ‘message.name’ value to the ws client’s ‘name’ property (you will have to make it). Then broadcast the return value of the userList function with this WS server as the argument. This will send out all of the names in the server to everyone.
   2. Create an event listener for the ‘close’ event. In this callback:
      1. Broadcast the return value of the userList function with this WS server as the argument. This will send out all of the names in the server except the one that just left
4. Back in the ‘/chat’ template’s Javascript, create a **new Websocket** connection at your PC’s local IP address.
   1. Change the WS’s ‘onopen’ function to use the WS’s send() method to send the ‘name’ variable saved to this page
   2. Change the WS’s ‘onmessage’ function to pass the argument ‘message’
      1. If the message has a ‘list’ property, use a for loop to display each name in the list in the user list div
      2. If the message has a ‘text’ property, ADD the message’s ‘user’ property followed by the value of the ‘text’ property to the messages div layer.
         1. Note you should be able to see all previous messages as well
   3. When the user clicks the button to send the message:
      1. If the text box has content, create an object with the ‘text’ and ‘name’ properties, whose values are the message text and the saved name, respectively
      2. Use the WS connection’s send() method to send the object

As bonus, you may use a database to restore all past messages when the user connects.

# Reflection

**What is a project you could accomplish using all of the things you’ve learned in this year’s lessons so far? What techniques would you need to complete it?**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**What was one mistake you made in school or otherwise that you can recognize? What can you do in the future to prevent it from happening again?**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**If your name is Hayden, what is a second mistake you made this week? What can you do in the future to prevent it from happening again?** (If your name is not Hayden, skip this question)

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**How difficult was the assignment this week? What made it difficult/easy for you?**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

# A picture containing text, monitor, screen, clipart Description automatically generatedGrading

|  |  |  |
| --- | --- | --- |
| 10 | You went above and beyond expectations. You applied knowledge that was not taught in this class in addition to what was taught. Additional rewards are given | * ***If the assignment does not have its own rubric, it will default to the rubric on the left.*** * All assignments start at 10/10 possible points * 1 point is deducted per infraction   + Lateness   + Mistakes   + Unprofessionalism   + Not following instructions * Outstanding submissions, or submissions on assignments not marked in “Required Documentation” can reward pogs |
| 10 | You performed as well as can be expected for this class. You show a complete understanding and made no mistakes. You have mastered the subject. |
| 8 | Assignment is complete. You show a good understanding of the subject, but there are mistakes or minor incorrect details. You are ready to move to new subjects. |
| 7 | You show and understanding of the subject, but there are serious errors, or there are pieces you can practically use without understanding them. Remediation needed. |
| 6 | Assignment is incomplete but/or you showed that you understand at least the fundamentals of the subject. Assignment is low effort. Serious need of remediation. |
| 5 | You show minimum effort, assignment is incomplete, or have serious mistakes. You did not demonstrate that you understand the content or purpose of the submission. |
| 0 | The work was not submitted, damaged, seriously incorrect, or unprofessional. The submission is rejected. |

# User Login Handling

|  |  |  |
| --- | --- | --- |
|  | Spelling and handwriting |  |
|  | Correct information |  |
|  | Complete answers |  |

# CIW Lesson 5 *CIW Submission Rules, DP09*

|  |  |  |  |
| --- | --- | --- | --- |
| **1-2, 4** | Correct directory structure, with correct naming convention | |  |
| **3** | Read all pages | Used all Flash Cards |  |
| All Quizzes over 80% | All Exercises over 80% |
| **4** | All labs completed as assigned | |  |

# Websockets Chat *Nodejs Submission Rules, DP09*

|  |  |  |
| --- | --- | --- |
| **1-2** | Correct directory structure, with correct naming convention |  |
| **3-5** | Intact package.json, correct file name, test user ready. |  |
|  | Program works as intended |  |
|  | Code is functional, efficient, and *your own code* |  |
| **7** | Deleted node\_modules folder |  |
| **8** | Files structure intact (did not change or delete files not part of your assignment |  |

# DocPac and Reflection *DocPac Submission Rules, DP09*

|  |  |  |
| --- | --- | --- |
| **1** | DocPac is turned in on time |  |
| **3** | a. DocPac is neatly folded |  |
| **3** | b. DocPac is not stained or damaged |  |
| **3** | c. No doodles, scribbles, or unnecessary writing |  |
| **4** | a. Answered each question in each prompt |  |
| **4** | b. Spelling and handwriting |  |
| **4** | c. No repeated answers from other DocPacs |  |
| **4** | d. Did not paraphrase assigned work |  |
| **6** | You are prepared to justify the use of any AI (you know what it does and why) |  |