***Documentation Packet [ 09 28 10 22 ] Oct. 28th, 22***

|  |  |
| --- | --- |
| Student Name: |  |
| Goals:  1. Review JSFighter 2. Learn about Websockets 3. Establish new project review process | Events: |
| Included Documentation  1. [S] Weekly Contribution 2. Reflection | Required Documentation:  1. [S] Weekly Contribution 2. Reflection |
| Changes/Notes:  * Juniors now have recurring weekly assignments as well. * Reflections are now recurring questions. | |

# [S] Weekly Contribution

You must contribute for the csmith1188/formbar and document your contribution.

* Pick an open issue and have the Project Manager assign you to it.
* Submit a Pull Request from your working branch to the formbar RC branch
  + If the issue is too large to be reasonably completed by one person in a week’s time, create new issues to break the problem into smaller steps, link them in the original issue, and have this change approved by the Project Manager immediately.
* Have the Pull Request reviews and merged (or closed with approval) by the Project Manager at the next Code Review.
* Submit additional Pull Requests for other issues in separate branches.

In the box below, write the identifying number of every Pull Request you wish to be reviewed and graded:

|  |
| --- |
|  |

# [J] Weekly Study

Complete any two chapters (must have lessons, quizzes, and/or projects) from any course in Codecademy. Must be 100% complete and quizzes must be 80% or higher.

In the box below, write the identifying number of every Pull Request you wish to be reviewed and graded:

|  |
| --- |
|  |

# [J] NodeJS Chatroom

Summary: Build a webchat server that asks a user’s name when connecting to a page, then adds their name to a list of users, and tells all connected users that list. Users can type in messages and add them to a live board that everyone can see.

1. Using nodeJS, ExpressJS, and socket.io, create a webpage that allows anyone who connects to it to chat with others viewing the same page after providing their name.
2. When a user types in text in a textbox and hits “submit”, it will emit a “chat” event to the server, whose content is the text the user input.
3. When the server receives a “chat” event, broadcast it to all users in the namespace.
4. When the client receives a “chat” event, display it in a box containing all received messaged.
5. Whenever someone connects, add their name to an array that tracks all users.
   1. Broadcast a “users” event to all clients, whose content is the array that tracks all users.
6. Whenever a client receives a “user” event, parse the content as an array, and use that array to update a box containing the names of all connected users.
   1. (Note that this program will not clear out any users
7. Create a folder called “FirstnameLastname” inside of the “Chatroom” folder of this DocPac
   1. If there is no “Chatroom” folder, create it
8. Put all of your project files in this new folder
   1. DO NOT include the node\_modules folder.
9. Open a Pull Request to the DocPac Repo on Github, or commit to the branch you used for your open Pull Request this week.

# [J] JSFighter Battle Royal

Summary: Building off of the NodeJS Chatroom, make every user a Fighter that can be attacked with the chat command “!attack <user to target>”. This tells all users the details about the attack. Remove fighters when the user disconnects.

1. Using the code from NodeJS chatroom, make another nodeJS/ExpressJS/socketio project.
2. Using the Fighter class from JSFighter2, create a new Fighter object and add it to an array that tracks all fighter objects instead of a list of names.
   1. The Fighter’s id will be *the unique websocket session ID of the connecting user*
   2. The Fighter will also have a *name* property, which is the name provided by the connecting user.
   3. Make a list of fighter names from all of the fighters in the fighter list, and broadcast that as the content of a “user” event instead of a normal list of names
3. If a user starts a “chat” event message with “!attack”
   1. Use split() on the chat event content to split the string at every space
   2. If the array that split() returns has more than one element:
      1. Use find() on the list of fighters to find a fighter object whose name property matches the second element of the return split() array.
      2. If you find a match:
         1. Use the attack() method of the user’s Fighter targeting the found fighter object.
         2. Broadcast a “chat” message so everyone can see the results of the attack.
4. If a user disconnects, use find() to find the fighter whose id is equal to the disconnecting user’s unique websocket session ID, and remove it from the fighter list.
5. Create a folder called “FirstnameLastname” inside of the “BattleRoyal” folder of this DocPac
   1. If there is no “BattleRoyal” folder, create it
6. Put all of your project files in this new folder
   1. DO NOT include the node\_modules folder.
7. Open a Pull Request to the DocPac Repo on Github, or commit to the branch you used for your open Pull Request this week.

# Reflection

**What is one technique you learned in your coding this week, and what kinds of things can you use it for (the more specifics the better) ?**

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

**What was a challenge to your time management this week, and what can you do to avoid it in the future (specific plan) ?**

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

**Why did you pick the specific Codecademy chapters or Project Issue that you did this week? How do you think it will benefit you in the future?**

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

# 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. | * All assignments start at 9/10 possible points * 1 point is deducted per infraction   + Lateness   + Mistakes   + Unprofessionalism   + Not following instructions   + Etc. * Outstanding submission increase by 1 point and yield a pog. |
| 9 | 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. |

# [S] Weekly Contribution / [J] Weekly Study

# DocPac/Reflection