IMY 220
Assignment 2

Node & NPM

General Instructions

- This assignment must be completed and submitted by the due date which is available on ClickUP.
- This assignment is a take-home assignment and may take one or two days to complete.
- This assignment should be completed on your own, but you may come to the tutorial session or email the assistant lecturer if you need further assistance.
- No late / email submissions will be accepted.

This assignment focuses on working with **NodeJs and NPM packages (Express and Socket.io)**. You will be required to create a simple voting application that allows multiple users to connect and vote on a topic.

Download *index.html* from ClickUP. This file contains the basic HTML for a voting application where users can vote for what they think a cat's name should be.

Part 1 - Setting up a Basic Project with Express and Socket.io

This assignment requires you to have the latest version of Node.js installed. If you haven't already, install it here: https://nodejs.org/en (This will install Node.js and NPM)

- You can test whether Node and NPM are installed correctly by running the commands `node -v` or `npm -v` respectively in your terminal.

Once that is done, run `npm init` from your terminal in the folder where you would like your assignment code to be, to initialise a simple NPM package as described in the lectures. Follow the prompts. Your *package.json* should have the **following field values**:

- **name**: assignment-2
- **description**: IMY 220 Assignment 2
- **author**: [Your name and Student Number]

The rest of the values can be the default values from running the command.

Next, install **express** and **socket.io** from npm as described in the lectures. Afterwards these should appear as dependencies in your *package.json* file. You should also have a *package-lock.json* file, as well as a *node_modules* folder. **Do not submit your node_modules folder in your assignment submission**.

Ensure that *index.html* is also in the root folder. Then, create three JS files, also in the root folder:

- **index.js** - for your client-side code
- **server.js** - for your server-side code
- **poll.js** - your own module for your `Poll` class

The functionality in each of these files is described in the following sections.

Part 2 - Express Server

In your **server.js** file, create a simple express server as described in the lectures. The server should listen on port 3000. On startup, `Listening on http://localhost:3000` should be logged to the console.

When the user visits `http://localhost:3000` they should be served *index.html*.

Test this functionality before moving on to the next section.

Keep in mind *throughout this assignment that your server code will be running in whatever terminal you run the npm command to start the server (e.g., in your text editor or a windows / mac terminal),* ***not*** *the browser console. Only your client-side code will show in the browser console.*

Part 3 - Basic Socket.io Message Passing

In your **server.js** file, set up the necessary code to have sockets on the **server-side** as described in the lectures. This will involve modifying the express server code you just wrote.

- When a user **connects**, the server should log to the console '`A user connected with ID:`' and then the ID of the client's socket that just connected (*hint: google how to get the socket ID*).
- When a user **disconnects**, '`A user disconnected`' should be logged to the console.

Note that this code won't work until the client side code is implemented (below).

In your **index.js** file, write all the code to allow sockets on the **client-side** (i.e., in the browser) as described in the lectures.

- When the user **connects**, the client should log '`I connected with ID:`' and then the client's socket ID.

Include index.js in *index.html*.

At this point you should notice that when you try to test this code, **it doesn't work**. That is because the client doesn't have access to your *index.js* file. The server is not sending it because it doesn't know how to handle the request. Thus, to fix this, add a **route** to your express code in *server.js* that points to *index.js* (i.e., '**/index.js**') and then serve the corresponding file. Make sure to set the '**Content-Type**' header to '**application/javascript**' so the client knows it's a JS file.

Refer to this discussion for further clarification:
https://stackoverflow.com/questions/62142037/how-to-include-socket-io-in-external-js-file-linked-to-html-file

Test this before moving on to the next section.

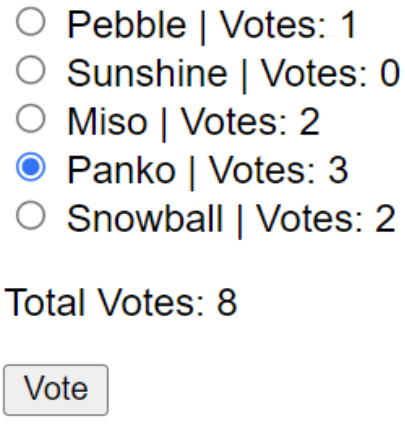
Part 4 - Voting

Once the previous sections have been completed, you should have a basic socket.io application that allows you to pass messages from server to client and vice versa.

Now work on creating the following functionality, making use of the **existing HTML** (you do not need to modify this at all), **basic JS**, and **sockets**:

1. A user should be able to **vote for a cat name** by clicking on one of the names, and pressing 'Vote'. The option that was voted for should be sent to the server. A user **can vote multiple times** for ease of testing.
2. All users should have the **votes live-update** every time a user votes for a name. That means the **total** number of votes as well as the votes for **each name** should update to the screen each time a user votes. This information should be sent from the server so all clients have access to the same information simultaneously.

For example:



It's **strongly recommended** that instead of defining this functionality in the socket events, to create **separate functions** and call them from within socket.on(). This will make your code much easier to manage.
For now, when the user **reloads** the page (i.e., disconnects and reconnects), the votes can be reset to zero. Test this functionality before moving on to the next section.

Part 4 - Poll Module & State Management

Right now, our server has no way of storing how many votes were cast or which names were voted for. It simply updates the clients when someone votes. Let's fix this by creating our **own module**, a `Poll` class that stores and manages the votes cast for each cat name.

In **poll.js**, create an ES6 class called `Poll` with the following functionality:

- A **constructor** that takes in a JS array of objects where each object has a field for the cat name, and the number of votes cast for that cat name. This array should be saved to a member variable.
- **`vote(name)`** - A function that takes in the cat name (string) that was voted for and updates the corresponding object in the member variable array accordingly.
- **`getVotes()`** - A function that returns the member variable.

Export this class as taught in the lecture videos and **import** it into **server.js**.

- **Instantiate** the class with an array of objects which has one object for each cat name and 0 to represent no votes (yet) for the cat name.
- **Change the socket functionality to make use of the `Poll` class**. Users that connect and/or reload the page (reconnect) should be sent the current number of votes for each cat (and from this should be able to calculate the total votes, however you may add this to your `Poll` class if you wish).

The server only has to keep track of the number of votes as long as it's running. That is, you do **not** need to write the votes permanently to a file, the `Poll` class can just be used in *server.js* and the data can be lost when the server shuts down.

Bonus - Live Feed

For bonus marks, create a **live feed** that logs which user has voted for which cat name as a user votes for it. Users should be differentiable to get the mark, so create some sort of way of identifying users (e.g., an ID or a substring of it, a username, etc.) This can be automatically given by the server, that is, you don't need to create any sort of account functionality, just a way of identifying different connections (*hint: we have already done most of this above*).

An example of this is as follows:

> Live Voting Feed (bonus):
>
> User MBmtF voted for pebble
>
> User 3Cr_L voted for panko
>
> User MBmtF voted for snowball

Submission Instructions

Place these in a folder named `A2_u12345678` where `12345678` is your student number.

- index.html
- index.js
- server.js
- poll.js
- package.json
- package-lock.json
- **Do not submit your node_modules folder. This will result in -10% from your assignment mark.**

Zip the **folder** and upload this to ClickUP in the relevant submission slot before the deadline.