Group 40 Portfolio 3

Scott Mueller & Sam Eue

Our project is an online Jeopardy! game that runs on a Node.js server and uses Angularjs and HTML front-end.

Contents

1 Overview

Inspired by our previous project and an event hosted by the same club we worked with for the previous project we decided to make a system to run Jeopardy! on any web enabled device. This game is divided into 3 separate ‘views’; *player*, *spectator* and *‘Trebek’*. The spectator view is designed to be put onto a projector or main screen for all individuals to involved to see, when a category and value are selected the gameboard is replaced by the ‘Hint’. On the player screen we have a login that communicates the player’s name to the server and moves to a screen that displays a button to buzz in for the question and the current scores of the players. Finally on the Trebek view we have the game board which the host can select a question based on the player’s input, this changes his view to show the question, answer and three buttons allowing him to score the player’s answer or back out if no answer is given. The system uses a Node.js backend and Angularjs to control the front end views.

2 New and Complex

2.1 New

For the display of data on the front end html pages we decided to use angularJS to speed up the time it takes to display new information. We have several NG-Views for different stages of the game that we swap into and out of; for the player we have a login to obtain user information which transitions to gameplay, for spectator we have the game board that transitions to the selected question and for Trebek we have the game boar that transitions to a game control. In each view we use Angular expressions to pull data real time from the Node file reads. Finally we had several controllers to help with inner logic and page transitions, giving us a simple way to gather data to report to the server for backend processing.

The server runs on Node.js using the HTTP module to handle page requests as well as Socket.io to handle server/client communication. The Node.js backend allowed us to read files on the server and send them to the client when a request was made for them using the HTTP module. Socket.io allowed us to pass data between the server and its clients by creating a persistent websocket connection between the two points. The game is intended to be run on at least 3 different clients and requires communication between the clients to function. Due to this we maded extensive use of Socket.io to allow the server to act as a relay between clients.

2.2 Complex

When clients connect to the server we want to send them the questions file for the game. The questions are stored in the questions.json file on the server. Since we want to cut down on the response time when clients connect, we have a function load the file into the server's memory when the server starts and the data is then sent to the client when it connects. Anytime a player connects we need to let the clients know. Here is an example then when we use the server as a relay. A player connects they send a username to the server. The server stores this and then sends it to all players. Another example of the server as a relay is when Trebek selects a question. We need the spectator view to change to display the question and so Trebek sends the category and value of the question to the server and the server retransmits it to the spectator view. A similar thing is done when we want to go back to the table view.

For the Angular side of the code we set it up so the data read in from Node would display through modifications such as with the player scores, as well we used Angular directives to build the question board and scoreboard, this allowed for a simple way to have modular structure and combined with NG-View helped create a cleaner structure to the overall code and front end appearance. Finally the use of controllers made it simple to control the way each view interacted with the backend code, for example Trebek controls the game from his ability to change views to his ability to change player scores based on their answers.

3 Bloom's Taxonomy

Synthesis/Creation, Analysis, Evaluation

Node.js comes with many modules and allows for the inclusion of many more through its package manager npm. Built on JavaScript it runs on the desktop/server side of things in the terminal instead of the traditional execution environment of JavaScript which is in a web browser served to the client by a remote server. While the HTTP module comes pre-included in the installation of Node.js, setting up a working HTTP server is not as install and forget as Apache HTTP Server is. The start of a server requires the user to write a JavaScript file that includes commands on what the server is to do when a client connects to it and makes a request for a file as opposed to Apache which natively handles the reading of the data and transmission to the client. Since Node.js is server side JavaScript, it is able to read files using the pre-included module fs, which stands for File System and provides simple wrappers around standard POSIX functions. The file data can then be sent to the client fulling its request through the HTTP module.

Further communication to the client however requires a websocket to be opened to the client and requires the user to download the Socket.io module. Socket.io enables real-time bidirectional event-based communication. This persistent connection to the clients allows us to send data besides just HTML files and the like through the HTTP connection. We initially use the websocket to transmit the question data to all the clients when they connect. Later we use the websocket's bidirectional communication ability to send event notifications to control the each of the player's views based on the actions Trebek takes throughout the game.

The use of angular helps control the front end appearance of our web app without having too much extra backend code to deal with. Our use of directives help create the overall appearance by using the repeat function to run simple loops in the html for the inclusion of data from an ordered set up, expressions were useful to put data from the server into the pages quickly and simply while still being able to edit the data and update it on the page without needless page reloads. Applications like NG-View are useful for the multiple views needed to make this app function smoothly, from login to the question board to the questions themselves, having these as small elements we can swap in and out of the page keeps the system neat. To top it all off, by attaching angular controllers to each view we can have it function in different way depending on what page the user is loading the view from, such that the spectators go from the board to a display with just the question instead of a view that contains the answer such as what happens for Trebek. Overall these systems help to streamline the process of operating the app so that the end user doesn't need to worry about the technical details of how various elements are handled, from one page we can handle the control of all three different views without the user needing to do anything terribly complex.