Switch from PPT --> new Node.js CMD process

>cd ..

>cd ..

>cd git

>dir

======

We're going to create a new folder, and later initialize it as a Git repository after we've made some code files.

======

>mkdir ThatChat

>cd ThatChat

(I’ll call mine ThatChat, for ThatConference).

======

In this demo, we’re creating a basic chat application using Node.js and Socket.io. The first goal is to setup a simple HTML webpage that serves out a form and a list of messages. We’re going to use the Node.JS web framework 'express' to accomplish this.

First let’s create a 'package.json' manifest file that describes our project.

All npm packages contain a file called package.json. This file holds metadata for the project, such as the project’s name, version, a description, and its dependencies.

======

>code package.json

{

"name": "socket-chat-example",

"version": "0.0.1",

"description": "my first socket.io app",

"dependencies": {}

}

======

Now, in order to easily populate the dependencies space with the things we need, we’ll use npm install --save:

======

npm install --save express@4.10.2

npm install --save socket.io

======

Express and Socket.io are a few of the things that the Node Chatroom application needs to function properly. That's why they're called dependencies. Express is a web framework that lets you structure your web app to handle multiple different HTTP requests at a specific URL.

You also implement Express to use multiple templating engines to generate HTML.

Socket.IO is a simple JavaScript library and Node.js module that allows you to create real-time bidirectional event-based communication apps. It simplifies the process of using Web Sockets significantly. In short, implement Socket.IO to connect, send, and receive messages.

Now that those are installed, we can create an app.js file that will setup our application.

======

code app.js

var express = require('express');

var app = express();

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

var io = require('socket.io')(http);

var port = process.env.PORT || 3000;

app.get('/', function(req, res){

res.sendFile(\_\_dirname + '/index.html');

});

io.on('connection', function(socket){

socket.on('chat message', function(msg){

io.emit('chat message', msg);

});

});

http.listen(port, function(){

console.log('listening on ' + port);

});

Explaination: In the first line, we're requiring a module called 'Express.' Then we create a variable that's instantiated to 'Express.'

In line 2, Express initializes 'app' to be a function handler that you can supply to an HTTP server (as seen in line 3).

In line 4, I initialize a new instance of 'socket.io' by passing the 'http' (the HTTP server) object. Then I listen on the 'connection' event in line 9 for incoming sockets, and at the end I log it to the console.

On line 5, we make the http server listen on port 3000 -OR- use the process environment port for hosting it on Azure online.

We define a route handler '/' that gets called when we hit our website home. We’re also calling 'res.sendFile' to pass it an HTML file, which we will now create.

======

C:\git\ThatChat\>code index.html

<!doctype html>

<html>

<head>

<title>Socket.IO chat</title>

<style>

\* { margin: 0; padding: 0; box-sizing: border-box; }

body { font: 13px Helvetica, Arial; }

form { background: #000; padding: 3px; position: fixed; bottom: 0; width: 100%; }

form input { border: 0; padding: 10px; width: 90%; margin-right: .5%; }

form button { width: 9%; background: rgb(130, 224, 255); border: none; padding: 10px; }

#messages { list-style-type: none; margin: 0; padding: 0; }

#messages li { padding: 5px 10px; }

#messages li:nth-child(odd) { background: #eee; }

</style>

</head>

<body>

<h1>Node.js & Socket.io Chat Room!</h1>

<ul id="messages"></ul>

<form action="">

<input id="m" autocomplete="off" /><button>Send</button>

</form>

<script src="https://cdn.socket.io/socket.io-1.2.0.js"></script>

<script src="http://code.jquery.com/jquery-1.11.1.js"></script>

<script>

var socket = io();

$('form').submit(function(){

socket.emit('chat message', $('#m').val());

$('#m').val('');

return false;

});

socket.on('chat message', function(msg){

$('#messages').append($('<li>').text(msg));

});

</script>

</body>

</html>

======

All it takes to load the socket.io-client, which exposes a io global, and then to connect, is this script that says var socket = io();. I’m not specifying any URL when I call io(), since it defaults to trying to connect to the host that serves the page. The main idea behind Socket.IO is to send and receive any events you want, with any data you want.

When the user types in a message, the server gets it as a chat message event. The scripts section in index.html contains socket.emit. So when the program captures a chat message event, it's included in the page.

Now if we navigate back to node and run app.js, we should see the following:

======

>node app.js

"listening on 3000"

Navigate to http://localhost:3000/ in In Private Internet Explorer & Firefox

(((website: jlord.us/git-it/challenges/repository.html )))

C:\git\ThatChat\>node app.js --> ^C

======

Now we would like to initialize our ThatChat folder as a Git repository.

Open new instance of Git Bash

======

>cd .. >cd .. >cd git >cd ThatChat

To create a new Git instance for a project:

>git init

That's it! It will just return us to a new line. If we want to be extra-sure that it's a Git repository, type:

>git status

Now that we've got a repository started, let's add our local files to it.

>git add .

Next, commit those changes to the repository's history with a short description of the updates.

>git commit -m "Initial Commit"

At this point, it's time to go to github.com, log in, and click the '+' in the top right to create a new repository.

Give it a name that matches your local repository's name, 'ThatChat', and a short description.

Make it public. DON'T Initialize with a README.

Leave .gitignore and license on 'none'.

Click create repository!

Copy to clipboard the URL from GitHub.

======

Back in your terminal, and inside of the 'ThatChat' folder that I initialized as a Git repository a second ago, I want to tell Git the location of the remote version on GitHub's servers. You can have multiple remotes, so each requires a name. For the main one, this is commonly named 'origin'.

======

Git Bash:$git remote add origin https://github.com/SarahSexton/ThatChat.git

======

Now my local repository knows where my remote one named 'origin' lives on GitHub's servers.

Next, we want to push (send) everything we've done locally to GitHub.

Git has a branching system so that you can work on different parts of a project at different times. By default, the first branch is named 'master'. When you push (and later pull) from a project, you tell Git the branch name you want and the name of the remote that it lives on. In this case, we'll send our branch named 'master' to our remote on GitHub named 'origin'.

======

git push origin master

Now go to GitHub and refresh the page of your repository. Ta-da! Everything is the same locally and remotely. Now we're ready to publish our chat room to Azure to share it with the rest of the world.

Now go to https://azure.com > click "Portal" > Web Apps

Click '+' > Quick Create > Find a unique URL.

Create Web App

Click on what you just made > Dashboard > Set up deployment from source control.

======

Deploying your web app from GitHub or any other External Repository results in a continuous deployment process where Azure will pull in the most recent updates from your project.

======

GitHub > Authorize > Choose a repository to deploy > ThatChat > master.

Linking GitHub repo... Deploying... Fetching changes... Active

Dashboard > URL > Navigate to page <UniqueURL.azurewebsites.net>

======

Continuous deployment is useful when you have multiple people working on a project and want to ensure that the latest version is always published regardless of who made the most recent update. Continuous deployment is also useful if you are using an online tool as the central repository for your application.

======

================

To delete everything:

Close all notepads, Visual Studio windows, browsers, and file explorers referencing ThatChat

Delete git/ThatChat folder

Navigate to GitHub.com, delete ThatChat repo

Navigate to Azure.com, delete UniqueURL web app