

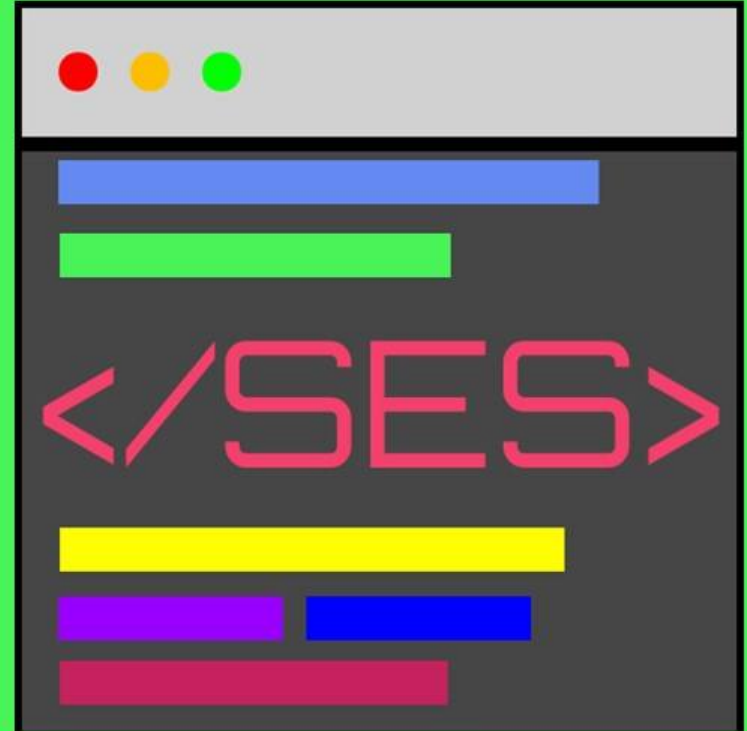


GIT Ahead

The development tutorial series to help you get an edge in a competitive job market.



Presented by:



And

DevShare

DevShare



// What is DevShare

An initiative to have developers share interesting and relevant topics, with their fellow students and developers

// Today's Dev's



Pavithran Pathmarajah
Sfwr-Eng IV

GIT Ahead Pt.2

A two part workshop series

// Motivation?

By the end of today we'll have a functional chatroom. That you can use to send messages securely. On our own URL!

Checkout: devshare.space/chat

// What you will learn

Today:

- JavaScript
- Node.js Server
- Web Sockets using Socket.io
- Microsoft Azure Webapps
- Flex on literally **EVERYONE**

// Follow along

Get the slides

Slides: devshare.space/slides

// Agenda



Part 1.

JavaScript

“

*JavaScript is the world's
most misunderstood
programming language*

- Douglas Crockford

// What is JavaScript

HTML

Provides the structure for a website. With pre-build structures for headings, tables, videos, etc.

CSS

Provides styling for the structure. Allowing animations, backgrounds, overlays etc.

JavaScript(JS)

A system that enables websites to be dynamic. When a website does anything more than sit there. It is because of JavaScript

JavaScript - THE MINIMUM



// Data Types

- `let someString = "Hello World"`
- `let someNumber = 47`
- `let someBoolean = true`
- `let literalNothing = null`
- `let someArray = [1, 2, 3, 4, 5]`
- An object is like a dictionary
 - `let someObjet = {}`
 - `someObject['Hello'] = "World"`
 - `someObjet = {"Hello": "World"}`
 - JavaScript Object Notation (JSON) is born

// Boolean Operations

- Basic Boolean Operations
 - `True != False, True == True, "Hello" === "Hello"`
- What happens with other data types?
 - `!0 == True, !1 == False, !2 == False`
 - `!{"Hello":"World"} == False, !null == True`
 - `'!'` will be false for anything that has a value
- Ternary operator, lets you select based on boolean
 - `(True ? "A" : "B") == "A", (False ? "A" : "B") == "B"`

// Selectors

- `if (some statement) {
 } else if {
 } else {
 }`
- `Switch (name) {
 case 'Jeff':
 // Do Something
 break // leaves the currently executing statement
 default:
 // Do Something
 }`

// Loops

- `while(true) {`
 `// While Loop`
`}`
- `do {`
 `// Do While Loop`
`}while(true)`
- `for(let i=0; i<10; i++) {`
 `// Basic For Loop`
`}`
- `continue` `//Will move onto the next iteration of the loop`

// functions

- `function name(parameter, parameter) {
 // Do something with the parameters
 return "Hello World"
}`
- `let name = function(parameter = "Default") {
}`
- `let name = (parameter) => {
 // Arrow function a shorthand function
}`

// User/Page Interaction

- Retrieve an element from the HTML document
 - `let el = document.getElementById('#ID')`
 - `el.innerHTML = "HELLO WORLD"`
 - Can do everything HTML supports via code
- **Popups - Launch a built in browser dialog box**
 - `alert("Hello World")`
 - `confirm("Ok/Cancel")` // Returns True/False
 - `prompt("Input Plz", "Default Text")` // Returns user input

// There is way more

- JavaScript
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript
- Interacting with the DOM
 - https://www.tutorialspoint.com/javascript/javascript_html_dom.htm
- Youtube Tutorial:
 - <https://www.youtube.com/watch?v=PkZNo7MFNFg>

Part 2.

Our Chat App

// We have 3 main parts

The website that the user interacts with, we made a rough one last week.

The server that will be accepting connections, and managing the chatting. This will be done with a physical Azure Server and Express on Node.js to run the server.

The socket connection handling on the server, to create pipes between our different users. For this we are going to be using socket.io

Node.js



// How to install Node.js & NPM

MacOS/Windows/Linux

- Download and install Node.js LTS Version
- <https://nodejs.org/en/download/>
- NPM is Node Package Manager and will install alongside Node.js

// What is Node.js

- JS stands for JavaScript
- Node is a runtime environment so that you may execute JS code without having to use a web browser.
- Node.js is typically used as a back-end, and are used in production on backend servers and API's
 - Used by Uber, Netflix, PayPal, LinkedIn, etc.
 - We are going to be using it to host a web server, to allow multiple browser sessions exchange data
- Node.js is actually based on the same underlying engine as Google Chrome

// What is NPM

- NPM is the Node Package Manager
- When working with Node.js it is expected to use a variety of packages, that have already been built some of which that interface directly with the computer.
- Node Packages are essentially, Java libraries

// Setting up our Node App

- Go to your folder and run:
 - `npm init`
- Fill in all the necessary info
- We have a node app, a package.json has been created
 - This package tracks all dependencies needed to run your application
- Now in the package.json to simplify things we'll add a line under "scripts"
 - `"start": "node index.js"`
 - Now if we call `npm run start` our index.js will run

Express

Let's make this app a web-server

// What is Express

- A Node Package that acts as a web server. Handling everything under the hood.
- Makes hosting a web server that supports, requests very easy.

// Adding Express

- In our directory with our package.json
 - Call `npm i express --save-dev`
 - This installs the package, and adds it as a dev dependency so it can be easily installed later, by other devs
 - You'll also see a file called `package-lock.json` this file tracks the current version of your installed packages for compatibility (ensuring the min/max version is installed)

// Using Express

- Basic Code

```
// Import express and create instance
const app = require('express')();
// Create Instance of http server
const http = require('http').Server(app);
const port = 8080; // Port to use for server

// Return HI to localhost:8080
app.get('/', (req, res) => res.send('HI'))

// Server starts listening
http.listen(port, () => console.log(`App Online`))
```

// The GET request

- `server.get('/', (req, res) => res.send('HI'))`
- Two Parameters the URL, and the function to handle the request and response
- The URL to catch
 - `'/'` Catches that URL `localhost:8080`
 - `'/Hi'` Catches that URL `localhost:8080/Hi`
 - `/*` Catches all URLs `localhost:8080/Hi/test/dsadsa`
- Handle the request and the response
 - `(req, res) => res.send('HI')` is a short arrow function
 - Ignores the 'request' and only sends a 'response'

// The Other Requests

- Really easy to handle all REST requests
- GET
 - `server.get('/', (req, res) => res.send(GET))`
- POST
 - `server.post('/', (req, res) => res.send('POST'))`
- PUT
 - `server.put('/', (req, res) => res.send('PUT'))`
- DELETE
 - `server.delete('/', (req, res) => res.send('PUT'))`

// Static Hosting

- We want to share our public folder as it is
 - `server.use('/public, express.static('public'))`
- Now whenever we add something to our public folder it will be available at /public

// Just the Surface

- Express can do a lot more, this is just some basic functionality
- <https://expressjs.com/>

SOCKET.IO



// What are Sockets

- A socket is a hollow that you plug into
- Sockets are used to connect two pieces of software together using a pipe
- In our case we are referring to network sockets, which act as an over the network connection between two programs.
- We are going to be using this to establish a connection to our server so that our chat app can talk to other chat apps.

// Adding SOCKET.IO

- In our directory with our package.json
 - Call `npm install socket.io --save-dev`
- In our index.js we need a new import

```
const io = require('socket.io')(http);
```

- Imports socket.io and uses out http server
- Add a simple connection method to our server

```
io.on('connection', (socket) => console.log('Connected'));
```

- Called when a socket connection is established

// Connect on our webpage

- Include socket.io script in our chat app

```
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.slim.js"></script>
```

- Connect to the socket on the web app we've created

```
<script>  
  let socket = io(http://localhost:8080);  
</script>
```

// Cross-Origin Resource Sharing

- A common issue you have heard of or may run into is CORS, which is when we try and access things across origin (URL Based).
- In this case we are trying to have out local file access a server running on localhost, throwing the security error
- We can handle this by allowing CORS

// Allowing CORS

- In our directory with our package.json
 - Call `npm install cors --save-dev`
- In our index.js we need a new import

```
const cors = require('cors')
```
- Add cors to our application

```
app.use(cors())
```
- Now we're done!

<https://www.npmjs.com/package/cors>

// Back To Sockets

- Now we can connect!
- Socket.io is simple in it uses these 'on event' receives to process and send messages.
- To send data
 - `socket.emit('identifier', data)`
- To receive data
 - `socket.on('identifier', data)`

// Sending Data 3 ways

- Direct one connection / To server
 - `socket.emit('event', data)`
- Broadcast to everyone
 - `io.emit('event', data)`
- Broadcast to all but one connection
 - `socket.broadcast.to('room').emit('event', data)`

// Rooms

- On the server join room
 - `socket.join('room')`
- On the server leave room
 - `socket.leave('room')`
- The benefit of a room is that communications stay within that room, allowing for multiple simultaneous sessions with minimal tracking of our own needing to be implemented

// Let's get our app working

- Now we're going to use everything we got so far to get our app online!

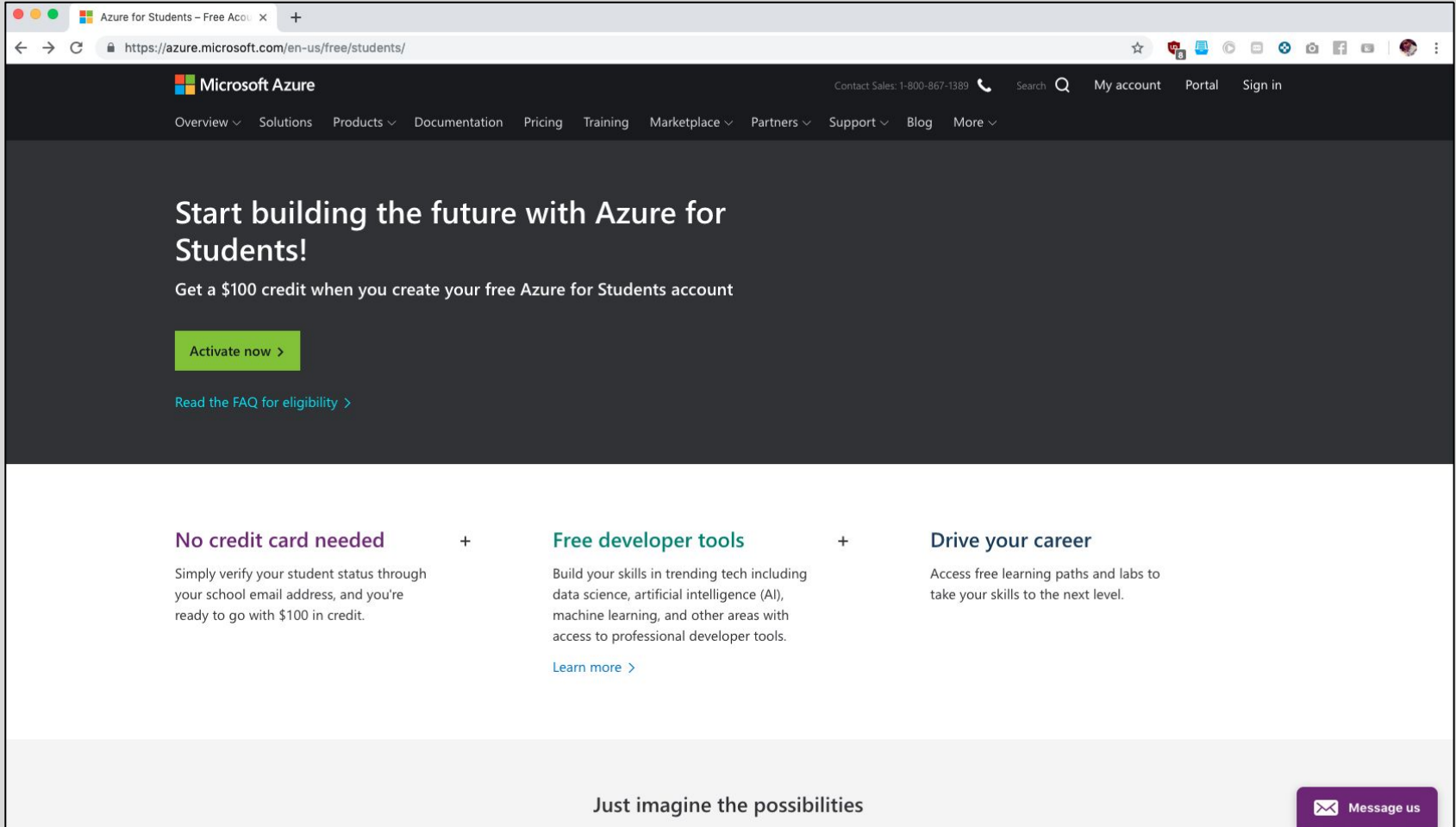
Part 3.

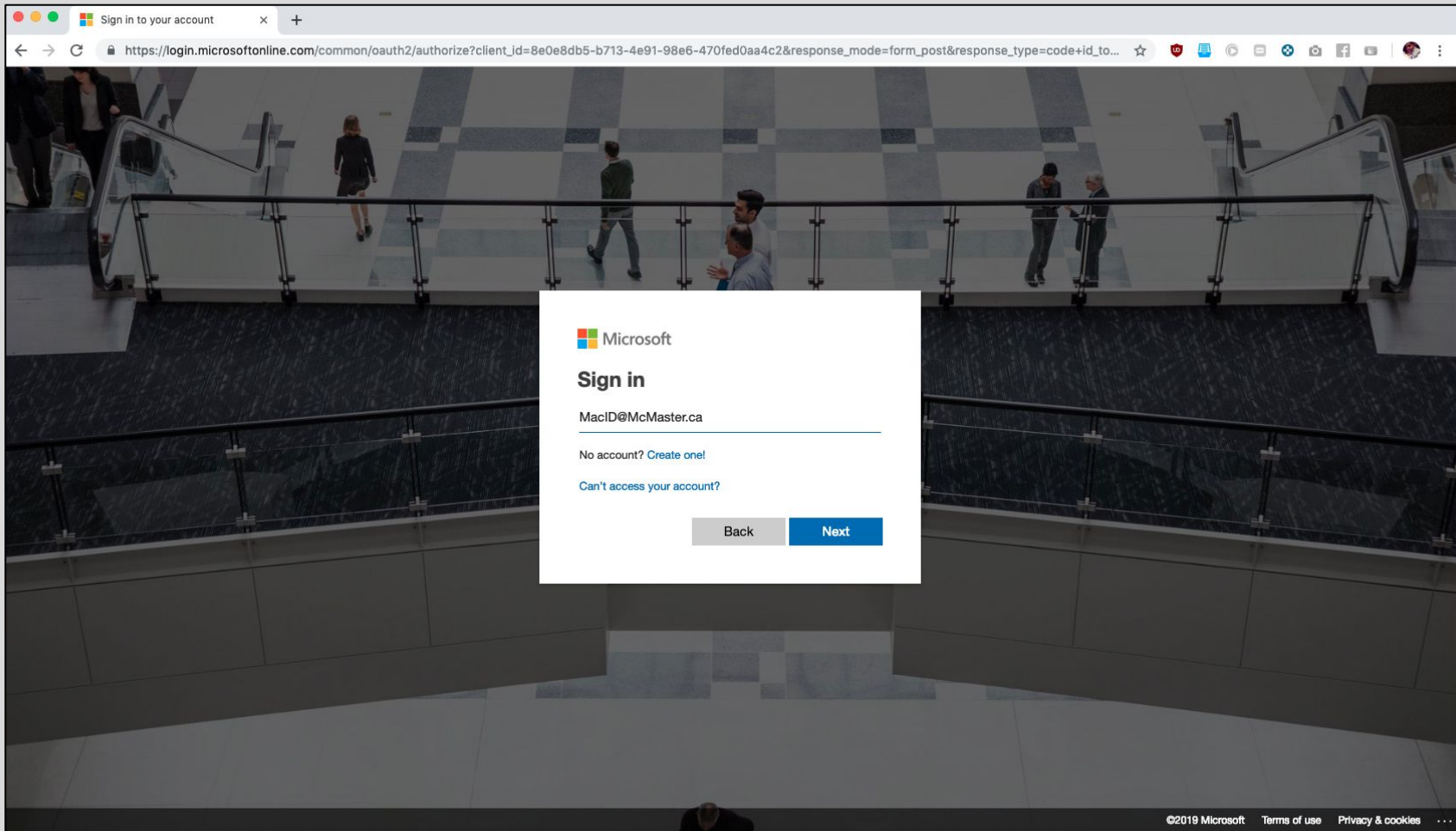
Microsoft Azure

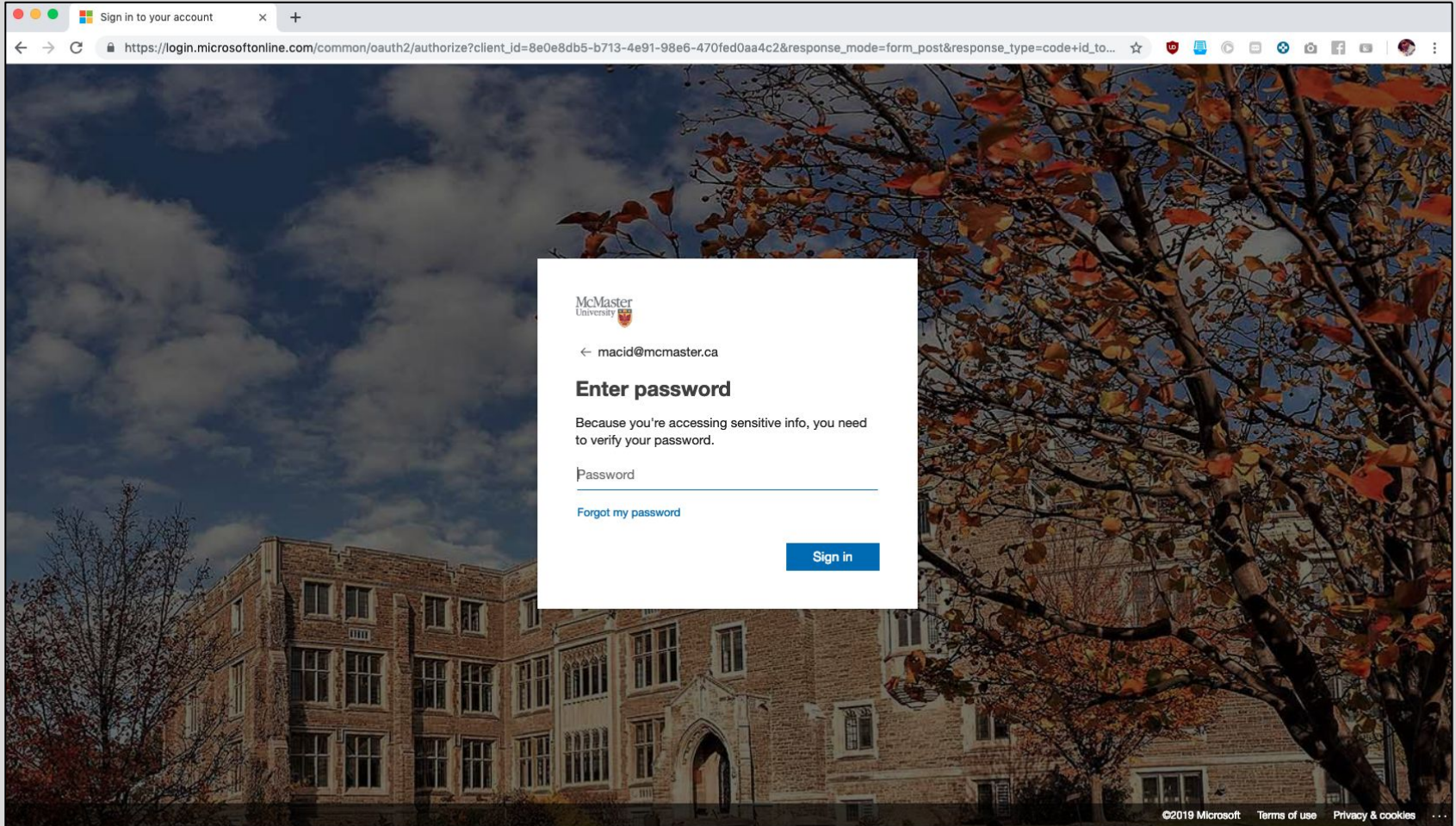
// Register

- As students we get access to a lot of cool stuff for free, one of those things is some free hosting on Microsoft Azure

<https://azure.microsoft.com/en-us/free/students/>







Dashboard - Microsoft Azure

https://portal.azure.com/#@mcmaster.ca/dashboard/private/00834822-db6c-4d8c-be05-bc6d26fa800d

Microsoft Azure

Search resources, services, and docs

pathmap@mcmaster.ca
MCMMASTER UNIVERSITY

Dashboard

+ New dashboard ↑ Upload ↓ Download ✎ Edit ➦ Share ↗ Full screen 📄 Clone 🗑 Delete

Create a resource

Home

Dashboard

All services

FAVORITES

All resources

Resource groups

App Services

Function Apps

SQL databases

Azure Cosmos DB

Virtual machines

Load balancers

Storage accounts

Virtual networks

Azure Active Directory

Monitor

Advisor






Security Center

Cost Management + Billing

Help + support

All resources
All subscriptions

Refresh

	EltroTech	App Service
	capstone	SQL database
	ServicePlanda210848-8556	App Service plan
	PPATH	App Service
	eltrotech	SQL server

Azure getting started made easy!

Launch an app of your choice on Azure in a few quick steps

Create DevOps Project

Quickstarts + tutorials

Windows Virtual Machines

Provision Windows Server, SQL Server, SharePoint VMs

Linux Virtual Machines

Provision Ubuntu, Red Hat, CentOS, SUSE, CoreOS VMs

App Service

Create Web Apps using .NET, Java, Node.js, Python, PHP

Functions

Process events with a serverless code architecture

SQL Database

Managed relational SQL Database as a Service

Service Health

Marketplace

// We want to create a webapp

- New Resource
 - Search WebApp
 - Create!
 - Node.js LTS
- Once Created we want to deploy the application from our Github REPO
 - Web-App
 - Deployment Center
 - Authorize with Github and select the Repo
- Now our changes will automatically be deployed

// Our app is on Azure

- New Resource
 - Search WebApp
 - Create!
 - Node.js LTS
- Once Created we want to deploy the application from our Github REPO
 - Web-App
 - Deployment Center
 - Authorize with Github and select the Repo
- Now our changes will automatically be deployed

// We are Online!

- This is just the basics
- It was a lot of information but, you can use this to build a variety of applications
- Everything from online room based games, to a crude

GIT Ahead

Now you got a ChatApp!