**Back-End Overview**

1. There is no single consensus of the ‘best backend’ framework/library/languages
   1. Unlike front-end, where React dominates
   2. Most common back end types:
2. CRUD
   1. Create
   2. Read
   3. Update
   4. Destroy
3. MEAN Stack - MongoDB, ExpressJS, AngularJS, NodeJS
4. End Points - what comes after the API address, path + verb
   1. Exs: myapi.com/search, myapi.com/findCollection, myapi.com/showRandom
5. Build a super simple API
   1. Build a package JSON file:
      1. yarn init (or yarn init -y)
   2. Create index.js
      1. Add node package of express ‘yarn add express’
      2. Terminal: ‘yarn install’
      3. index.js: const express = require(‘express’)
      4. index.js: const app = express()
      5. index.js: app.get(‘/hello’, (request, response)=>{

response.json( {hello: ‘world’} )

})

* + 1. index.js: app.listen(5000)
    2. index.js: console.log( ‘api is up and running’ )
  1. Run it
     1. Terminal: node index.js
  2. Nav to localhost:5000/hello in browser
     1. See result
     2. 3000, 5000, 8000, 9000 are typically safe ports to use
  3. In package.json: “scripts”: { “start”: “node index.js” } //tells yarn to ‘node index.js’ when ‘yarn start’ is input in terminal

1. Things you can do...
   1. Dependencies to add (think boiler plate)



* 1. Example Get Request for all of ‘tasks’:



* 1. Example Get Request for a single ‘task’



* + 1. How it works
       1. Pulls the ID from the request and saves it in a local variable
       2. Create/assign variable ‘task’ with a corresponding task in ‘tasks’ that matches the id
       3. Responds with a JSON response of ‘task’
  1. Example Post Request



* + 1. Add app.use(express.json()) to top of file, enables the app to use incoming JSON
    2. Allows post requests (ex, a post request from postman)
    3. How it works
       1. Creates a variable ‘task’ that reads JSON
       2. Contents of ‘task’ is pushed into ‘tasks’
       3. Server responds with ‘task’ as validation
    4. Things to add:
       1. Global constant ‘id’
       2. Add an id to each incoming post request, store in ‘tasks’ as an object with id + content
       3. Increment ‘id’ each time a post is submitted



* 1. Example Delete Request



* + 1. How it works
       1. Creates the var ‘task’ and assigns it to the task in ‘tasks’ matching the parameter id
       2. Creates the var ‘index’ and assigns it the value of the task in ‘tasks’
       3. Splice ‘removes’ the array in the position of ‘tasks’ that matches the index
       4. Responds with the removed value
  1. Example Put Request



* + 1. How it works
       1. Creates a var ‘task’ with the body of the request
       2. Confused on step 2
       3. Creates a var ‘index’ with
  1. Create a function to store data into a file
     1. Require the file system library
        1. In app.js or index.js: const fs = require (‘fs’)
     2. .js file: function saveTasks(){



* + 1. Call this function in .put, .delete, .post functions to make the data live on, forever!
       1. Be sure to edit your id system to account for server stop/starts
  1. Create a function to read data from a file
     1. Require the file system library
        1. In app.js or index.js: const fs = require (‘fs’)
     2. .js file: function getTasks(){



1. .parse and .JSON are opposites
   1. .parse() takes a JSON file and turns it into a JavaScript object
   2. .json() take a JavaScript object and turns it into a JSON file
2. Idempotent - does not affect the data, can run it infinite times without modifying the original
3. ‘client’ or ‘frontend’ are good names for half your app
4. CORS - Cross Origin Resource Sharing
   1. Because the database lives on localhost:5000 and my React page is on localhost:3000, the browser will complain about CORS
   2. To solve, in package.json in the client folder, add a key before the end of the object:



* 1. Now you need to change all references of localhost:5000 to say 3000 or remove the beginning (localhost:5000) so the routes simply say ‘/api/tasks’

1. Axios - for the Client Side
   1. In terminal in client: yarn add axios
   2. Import Axios into your JavaScript file



* 1. Example Get:



* 1. Example Post:

