**Tech Notes to help with final project**

**Mongoose**

* GET/FETCH methods data + display with React
* State = { }
* Elegant mongodb object modeling for node.js “row of data”
* Cross-Platform document-oriented program (BSON = compressed JSON)
* Can also use custom methods
* findOneandUpdate(); method
* \*Switch Statements
* Use \_\_\_\_ creates db
* db. \_\_\_\_\_ .remove
* db. \_\_\_\_\_ .find()
* db. \_\_\_\_\_ .drop()
* db. \_\_\_\_\_ .update()
* db. \_\_\_\_\_ .insert() One/Many
* db. \_\_\_\_\_ .find().sort({})
* db. \_\_\_\_\_ .save
* Object Relational Mapping
* Creating an object store for db
  + Indexdb.db.open
    - Const = event.target.result
    - Const.createObjectStore(‘nameofdb’)
      * Req.onSuccess = evet { }
    - Cavoite: hard to query, must create indexes to query columns
      * objectStore.createIndex(“nameofindex”, “nameofparameter“ )
        + cursor.value.status

cursor.continue

Example syntax to connect to db

* Good practice to group all route definitions into a file / directory of its own
* Can create sub routes

|  |
| --- |
| var express = require('express'); |
|  | var mongodb = require('mongodb'); | |
|  | var app = express(); | |
|  |  | |
| var MONGODB\_URI='mongodb-uri'; |
|  | var db; |
|  | var coll; |
|  |  |
|  | // Initialize connection once |
|  |  |
|  | mongodb.MongoClient.connect(MONGODB\_URI, function(err, database) { |
|  | if(err) throw err; |
|  |  |
|  | db = database; |
|  | coll = db.collection('test'); |
|  |  |
|  | app.listen(3000); |
|  | console.log('Listening on port 3000'); |
|  | }); |
|  |  |
|  | // Reuse database/collection object |
|  |  |
|  | app.get('/', function(req, res) { |
|  | coll.find({}, function(err, docs) { |
|  | docs.each(function(err, doc) { |
|  | if(doc) { |
|  | res.write(JSON.stringify(doc) + "\n"); |
|  | } |
|  | else { |
|  | res.end(); |
|  | } |
|  | }); |
|  | }); |
|  | }); |

**Express**

* Object Data Model
* Object Relational Model

**React**

* works in flux pattern
  + action -> dispatcher -> store -> view
* “state”
  + Vault that holds info.
* Next.js
  + For work with Node
* ReactDom.render
  + To create object/elements

**Node**

* Template Literal
  + Back ticks allow you to create a string with objects
    - Ex: `this is an example of {$object: answer}`
* Get – reads the file
* Post – receives new note and adds to db.json
* Delete – receives id of a note to delete
  + Then rewrites db.json

**Other**

* Dotenv file
  + Add db name, mongoose user and password, then able to connect
* Database practices:
  + Use POST methods to upload data to db
* IndexDB
  + Js-based object oriented db
  + To store data within browser when offline in user browser.
    - Will store data when connect is persistent
    - 100% asynchronous
    - DOM events
    - Copy path in web browser – check console.log
  + Built in transactional db model
  + Must create an object to store data
* Class = library of functions / array of functions
  + Only works when called in API
* Read(requires 2 paramaters)
  + 1st = location, format text ‘utf8’
* Write(location, format)
* Use onchange/useref to capture inputs
* Service Workers
  + Virtual proxy between browser+network
  + How to properly cache assets of a website + make available when user is offline
  + Promise-based approach

Common Tips

* Set up server first
* Util is for promises
* Don’t move on before completing next > ensures synchronous js
* Look up body-parser , cors packages
* Controllers = where new info is added/updated/deleted