**Lead Em**

Before Release:

* Calendar styling
* Overall search page styles (include <section>?)
* Search criteria return proper leads
* Search Form component should be a form
* Make form actually use FormGroup
* Calendar doesn’t clear the date when clicking ‘x’
* Fix styling on label
* Change Application.html to index.html
* Make detail page able to save
* Integrate markup for Details page
* Integrate landing page
* Modal functionality appears broken - fix
* Remove /edit from path to edit a lead
* Allow creation of new users – new component
* Integrate Markup into Lead Form Component
* Integrate IsActivelySeeking checkbox and Boolean property
* Integrate Notes field and text property
* Allow user to create a brand new lead
* Rework header to have a logout button that isn’t there unless you’re logged in, and the name of the user
* Make Logout work in the header
* Try using Mongoose AutoIncrement for integer Ids instead of string
* Lead Detail page and component
* Integrate Mongo
* Write tests
* Implement Users
* Make app land on sign in page
* Implement sign-in and sign-up features
* Implement session checking and tokens
  + Finish implementing middleware that validates tokens
* Make Lead Detail a form
* Apply validation rules
* Factor all controls out of the form
* Debounce Search
* Make Search by UserId works (test by putting userIds in the leads at loading)
* Save UserId on Lead document
* Make a decision as to how Neighborhoods will be handled now that Search and detail use the same form
* Make sure versioning (\_\_v) works – if there’s time
* Implement Neighborhoods in Search and LeadModel
* Put Try-catch blocks in the relevant repo methods
* Modal for success in signing up
* Make search fire automatically for all fields in the lead form (so far all text fields work)
* Style Login button and Auth form
* Put route guards to avoid having unauthorized access to detail/search
* Something is wrong with the repo method for search. It returns as many copies of each lead as there are cities selected in the search criteria – probably happening in the SortByNeighborhood method. Fix line 70 to ensure the lead doesn’t get added more than once to the list if two cities match its criteria
* Add Email for User Auth and make sure it is unique
* Add wait spinners for CRUD calls
* Aot and rollup compilation (issues: aot broke because of ‘/Common/….” Having to be replaced with “Common/….” In every component file, and rollup was saying guid import didn’t work in BaseModel (replaced with a random js Math number generator))

Bugs:

* Phone Number Missing
* Modals not showing up on CRUD calls
* LeadModel.Phone.AreaCode error
* Aot or rollup breaks modals and some CSS
* CSS Display is wonky on smart phone:
  + Fields in Lead form are wider than the screen resolution, should be 100% of screen width
  + Green lines are wider than the display, should be 100% of screen width
  + Search Dictionary is shorter than phone width, should be 100%
  + The two buttons in the landing screen are too narrow, should be 100% of the phone screen width
  + Login screen is all messed up. Login form should be 100% of screen width

Deployment:

* Make Deployment build script:
  + Produces Dist and Aot folders in Client
  + Cleans all TypeScript files
  + Installs npm on both Client and Root
  + Creates production build folder with this structure:

leads.care

|

|-client

|-aot, dist folders

|-server

|-All js files in their regular folder structure

|-install

|-firstInstall.sh, Install.sh, uninstall.sh

|-node\_modules (root)

|

|-Main.js

* Make deploy script
  + Installs Node 6.10.3
  + Installs Mongo
  + Installs pm2

Milestones:

* Deploy to AWS or Digital Ocean:
  + SSH key for droplet
    - In command line:
      * ssh-keygen -t rsa
      * file to save in: /Users/Nicolas/.ssh/leadDroplet
      * cat /Users/Nicolas/.ssh/leadDroplet.pub to verify that the pub key is there, and to copy what this command returns to paste it into the SSH field requiring it. Copy the contents of that ssh file using the terminal command pbcopy < ~/.ssh/<ssh file name>.pub. (copy the whole thing, including the MacBook – Pro.local and ssh – rsa parts)
      * Paste this into the SSH key content field and in the name field, write in leadDropletSsh for the name
      * Ssh-add ~/.ssh/leadDroplet, give the mac password. – This is for Filezilla to log into the droplet. Because of this step, there is no need to add a keyfile in FileZilla
      * You can delete and create neww ssh keys on a droplet here: <https://cloud.digitalocean.com/settings/security?i=f09016>
  + Create the droplet (LeadDroplet)
  + Once it’s created, you have the ip address: 165.227.82.109
  + Password has been reset to qazplmQAZPLM135!#%
  + To SFTP into the instance, use FileZilla:
    - Host: 165.227.82.109
    - Port: 22
    - Use SFTP protocol
    - Logon Type: Normal
    - Password: Blank (because we ssh-add’ed the leadDroplet
  + To ssh into the droplet:
    - Simply go to the mac bash terminal and enter ssh [root@165.227.82.109](mailto:root@165.227.82.109)
  + Added new user: with ‘adduser nick’
    - Name: nick
    - Pw: iojknmIOJKNM90()
  + Gave the new user sudo priviledges:
    - Usermod –aG sudo nick (the ‘-aG’ stands for append group, because we are appending the sudo group to the nick user)
    - To test that this worked, you can enter id nick, and in the groups, you should see sudo (27) as one of the groups this user has
  + Steps to disable password login, and use ssh exclusively (for nick user only, not root)
    - Generate ssh keys on the server:
      * Mkdir ~/.ssh create the ssh folder
      * Chmod 700 ~/.ssh make it accessible only to our user
  + The app was moved to /home
  + Install pm2 globally: sudo npm install pm2 –g
  + To start the app using pm2: pm2 start Main.js “appname”

Future:

* Make checkbox label check/uncheck a box
* Make BaseComponent that includes modal functions, form functions
* Style modals with animation, refactor to use our own framework
* Make decent caching system
* Make LeadForm into a reliable pattern with associated API that is repeatable
* Datepicker and checkbox array should be its own component (neighborhoods)
* Use angular materials
* Fix modal problem with aot compilation
* Make Node/Yeoman-based app scaffolding CLI to automate app scaffolding and asset addition. In other words, you can add an asset the way you add a component in Angular-CLI, except adding an asset will make you the client-side a

TCR after first demo:

* Cache search parameters (works except Move-in date and neighborhoods)
* Clear search criteria button
* Make read-only text phone number field
* Make lead rows collapsable with content
* Add edit button in detail page
* Make content editable when edit mode is on

Issues often encountered after not touching the app for a while:

* If you’re getting an error when accessing the LeadsSearch page, where the content returned is { title: “error” }, the problem is in the server-side TokenValidate middleware. Make sure there are no errors in there like arguments not having a type and such.
* If GitHub desktop won’t include the Client folder, it is because Angular-CLI creates a .git and .gitignore resource on creation. Once you delete these two, you should see the insides of the Client folder show up in the GitHub desktop
* If you get the node side stopping at emitDestroyScript in async\_hooks.js, you’re likely trying to reach the server with a CRUD request with an expired token. Log out and restart the app

How to get the app running after a while of not running it:

* Run mongod
* In the top leads directory, build TypeScript (shift-cmd-B)
* In the Client folder, run ng build
* In the browser, go to <http://localhost:3000/#/>, this is the login window

How certain things work:

* App settings:
  + Server: in the Shared/Enums folder, for instance, you have an index.ts that export all the files in the Enums folder so that the AppSettingsController has only to require the folder to get every file within. It then sends that to the client wrapped in an object
  + Client: the app settings service Get is called by the AppSettingsGuard as soon as it is invoked by a router navigation that requires it via canActivate. The AppSettingsService.get() immediately sets all the app setting fields, which is a registry, so when the AppSettingsGuard calls it, it doesn’t return true until the get is done by .map’ing the get(). These fields aren’t persisted in memory when the browser is refreshed, so the AppSettingsGuard is invoked then too, since a refresh has to go through the route’s canActivate.

Docker:

* Download Docker at <https://docs.docker.com/install/#supported-platforms>
* In a terminal, run docker –v to make sure it works and get the version
* Mkdir mean-docker-template
* Angular client:
  + Cd into mean-docker-template and ng new angular-client
  + In the package.json scripts, change the start script from “ng serve” to “ng serve -H 0.0.0.0" or –host 0.0.0.0. The option –h sets the default host to serve from 0.0.0.0
  + Build the image: docker build -t <image\_tag>:<tag> <directory\_with\_Dockerfile>. The –t is shortform for –tag, a name given to the image to be built. Our command will look like “docker build -t angular-client:dev .” The <directory\_with\_Dockerfile> in this case is “.” meaning the current directory, since we’re running this command where the Dockerfile is located. This will build the image, which can take a few minutes depending on the internet connection. The image will expose the app through port 4200.
  + Next, we can run the container based on the image we just created using “docker run -d --name <container\_name> -p <host-port:exposed-port> <image-name>”. Our command will look like “docker run -d --name angular-client -p 4200:4200 angular-client:dev”. The –d flag means it will run in detached mode and get you back to your host without going into the container. So the container will be started in the background and then give you back control of the command prompt instead of going into the execution view inside the container. The “—name” flag attaches the name that will be assigned to this container so you can refer to it later. The <host-port:exposed-port> translates to <localhost:4200:dockerhost4200>, so the host-port is the port you will communicate with the image through, and the exposed port is the port the port you declared in the EXPOSE command in Dockerfile. The command itself will return something like “8045a76a96076029ad905f6c43a0538bbddc290a112acf96ae114263bb36239c” which is the container ID. At this point, you can go to your browser and view the client.
  + To stop the container, you can run “docker ***stop*** angular-client”. Once you do this and refresh the browser, you will see that your app is no longer served.
  + To restart the same container, you can run “docker ***restart*** angular-client” or “docker ***start*** angular-client”, and wait a moment (typically about 15 secs) before refreshing the browser to see it run.
  + Note: most of the Node.js images are built from Debian linux
* Node-Express:
  + Mkdir express-server at the top level of mean-docker-template
  + Give it this minimal package.json file:

{

"name": "express-server",

"version": "0.0.0",

"private": true,

"scripts": {

"start": "node server.js"

},

"dependencies": {

"body-parser": "~1.15.2",

"express": "~4.14.0"

}

}

* + Make a minimal server.js file like this:

// Get dependencies

const express = require('express');

const path = require('path');

const http = require('http');

const bodyParser = require('body-parser'); // Get our API routes

const api = require('./routes/api');

const app = express(); // Parsers for POST data app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: false })); // Set our api routes

app.use('/', api); /\*\* \* Get port from environment and store in Express. \*/

const port = process.env.PORT || '3000';

app.set('port', port); /\*\* \* Create HTTP server. \*/

const server = http.createServer(app); /\*\* \* Listen on provided port, on all network interfaces. \*/

server.listen(port, () => console.log(`API running on localhost:${port}`));

* + And an routes/api.js file in the Routes folder at top level:

const express = require('express'); const router = express.Router(); /\* GET api listing. \*/

router.get('/', (req, res) => {

res.send('api works'); });

module.exports = router;

* + Then you can npm install, and npm start (since the start script points to “node server.js”). You should see the api served on localhost:3000
  + Just like before, you run the build and run commands:
    - docker build –t express-server .
    - docker run -d --name express-server -p 3000:3000 express-server:dev
  + after those two, you should see your app come alive again in the browser if you had stopped it previously
* Mongo:
  + We can’t have a Dockerfile to run Mongo, but one already exists on DockerHub. We only need to know how to run it. Assuming we already had a MongoDB image to run locally, we could simply run it using “docker run -d --name mongodb -p 27017:27017 mongo”. So here, the container name is mongodb, the image name mongo, and it runs on port 27017. Docker will first check to see if you have a Mongo image already downloaded or built. If you don’t, it will look for it on DockerHub. So running that command will actually start a mongo image served on 27017. When you do, you will get a line that says “can’t find mongo locally”, then proceed to download it. When you go to the browser to localhost:27017, you will see a message that says “It looks like you are trying to access MongoDB over HTTP on the native driver port.”. That means it’s working! So you now have a Mongo image running from a Docker container named mongo. Alternatively, you should also be able to go to the terminal and run sudo ./mongo without having to run mongod first, since it is already running, and not get an error. You could uninstall mongo altogether, and simply run mongo from a Docker container locally, since you basically don’t need a local installation anymore.
* Connect it all:
  + Docker comes with docker-compose. You can check that you have it installed by running docker-compose –v. At the project level (same as angular-client and express-server), make a new file called ‘docker-compose.yml’. That file will simply tell Docker which containers to build (note: yaml (Yaml Ain’t Markup Language) is another way to serialize key-value pairs like json does. It uses whitespace to denote execution hierarchy the way Python does). Essentially, this file is like running a bunch of docker run commands end-to-end.
  + Run the command “docker-compose up -d”. The ‘-d’ is optional for detached; remove it if you want to see specific process and error messages, or leave and see the logs with “docker-compose logs”. That will build all the containers if they aren’t already, then run them simultaneously. You should be able to see the apps served on localhost 3000 and 4200, but not mongo on 27017. For some reason, the mongo app doesn’t show up on browsers when using docker-compose up, but it still tests successfully when running “sudo ./mongo”.
  + To stop the whole docker-compose set, use “docker-compose stop”, and to remove all docker-compose containers, “docker-compose rm”
  + Everytime you make changes to your app
  + In the yaml file, you have to add another field that links to the name of the service:

links:

- mongodb # link this service to the database service

* + If you run the app now and try to make a CRUD call like posting a new item to the database, you will get a CORS error since there Angular client and Express apps are running in different containers
  + To get the debugger attaching to the node process inside vs-code, you have to follow these steps:
    - In the debug dropdown, add configuration
    - Select Docker: attach to Node
    - In launch.json, make sure that the localRoot points at the top level of the Express/Node project like this: "localRoot": "${workspaceFolder}/express-server/"
    - In vs-code, cmd-shift-p
    - Select compose up, and select the docker-compose.debug.yml file to debug or docker-compose.yml in production
    - Once the process is started, you can attach Docker to Node in the debug dropdown
    - This video was helpful: <https://www.youtube.com/watch?v=WDqnaZFE7Oc>