

Tssg Database Application

When I wrote the first version of this document, I made a few assumptions that I should not have and depended partially upon a previous document that I had sent out. In this revision I plan to eliminate any assumptions I made, incorporate some of the previous document, rearrange this document into a more logical implementation order and include a couple of new items. Hopefully this will allow anyone to successfully implement this application.

These instructions are for a Windows environment because that is what I developed it in. There is no reason that it cannot be installed in a Linux environment. I assume that anyone doing that I would be knowledgeable enough to know what changes would be necessary to accomplish that.

I highly recommend that before over complicating this installation, that you run through this installation and get everything running in the simplest environment you can create. By then, you will understand more about what happens where and when and know where to look in isolating any problems.

This project is in three parts, the Mongo database, the backend code tssgDbApi, and the frontend code tssgDbApp. I will cover them in that order.

The first assumption I made is that you would already have npm/node installed. As we discovered yesterday in our Monday meeting that was incorrect. Here is one way to install npm/node. You will install node which includes npm. The following link provides a full set of instructions for installing node.

<https://blog.teamtreehouse.com/install-node-js-npm-windows>

During installation, make sure the path option is selected. This will ensure that the system environment path is appropriately updated.

As the documentation instructs, in a command window type 'node -v ' and then 'npm -v ' to display the installed versions of node and npm.

If you don't use an msi file to install node and npm, include the -g option to ensure they are installed globally. Also, you may have to update the system environment variable manually.

Next, install the Angular CLI:

In a command prompt, type:

```
npm install -g @angular/cli
```

I recommend using the -g option because twice now I have seen by not using it the system environment path variable does not get set up correctly and the angular commands (ng ect.) fail (as in cannot be found).

Next, either download or clone the two project folders:

<https://github.com/pscheid1/tssgDbApi>

<https://github.com/pscheid1/tssgDbApp>

You can place the two project folders anywhere you desire.

MongoDB

Next, install MongoDB and setup the database:

Use one of the below links (msi or zip) to download and install MongoDB:

https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2012plus-4.2.0-signed.msi

https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2012plus-4.2.0.zip

<https://www.mongodb.com/download-center/community> This link will take you to the MongoDB Download Center and allow you to make other choices.

I'm not sure MongoDB sets up the system path variable. You might want to check and add it yourself. It's not required but it makes life easier. Depending on where MongoDB is installed it would look something like this: C:\Program Files\MongoDB\Server\4.0\bin. Also, change the version to match what you have installed.

In Windows, run the services app and check to make sure MongoDB is installed and running.

If you have an existing tssg-tech database and want to delete it, use the mongo shell and the drop database command.

To drop an existing tssg-tech database:

The following commands depend on the system path variable setup for MongoDB. If not, you will have to preface mongo with the proper path.

Open a command window.

Type: mongo (runs the mongo shell)

Type: show dbs (should display tssg-tech along with other databases, proves mongo service is running)

Type: use tssg-tech (connects the shell with tssg-tech)

Type: db.dropDatabase(); (case is imperative)

Type: show dbs (tssg-tech should not be listed)

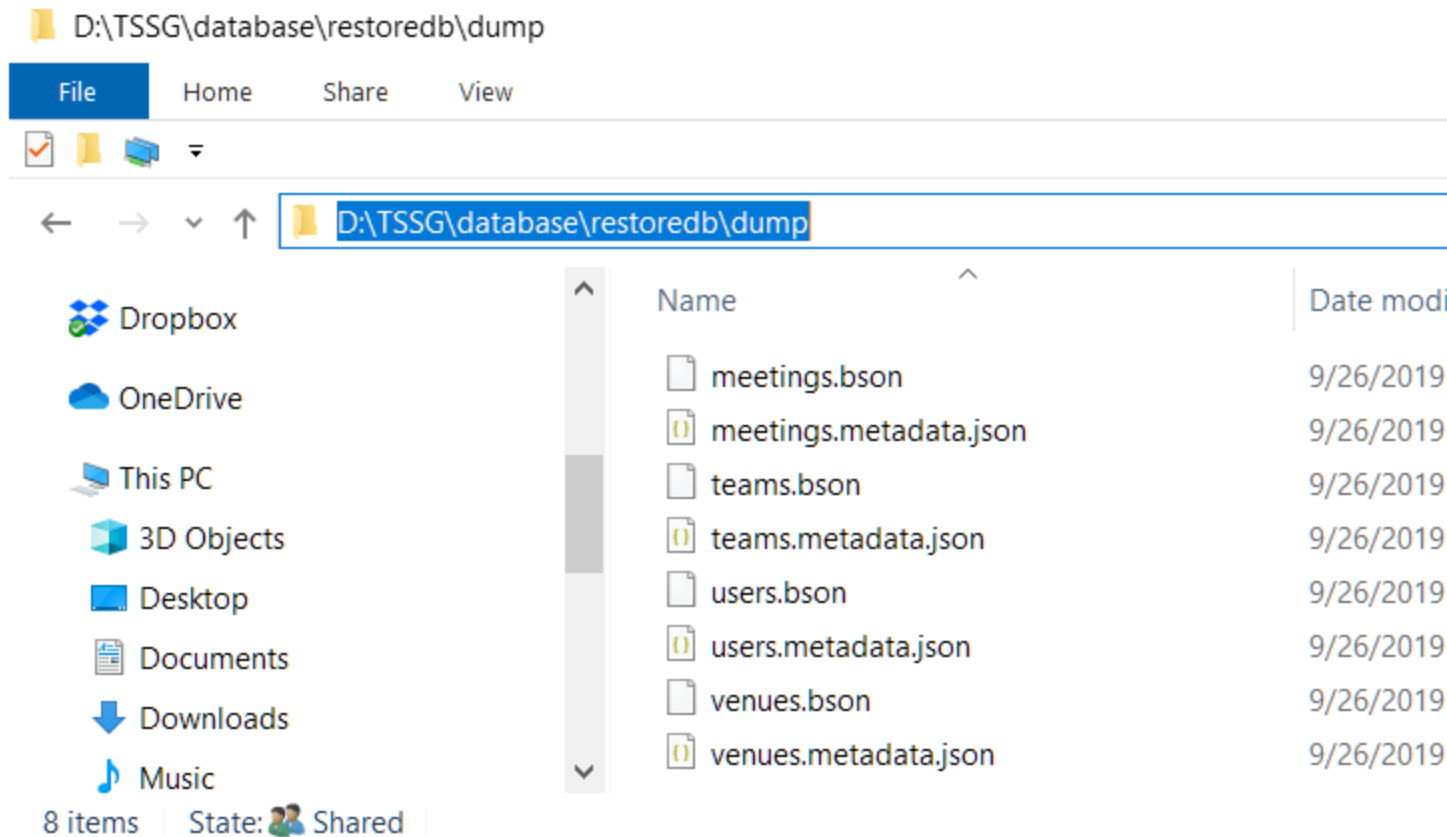
To initialize a startup database, I'm attaching a small dump file. To use this file, choose any folder you want and create a sub folder named dump.

Extract the files from tssg-tech.zip into dump.

Cd to the parent folder of dump.

In the example below my folder is d:\tssg\database\restoredb with a dump child folder.

Example:



Make sure your MongoDB service is running. If you are using the default port (27017) you don't need to do anything else. If you have changed ports, url, and other items, you will need to check the mongorestore documentation on how to declare parameters for the changes. Use the following link:

<https://docs.mongodb.com/manual/reference/program/mongorestore/#cmdoption-mongorestore-db>

In a command window, cd into tssg/database/restoredb. This folder should be empty, but it doesn't matter if it contains other files. If you have set up the system environment path variable, execute mongorestore --db tssg-tech. Mongorestore will automatically look for the dump folder and install the files. If there is no tssg-tech database, it will create one. If there is a tssg-tech database, it will add any new items.

Here is the output from my mongorestore command example:

```
PS D:\tssg\database\restoredb> mongorestore --db tssg-tech
PS D:\tssg\database\restoredb> mongorestore --db tssg-tech
2019-09-24T18:27:09.838-0400 the --db and --collection args should only be used when restoring from a BSON file. Other uses are
2019-09-24T18:27:09.839-0400 deprecated and will not exist in the future; use --nsInclude instead
2019-09-24T18:27:09.839-0400 using default 'dump' directory
2019-09-24T18:27:09.841-0400 building a list of collections to restore from dump dir
2019-09-24T18:27:09.843-0400 reading metadata for tssg-tech.venues from dump\venues.metadata.json
2019-09-24T18:27:09.844-0400 reading metadata for tssg-tech.users from dump\users.metadata.json
2019-09-24T18:27:09.845-0400 reading metadata for tssg-tech.meetings from dump\meetings.metadata.json
2019-09-24T18:27:09.928-0400 restoring tssg-tech.venues from dump\venues.bson
2019-09-24T18:27:09.932-0400 no indexes to restore
2019-09-24T18:27:09.932-0400 finished restoring tssg-tech.venues (8 documents)
2019-09-24T18:27:11.925-0400 restoring tssg-tech.users from dump\users.bson
2019-09-24T18:27:12.002-0400 restoring tssg-tech.meetings from dump\meetings.bson
2019-09-24T18:27:12.009-0400 restoring indexes for collection tssg-tech.users from metadata
```

```

2019-09-24T18:27:12.011-0400 restoring indexes for collection tssg-tech.meetings from metadata
2019-09-24T18:27:12.098-0400 finished restoring tssg-tech.users (3 documents)
2019-09-24T18:27:12.100-0400 finished restoring tssg-tech.meetings (1 document)
2019-09-24T18:27:12.100-0400 done
PS D:\tssg\database\restoredb>

```

You can use the following commands to verify the database:

```

PS D:\tssg\database\restoredb> mongo
MongoDB shell version v4.0.5
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongod
Implicit session: session { "id" : UUID("e47e1741-c62c-4ef8-9aba-e00ab86becda") }
MongoDB server version: 4.0.5
Server has startup warnings:
//
Output removed to conserve space
//
---

> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
tssg-tech  0.000GB
> use tssg-tech
switched to db tssg-tech
> show collections
meetings
users
venues
> db.users.find().pretty();
{
  "_id" : ObjectId("5d449d578b7d7853fcc08652"),
  "inactive" : false,
  "username" : "admin",
  "firstname" : "Super",
  "lastname" : "Admin",
  "role" : "Admin",
  "email" : "noEmail@none.org",
  "mobile" : "9998887777",
  "createdAt" : ISODate("2019-08-02T20:29:02.772Z"),
  "hash" : "$2a$10$nPnV9xDvXYI7G/U6Blda8eVylGIVR/lbx/S5RVBI3zgkPizoxuuDG",
  "_v" : 0
}
{
  "_id" : ObjectId("5d449dac8b7d7853fcc08654"),
  "inactive" : false,
  "username" : "user",
  "firstname" : "Plane",
  "lastname" : "User",
//
Output removed to conserve space
//
}
{
  "_id" : ObjectId("5d449e288b7d7853fcc08656"),
  "inactive" : false,
  "username" : "contact",
  "firstname" : "Lonely",
  "lastname" : "Contact",
//
Output removed to conserve space
//
}
>

```

If you received results like the above then your database is up and running.

Securing the Database

When you secure the database does not matter. It will not change this effort whether you do it now or later. There are a few methods and options available for securing a Mongo database. The following notes explain one of the simpler ways to accomplish securing the database while at the same time providing a reasonable amount of security. Consult the MongoDB documentation for a full explanation on accomplishing this. In this document I'm only going to cover the minimum requirements.

When the Mongo database software is installed, three databases are automatically created, admin, config, and local. Above, we created a new database, tssg-tech. At this point there is no security as to who can do what. Anyone with a little knowledge can affect any database. What we will do here is enable authentication so that only specific users with specific roles will be able to administer specific databases. By default, when we enable the security features, all databases will be protected. Although any database could be setup, authentication is normally implemented by using the admin database. We are going to add one or more users to the admin database and give them roles to limit or allow operations on some or all other databases. Once we do this we will enable authentication in the mongod.cfg file. This will require anyone attempting to connect to a mongo database to supply a valid username and password that will be verified against an account in the admin database.

Run the mongo shell

Mongo

Switch to the admin database

use admin

Create a new user for global mongo administration

Multiple accounts may be created.

Change the username and password and cut and paste (or type) for each account

Connecting with this username and password will allow administration of any mongo database.

```
db.createUser(  
  {  
    user: "mongoDbAdmin",  
    pwd: "mongoDbAdminPW",  
    roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAnyDatabase" ]  
  }  
);
```

Create one or more administrator accounts for the tssg-tech database.

Multiple accounts may be created.

Change the username and password and cut and paste (or type) for each account.

These accounts will be limited to the tssg-tech database.

```
db.createUser({  
  user: "tssgAdmin",  
  pwd: "tssgAdminPW",  
  roles: [  
    { role: "userAdmin", db: "tssg-tech" },  
    { role: "dbAdmin", db: "tssg-tech" },  
    { role: "readWrite", db: "tssg-tech" }  
  ]  
});
```

Make the following changes to the mongod.cfg file:

security:

authorization: enabled

setParameter:

enableLocalhostAuthBypass: false

Restart MongoDB to enable authorization.

To disable authorization, set authorization: disable and restart MongoDB

Changes to our mongo connection string.

This command is in the tssgDbApi server.js file and possibly in your system environment variable.

Most mongo applications will support the -p (no password) option and will request a password. Obviously for command line entries.

Insert username:password@ between //<host>

Append the query string ?authSource=admin (this informs mongo which database to use for authentication)

mongodb://tssgAdmin:tssgAdminPW@localhost:27017/tssg-tech?authSource=admin

Getting into the code:

At this point, you should have npm and node installed, both project folders installed (tssgDbApi and tssgDbApp) and a MongoDB database. The following instructions will cover building, testing and use of the tssg database application.

System environment variables:

The code does not depend on any system environment variables. It will use default settings (indicated below) or you can supply command line parameters. Setting up system environment variables makes life easier. If present, the following system environment variables will be used by various modules. The defaults that are declared are built into the code and are enabled when a variable is empty or has not been declared.

- tssgApiDefaultTeam Team name for Wednesday Group Meeting (default: WedGenMtg)
- tssgApiMtgDebug Enable/disable additional debug output in tssgDbApi/controllers/meeting.controller.js
- tssgApiPort port number for tssgDbApi node server (default: 7010)
- tssgApiURL web address for the backend api (default: localhost)
- tssgAppPort currently not used (default: 4200)
- tssgAppURL currently not used (default: localhost)
- tssgMongoDB_URL mongodb:<host and port>/<database name> (default: mongodb:localhost:27017/tssg-tech)

These are my settings:

- tssgApiDefaultTeam WedGenMtg
- tssgApiMtgDebug false
- tssgApiPort 7010
- tssgApiURL http://192.168.1.2
- tssgAppPort 4200
- tssgAppURL http://localhost
- tssgMongoDB_URL mongodb://localhost:27017/tssg-tech

Command line variables:

tssgDbApi

This is the backend code. It is developed in express/javascript and executes on a node server. The major parts of this project are in the routes, models and controllers' folders. Each of these folders contains four files. One each for users, teams, meetings and venues.

A model defines the scheme for a mongo collection. A mongo collection is like a SQL table. Each model declares the necessary items to create a collection for the like named model in tssgDbApp. Therefore, our mongo database contains a minimum of four collections. One each for users, teams, meetings and venues. Notice the

collection names are plural. This is not required but is standard convention as collections usually contain multiple entries.

A routing file defines an AJAX/REST mapping of input data to controller functions. As mentioned above, there is a router file for each collection. One each for users, teams, meetings, and venues. If you look in the server.js file you will find four statements like "app.use('/meetings', meeting)". This will map a REST command say GET to a routing file. For example, GET/meetings/<data> would search the meeting.route.js file and look for a match. The mapping in the route file omits the leading name (in this example meetings). When a match is found, then if specified, authorize is called to verify the logged-on user has the required role. If the logged-on user has authorization, the declared controller function is called. For public routes there is no authorize parameter declared in the route.

A controller file implements the CRUD (and other) functions for a specific collection. Each function takes a request (front-end or external app) and formats a call to the database. It then returns a result back to the caller which may contain data, an error, or status code.

The server.js file contains the setup code and is what gets executed. Execution is covered below.

The _helpers folder contains three files:

- Authorize.js gets called during route processing. It basically compares the user.role stored in the JWT token at logon with an array of roles passed in with the call. If the user's assigned role matches a role in the passed-in array, then user is authorized, otherwise authorization fails.
- Error-handler.js is a middleware global error handler.
- Role.js is an enumeration and defines the roles for authorization.
 1. Admin: All CRUD rights (create, read, update, delete)
 2. User: Read only. Limited to their own accounts.
 3. Contact: No CRUD rights. Cannot logon. Used for contact data only.

How to build/run tssgDbApi

Open a command window and cd to the project folder (tssgDbApi)

Normally, npm modules are not stored in git repositories and this is also true for tssgDbApi. When you download or clone tssgDbApi it will be missing all the npm modules required to successfully build it.

If this is a new clone or download, type:

npm install (this will use the package.json file and install the required packages locally)

You only need to do this after starting from a new clone or download of the project.

The following commands are created as scripts and are set up in the scripts section of the package.json file in the project folder.

For development, type:

npm run dev

This command will run server.js with nodemon. Nodemon monitors project files and if a change is made to any project file, the node server will be stopped and restarted. To stop a running session, enter Ctrl-C.

Example:

```
PS D:\TSSG\tssgDbApi> npm run dev
```

```
> tssgdbapi@1.0.0 dev D:\TSSG\tssgDbApi
```

```
> nodemon server.js --scripts-prepend-node-path
[nodemon] starting `node server.js --scripts-prepend-node-path`
mongoDB: mongodb://localhost:27017/tssg-tech
Server is up and running on port number 7010
mongodb://localhost:27017/tssg-tech - we're connected!
```

For production, type:

```
npm run prod
```

This command will run server.js with node. Node will not monitor file status. You will need to stop and start the server manually .

Command line variables:

Unfortunately, npm run does not support command line arguments. To modify the run environment, you must use environment variables (defined above) or replace the npm run command with native node run commands or modify the command definitions in package.json. Native node run commands are no more complicated than the npm run commands. There are many ways, methods, parameters etc. to run a node server. For our purposes we simply need to invoke node or nodemon and a server module.

Simply cd to the project folder and start a server:

```
Cd .../tssgDbApi
```

```
node server.js or nodemon server.js will create a tssgDbApi server instance.
```

This will create a tssgDbApi server identical to npm run dev or npm run prod and can be modified by environment variables. Environment variables are usually used for more permanent modifications to the run time environment. Command line arguments are normally preferred for testing and temporary changes. The Currently, tssgDbApi supports the following command line variables:

-p -port	port number
-h -host	ip hostname localhost
-mtgDbg	true (anything else is false)

tssgDbApp

This document includes a few changes to the project since its initial release. The biggest change is the introduction of the team's collection. What does this mean in terms of the database that is currently being used? The reality is "nothing". This is one area where mongo is very accommodating. Everything in the existing database will work just fine and any new objects will be added as if the database knew of them from day one. There are however two areas to be aware of. The first is that some existing items will need to be updated in order to populate new fields. For example, an existing meeting will need to be edited to add a team name to the meeting. All meetings are now team specific. The second thing is that if you restore the database from a backup made prior to new data being entered, you will lose the new data. This is no different than losing changes made to any data after a database backup.

tssgDbApp is the front-end code. It executes on the client browser. It was develop using Angular 8.

(<https://angular.io/docs>)

The meat of the project is in the tssgDbApp/src/app folder.

The 4 top folders are prefaced with an underscore and are named `_guards`, `_helpers`, `_models`, and `_services`.

There is no significance to the underscore character. It just visually separates these folders from the component folders. I will cover these later.

Next are the component folders. These are the primary Angular structures. The 'normal' default for components is a parent folder with the component name containing four files named:

```
<component name>.component.css  
<component name>.component.html  
<component name>.component.spec.ts  
<component name>.component.ts
```

Each component is 'usually' designed for a specific 'CRUD' function so you will have name-add (create), name-edit (update) and name-get (read/list and delete). As far as angular is concerned, names have no implicit meaning. Meeting-add could also be xyzq-who-cares. However, we attempt to use meaningful names as to what a component is to accomplish. Therefore, looking at the code you will find components for Meeting (add, edit, get, get-schedule), team (add, edit, get) user (create, edit, get-all, get-curr, home, login, logout) and venue (add, edit, get). Meeting, team and venue are standard. User is standard with additional functions. Get-all will list all users. Get-curr will list the current logged on user. Home is the home landing page. Login provides the logon form and logoff performs a user logoff.

The navbar component creates the navbar used by all the other components. It gets injected into `app.component.html`.

At the bottom of the app folder is the app component. `App.component` is the root page. All other components are rendered through this page.

Also, the `app-routing.module.ts` file is here. This is where the Angular routing is set up. It looks complicated but once understood it enables easy page to page routing.

Finally, we have the `app.modules.ts` file. An Angular module class describes how the application parts fit together. Every application has at least one Angular module, the *root* module that you bootstrap to launch the application. You can call it anything you want. The conventional name is `AppModule`.

(<https://v2.angular.io/docs/ts/latest/guide/appmodule.html>)

`_guards` folder contains the `auth.guard.ts` file. This implements a `JwtHelperService` that verifies the logon state during transaction processing. The JWT token is verified and route restrictions based on the logged-on users' role is enforced.

`_helpers` folder contains `error.interceptor.ts`, `JWT.interceptor.ts` and `tssg.ErrorHandler.ts`. These files provide various middleware error handlers and the `JWT.inteceptor` inserts the JWT token into transaction headers.

`_models` folder contains the model for each of the four major components. These map to the schemes declared in the backend code (`tssgDbApi/models/*`).

`_services` folder contains services that that connect `tssgDbApp` to `tssgDbApi`. Each service makes AJAX calls to the like named backend services.

Back out one level to the `tssgDbApp/src` folder:

In the `assets` folder is the `tssg_logo.png` file used in the navbar.

In the environments folder there are two very important files. `Environment.prod.ts` and `Environment.ts`. During production builds, angular will replace the `environment.ts` file with the `environment.prod.ts` file. If you look in the `angular.json` file under `configurations/production/fileReplacements` you will find the declaration for this. This is how we get system environment variables to the angular code. Remember, the frontend code runs on the client and does not have access to system environment variables. In the non-production file (`environment.ts`) we have constants set up for the variables. During development, you must change this if it is necessary. The `typings.d.ts` file sets up an interface for these variables.

`Index.html` is the root page. Links and scripts are declared here, and the angular rendered pages will be stuffed into `<body><app-root></app-root></body>`.

At the very top level is the `dist` (distribution) folder. It is immediately under the `tssgDbApp` folder. When you do a build for production, the minimized code bundle will be placed in this folder in a sub-folder named `tssgDbApp`. You can point to this folder or move it to any location you may wish. Then configure your web server (apache, nginx, ...) to that folder. You can also use the `npm server` if you desire.

How to build tssgDbApp

Open a command window and `cd` to `tssgDbApp` (the project folder not the `tssgDbApp/dist/tssgDbApp` folder)

Normally, `npm` modules are not stored in git repositories and this is also true for `tssgDbApp`. When you download or clone `tssgDbApp` it will be missing all the `npm` modules required to successfully build it.

If this is a new clone or download, type:

```
npm install
```

 (this will use the `package.json` file and install the required packages locally)

You only need to do this after starting from a new clone or download of the project.

The following commands are created as scripts and are set up in the `scripts` section of the `package.json` file in the project folder.

This is a very abbreviated list of some angular cli commands. See <https://angular.io/cli> for a full list and explanation of the angular cli commands. The few listed below are the ones I regularly use.

Development: `ng [s | serve] [-o | --open]` (the `|` character is an 'or', not a pipe)

This will build a development version of the project (uses the `environment.ts` file) and start a `npm` server. Each time changes are made and successfully compiled, the server will be automatically restarted. If the `open` option is supplied, each time the server is started a copy of the default browser will be launched with the correct URL and port. A new browser will only be launched if one is currently not in use.

Production: `ng [b | build] [--prod]`

This will build a production version of the project (uses the `environment-prod.ts` file) and copy it to the `<project> dist` folder. In our case, that would be `tssgDbApp/dist/tssgDbApp`.

As I previously mentioned, this is a very abbreviated list of Angular build options. Angular has a very extensive list of build and deploy options. Please consult the following document for a full explanation of these options.
<https://angular.io/guide/deployment>

Angular generates testing structures for both unit testing and end-to-end (e2e) testing. I did not use this as I believed it would take me just as long to understand and use it as it did for me to learn and understand angular development. I know this seems backwards but that is the way I did it. Now that the project is in fairly good shape, I might take some time and investigate using these features. For those of you interested in pursuing this on your own, this might be a good place to start. <https://angular.io/guide/testing>

Finally, a few instructions on using the tssg database application:

These instructions assume that you are working with a newly installed system.

Ensure the mongo db server is running.

In a command window, navigate to the backend project folder – tssgDbApi

Start the tssgDbApi service: run npm dev | run npm prod | node server.js

Note: run npm [dev | prod] runs the like named script declared in package.json.

Looking at the scripts in package.json you will find that the dev script runs server.js with nodemon and prod runs server.js with node.

Node server.js executes server.js (same as npm run prod)

If you use the npm run dev command, you should see output like the following:

```
PS D:\TSSG\tssgDbApi> npm run dev
> tssgdbapi@1.0.0 dev D:\TSSG\tssgDbApi
> nodemon server.js

[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
Server is up and running on 192.168.1.11:7010
we're connected!
```

A simple test to see if the backend code is up and running is:

Enter through browser:

[http://\(ip or localhost\):7010/venues/test](http://(ip or localhost):7010/venues/test) - This should return:

```
globalRoot: D:\TSSG\tssgDbApi - folders: D:\TSSG\tssgDbApi - packageName: tssgDbApi - __dirname: D:\TSSG\tssgDbApi\controllers
```

Values will be different. The above command is using a venue route that is not restricted and proves backend code is up and running. It does not connect to the database and therefore does not prove the database path is functional.

To prove the backend/database path is operating, enter:

[http://\(ip or localhost\):7010/meetings/webSchedule/WedGenMtg](http://(ip or localhost):7010/meetings/webSchedule/WedGenMtg) - This should return 0 – 3 meetings for the WedGenMtg team, depending on the current date and time and the meetings existing in the current database. Also, the database must contain a team with the name of WedGenMtg. If you have different teams' setup, use one of existing team names. Case is significant.

If no meetings qualify, you will receive an empty array: []

If one, two or three meetings exist, the array will contain the meeting data: [...]

If you are using angular in dev mode:

Navigate to the project folder tssgDbApp

Start a node server: `ng [s | serve] [-o | --open]` This runs the server in a manner like the nodemon method. A file change/save will generate a compile and a successful compile will restart the server.

If you didn't use the open option or are running a production build, browse to localhost:4200 (or whatever ip and port you have setup)

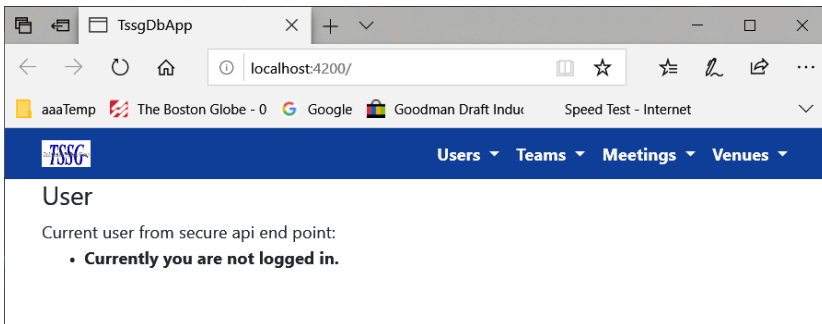
If you use the “`ng serve -o`” command, you should see output like the following:

```
PS D:\TSSG\tssgDbApp> ng serve -o
10% building 3/3 modules 0 active [wds]: Project is running at http://localhost:4200/webpack-dev-server/
i [wds]: webpack output is served from /
i [wds]: 404s will fallback to //index.html

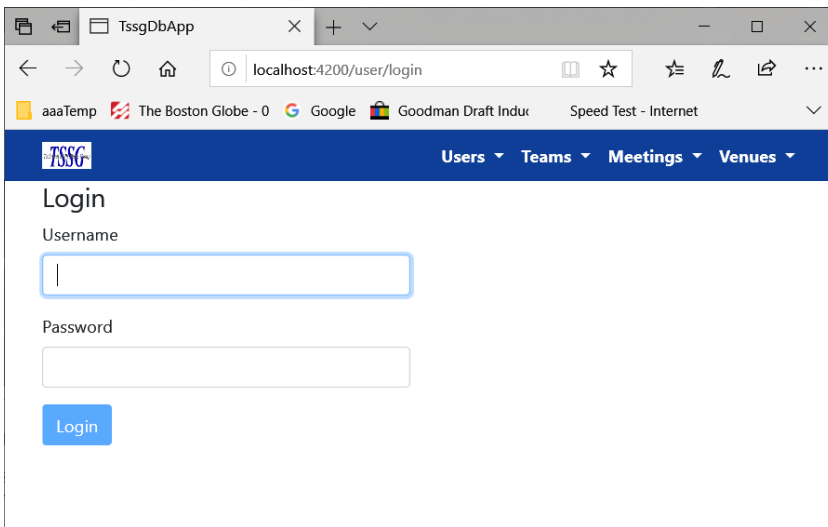
chunk {main} main.js, main.js.map (main) 225 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 251 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.09 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 83.6 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 5.41 MB [initial] [rendered]
Date: 2019-10-24T16:49:39.965Z - Hash: e0d06cf376b574ca7d7d - Time: 12390ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/
**

i [wdm]: Compiled successfully.
```

The home page with a navbar and a few lines stating you are not logged in should display in the browser window.



Select users/login from the navbar.



If you are using the raw database supplied with this document, login with the super admin account. If other accounts have been added, login with an admin account. In startup database supplied with this document, there are two accounts that you can logon with.

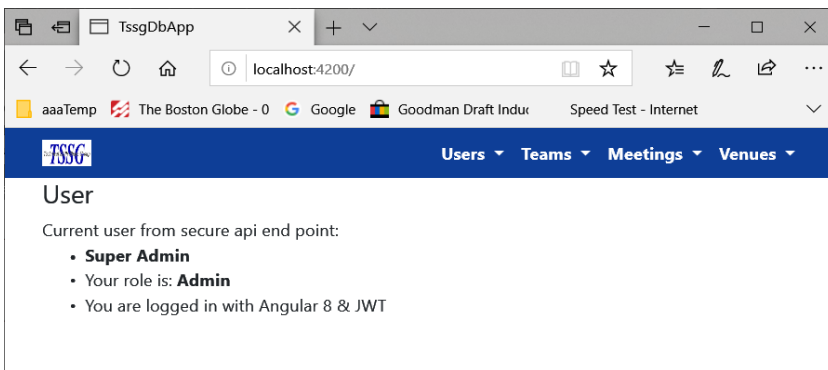
Their username and passwords are:

admin/adminpw

user/userpw

The admin account has role admin and user account has role user.

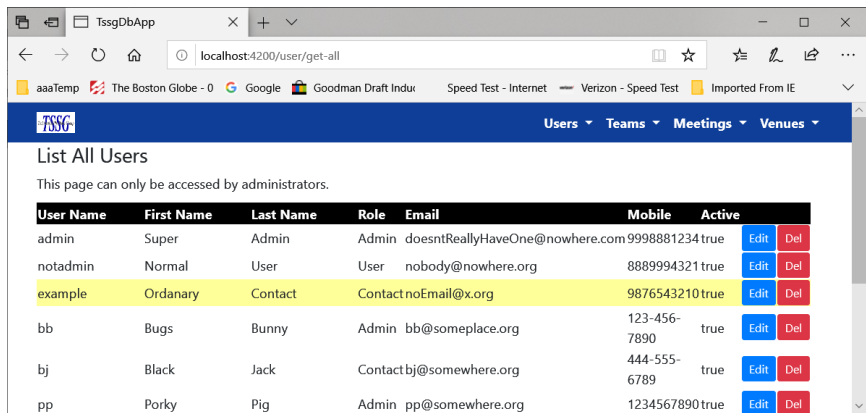
If successful, the user home page will display account information.



When a logon is successful, a Java Web Token (JWT) is created and stored in web page headers. The JWT encodes (not encrypts) the username and role. This information is used to authenticate and authorize requests. Because the JWT is not encrypted it is not secure. Using an encryption protocol (TLS/https) provides the necessary security. Currently, the JWT is valid for 2 hours. If you are logged on greater than 2 hours, you will be notified that your JWT has expired when you attempt a request. When this occurs, you are automatically logged off. Simply logon and a new JWT will be created and be valid for 2 hours.

We can have a conversation about the JWT validity length of 2 hours. It's just a value I chose that seemed to be reasonable. There are also methods to consider as to how to change this value as it currently requires a coding change. It wouldn't take much to convert it to a system variable. Something to consider about the JWT validity length is that the browser maintains the JWT for the remainder of the two-hour period until you log off or close the browser. This is browser dependent. If you are logged on and kill the browser, Chrome and Firefox will maintain the JWT and Edge will clear it. This means that if you are logged on with Chrome or Firefox and kill the browser, anyone using your machine can restart the browser and be logged on to tssg-tech. Convenient for developers, bad for security.

From the user's dropdown you can add a user, list all users, list current user, login and logout. From the list all users, you can edit accounts or delete them. List current user lists the user that is currently logged on. This user can edit their account but not change their role. Also, they cannot delete their account. If a logged-on user does not have the role admin, the only options available are logout or list current. Again, these are choices I made, and we can certainly make changes.



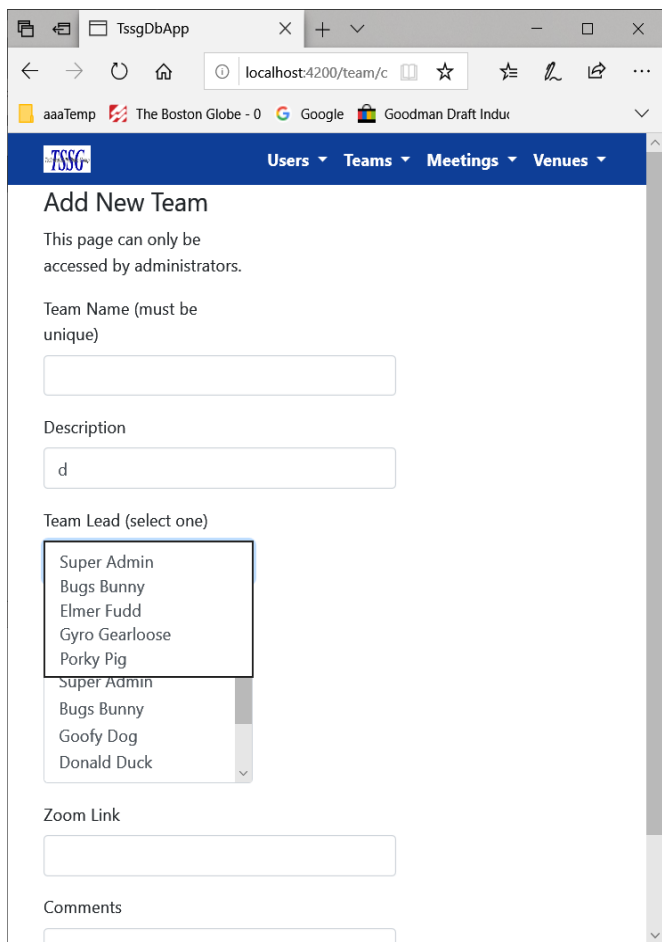
User Name	First Name	Last Name	Role	Email	Mobile	Active
admin	Super	Admin	Admin	doesntReallyHaveOne@nowhere.com	9998881234	true
notadmin	Normal	User	User	nobody@nowhere.org	8889994321	true
example	Ordinary	Contact	Contact	noEmail@x.org	9876543210	true
bb	Bugs	Bunny	Admin	bb@someplace.org	123-456-7890	true
bj	Black	Jack	Contact	bj@somewhere.org	444-555-6789	true
pp	Porky	Pig	Admin	pp@somewhere.org	1234567890	true

Because this is an initialized system, spend some time familiarizing yourself with the navbar.

Add some users. They can be test or real. It would help to have at least two each with role admin, user, and contact. It is important to add users first as teams and venues require user entries.

The team lead selection is limited to users with role admin. The team members selection will display all users with role admin or user but not contact. Choices are limited to what is in the database which is why it helps to add a few users for each role.

Create one or more teams.



Add New Team

This page can only be accessed by administrators.

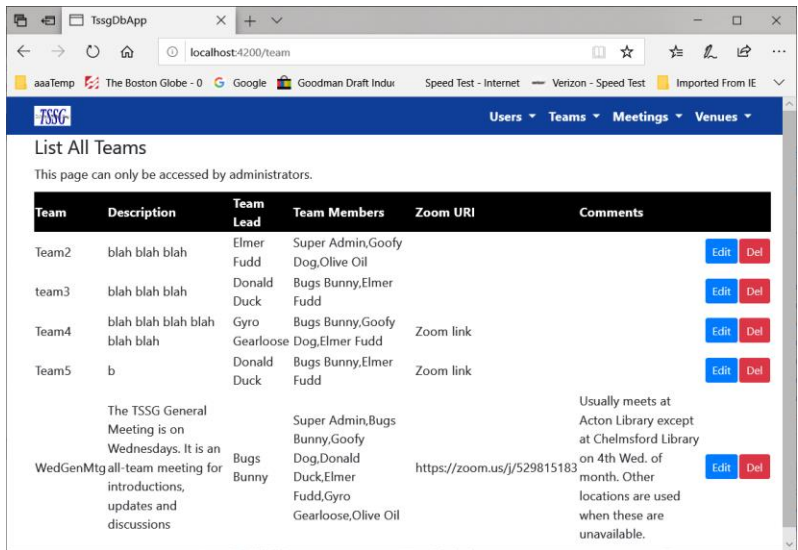
Team Name (must be unique)

Description

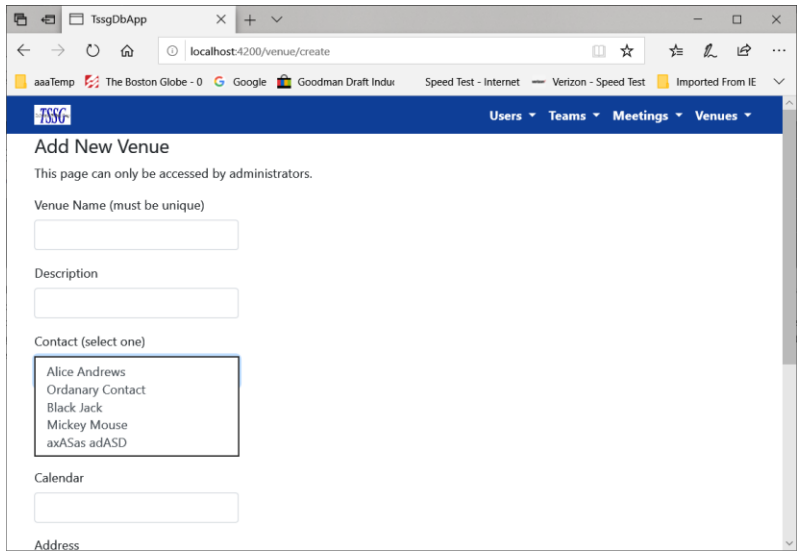
Team Lead (select one)

Zoom Link

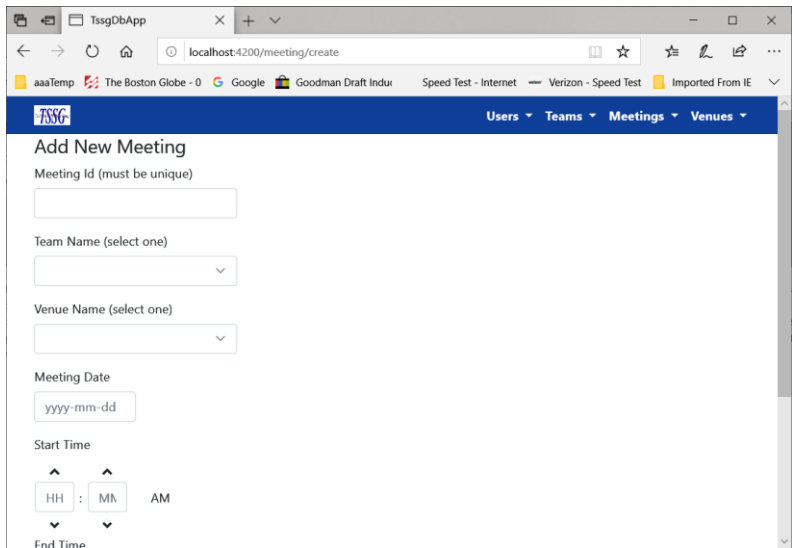
Comments



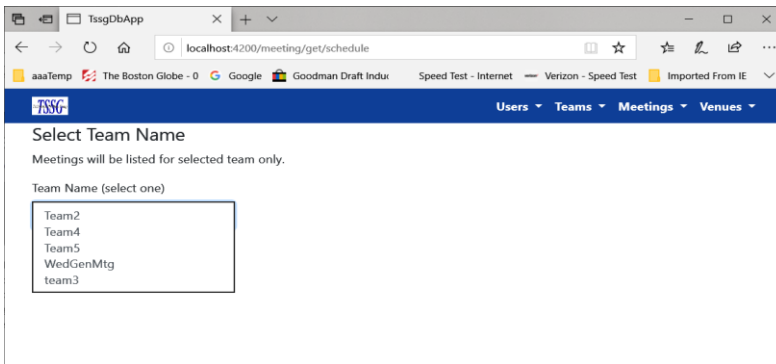
Then add one or more venues.



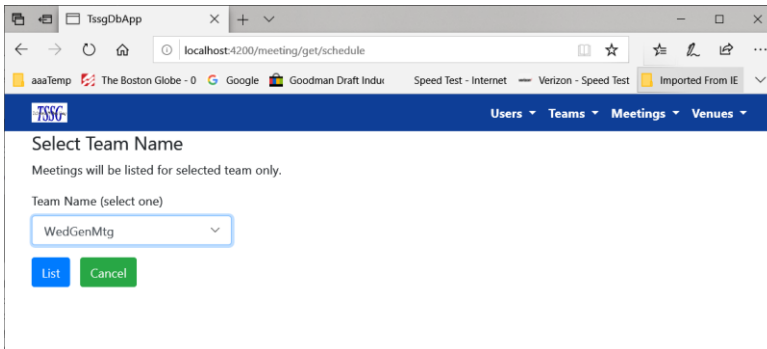
Now you can create meetings. Meetings require both team and venue entries.



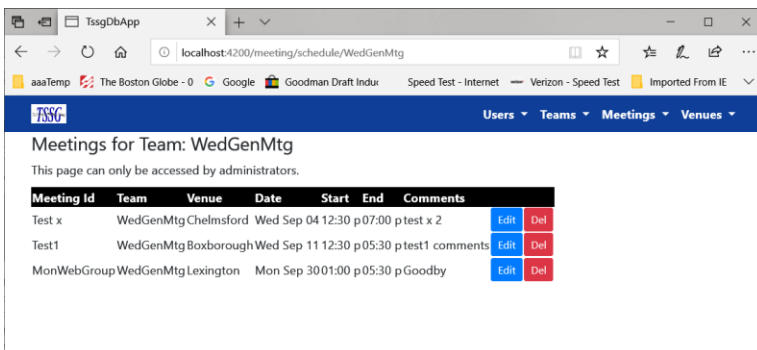
Select Meetings/List Team Meetings. The following page will display:



Select a team name



Select List. The scheduled (0, 1, 2, or 3) meetings for that team will display.
To qualify, a meeting date must be \geq today's date and for meetings for today's date the meeting end time must be $>$ current time.



Once you have a few meetings you can try one of the webSchedule test methods included in the Public folder section.

Public folder:

Items in this folder are not part of the project. I have this setup as a public test web site. If you use the api url without one of the api paths (without users, teams, meetings, venues) server.js will route the request to the public folder. If no argument was provided, the server will return the index.html file in the public folder. If an argument is provided, the server will attempt to locate it in the public folder and server it. To access the public folder, you use the backend url:

For example:

<http://localhost | ip :7010> – returns index.html

[http://localhost | ip :7010/index.html](http://localhost:7010/index.html) – returns index.html

[http://localhost | ip :7010/schedule.html](http://localhost:7010/schedule.html) -returns schedule.html

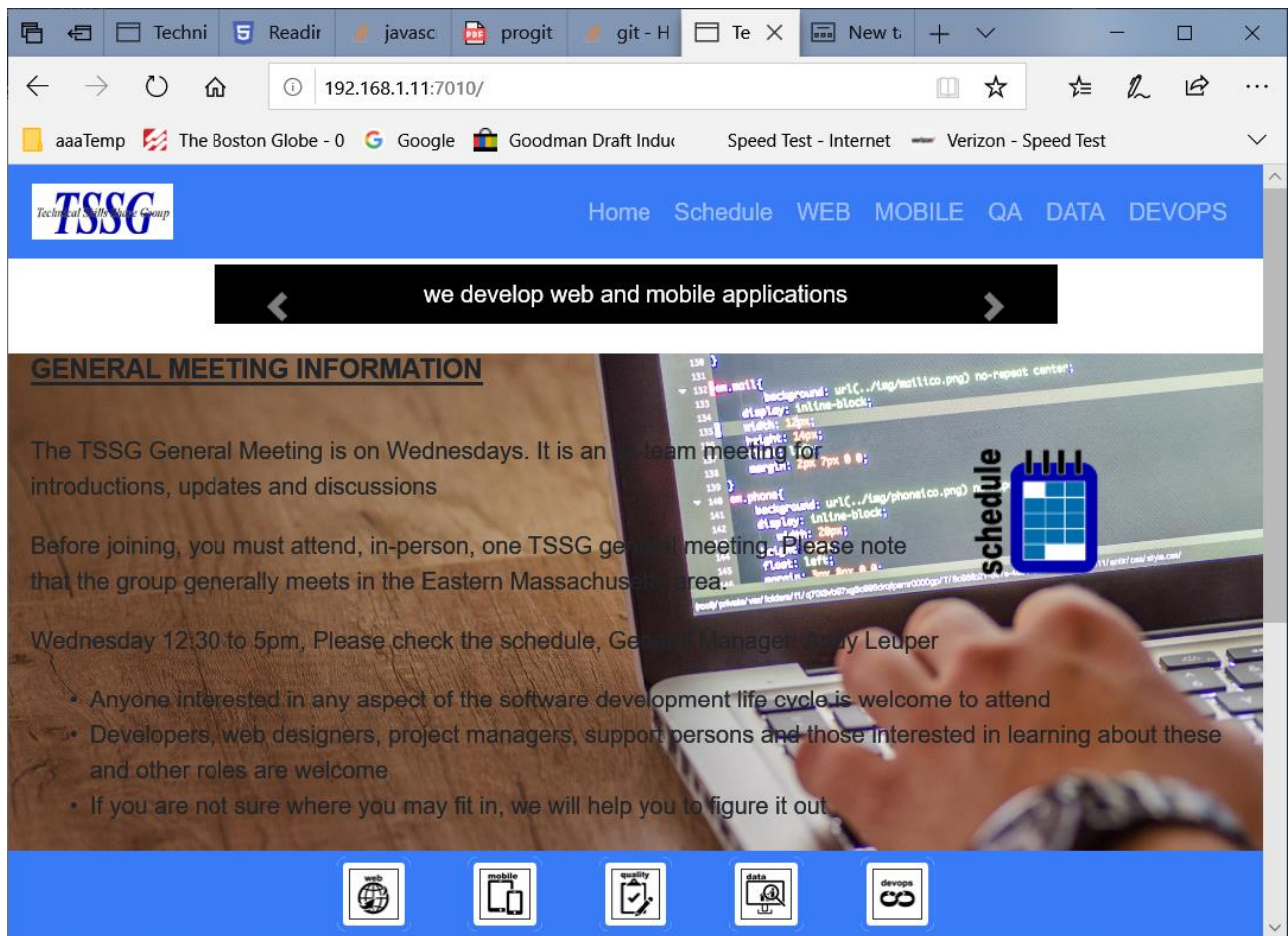
Index.html and schedule.html are the only files included in the project. There is no reason why you could not include other files if you want to.

To access the public folder, you do not need the frontend code. It does not need to be running or even present. These operations use only the backend code and the database code. If you call for the schedule from the index.html page or by accessing schedule.html, the code uses the public access port (not to be confused with the public folder) just as the TSSG web site will do once it is set up. The public folder is not required for the TSSG web site and may be removed if desired. The public/index.html, public/imports/tssgPageTemplate.html and public/schedule.html files demonstrate exactly what is needed to modify the code in the TSSG web site.

index.html is my version of the tssg classic website. Using this, the tssg schedule can be used to test the api schedule calls. When using the schedule call from the web page (index.html), the schedule is built in tssgPageTemplate.html. This is due to my single page implementation. There is no problem with this, but it divides the effort into two parts. The java script located in index.html and schedule page modeled on tssgPageTemplate.html which is in the index.html shadow dom.

If you want to use something closer to what your classic website is doing (your classic multipage) you can use localhost:7010/schedule.html. This will pull schedule.html from the tssgDbApi/public folder. This version of schedule.html contains the required java script and more closely emulates the way your current multi-page website works. The schedule.html page produces a full-page load just as your current classic website does. It probably does not exactly duplicate your current website schedule page but shows you what is required to modify your current classic website.

Page displayed for index.html



All the footer buttons, navbar links and the TSSG logo also work.

Page displayed when selecting the navbar schedule or home page schedule button:

Technical Staffing Solutions Group

Home Schedule WEB MOBILE QA DATA DEVOPS

we perform data analytics

WEDNESDAY		
Oct 30 2019	<div>library</div> <div>BOXBOROUGH</div> <div>more information...</div>	12:30 to 5 pm
Nov 06 2019	<div>library</div> <div>ACTON</div> <div>more information...</div>	12:30 to 5 pm
Nov 13 2019	<div>library</div> <div>ACTON</div> <div>more information...</div>	12:30 to 5 pm

web mobile quality data devops

All the footer buttons, navbar links and the TSSG logo also work.

Page displayed for schedule.html:

Techni

Readir

javasc

progit

git - H

192.168.1.11:7010/schedule.html

aaaTemp

The Boston Globe - 0

Google

Goodman Draft Indur

WEDNESDAY		
Oct 30 2019	<div> <div>library</div> <div>BOXBOROUGH</div> <div>more information...</div> </div>	12:30 to 5 pm
Nov 06 2019	<div> <div>library</div> <div>ACTON</div> <div>more information...</div> </div>	12:30 to 5 pm
Nov 13 2019	<div> <div>library</div> <div>ACTON</div> <div>more information...</div> </div>	12:30 to 5 pm