Contents

[Assignment Part 1: 2](#_Toc520283486)

[Assignment Part 2: 3](#_Toc520283487)

[Questions: 3](#_Toc520283488)

[Why did you use the design/framework (if any) you chose? 3](#_Toc520283489)

[High Level design: 4](#_Toc520283490)

[Are there better frameworks for this task which you did not choose for some reason? 4](#_Toc520283491)

[When you added the testing, why did you select those particular tests? 5](#_Toc520283492)

[How can the API calls be made secure so that not anyone with the API can call them? 5](#_Toc520283493)

[Explain how bad input should be handled on the back end if it gets past the front-end checks? 5](#_Toc520283494)

[Which JavaScript best practices do you find valuable and adhere to? 5](#_Toc520283495)

# Assignment Part 1:

* ./GranifyApp directory
* Dependencies:
  + Angular/Angular CLI (<https://angular.io/>)
  + Amexio (<https://amexio.org/>)
  + npm JavaScript package manager (<https://www.npmjs.com/>)
* All dependencies required downloading some JavaScript packages from ‘npm’
* See the following web page for proper deployment
  + <https://angular.io/guide/deployment>

Backend API Definition:

The backend server API would require the following functions to support the front end (in Pseudo JavaScript code). I do not have experience with HTTP RPC calls for proper syntax.

class Record {

id: string

name: string

number: string

}

getAllRecords(): record[];

getAddCount(): number();

subscribeAddCount(callback): void;

getDeleteCount(): number;

subscribeDeleteCount(callback): void;

addsPerHour(): number

subscribeAddsPerHour(callback): void;

deletesPerHour(): number;

void subscribeDeletesPerHour(callback): void;

// addRecord return values

// 1 == record added successfully

// 0 == record exists

// -1 == invalid inputs

addRecord(name: string, number: string): number;

deleteRecord(id: string, name: string, number: string): boolean;

findRecord(id: string, name: string, number: string): boolean;

# Assignment Part 2:

Unfortunately, the Amexio common control library added syntax to the HTML markup that the AngularCLI testing framework choked on syntactically. By the time I discovered this most of the application had been built. I did not have time to determine how to solve this problem or how to expose the Amexio components to the testing framework properly.

# Questions:

## Why did you use the design/framework (if any) you chose?

I researched several alternatives for web development frameworks and chose Angular with a Visual Studio Code IDE for project management, GitHub source control support and JavaScript debugging.

There is a lot of community support, Visual Studio Code extensions and tutorials for this setup of web application development. AngularCLI (command line interface for angular) also has commands to work within the Visual Studio Code IDE directly (creating projects, components and services…etc.). Angular also has a built-in testing framework for unit tests for each component/service and app created with the AngularCLI command line interface (i.e. when you create a component… its test script is auto generated for you).

I also used Amexio Angular components to build my application with (https://amexio.org/). This ensured most of the controls in my application were well tested and sped up the development of my project.

### High Level design:

Angular Application (Client Web browser)

AppComponent (root Angular Component)

Backend Service (Web server)

GranifyAppService

Future Components

GranifyAppService:

This is the heart of the application. It is a global Angular service that provides the business logic and data to this application. It is split out into its own service to follow Model View Controller pattern and to facilitate modularization. With the business logic/data separated from the UI we can add additional views or Angular components such that the views and components are super light weight and in a decoupled way from the business logic.

This service would be responsible for marshalling calls from the view to the backend and vis versa. This keeps the views/components lightweight and decoupled from the location and format of the backend data. This service acts as the proxy between the Data Model and the view/controllers within MVC pattern.

App Component:

This is the root component for the Angular application. I used this as the front-end view/component for the GranifyAppService. It communicated with the GranifyAppService only. Its up to the GranifyAppService to provide data and functions for the view/components to use.

## Are there better frameworks for this task which you did not choose for some reason?

I believe each framework has its positives and negatives and in given situations different frameworks should be used based on the desired end goals. I have very little JavaScript and web development experience. For me I needed a framework that was quick modular, plug and play and easy to learn/understand. The lower level frameworks like Bootstrap and jQuery did not have the plugin type of support I was looking for nor the automatic testing mechanisms supported by Angular.

React also seems to have wide spread usage in the community with the same support for Visual Studio Code IDE, extensions and testing. I chose Angular simply because it came up more in my research and I felt more comfortable using it.

## When you added the testing, why did you select those particular tests?

Unfortunately, I could not sort out my issues with the testing framework in my Angular Project. The test scripts are auto generated but the tests all failed due to the Amexio 3rd party Angular Components. The errors appear to be related to HTML syntax that seem fine in the Angular runtime application but threw many exceptions at the testing framework layer. I will highlight what tests I wanted to write instead:

For each Angular component/service I would create a test for each public API function to that component. Each API function would have 2 tests, one with valid input and an expected result check. The other with invalid input and an expected error check. This would ensure each API call is working as expected in the known cases as well as proper error conditions for invalid inputs on the error test cases.

## How can the API calls be made secure so that not anyone with the API can call them?

Client-side JavaScript is not secure. Anyone can see the code and the APIs it calls. We can protect the backend APIs on the service with client login/authentication that generates a valid token (some piece of data) on both the client and server that can be passed into each API request on the server from the client. That token would be checked against its counter part on the server to verify the client side is authenticated and allowed to call the server-side API calls.

## Explain how bad input should be handled on the back end if it gets past the front-end checks?

Each client->server API call should return a success / failure code. Server API functions should immediately verify the inputs received and send a failure message on bad data. It would be up to the client-side code to gracefully handle these error cases for a clean user experience.

## Which JavaScript best practices do you find valuable and adhere to?

Unfortunately, this is my first jump into JavaScript. I do not have much knowledge to properly answer this question.