National Aeronautics and
Space Administration

KSC-IT-G

RELEASE DATE: NOVEMBER 16, 2018

COMPUTER PROGRAMMING RESOURCES 2018

Brandeis Bellamy, Intern

NASA Internships and Fellowships (NIFs)
IT-G Technical Integration Office
Information Technologies and Communications Services Directorate
NASA Kennedy Space Center

REVISION AND HISTORY PAGE

NAME	WHAT WAS CHANGED/ADDED	DATE
Brandeis Bellamy	Created Entire Document. Input from Dan Simons on sections: "PF Project Structure" and "Additional Competencies Needed"	Nov 2018

Table of Contents

REVISION AND HISTORY PAGE 2
ACRONYMS 9
DEFINITIONS & DESCRIPTIONS 10
Angular 10
ASP.NET 10
C Sharp 10
Cascading Style Sheets (CSS) 10
Graphic User Interface (GUI) 10
Hypertext Markup Language (HTML) 10
JavaScript 10
Microsoft Visual Studio 2017 10
Microsoft SQL Server 2014 11
Report Definition Language Client-side (RDLC) 11
Structured Query Language (SQL) 11
TypeScript 11
NASA KSC WEB DEVELOPMENT 12
1. REQUIRED ONLINE ACCESS 12
Confluence 12
Project Foundation 2.0 12
Quality Assurance System 12
2. REQUIRED SOFTWARE DOWNLOADS 12
Microsoft SQL Server Management Studio 2014 12
Microsoft Visual Studios 2017 Professional Edition 12
Angular 12
PF Visual Studio Extensions 13
Authos Key 13
Sourcetree 13
Bitbucket 13
3. REQUIRED NAMS REQUESTS (IdMAX) 14
ACES workflow for Elevated Privileges 14
IT152 Center API Manager (CAPI) 14
KSC App Logging Service (AILS IM101) 14

KSC Atlassian Products Suite (IM123) 14
KSC OneClick Deployment System 14
4. REQUIRED KSC WEB DEVELOPMENT ACCESS 14
KSC Fixed Ladder Inventory System (KFLIS) URLs 14
KFLIS Home Directories 15
QAS URLs 15
QAS Home Directories 15
5. PROJECT FOUNDATION (PF) PROJECT STRUCTURE 15
Data Project: PF.Core.Data 15
Logic Project: PF.Core.Logic 16
Unit Test Project: PF.Core.Test 16
Utility Project: PF.Core.Utility 16
Web Project: PF.Core.Website.Angular 16
Database Schema Project: PF.Core.Schema 17
Services Project: PF.Services.Employee 17
6. ADDITIONAL COMPETENCIES NEEDED 17
Bamboo 17
OneClick 18
DACPAC 18
SANDBOX 18
Bitbucket 19
Unit Tests 19
508 Compliance 19
SortSite 19
18F Web Design Standards 19
Fontawesome 19
Acunetix Scan 20
Subversion 20
SourceTree 20
gitignore 21
WEB DEVELOPMENT 21
1. BACK-END WEB DEVELOPMENT 21
C# 21

Linking SQL with C# 22
RDLC Report 22
SQL 23
NuGet 24
2. FRONT-END WEB DEVELOPMENT 28
CSS 28
HTML 28
JavaScript 29
3. ANGULAR 6 30
Angular 30
Bootstrap 31
JSON 31
Typescript 32
ANGULAR 6 TUTORIAL NOTES AND GENERAL FILE DESCRIPTIONS 32
SRC FOLDER 32
APP FOLDER 32
SERVICE FILE 33
COMPONENT FILES 34
ANGULAR 4 REACTIVE FORMS TUTORIAL NOTES 37
ADDING AN ALERT TO TEXT FIELD 38
USING THE CHECKBOX 38
NODE.JS NOTES FROM ANGULAR 6 AND ANGULAR 4 TUTORIALS 41
Acronyms 41
Flags 41
Command Prompts 41
4. ASP.NET 42
QUALITY ASSURANCE SYSTEM (QAS) WEB DEVELOPMENT 45
1. SA QAS WEBSITE: SPECIFIC FILE DESCRIPTIONS, SCREEN VIEWS & PSEUDOCODE 45
COMPONENTS 45
Screen Shot: All Component Files 46
PAGE 1: WELCOME PAGE 46
Screenshot: Welcome Page Component Files 46
Screen View: Welcome Page 46

PAGE 2: ADD EMPLOYEE CONFIGURATION FILE 46
Screenshot: Add Employee Component Files 47
Screen View: Add Employee Page 47
employee.component.ts 47
employee.component.html 48
PAGE 3: SEARCH ENGINEER 49
Screen Shot: Search Engineer Component Files 49
Screen View: Search Employee Page 49
Screen View: Search Employee Details Page 50
engineer.component.ts 50
engineer.component.html 50
engineer-details.component.ts 50
engineer-details.component.html 51
PAGE 4: ADD PROJECT CONFIGURATION FILE 51
Screen Shot: Add Project Component Files 51
Screen View: Add Project Page 52
project.component.ts 52
project.component.html - values 53
PAGE 5: NEW JOB REQUEST FORM 53
Screen Shot: New Job Request Form Component Files 53
Screen View: New Job Request Form Page 54
request.component.ts 54
request.component.html 55
PAGE 6: SEARCH JOB REQUEST FORM 56
Screen Shot: Search Job Request Form Component Files 56
Screen View: Search Job Request Form Page 56
Screen View: Search Job Request Form Details Page 57
job.component.ts 57
job.component.html 58
job-details.component.ts 58
job-details.component.html 58
2. COMPLETE ANGULAR SA QAS CODE BEFORE ADDING IT TO ASP.NET 59
Employee 59

employee.component.ts 59
employee.component.html 60
employee.component.scss 61
employee.component.spec.ts 62
Engineer 63
engineer.component.ts 63
engineer.component.html 63
engineer.component.scss 64
engineer.component.spec.ts 64
Engineer Details 64
engineer-details.component.ts 64
engineer-details.component.html 65
engineer-details.component.scss 65
engineer-details.component.spec.ts 65
Home (Not added to ASP.NET) 66
home.component.ts 66
home.component.html 66
home.component.scss 67
home.component.spec.ts 67
Job 67
job.component.ts 67
job.component.html 68
job.component.scss 69
job.component.spec.ts 69
Job Details 69
job-details.component.ts 69
job-details.component.html 70
job-details.component.scss 70
job-details.component.spec.ts 70
Project 71
project.component.ts 71
project.component.html 72
project.component.scss 73

project.component.spec.ts 74
Project Details 74
project-details.component.ts 74
project-details.component.html 75
project-details.component.scss 75
project-details.component.spec.ts 75
Request 76
request.component.ts 76
request.component.html 77
request.component.scss 79
request.component.spec.ts 80
Sidebar (Not added to ASP.NET) 80
sidebar.component.ts 80
sidebar.component.html 81
sidebar.component.scss 82
sidebar.component.spec.ts 82
Other App Files 83
app.module.ts 83
app-routing.module.ts (Not added to ASP.NET) 84
data.service.ts 85
filter.pipe.ts (Not added to ASP.NET) 86
Other SRC Files 86
index.html 86
styles.scss 87
APPENDIX I – TUTORIAL: CONNECTING SQL DATABASE TO ASP.NET 88
APPENDIX II - INTERVIEWING CLIENTS: REQUIREMENTS ELICITATION 104

ACRONYMS

ASP = Active Server Page
API = Application Programming Interface
C# = C Sharp Programming Language
CLI = Command Line Interface
CSS = Cascading Style Sheets
DACPAC= Data-tier Application Component Packages
DBA = Database Administrator
DOM = Document Object Model
GUI = Graphic User Interface
HTML = Hypertext Markup Language
HTTP = HyperText Transfer Protocol
IT-G = Technical Integration Office
IDE = Integrated Development Environment
JSON = JavaScript Object Notation
KFLIS = KSC Fixed Ladder Inventory System
KSC = Kennedy Space Center
MVS = Microsoft Visual Studios
NPM = Node Package Manager
PF = Project Foundations
QAS = Quality Assurance System
R&T = Research and Technology
RDLC = Report Definition Language Client-side
SA = Safety and Mission Assurance Directorate
SRC = Source
SSMS = Microsoft SQL Server Management Studio
SQL = Structured Query Language
SVN = Subversion
T-SQL = Transact-SQL
UB = Exploration Research and Technology Programs Directorate

DEFINITIONS & DESCRIPTIONS

Angular

Angular is a TypeScript-based open-source front-end web application platform led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS. Additional technologies that can be used with Angular include JSON, Node.js, and Bootstrap. As of 10/05/2018, the latest version of Angular is Angular 6.

ASP.NET

ASP.NET is a unified Web development model that includes the services necessary for you to build enterprise-class Web applications with a minimum of coding. ASP.NET is part of the .NET Framework, and when coding ASP.NET applications you have access to classes in the .NET Framework. You can code your applications in any language compatible with the common language runtime (CLR), including Microsoft Visual Basic and C#. These languages enable you to develop ASP.NET applications that benefit from the common language runtime, type safety, inheritance, and so on.

C Sharp

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative.

Cascading Style Sheets (CSS)

CSS is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

Graphic User Interface (GUI)

GUI is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.

Hypertext Markup Language (HTML)

HTML is the standard markup language for creating web pages and web applications. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

JavaScript

JavaScript is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm. JavaScript enables interactive web pages and thus is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.

Microsoft Visual Studio 2017

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as web sites, web apps, web services and mobile apps. Visual Studio

includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML and CSS.

Microsoft SQL Server 2014

Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

Report Definition Language Client-side (RDLC)

RDLC is used in Microsoft Visual Studio for client-side reporting. RDLC reports are local reports running completely on the client-side using Visual Studio ReportViewer control.

Structured Query Language (SQL)

SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS).

TypeScript

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing to the language. TypeScript is designed for development of large applications and transpile to JavaScript. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs. TypeScript may be used to develop JavaScript applications for both client-side and server-side (Node.js) execution.

NASA KSC WEB DEVELOPMENT

1. REQUIRED ONLINE ACCESS

Confluence

- NASA KSC IT's online collaboration tool
- https://confluence.ksc.nasa.gov/dashboard.action#all-updates

Project Foundation 2.0

- Page located in Confluence
- Project Foundation (PF) is the starting point for all applications. It contains many of the features necessary on nearly every project we produce, including integration with API Services (Employee, Authorization, Logging, etc.) and provides a starting point for your project
- Gives step-by-step instructions on how to create an Angular project
- https://confluence.ksc.nasa.gov/display/PF2

Quality Assurance System

- Page located in Confluence
- Workspace for the SA QAS project
- https://confluence.ksc.nasa.gov/display/QAS/Quality+Assurance+System+Home

2. REQUIRED SOFTWARE DOWNLOADS

Microsoft SQL Server Management Studio 2014

- https://www.microsoft.com/en-ca/sql-server/sql-server-downloads

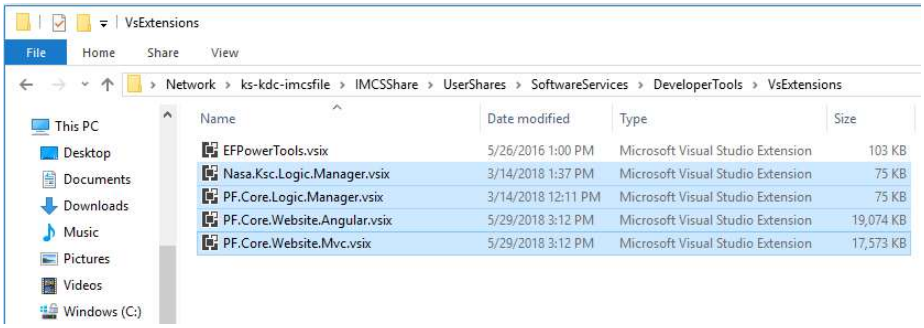
Microsoft Visual Studios 2017 Professional Edition

- Needs registration key from Dr. Ali
- https://visualstudio.microsoft.com/downloads/
- Microsoft Visual Studio IDE is not the same as Visual Studio Code code editor

Angular

- Is included Microsoft Visual Studios ASP.NET
- Is included in NASA's Confluence/Project Foundation 2.0
- Do not need to download Anglular CLI from https://angular.io/guide/quickstart for QAS Project
- Can use Angular CLI and tutorial to learn how Angular works
- If you practice with Angular CLI, make sure to download Node.js

PF Visual Studio Extensions

- The extensions are needed to use Project Foundations (PF) in Microsoft Visual Studio
- https://confluence.ksc.nasa.gov/display/PF2/Installing+the+PF+Visual+Studio+Extensions
- Copy/Paste the file extension into your computer's file explorer: \\ks-kdc-imcsfile\IMCSShare\UserShares\SoftwareServices\DeveloperTools\VsExtensions
- Double click and install the extensions:
    - Nasa.Ksc.Logic.Manager.vsix
    - PF.Core.Logic.Manager.vsix
    - PF.Core.Website.Angular.vsix
    - PF.Core.Website.Mvc.vsix
- 

Authos Key

- A step in creating PF.Core.Website.Angular through the Confluence website
- https://confluence.ksc.nasa.gov/display/PF2/Authos+-+Retrieving+Your+Apps+Authos+Key
- http://authos.ksc.nasa.gov
- For Authos Authorization:
    - Contact KSC IMCS Care Center KSC-IMCS-Care-Center@mail.nasa.gov
    - Create IM102 Authos NAMS request

Sourcetree

- Sourcetree is a good tool if multiple people are working on one project
- It simplifies how you interact with your Git repositories so you can focus on coding. Visualize and manage your repositories through Sourcetree's simple Git GUI.
- You might be prompted to update .NET, which you should.
- https://www.sourcetreeapp.com/

Bitbucket

- Bitbucket is used with Sourcetree. It is also called Stash
- https://confluence.ksc.nasa.gov/pages/viewpage.action?pageId=2883769
- Bitbucket Server is self-hosted Git repository collaboration and management for professional teams.

- https://confluence.atlassian.com/bitbucketserver

3. REQUIRED NAMS REQUESTS (IdMAX)

- Each requestor must have up-to-date IT security training
- Each request must summarize purpose of request (eg. “Working with mentor on QAS project”)
- Each request must be authorized by you mentor
- Certain requests might need to be renewed (i.e. Elevated Privileges)
- https://idmax.nasa.gov/nams/user

ACES workflow for Elevated Privileges

- This workflow is used to request and obtain administrative rights for ACES computers (FTP, Admin roles, etc)
- Can only request access on a weekly basis

IT152 Center API Manager (CAPI)

- The Center API Manager (CAPI) is used for the administration and monitoring of KSC API Services.

KSC App Logging Service (AILS IM101)

- Application Information Logging Services (AILS) will be a series of services that will allow applications to log application events using a centralized set of services. Additionally administrative and report functions will be available to configure the application as well as to view the logs.

KSC Atlassian Products Suite (IM123)

- KSC Atlassian Products Suite for JIRA, Bamboo, Crucible, Confluence and Stash. (IM123)
- For each suite element, choose user, not administrator

KSC OneClick Deployment System

- Used to deploy a website with the click of a button
- Choose “Developer” role

4. REQUIRED KSC WEB DEVELOPMENT ACCESS

KSC Fixed Ladder Inventory System (KFLIS) URLs

- Development: https://kflis-dev.ksc.nasa.gov
- Review: https://kflis-review.ksc.nasa.gov

- Test: https://kflis-test.ksc.nasa.gov

KFLIS Home Directories

- Development: \\ks-kdc-imcsfile\internal$\Dev\kflis-dev
- Review: \\ks-kdc-imcsfile\internal$\Review\kflis-review
- Test: \\ks-kdc-imcsfile\internal$\Test\kflis-test

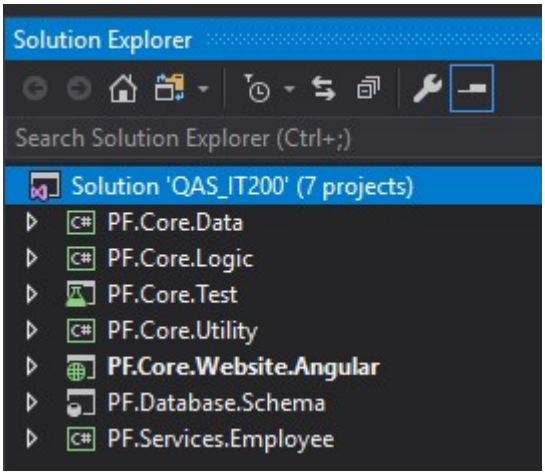
QAS URLs

- Development: https://qas-dev.ksc.nasa.gov
- Review: https://qas-review.ksc.nasa.gov
- Test: https://qas-test.ksc.nasa.gov

QAS Home Directories

- Development: \\ks-kdc-imcsfile\internal$\Dev\qas-dev
- Review: \\ks-kdc-imcsfile\internal$\Review\qas-review
- Test: \\ks-kdc-imcsfile\internal$\Test\qas-test

5. PROJECT FOUNDATION (PF) PROJECT STRUCTURE



Data Project: PF.Core.Data

- Create SQL Server Database Project With Visual Studio
    - When we work on any project, a database plays an important role and after time when the number of tables, views and procedures increases - it becomes too difficult to manage the scripts… We can create a new database project and import database schema from an existing database, a .sql script file or a Data-tier application (.dacpac). We can then invoke the same visual designer tools (Transact-SQL Editor, Table Designer) available for connected database development to make changes to the offline database project, and

publish the changes back to the production database. The changes can also be saved as a script to be published later. Using the Project Properties panel, we can change the target platform to different versions of SQL Server (including SQL Azure).
    - https://www.c-sharpcorner.com/article/create-sql-server-database-project-with-visual-studio

Logic Project: PF.Core.Logic

- Quickstart: Create and automate tasks, processes, and workflows with Azure Logic Apps - Visual Studio
    - With Azure Logic Apps and Visual Studio, you can create workflows for automating tasks and processes that integrate apps, data, systems, and services across enterprises and organizations. This quickstart shows how you can design and build these workflows by creating logic apps in Visual Studio and deploying those apps to Azure in the cloud.
    - https://docs.microsoft.com/en-us/azure/logic-apps/quickstart-create-logic-apps-with-visual-studio

Unit Test Project: PF.Core.Test

- Create a unit test project
    - Unit tests often mirror the structure of the code under test. For example, a unit test project would be created for each code project in the product. The test project can be in the same solution as the production code, or it can be in a separate solution. You can have multiple unit test projects in a solution.
    - https://docs.microsoft.com/en-us/visualstudio/test/create-a-unit-test-project?view=vs-2017

Utility Project: PF.Core.Utility

- Using Visual Studio Utility Project Types
    - To help you more easily maintain files within a solution, Visual Studio makes available various utility projects. These utility projects allow you to keep track of files that are not part of any other project that is loaded into a solution. Because any file type can be stored within these projects, such as program source files or Microsoft Word documents, these projects can't be compiled into a program. And because utility projects are not associated with any particular programming language, they are available to all users of Visual Studio and don't require Visual Basic, Visual C#, or Visual C++ to be installed.
    - https://flylib.com/books/en/3.50.1.52/1/

Web Project: PF.Core.Website.Angular

- Creating a Web Project to Deploy
    - In the following procedure, you create a Web application project to deploy by using the Visual Studio ASP.NET Web Application project template. You will create the ASP.NET default membership database, add the AdventureWorksLT database, and add controls to display data from the AdventureWorksLT database.
    - https://msdn.microsoft.com/en-us/library/dd483479(v=vs.100).aspx#Anchor_1

Database Schema Project: PF.Core.Schema

- DOME database schema (project-specific data)
  - http://www.comp-sys-bio.org/DOME/domeschema.html
- Using Visual Studio database projects in real life
  - To get changes made to database to schema project we create new database schema comparison. Schema comparison allows us to compare two different schemas. Right click on database project and select Schema Compare
  - https://weblogs.asp.net/gunnarpeipman/using-visual-studio-database-projects-in-real-life

Services Project: PF.Services.Employee

- Lesson 1: Defining a Data Source View within an Analysis Services Project
  - Designing a business intelligence application in SQL Server starts with creating an Analysis Services project in SQL Server Data Tools (SSDT). Within this project, you define all the elements of your solution, starting with a data source view.
  - https://docs.microsoft.com/en-us/sql/analysis-services/lesson-1-defining-a-data-source-view-within-an-analysis-services-project?view=sql-server-2017

6. ADDITIONAL COMPETENCIES NEEDED

Bamboo

A Bamboo build will need to be created. This will create a build package and automatically deploy the application to the DEV website, after which that package (a NuGet package) will be pushed to the OneClick environment, where people will be able to push a deployment to the Review, Test or Production websites.

- Atlassian Product: Installation Needed
- Understanding the Bamboo CI Server
  Bamboo is a continuous integration (CI) server that can be used to automate the release management for a software application, creating a continuous delivery pipeline. CI is a software development methodology in which a build, unit tests and integration tests are performed, or triggered, whenever code is committed to the repository, to ensure that new changes integrate well into the existing code base. Integration builds provide early 'fail fast' feedback on the quality of new changes.
  - https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html
- Bamboo Upgrade and Installation
  - https://confluence.ksc.nasa.gov/display/ATLS/Bamboo+Upgrade+and+Installation
- Bamboo Setting up a new Build
  - https://confluence.ksc.nasa.gov/display/ATLS/Bamboo+Setting+up+a+new+Build
- Bamboo Pre and Post Upgrade Steps
  - https://confluence.ksc.nasa.gov/display/ATLS/Bamboo+Pre+and+Post+Upgrade+Steps

OneClick

A OneClick plan will need to be set up. OneClick is basically a deployment workflow that allows people to deploy to a website with the click of a button. As a workflow, there are steps in that process that must be followed by various people (i.e., there are developer steps, project manager steps, DBA steps and IT Security steps). The big thing to know about OneClick is that you must define many variables, most of which are within your web.config file, which will be transformed using those specified variables, based on the selected deployment environment.

- 1-Click
    - 1-Click, also called one-click or one-click buying, is the technique of allowing customers to make online purchases with a single click, with the payment information needed to complete the purchase having been entered by the user previously.
    - https://en.wikipedia.org/wiki/1-Click
- Standards and Practices
    - https://confluence.ksc.nasa.gov/display/SA/Standards+and+Practices#

DACPAC

This is our new way of creating and deploying database changes. We no longer have direct modify access to the DEV database. We have read-only access now. The correct way to develop our applications that contain databases, is for each developer to make the changes in his or her SANDBOX database, update the database schema project, commit those changes to the Bitbucket repository, then merge them with the master branch in the repository after which, the DEV database will be updated with those schema changes (and pushes to each environment thereafter will update the applicable database (for Review, Test and Production).

- About
    - DACPAC, or Data-tier Application Component Packages, is a feature in SQL Server 2008 that allows developers to package database changes into a single file in Visual Studio and send it to the DBAs for deployment.
    - https://stackoverflow.com/tags/dacpac/info
- DACPAC is a reason to upgrade to SQL Server 2008 R2
    - https://web.archive.org/web/20140815170457/http://infinitecodex.com/infinitecodex/post/2010/12/09/DACPAC-is-a-reason-to-upgrade-to-SQL-Server-2008-R2.aspx
- Data-tier Applications
    - https://docs.microsoft.com/en-us/sql/relational-databases/data-tier-applications/data-tier-applications?view=sql-server-2017
- Deploying a DACPAC with SQL Server Management Studio
    - https://blogs.msmvps.com/deborahk/deploying-a-dacpac-with-sql-server-management-studio/

SANDBOX

A sandbox is a testing environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including Web development and revision control. Sandboxing protects "live" servers and their data, vetted source code distributions, and other collections of code, data and/or content, proprietary or public, from changes that could be damaging to a mission-critical system or which could simply be difficult to revert. Sandboxes

replicate at least the minimal functionality needed to accurately test the programs or other code under development.

- Sandbox (software development)
  - https://en.wikipedia.org/wiki/Sandbox_(software_development)
- Document conversion sandbox for Confluence Data Center
  - https://confluence.atlassian.com/conf611/document-conversion-sandbox-for-confluence-data-center-956138729.html

Bitbucket

Bitbucket Server is self-hosted Git repository collaboration and management for professional teams.

- Atlassian Product: Installation Needed
- Bitbucket Server documentation
  - https://confluence.atlassian.com/bitbucketserver
- Install a Bitbucket Server trial
  - https://confluence.atlassian.com/bitbucketserver/install-a-bitbucket-server-trial-867192384.html
  - Stash/Bitbucket
    - https://confluence.ksc.nasa.gov/pages/viewpage.action?pageId=2883769

Unit Tests

Although they are not a dire necessity, it's a good practice to create them as the team sees fit. HOWEVER, when you transfer your project to KIAC, they WILL require them, although the amount, quality, and coverage is open and vague.

508 Compliance

We use a tool called Sortsite to test our applications for any 508 Compliance shortcomings. Although, we've found, that on Angular applications, being SPA (Single Page Applications), unless you run Sortsite on a page-by-page basis, a single run will pretty much always pass.

SortSite

SortSite is a web crawler that scans entire websites for quality issues including: accessibility; browser compatibility; broken links; legal compliance; search optimization; usability and web standards compliance.

- https://en.wikipedia.org/wiki/SortSite

18F Web Design Standards

It also must comply with 18F Web Design Standards (which Project Foundation 2 adheres to). I believe Twitter Bootstrap and **Fontawesome** comply as well, so we are free to use them.

- https://18f.gsa.gov/2015/09/28/web-design-standards/

Fontawesome

Font Awesome makes it easy to add vector icons and social logos to your website

- https://fontawesome.com/
- The iconic SVG, font, and CSS toolkit
    - https://github.com/FortAwesome/Font-Awesome

Acunetix Scan

This must be run any time you deploy an application to Production and it must be approved by IT Security. There will be a step in OneClick for their approval.

Acunetix (Acunetix Web Vulnerability Scanner)

- Tutorial: Using Acunetix with targeted technology profiles
    - https://confluence.ksc.nasa.gov/display/AVS/Tutorial%3A+Using+Acunetix+with+targeted+technology+profiles

Subversion

The NASA Development Teams use Bitbucket Repositories. The KIAC Teams use Subversion (SVN). We do not have to be concerned about converting our application to a Subversion Repository when we hand them over to KIAC. We use either SourceTree *or* Visual Studio 2017 to manage versioning our code (commits, merges, pushes pull requests, etc).

- Used by Atlassian Product Bamboo: Installation might be needed
- What is a SVN repository?
    - Subversion is a powerful open-source version control system that is typically used to manage the collections of files that make up software projects. However, a SVN repository it may actually be used for managing any collection of files that are changed or modified over time… SVN repository (or Subversion repository) is a collection of files and directories, bundled together in a special database that also records a complete history of all the changes that have ever been made to these files.
    - Where can I get a SVN repository?
        - Download the SVN tools and setup a server yourself **or**
        - Using a SVN hosting service to serve your SVN repository
    - https://www.projecthut.com/what-is-svn-repository/
- Subversion with Bamboo
    - https://confluence.atlassian.com/bamboo0606/using-bamboo/linking-to-source-code-repositories/subversion

SourceTree

Sourcetree simplifies how you interact with your Git repositories so you can focus on coding. Visualize and manage your repositories through Sourcetree's simple Git GUI.

- Atlassian Product: Installation Needed
- Sourcetree installation
    - https://www.sourcetreeapp.com/

- Sourcetree knowledge base
  - https://confluence.atlassian.com/sourcetreekb
- How to set up SSH keys for authentication with SourceTree
  - https://confluence.ksc.nasa.gov/display/ATLS/How+to+set+up+SSH+keys+for+authentication+with+SourceTree
- SourceTree Basics
  - https://confluence.atlassian.com/sourcetreekb/sourcetree-basics-780870007.html

gitignore

The gitignore file is important so as not to commit unnecessary files, especially the node-modules folder for Angular applications, which can be huge (300MB's or larger).

- .gitignore explanation
  - https://www.atlassian.com/git/tutorials/saving-changes/gitignore
- gitignore.io
  - https://www.gitignore.io/

WEB DEVELOPMENT

1. BACK-END WEB DEVELOPMENT

C#

- **Ali Farhat, "Payroll Example of GUI C# Part 1"**
  URL: https://www.youtube.com/watch?v=SgMtj_mAwtE [cited 13 April 2018]
  Description: GUI Elements in Visual Studio Payroll example

- **Ali Farhat, "Visual Studio C# GUI Part 1, 2, & 3"**
  URL: https://www.youtube.com/watch?v=xpXtHgzOniU [cited 13 April 2018]
  Description: GUI Elements in Visual Studio

- **Derek Banas, "C# tutorial"**
  URL: https://www.youtube.com/watch?v=lisiwUZJXqQ [cited 13 April 2018]
  Description: C# tutorial using the Console in Visual Studios

- **programminghelporg, "C#.Net Tutorial 22 - Introduction to the Graphical User Interface"**
  URL: https://www.youtube.com/watch?v=GIcgONHznuU [cited 13 April 2018]
  Description: Learn GUI controls such as group boxes, check boxes, checklist boxes, radio buttons, combo boxes, numeric ups and downs, picture boxes, richtext boxes, tooltips, and link labels.

- **Tutorials Point (India) Pvt. Ltd., "C# - Introduction to GUI"**
  URL: https://www.youtube.com/watch?v=sHUXbCXmZKU [cited 13 April 2018]
  Description: C# - Introduction to GUI

Linking SQL with C#

- **codeproject.com, "How to connect SQL Database to your C# program, beginner's tutorial"**
  URL: https://www.codeproject.com/articles/823854/how-to-connect-sql-database-to-your-csharp-program [cited 13 April 2018]
  Description: Linking Databases in Microsoft SQL Server to C# GUI in Visual Studios

- **connectionstrings.com, "SQL Server 2012 Connection String"**
  URL: https://www.connectionstrings.com/sql-server-2012/ [cited 13 April 2018]
  Description: C# Connections Strings for different security levels to link SQL to Visual Studios

RDLC Report

- **Abbas Al-Eryani, "Creating reports using report viewer and passing parameter C#"**
  URL: https://www.youtube.com/watch?v=G1OuPZNsCr4 [cited 13 April 2018]
  Description: Creating a report viewer with parameters in C# GUI RDLC Report

- **Dotnet Awesome, "How to create Microsoft Report (.rdlc) with multiple datasources in** ASP.NET."
  URL: http://www.dotnetawesome.com/2013/11/how-to-create-microsoft-report-rdlc.html [cited 13 April 2018]
  Description: Show RDLC report with multiple datasources

- **FoxLearn, "C# Tutorial - How to create RDLC Report with parameters | FoxLearn"**
  URL: https://www.youtube.com/watch?v=WGKzFKgeoo8 [cited 13 April 2018]
  Description: How to create RDLC Report with parameters using Microsoft Report Viewer in C#. This is the best way to learn C# for beginners.

- **Programming-Tech, "C# how to create rdlc report with parameters"**
  URL: https://www.youtube.com/watch?v=hZ9JcT1d0SM [cited 13 April 2018]
  Description: This video provides a guidance on how to create a simple RDLC Report with parameters.

- **ProgrammingKnowledge, "C# Tutorial 10: How to Link Combobox with Database values"**
  URL: https://www.youtube.com/watch?v=cdkDHkXyVFI [cited 13 April 2018]
  Description: GUI C# - How to show and bind a combobox to a textbox

- **ProgrammingKnowledge, "C# Tutorial 11: Database values in textbox if select Combobox"**
  URL: https://www.youtube.com/watch?v=dwG6eg5UofI [cited 13 April 2018]
  Description: Continuation of C# Tutorial 10

- **stackoverflow.com, "Comparing strings in C# with OR in an if statement"**
  URL: https://stackoverflow.com/questions/11283764/comparing-strings-in-c-sharp-with-or-in-an-if-statement [cited 13 April 2018]

Description: As a user inputs information, this helps to check if certain textboxes are still empty.

- **stackoverflow.com, "Get integer from Textbox"**
  URL: https://stackoverflow.com/questions/11931770/get-integer-from-textbox [cited 13 April 2018]
  Description: C# - If the user needs to input a number into a textbox, it needs to be parsed in order to be read as a string

- **stackoverflow.com, "Getting Error Msg - Cannot implicitly convert type 'string' to 'bool'"**
  URL: https://stackoverflow.com/questions/2357354/getting-error-msg-cannot-implicitly-convert-type-string-to-bool [cited 13 April 2018]
  Description: C# - Using a Boolean expression to read a string

- **stackoverflow.com, "How can I get column names from a table in SQL Server?"**
  URL: https://stackoverflow.com/questions/1054984/how-can-i-get-column-names-from-a-table-in-sql-server [cited 13 April 2018]
  Description: How to make the SQL table column names appear in a C# GUI dropdown combobox

SQL

- **dofactory.com, "SQL SELECT DISTINCT Statement"**
  URL: http://www.dofactory.com/sql/select-distinct [cited 13 April 2018]
  Description: Used so multiple results with identical information are not shown

- **Human Machine-Interfaces, "Microsoft SQL Server 2014 - Introductory Tutorial"**
  URL: https://www.youtube.com/watch?v=osd_JVYDDHs [cited 13 April 2018]
  Description: Learn the basics of using Microsoft SQL Server 2014. This video tutorial demonstrates how to connect to a database, how to create new database, create a new table, add data, and the basics of using scripts.

- **Joey Blue, "Basic SQL Training Tutorial"**
  URL: https://www.youtube.com/watch?v=2HVMiPPuPIM&list=PLD20298E653A970F8 [cited 13 April 2018]
  Description: Multiple videos on how to use SQL

- **Joey Blue, "Learn SQL in 1 Hour - SQL Basics for Beginners"**
  URL https://www.youtube.com/watch?v=9Pzj7Aj25lw [cited 13 April 2018]
  Description: A crash course in SQL. Creating a Database, Creating Tables, Inserting, Updating, Deleting, Selecting, Grouping, Summing, Indexing, and Joining.

- **Microsoft SQL Server Management Studio (SSMS) Tutorials, "Tutorial: Connect to and query a SQL Server instance by using SQL Server Management Studio"**
  URL: https://docs.microsoft.com/en-us/sql/ssms/tutorials/connect-query-sql-server [cited 13 April 2018]

Description: This tutorial teaches you how to use SQL Server to connect to your SQL Server instance and run some basic Transact-SQL (T-SQL) commands. The article demonstrates how to: Connect to a SQL Server instance, Create a database ("TutorialDB"), Create a table ("Customers") in your new database, Insert rows into your new table, Query the new table and view the results, Use the query window table to verify your connection properties, and Change the server that your query window is connected to.

- **Microsoft T-SQL Language elements, "Language Elements (Transact-SQL)"**
  URL: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/language-elements-transact-sql [cited 13 April 2018]
  Description: This site teaches all of the skills needed to be considered a SQL Administrator.

- **Microsoft TechNet, "How to insert data from one table to another table"**
  URL: https://blogs.technet.microsoft.com/mdegre/2009/09/08/how-to-insert-data-from-one-table-to-another-table/ [cited 13 April 2018]
  Description: Inserting data from one table into another.

- **SQL Server Helper, "SQL Server Error Message"**
  URL: http://www.sql-server-helper.com/error-messages/msg-1-500.aspx [cited 13 April 2018]
  Description: This explains the common errors displayed when there are SQL syntax errors

- **stackoverflow.com, "Select data from date range between two dates"**
  URL: https://stackoverflow.com/questions/14208958/select-data-from-date-range-between-two-dates [cited 13 April 2018]
  Description: When using C# GUI DateTimePicker, this helps to show a range of dates in a RDLC report

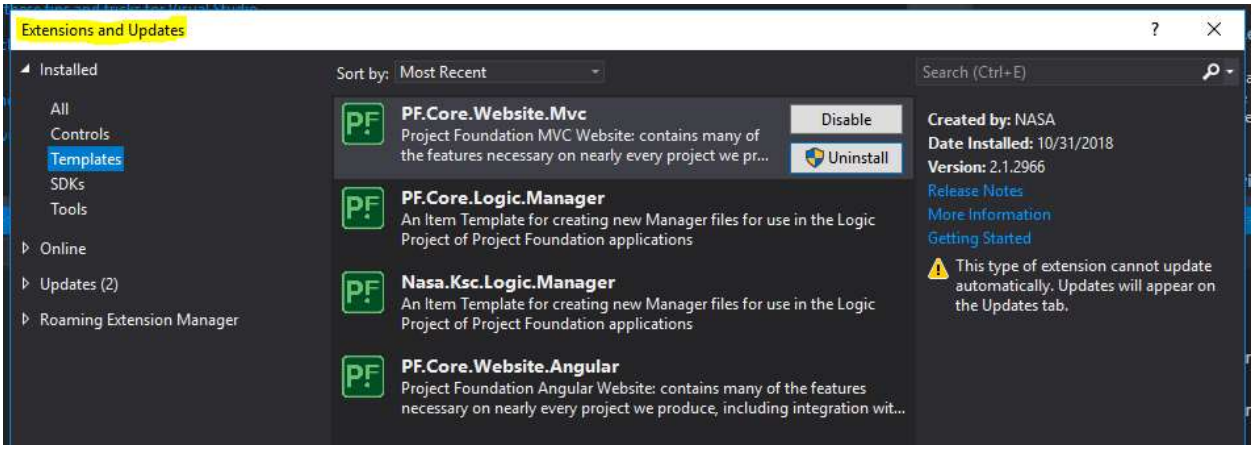
- **w3schools.com, "SQL Tutorial"**
  URL: https://www.w3schools.com/sql/default.asp [cited 13 April 2018]
  Description: A very good website to learn SQL with "Try It Yourself" examples

- **Web and Database Programming, "Database - Getting Started With SQL Server Management Studio"**
  URL: https://www.youtube.com/watch?v=4n63lvs7LyU [cited 13 April 2018]
  Description: As part of Database II - this first lecture is a review and is intended to get students up and running in SQL Server Management Studio

NuGet

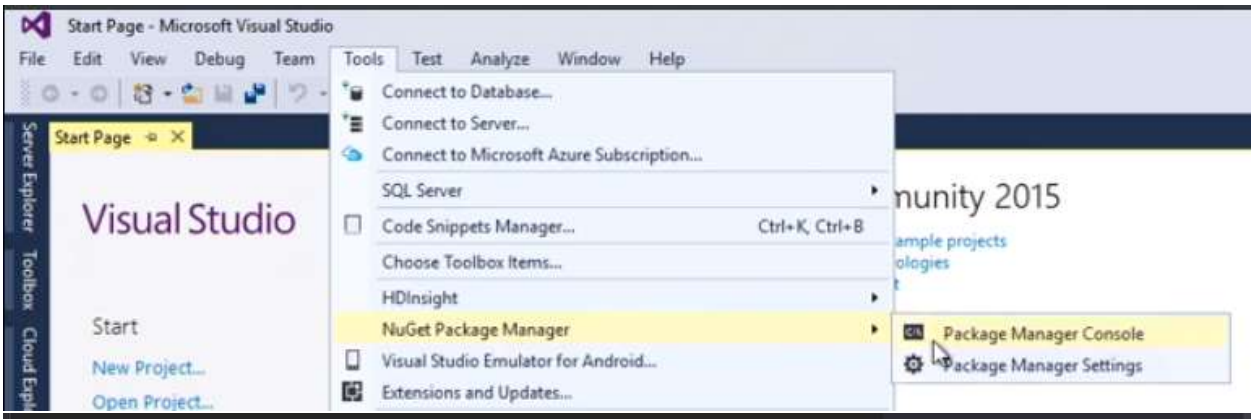
- An introduction to NuGet
  - https://docs.microsoft.com/en-us/nuget/what-is-nuget
- NuGet Website
  - https://www.nuget.org/packages
- NuGet Package Explorer
  - https://github.com/NuGetPackageExplorer/NuGetPackageExplorer

- [1]**LinkedIn Learning/ Lynda.com, "Overview of NuGet"**
  - https://www.linkedin.com/learning/navigating-dot-net-and-dot-net-standard-for-cross-platform-development/overview-of-nuget
- [2]**LinkedIn Learning/ Lynda.com, "Windows Package Management: NuGet and Chocolatey"**
  - https://www.linkedin.com/learning/windows-package-management-nuget-and-chocolatey
    - Extensions and Updates



    - Package Manager Console

[1] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[2] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research




```
Package Manager Console
Package source: All    Default project:
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-
party packages. Some packages may include dependencies which are governed by additional licenses. Follow the package
source (feed) URL to determine any dependencies.

Package Manager Console Host Version 4.8.0.5385

Type 'get-help NuGet' to see all available NuGet commands.

PM> get-help nuget
TOPIC
    about_NuGet

SHORT DESCRIPTION
    Provides information about NuGet Package Manager commands.

LONG DESCRIPTION
    This topic describes the NuGet Package Manager commands. NuGet is an integrated package
    management tool for adding libraries and tools to .NET projects.


    The following NuGet cmdlets are included.

        Cmdlet                Description
        ------------------    -------------------------------------------
        Find-Package          Get the set of packages available from the package source,
                              based on the package Id/keyword. This is a new command that
                              will replace Get-Package -ListAvailable.

        Get-Package           Gets the set of installed packages. With -Updates switch,
                              gets the set of package updates available from the package source.

        Install-Package       Installs a package and its dependencies into the project.

        Uninstall-Package     Uninstalls a package. If other packages depend on this package,
                              the command will fail unless the -Force option is specified.

        Update-Package        Updates a package and its dependencies to a newer version.

        Sync-Package          Get the version of installed package from specified/default project
                              and sync the version to the rest of projects in the solution.
```

```
        Add-BindingRedirect     Examines all assemblies within the output path for a project
                                and adds binding redirects to the application (or web)
                                configuration file where necessary.

        Get-Project             Returns a reference to the DTE (Development Tools Environment)
                                for the specified project. If none is specifed, returns the
                                default project selected in the Package Manager Console.

        Open-PackagePage        Open the browser pointing to ProjectUrl, LicenseUrl or
                                ReportAbuseUrl of the specified package.

        Register-TabExpansion   Registers a tab expansion for the parameters of a command.

SEE ALSO
    Online documentation: http://go.microsoft.com/fwlink/?LinkID=206619
    Find-Package
    Get-Package
    Install-Package
    Uninstall-Package
    Update-Package
    Sync-Package
    Add-BindingRedirect
    Get-Project
    Open-PackagePage
    Register-TabExpansion
PM>
```

- NASA NuGet Repo Package - find-package

```
Package Manager Console
Package source: NASA NuGet Repo Packages    Default project:
PM> find-package

Id                                  Versions                  Description
--                                  --------                  -----------
PF.Core.Rulesets                    {1.0.2865}                Copies the latest ruleset files to the root directory for the solution.
SitecoreFramework8                  {2.0.23}                  This is the framework that is used for Sitecore 8 development at KSC
PF.Core.Rulesets                    {1.0.2865}                Copies the latest ruleset files to the root directory for the solution.
PF.Services.Employee                {1.0.2939}                Adds the PF.Services.Employee C# Project to your solution in order to s...
SitecoreFramework8                  {2.0.23}                  This is the framework that is used for Sitecore 8 development at KSC
PF.Core.Reporting.Mvc               {1.0.2458}                Adds Reporting Services files and web.config settings to support the Re...
SitecoreFramework8                  {2.0.23}                  This is the framework that is used for Sitecore 8 development at KSC
PF.Core.Reporting.Angular           {1.0.2076}                NOTE: this is for Angular Apps. Adds Reporting Services files, Angular ...
PF.Core.Rulesets                    {1.0.2865}                Copies the latest ruleset files to the root directory for the solution.
PF.Services.Employee                {1.0.2939}                Adds the PF.Services.Employee C# Project to your solution in order to s...
PF.Core.Rulesets                    {1.0.2865}                Copies the latest ruleset files to the root directory for the solution.
PF.Core.EmployeeLookup.Mvc          {1.0.1429}                Adds the Employee Lookup views and controller files to the selected pro...
PF.Core.Rulesets                    {1.0.2865}                Copies the latest ruleset files to the root directory for the solution.
PF.Services.Employee                {1.0.2939}                Adds the PF.Services.Employee C# Project to your solution in order to s...
ComponentArt2012EmbedLicenseForS... {1.0.0}                   This package adds a license embedded ComponentArt 2012 dll.  Please see...
Datalist.Core                       {5.4.0}                   Datalist core dll without content files.
PF.Core.Rulesets                    {1.0.2865}                Copies the latest ruleset files to the root directory for the solution.
SitecoreFramework8                  {2.0.23}                  This is the framework that is used for Sitecore 8 development at KSC
PF.Services.Employee                {1.0.2939}                Adds the PF.Services.Employee C# Project to your solution in order to s...
imcs.techdocgateway                 {1.0.0.1}                 Encapsulates uploading and deleting documents to TechDoc.  Handles upda...
Time Elapsed: 00:00:05.3089506
```

2. FRONT-END WEB DEVELOPMENT

CSS

- [3]**LinkedIn Learning/ Lynda.com**, "**CSS Essential Training 1"**
  URL: https://www.linkedin.com/learning/css-essential-training-1 [cited 5 October 2018]
  Description: CSS is a stylesheet language that allows you to control the appearance of your webpages. In this hands-on course—the first installment in an ongoing series—Christina Truong demonstrates the concepts that form the foundation of CSS, and explains how to use this language to add colors and other design elements to take your webpages beyond just black text on a white background. Christina covers selecting content, how the box model defines the spacing and sizing of page elements, styling text, and managing basic layouts. She also explores the tools needed to work with CSS, how to use selectors to target elements, and guidelines for page layouts with floats. Plus, at the end of the course, you'll walk away with an actual project—an online résumé page.

- [4]**LinkedIn Learning/ Lynda.com, "Learning Responsive Design"**
  URL: https://www.linkedin.com/learning/learning-responsive-design [cited 30 April 2018]
  Description: Web projects need to work across multiple devices, screen sizes, and browsing contexts. Web designs need to be responsive to these variables, providing an optimal viewing experience for each scenario. The course covers concepts like screen density, fluid grids, and responsive images, as well as actual design strategies that guide you from mock-up to testing.

HTML

- [5]**LinkedIn Learning/ Lynda.com, "HTML Essential Training"**
  URL: https://www.linkedin.com/learning/html-essential-training [cited 5 October 2018]
  Description: HTML is the programming language that powers the web. And like any language, once you master it, you can begin to create your own content, whether that's simple websites or complex web applications. This course provides an in-depth look at the essentials: the syntax of HTML and best practices for writing and editing your code. Senior staff author James Williamson reviews the structure of a typical HTML document, and shows how to section pages and format your content with HTML. Plus, learn how to create links and lists, and find out how HTML works with CSS and JavaScript to create rich, engaging user experiences. So open a text editor, watch these videos, and begin learning to author HTML the right way.

- [6]**LinkedIn Learning/ Lynda.com, "HTML & CSS: Creating Forms"**
  URL: https://www.linkedin.com/learning/html-css-creating-forms [cited 5 October 2018]
  Description: Forms are a major component of modern websites, used to collect user data, conduct polls, and more. While you can use a form builder, they're relatively simple to build from scratch with basic web technologies: HTML and CSS. In this course, Clarissa Peterson

[3] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[4] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[5] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[6] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research

demonstrates best practices for creating and styling web forms that are easy to complete and accessible to all users and devices. Learn proper use of form fields, including the new HTML5 input types, and discover how to add interactivity such as form validation, style forms with CSS, create responsive layouts, add interactivity such as form validation, and design forms for usability *and* accessibility.

- **U.S. Web Design System**
  URL: https://designsystem.digital.gov/ [cited 5 October 2018]
  Description: The U.S. Web Design System is the easiest path for teams around the federal government to comply with the latest federal policy for websites and digital services. The Design System supports modern web development practices, such as mobile first and responsive frameworks. By offering open-source, quick-to-download code and assets, the Design System makes it easy to deliver the highest-quality government websites to the public.

JavaScript

- [7]**LinkedIn Learning/ Lynda.com, "JavaScript Essential Training"**
  URL: https://www.linkedin.com/learning/javascript-essential-training-3 [cited 5 October 2018]
  Description: JavaScript is a scripting language of the web. As the web evolves from a static to a dynamic environment, technology focus is shifting from static markup and styling—frequently handled by content management systems or automated scripts—to dynamic interfaces and advanced interaction. Once seen as optional, JavaScript is now becoming an integral part of the web, infusing every layer with its script. Through practical examples and mini-projects, this course helps you build your understanding of JavaScript piece by piece, from core principles like variables, data types, conditionals, and functions through advanced topics including loops, closures, and DOM scripting. Along the way, you will also be introduced to some ES6 and the basics of JavaScript libraries.

- [8]**LinkedIn Learning/ Lynda.com, "Learning the JavaScript Language"**
  URL: https://www.linkedin.com/learning/learning-the-javascript-language [cited 5 October 2018]
  Description: JavaScript is the lingua franca of the web, but before using it to create dynamic websites, you need to understand how it works. In this workshop trainer and developer Joe Chellman explores the syntax behind the JavaScript language. He shows how to "speak" JavaScript by gaining an understanding of variables, types, objects, arrays, operators, control structures, loops, and functions, through a series of hands-on examples that put these ideas into action. After completing this course, most developers will understand the core syntax of JavaScript and how this scripting language works to build powerful and complex functionality on the web.

[7] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[8] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research

- **w3schools.com, "**JavaScript HTML DOM"
  URL: https://www.w3schools.com/js/js_htmldom.asp [cited 5 October 2018]
  Description: With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

- [9]**LinkedIn Learning/ Lynda.com, "JavaScript for Web Designers**"
  URL: https://www.linkedin.com/learning/javascript-for-web-designers-2 [cited 5 October 2018]
  Description: Web designers can improve their skill sets and job prospects by getting comfortable with JavaScript. It's the only scripting language for client-side programming—for doing things that would be impossible with HTML and CSS alone—and is increasingly popular for server-side programming, too. Learn some basic, real-world uses for JavaScript in this course with Joe Chellman. Find out how to write JavaScript in the browser and in a code editor, and then put it straight to use in a shopping cart application: creating forms and form fields, reading data, checking input for errors, and performing calculations. Plus, learn how to use JavaScript time and date functions to create features like calendars and countdown clocks, and add interactive content through APIs like Google Maps. This course is a great jumping-off point for more intricate client-side programming (e.g., mobile apps or highly interactive websites) and server-side programming with frameworks like Node and Backbone. With the addition of JavaScript, your design possibilities are almost limitless.

3. ANGULAR 6

Angular

- [10]**LinkedIn Learning/ Lynda.com, "Angular Essential Training"**
  URL: https://www.linkedin.com/learning/angular-essential-training [cited 5 October 2018]
  Description: Angular was designed by Google to address challenges programmers face building complex, single-page applications. This JavaScript framework takes care of the back end so you can take care of the client side. *Angular Essential Training* introduces you to the essentials of this "superheroic" framework, including powerful features such as rich templates, change detection, user interactions, two-way data binding, comprehensive routing, and dependency injection. Justin Schwartzenberger steps through the framework one feature at a time, focusing on the component-based architecture of Angular. Learn what Angular is and what it can do, as Justin builds a full-featured web app from start to finish. After mastering the essentials, you can tackle the other project-based courses in our library and create your own Angular app.

- **Angular, "Tutorial: Tour of Heroes"**
  URL: https://angular.io/tutorial [cited 5 October 2018]
  Description: The Tour of Heroes tutorial covers the fundamentals of Angular. In this tutorial you will build an app that helps a staffing agency manage its stable of heroes. This basic app has many of the features you'd expect to find in a data-driven application. It acquires and displays a list of heroes, edits a selected hero's detail, and navigates among different views of heroic data. By the end of the

[9] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[10] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research

tutorial you will be able to do the following: Use built-in Angular directives to show and hide elements and display lists of hero data, Create Angular components to display hero details and show an array of heroes, Use one-way data binding for read-only data, Add editable fields to update a model with two-way data binding, Bind component methods to user events, like keystrokes and clicks, Enable users to select a hero from a master list and edit that hero in the details view, Format data with pipes, Create a shared service to assemble the heroes, and Use routing to navigate among different views and their components. You'll learn enough Angular to get started and gain confidence that Angular can do whatever you need it to do.

- **Egghead.io, "Get Started with Angular"**
  URL: https://egghead.io/courses/get-started-with-angular [cited 5 October 2018]
  Description: Angular is basically a collection of Components brought together within modules. The many tools, such as the Angular CLI, allow you to easily create Components. The key to understanding Angular is understanding how Components interact with each other through Services, Inputs, and Outputs. It's also essential to understand the basics of styling components and how CSS is shared so you can make your application look the way you want. This course will step us through creating a simple Angular application starting with the Angular CLI. We will build out a few components that will handle events through Angular Event handlers. The course will then finish up with styling the application.

Bootstrap

- [11]**LinkedIn Learning/ Lynda.com, "What is Bootstrap?"**
  URL: https://www.linkedin.com/learning/learning-bootstrap-2/what-is-bootstrap [cited 30 April 2018]
  Description: Bootstrap is needed for AngularJS and Angular. Bootstrap is a free web development tool from Twitter that, with a little bit of CSS and JavaScript experience, makes building websites quick, intuitive, and fun. The video explores its 12-column grid layout; typography and icon libraries; fully functional components like nav bars, buttons, and tabs; and much more. This course also shows how to add JavaScript extras like dropdown menus, modal windows, and photo carousels.

JSON

- [12]**LinkedIn Learning/ Lynda.com, "What is JSON?"**
  URL: https://www.linkedin.com/learning/javascript-and-json-integration-techniques/what-is-json [cited 30 April 2018]
  Description: Jason is needed for AngularJS and Angular. JavaScript Object Notation (JSON) has replaced XML as the core way of sharing data, especially when it comes to JavaScript, since it's so much faster, sleeker, and easier to parse. In this course, dive into working with JSON tools, designing JSON objects, and using different ways to handling JSON data.

[11] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[12] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research

Typescript

- [13]**LinkedIn Learning/ Lynda.com, "TypeScript Essential Training"**
  URL: https://www.linkedin.com/learning/typescript-essential-training/welcome [cited 30 April 2018]
  Description: Typescript is used in Angular instead of JavaScript. However Typescript can understand JavaScript syntax when written in a .ts file. TypeScript lets you write JavaScript "the way you really want to." TypeScript compiles to plain JavaScript, so it works in any browser, any host, and any OS. It adds a variety of helpful syntax and tools onto an already mature language, bringing the power and productivity of static typing and object-oriented development to core JavaScript. Plus, it's completely open source.

ANGULAR 6 TUTORIAL NOTES AND GENERAL FILE DESCRIPTIONS

- **Coursetro.com, "Learn Angular 6 in 60 Minutes - Free Beginners Crash Course"**
  URL: https://coursetro.com/posts/code/154/Angular-6-Tutorial---Learn-Angular-6-in-this-Crash-Course [cited 5 October 2018]

SRC FOLDER

index.html

- info put in <head> section
- can import custom Google fonts & material icons
  - material icons `<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">`
    - https://material.io/tools/icons/?style=baseline
  - custom Google fonts `<link href="https://fonts.googleapis.com/css?family=Montserrat:300,700" rel="stylesheet">`

styles.scss

- `styles here will apply to templating of all components`
- found on https://coursetro.com/posts/code/154/Angular-6-Tutorial---Learn-Angular-6-in-this-Crash-Course

APP FOLDER

app.components.ts

- basic building blocks of apps
- *import{}* - imports Component from angular core library
- *@Component{(…)}* - component decorator object
  - selector: 'app-root' - referencing <app-root> in index.html

[13] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research

    - templateUrl: - defines where the html template is coming from
    - styleUrls: - defines where the scss template is coming from
- *export class AppComponent {}* - logic section of component file, where all properties goes and dependency injections, where you define custom methods an functions

app.components.html

- where all html templating goes
- can imbed a component in another component's template
    - eg. <app-sidebar> is linked to selector: 'app-sidebar' in sidebar.components.ts, and references what's written in sidebar.component.html file
    - need
- need <router-outlet>
    - linked to the routing flag (--router) which generated the app-routing.module.ts file
- only need this code:

```
<div id="container">
  <app-sidebar></app-sidebar>

  <div id="content">
    <router-outlet></router-outlet>
  </div>
</div>
```

app.components.scss

- where all css for app.components.html goes

app.components.spec.ts

- test file

app-routing.module.ts

- Links pages together, makes icons work when clicked.
- Lists Components generated in the CLI command prompt
- import{…} from '…';
    - import components that you want to route
    - eg - import{UsersComponent} from './users/users.component';
- *const routes: Routes=[{ }]*
    - define three objects in Routes array
    - path: - each route will take a path to the URL where it's going to be
        - path: ' ', the root path, will be initial path that loads when the app loads
        - path: 'details/:id', - :id is a URL parameter
    - component: - shows which component resides at that path

SERVICE FILE

- named data, can be named anything

data.service.spec.ts

- test file

data.service.ts

- purpose of service file is to communicate with an API via the Angular 6 HTTP Client
- has reusable code that all Components to have access to
- can connect to an API through a database
- *import {HttpClient} from '@angular/common/http';*
  - connects with built-in HTTP client
- constructor() - to use the HttpClient, we can create an instance of it through dependency injection within the constructor
  - eg *constructor(private http: HttpClient){ }*
- custom method to call on the client
  - eg - *getUsers() { return this.http.get('…')}*
  - json users example at https://jsonplaceholder.typicode.com/users/
    - free API for testing websites
    - https://jsonplaceholder.typicode.com/
  - if you want information of a single user, can put
    - *getUser(userId) {return this.http.get('…' + userId)}}*
    - accepts *userId* as a parameter
    - make sure the **/** is before *+ userId*

app.module.ts

- before using HttpClient, need to import
  - *import { HttpClientModule } from '@angular/common/http';* **and**
  - *imports: [… HttpClientModule, ...]*
- when components & services are generated, they are auto-updated in the @NgModule decorator
  - components added to declarations array
  - services added as providers
- where the HttpClient is referenced
  - *import {HttpClientModule} from '@angular/common/http';*
- then have to add it to the *imports:[…]* section
- to import animations
  - *import { BrowserAnimationsModule } from '@angular/platform-browser/animations';*
  - add to imports - *imports:[… BrowserAnimationsModule]*

COMPONENT FILES

- named users, posts, sidebar & details, but can be named anything

users.component.ts

- where you import the data that you referenced from the json website in the data.service.ts file
  - *import {DataService} from '../data.service';*

- also where you import rxjs observables
    - *import {Observable} from 'rxjs';* - holds data returned from API
- *export class UsersComponent* - adding a property to hold return data from API, needed to display the results
    - *users$: Object;*
- gain access to service through dependency injection, create an instance of it:
    - *constructor(private data: DataService){}*
- execute code when component loads, a lifecycle hook for angular, subscribe to Observable, bind data to users to return data

```
ngOnInit() {
  this.data.getUsers().subscribe(
    data => this.users$ = data
  )
 }
```

- **Animation functions**: https://angular.io/guide/animations
    - *import { trigger,style,transition,animate,keyframes,query,stagger } from '@angular/animations';*
    - in Component decorator, add animations property as an array

```
animations: [
    trigger('listStagger', [
      transition('* <=> *', [
        query(':enter',
          [
          style({ opacity: 0, transform: 'translateY(-15px)' }),
          stagger('50ms',
            animate('550ms ease-out',
              style({opacity: 1, transform: 'translateY(0px)'})))
          ], { optional: true }),
        query(':leave', animate('50ms', style({ opacity: 0 })), {
          optional: true
        })
      ])
    ])
]
```

users.component.html

- what is shown in the user area of the browser
    - in unordered list, list item <li *ngFor= "let user of users$">
        - *ngFor directive lets you iterate over an array of objects
    - user name will be referred to through interpolation {{...}}
    - when the person's name is clicked, will be directed to the details page
        - <a routerLink= "/details/{{user.id}}">{{user.name}}
            - id is coming from the API json file
    - To view **animations**, add to unordered list
        - *<ul [@listStagger]="users$">*

sidebar.component.html

- *<a routerLink="">* - used instead of href to link to different routes

- add class binding from sidebar.component.ts.
    - only activates if template expression validates to true.
    - *<a routerLink="" [class.activated]="currentUrl == '/'">*
    - *<a routerLink="posts" [class.activated]="currentUrl == '/posts'">*

sidebar.component.scss

- component-specific CSS files only apply to that component's HTML template

sidebar.component.ts

- **class binding**: want sidebar to indicate which page you are currently on
- import a new function from the router
    - *import { Router, NavigationEnd } from '@angular/router';*
- create an instance of the router, subscribe to it when someone clicks on the link. Shows the current URL
    - *currentUrl: string;*
      constructor(private router: Router) {
    - *router.events.subscribe((_: NavigationEnd) => this.currentUrl = **this.router**.url) }*

details.component.ts

- also need to import data service and observable
- can copy & paste the same info from the users.component.ts file:
    - *import { DataService } from '../data.service';*
    - *import { Observable } from 'rxjs';*
        - Also need *import { ActivatedRoute } from '@angular/router';*
    - ***user$:** Object;*
    - *constructor(private data: DataService){}*
        - also need ActivatedRoute, so total looks like *constructor(private data: DataService, private route: ActivatedRoute) { }*
        - Inside constructor() {…}: reference the route, get the parameters in the URL, subscribe to them, and equate it to the id (referenced in the app-routing.module.ts file *'details/:id'*)
            - *this.route.params.subscribe(params => this.user$ = params.id)*
    - write ngOnInit to load details when a person clicks on a name *ngOnInit() { this.data.**getUser().**subscribe(data => this.users$ = data)}*
        - details.**component.html**
            - use interpolation inside of heading and unordered lists
                - eg *<h1>{{user$.name}}</h1>*
                - * Username: {{user$.username}}
…
*

posts.component.ts

- similar to details.component.ts and users.component.ts
- import data service and observable, create object, insert constructor, and link with ngOnInit

- *import { DataService } from '../data.service';*
- *import { Observable } from 'rxjs';*
- ***posts$:** Object;*
- *constructor(private data: DataService){}*
- *ngOnInit() {this.data.**getPosts**().subscribe(*
  *data => this.**posts$** = data)*

posts.component.html

- use *ngFor for iteration, routerLink for clickable title, and {{…}} for interpolation in an unordered list
  - *<h1>Posts</h1>*
    *
<li *ngFor="let post of posts$">*
    <a routerLink="">{{ post.title }}
    * {{post.body}} *

General notes

- (click)=”” - click event, when clicking on a button
- (mouseover)=”” - when rolling your mouse over something, it will show the mouse event
- $event - mouse event
- #myInput - ref, reference to an input and pass in other places
- @Inject - inside a class constructor(), shows a dependency injection
  - https://angular.io/api/core/Inject
- *ngFor=”let… of…” - repeats for each event
  - https://angular.io/api/common/NgForOf

ANGULAR 4 REACTIVE FORMS TUTORIAL NOTES

- **Coursetro.com, “Angular 4 Reactive Forms Tutorial”**
  URL: https://coursetro.com/posts/code/66/Angular-4-Reactive-Forms-Tutorial# [cited 5 October 2018]

app.module.ts

- import the reactive forms module from the angular forms library
  - `import { FormsModule, ReactiveFormsModule } from '@angular/forms';`
- add HttpModule if not already imported
  - `import { HttpModule } from '@angular/http';`
- Add to imports
  - `imports: [BrowserModule, FormsModule, ReactiveFormsModule, HttpModule],`

app.component.ts

- import form-specific functions in component that will house the form
  - `import { FormBuilder, FormGroup, Validators } from '@angular/forms'`
    - every form is bound to an instance of FormGroup
    - FormBuilder handles form control creation

    - Use Validators to set up validation for each of the form inputs
- define export class properties
    - `export class AppComponent {`

```
rForm: FormGroup;
post: any; // A property for our submitted form
description: string = '';
name: string = '';
 constructor (…) }
```

    - rForm is short for reactive form, fb is short for form builder
    - in constructor, have dependency injection `constructor(private fb: FormBuilder) {…}`
        - need to **specify validation** within constructor () {…}
        - *'name'*
            - the first input lets you specify a form value, leave as *null*.
            - the second input lets you specify a validation. if you want the name to be required, put *Validators.required*
        - 'description'
            - can have multiple validation requirements
            - eg minimum and maximum lengths
            - use *Validators.compose([…])*
        - 'validate'
            - need a reference to access it later for the checkbox to validate name input
        - `this.rForm = fb.group({`

```
'name': [null, Validators.required],
 'description': [null, Validators.compose([Validators.required,
 Validators.minLength(30), Validators.maxLength(500)])],
    'validate': ''});
```

        - also beneath constructor(){…}, need a custom method to handle the **submitted form**
            - if you had a back API, this is where you would call to store it

```
addPost(post) {
  this.description = post.description;
  this.name = post.name;}
```

ADDING AN ALERT TO TEXT FIELD

- Add in *export class AppComponent{…}*
    - `titleAlert: string = 'This field is required';`

USING THE CHECKBOX

- Need *ngOnInit* lifecycle hook in export class *AppComponent {…}* after *constructor(){}*
- when it loads, *this.rForm = fb.group({…})* will run
- all fields mentioned in *rForms* are tied into *valueChanges*

- can change the validation requirements if the checkbox is on, *validate == '1'*
- the *else* statement returns it to its default setting
- *updateValueAndValidity* updates the name textbox
- ngOnInit() {

```
    this.rForm.get('validate').valueChanges.subscribe(
      (validate) => {
        if (validate == '1') {
          this.rForm.get('name').setValidators([Validators.required,
Validators.minLength(3)])
          this.titleAlert = "You need to specify at least 3 characters";
        } else {
          this.rForm.get('name').setValidators(Validators.required);
        }
        this.rForm.get('name').updateValueAndValidity();
      }
    )
   }
```

index.html

- import front-end framework called Foundation, competitor to Bootstrap, to structure form or website
- `<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/foundation/6.3.1/css/foundation.min.css">`

app.component.html

- if the name property does not exist, show everything inside this * *
- `<div *ngIf="!name; else forminfo"> code shown below </div>`
- if the name does exist, will show *forminfo*, a local template variable
- If they submitted the form successfully, it will show *#forminfo*, not * *
- `<ng-template #forminfo>`

```
    <div class="form-container">
      <div class="row columns">
        <h1>{{name}}</h1>
        <p>{{description}}</p>
      </div>
    </div>
  </ng-template>
```

- "row columns" helps structure the container
- interpolation of name and description properties is defined in app.component.ts file export class *AppComponent{…}* and *addPost(…){…}*
- Form Element section
    - bind *formgroup* to *rForm.* use event binding on *ngSubmit*. When submitted, call method *addPost*. Pass in *rForm*, all input values submitted in the form to the method.
- Labels section

    - *formControlName = "name"* name has to match component name in app.component.ts
    - *formControlName = "description"* name has to match component description in app.component.ts
- Checkbox section
    - checkbox named validate, value is 1. If checked, will pass *On* value to *addPost()*
- Submit Button
    - have property binding on *[disabled]* attribute. disabled if form is not valid (won't be able to click it.). If it is valid, attribute will be removed.

```
<div *ngIf="!name; else forminfo">

  <form [formgroup]="rForm" (ngSubmit)="addPost(rForm.value)">
    <div class="form-container">
      <div class="row columns">

        <h1>My Reactive Form</h1>

        <label>Name
          <input type="text" formControlName="name" />
        </label>

        <label>Description
          <textarea formControlName="description"></textarea>
        </label>

        <label for="=validate">Minimum of 3 Characters</label>
        <input type="checkbox" name="validate" formControlName="validate" value="1" /> On

        <input type="submit" class="button expanded" value="Submit Form"
        [disabled]="!rForm.valid" />

      </div>
    </div>
  </form>

</div>
```

- Creating an alert to show a mandatory field. Can use interpolation. Must be referenced in app.component.ts
- Name textbox
    - `<div class="alert" *ngIf="!rForm.controls['name'].valid && rForm.controls['name'].touched">{{titleAlert}}</div>`
    - *touched* means it only shows if the user clicks the box then clicks out.
- Description textbox
    - `<div class="alert" *ngIf="!rForm.controls['description'].valid && rForm.controls['description'].touched">You must specify a description that's between 30 and 500 characters.</div>`
    - Text explanation must match *description* in app.component.ts

style.css

- one import and change body background color

```
@import url('https://fonts.googleapis.com/css?family=Muli:400,700');
body {
  background: #e8e8e8;
  font-family: 'Muli';}
```

app.component.css

- Animation
-
```
@keyframes alertAnim {

  from {
    opacity: 0;
    transform: translateY(-20px);
  }

  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

NODE.JS NOTES FROM ANGULAR 6 AND ANGULAR 4 TUTORIALS

Acronyms

- CLI: Command Line Interface
- API: Application Programming Interface
- npm: Node Package Manager
- ng: Prefix that stands for Angular

Flags

- -- : Anything with two dashes in front of it is a flag
- *--routing*: Creates different page URLs
- *--style=scss*: Specific type of css
- *--open*: Opens new browser
    - shorthand: *-o*
- *--inline-template --inline-style*: Generate new component, view in one form
    - shorthand: -it -is
- *--flat*: Puts the file in src/app instead of its own folder.
- *--module=app*: Tells the CLI to register it in the imports array of the AppModule
- *--prod*: For production, reduces size

Command Prompts

- Install Angular CLI:
    - *npm install –g @angular/cli*
- Create a new application:
    - *ng new application_name*
- Change directory:

    - *cd application_name*
- Install In-memory Web API to mimic communication with remote server:
    - *npm install angular-in-memory-web-api –save*
- Open new local host / launch application:
    - *ng serve --open*
- Generate new component (includes .ts, .html, and .css files):
    - *ng generate component insert_name*
        - shorthand: *ng g c insert_name*
        - (generates component and component.spec)
- Generate new service (includes .ts file):
    - *ng generate service insert_name*
        - shorthand: *ng g s insert_name*
- Loading and configuring a router:
    - *ng generate module app-routing --flat --module=app*
- To install Animations:
    - *npm install @angular/animations@latest --save*
- To upload app with a standard host (share with others):
    - *ng build --prod*
        - creates a dist folder for distribution, for production

4. ASP.NET

- **Pluralsight.com, "Intro to ASP.NET"**
  URL: https://app.pluralsight.com/player?author=dan-wahlin&name=webforms-01&mode=live&clip=0&course=aspdotnet-webforms4-intro [cited 30 April 2018]
  Description: A comprehensive overview of API.NET for Microsoft Visual Studios

- **Docs.Microsoft.com, "Connecting to Databases in ASP.NET"**
  URL: https://msdn.microsoft.com/en-us/library/ms178371(v=vs.100).aspx [cited 5 October 2018]
  Description: ASP.NET gives you flexibility in how you connect to databases. A simple way is to use data source controls, which allow you to encapsulate data access in a control that you can configure with connection and query information. Alternatively, you can write code to perform data access yourself using ADO.NET classes or LINQ queries.

- **Docs.Microsoft.com, "Getting Started With ASP.NET Core And Angular 4 Using WEB API"**
  URL: https://code.msdn.microsoft.com/Getting-Started-With-d448f7bf [cited 5 October 2018]
  Description: In this article, let's see on how to getting started with ASP.NET Core and using Web API.

- **Docs.Microsoft.com, "Use the Angular project template with ASP.NET Core"**
  URL: https://docs.microsoft.com/en-us/aspnet/core/client-side/spa/angular?view=aspnetcore-2.1&tabs=visual-studio [cited 5 October 2018]
  Description: The updated Angular project template provides a convenient starting point for ASP.NET Core apps using Angular and the Angular CLI to implement a rich, client-side user interface (UI). The

template is equivalent to creating an ASP.NET Core project to act as an API backend and an Angular CLI project to act as a UI. The template offers the convenience of hosting both project types in a single app project. Consequently, the app project can be built and published as a single unit.

- **Docs.Microsoft.com, "Get Started with ASP.NET Web API 2 (C#)"**
  URL: https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api#adding-a-controller [cited 5 October 2018]
  Description: Adding a Controller: In Web API, a controller is an object that handles HTTP requests. We'll add a controller that can return either a list of products or a single product specified by ID.

- **Docs.Microsoft.com, "Creating a Controller (C#)"**
  URL: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/creating-a-controller-cs [cited 5 October 2018]
  Description: The goal of this tutorial is to explain how you can create new ASP.NET MVC controllers. You learn how to create controllers both by using the Visual Studio Add Controller menu option and by creating a class file by hand.

- **Docs.Microsoft.com, ".NET Core command-line interface (CLI) tools"**
  URL: https://docs.microsoft.com/en-us/dotnet/core/tools/?tabs=netcore2x [cited 5 October 2018]
  Description: The .NET Core command-line interface (CLI) is a new cross-platform toolchain for developing .NET applications. The CLI is a foundation upon which higher-level tools, such as Integrated Development Environments (IDEs), editors, and build orchestrators, can rest.

- **Docs.Microsoft.com, "Work with SQL Server LocalDB in ASP.NET Core"**
  URL: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/working-with-sql?view=aspnetcore-2.1&tabs=aspnetcore2x [cited 5 October 2018]
  Description: The object handles the task of connecting to the database and mapping objects to database records. The database context is registered with the Dependency Injection container in the ConfigureServices method in the Startup.cs file

- **Coding Flow, "Building Single Page Applications on ASP.NET Core 2.1 with Angular 6 – Part 1: Getting Started"**
  URL: http://www.codingflow.net/building-single-page-applications-on-asp-net-core-2-1-with-angular-6-part-1-getting-started/ [cited 5 October 2018]
  Description: This is the first in a new series of posts on building Single Page Applications with ASP.NET Core 2.1 and Angular 6. There have been major improvements to both frameworks, so this is a good time to have a fresh look. This post covers the prerequisites, demonstrates how to create a new application using the built-in SPA template, and describes how to upgrade to Angular 6 (the template uses Angular 5).

- **SSW TV, "Building Single Page Applications with ASP.NET Core 2 | Jason Taylor @ DDD Brisbane 2017"**
  URL: https://tv.ssw.com/7393/building-single-page-applications-with-asp-net-core-2-jason-taylor-ddd-brisbane-2017 [cited 5 October 2018]

Description: In this talk, we'll look at using ASP.NET Core JavaScript Services to build single page applications using Angular, Aurelia, Knockout, React, React+Redux or Vue. With this approach you can develop on Windows, Mac, or Linux using your preferred source code editor. I'll demonstrate server-side prerendering, webpack dev middleware, hot module replacement (HMR), efficient production builds and much more. You will walk away ready to create single page applications using your favourite client-side framework and ASP.NET Core.
URL:https://github.com/JasonGT/DDDBNE2017/tree/master/Northwind/Northwind.Presentation/ClientApp/app [cited 5 October 2018]
Description: Code for the video above

- **Telerik, "Cooking with ASP.NET Core and Angular"**
  URL: https://www.telerik.com/blogs/cooking-with-aspnet-core-and-angular-2 [cited 5 October 2018]
  Description: Using cutting edge technology means overcoming that initial learning curve. Many times, we would like to just jump in and get started without starting from scratch. But as we all know the best meals are the ones that are well prepared, and rushing into things too quickly ends up making someone sick or leaving an entire meal in the trash. In this article, we'll find a happy medium—we'll look at how to get started with Angular using ASP.NET Core by following a simple recipe with clearly defined ingredients. We'll start by walking through all of the ingredients necessary to make a successful project. Next we'll follow the recipe, learning where each ingredient fits. Finally, the application will be fully baked and ready to serve with all of its cutting edge goodness.
  URL: https://www.youtube.com/watch?v=TbJSABCCtt8 [cited 5 October 2018]
  Description: In this episode, Robert is joined by Ed Charbeneau for a discussion on using Angular with ASP.NET Core. Ed shows how to create an Angular app using the new ASP.NET Core project templates [T01:45], how to create an Angular app with a Web API backend [T20:45] and how to use Angular to build a native iOS and Android app [T27:10].

- **Docs.Microsoft.com, "Adding a Controller"**
  URL: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-controller [cited 5 October 2018]
  Description: MVC stands for model-view-controller. MVC is a pattern for developing applications that are well architected, testable and easy to maintain.

- [14]**LinkedIn Learning/ Lynda.com, "ASP.NET Essential Training"**
  URL: https://www.linkedin.com/learning/asp-dot-net-essential-training [cited 5 October 2018]
  Description: Thousands of businesses have used Microsoft ASP.NET to build professional, dynamic websites. In this course, web developer David Gassner demonstrates the tools needed to build and deploy a dynamic site using ASP.NET 3.5 or 4.5. Covering everything from installing and configuring Visual Web Developer 2008 or Visual Studio Express 2012 for Web and SQL Server Express to

[14] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research

creating web form pages, this course is designed to give beginning and intermediate developers hands-on experience.

QUALITY ASSURANCE SYSTEM (QAS) WEB DEVELOPMENT

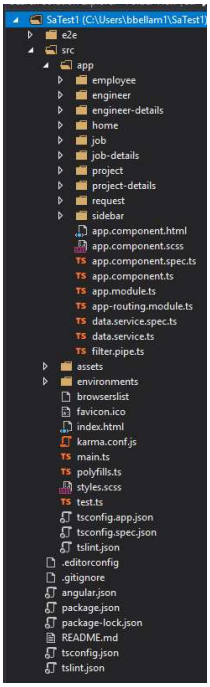
1. SA QAS WEBSITE: SPECIFIC FILE DESCRIPTIONS, SCREEN VIEWS & PSEUDOCODE

- Information as of 9/27/18
- All pages are shown the way the System Administrator would see it.
- Angular files written in SaTest1 Project, accessible through Microsoft Visual Studios and Node.js

COMPONENTS

- Component folders consist of a typescript file (.ts), HTML, CSS (or SCSS) and a test file (.spec.ts)
- Each visible QAS page is a component.
    - Welcome page
    - Add Employee: Configuration File
    - Search Engineer
    - Add Project: Configuration File
    - Job Request Form
    - Search Job Request
- Pages with a search function are linked to a component that shows the results
    - Search Engineer component is linked to Engineer Details component
    - Search Job Request component is linked to Job Details Component
- Sidebar is also a component, but it will be discussed in the “Other Components and Services” section

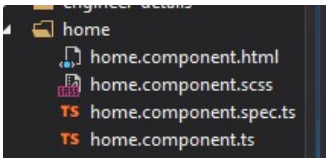
Screen Shot: All Component Files



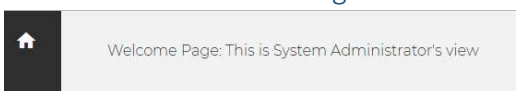
PAGE 1: WELCOME PAGE

- Only has default HTML at the moment.
- Want to have square boxes (similar to the format of Canvas student website)
    - Each box will be linked to page (similar to sidepanel links)
    - Each box will have a **bold** title and synopsis of each page (mini instructions to user)

Screenshot: Welcome Page Component Files



Screen View: Welcome Page

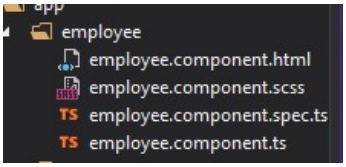


PAGE 2: ADD EMPLOYEE CONFIGURATION FILE

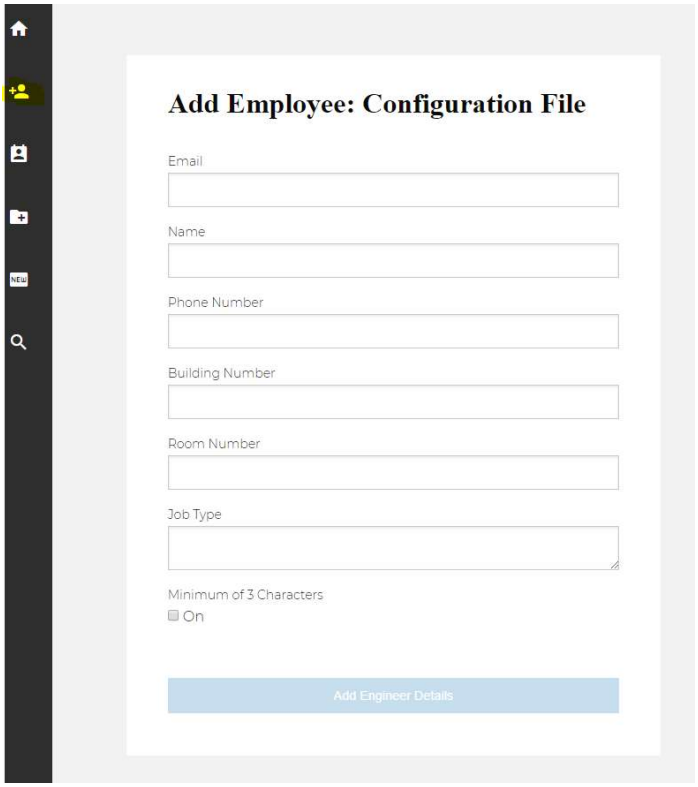
- This is where the System Admin would add new engineers/employees to the database
- When they input information and click “Add Engineer Details” button, a synopsis page is shown
- Need to find a way to link it to a SQL database through C#

- Want to create a button that gives System Admin the ability to add more “Job Type” rows
- The optional checkbox can change the minimum amount of characters required to enter into a textbox
- Want the name, phone number, building number, and room number to auto-populate when the email address is entered
-

Screenshot: Add Employee Component Files



Screen View: Add Employee Page



employee.component.ts

i. **Values used - name, email, phone, building, room, job**

ii. **Import**

```
import { FormBuilder, FormGroup, Validators } from '@angular/forms'; // added
for employee configuration file reactive form
```

iii. **export class EmployeeComponent implements OnInit**

```
  rForm: FormGroup;              //rForm is short for Reactive Form
```

```
  post: any;                      // A property for our submitted form
  name: string = '';              //entering the employee name
  titleAlert: string = 'This field is required'; //alert if nothing is entered
in textbox
  email: string = '';
  phone: string = '';
  building: string = '';
  room: string = '';
  job: string = '';
```

iv. `constructor(private fb: FormBuilder)`

```
  'name': [null, Validators.required],
  'validate': ''  ,  // needed for the checkbox
  'email': [null, Validators.required],
  'phone': [null, Validators.required],
  'building': [null, Validators.required],
  'room': [null, Validators.required],
  'job': [null, Validators.compose([Validators.required,
Validators.minLength(30), Validators.maxLength(500)])],
```

v. `addPost(post)`

```
  this.name = post.name;
  this.email = post.email;
  this.phone = post.phone;
  this.building = post.building;
  this.room = post.room;
  this.job = post.job;
```

employee.component.html

vi. Labels (match values above) - email, name, phone, building, room, job

```
  <label>
    Email <!--new value-->
    <input type="text" formControlName="email">
  </label>
  <div class="alert" *ngIf="!rForm.controls['email'].valid &&
rForm.controls['email'].touched">{{ titleAlert }}</div>
```

<. . . other labels . . . >

```
  <label>
     Job Type  <!--new value-->
     <textarea formControlName="job"></textarea>
  </label>
  <div class="alert" *ngIf="!rForm.controls['job'].valid &&
rForm.controls['job'].touched">You must specify a description that's between 30
and 500 characters.</div>
```

vii. After submit button is clicked `<ng-template #forminfo>`

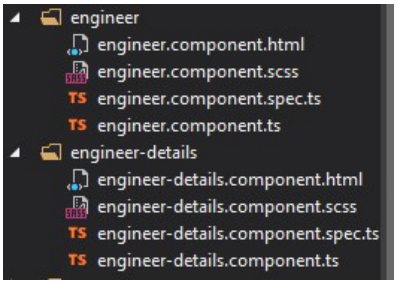
```
  <h1>{{ name }}</h1>
```

```
<p>
 {{ email }}    <br>
 {{ phone }}   <br>
 {{ building }}<br>
 {{ room }}    <br>
 {{ job }}     <br>
 {{ description }}
 </p>
```

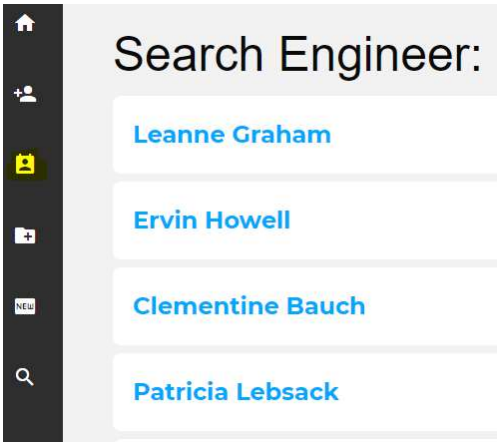
PAGE 3: SEARCH ENGINEER

- This is where the System Admin can search for engineers in the database
- List of current engineers in the system
- want to have a search box where they System Admin can search by name

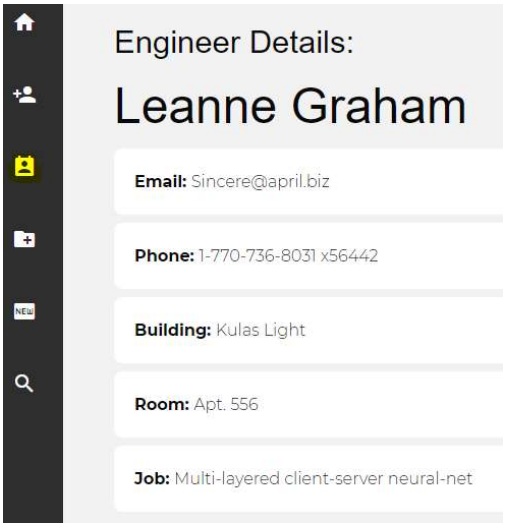
Screen Shot: Search Engineer Component Files



Screen View: Search Employee Page



Screen View: Search Employee Details Page



engineer.component.ts

1. **Values used - engineers$**

2. **Import**

```
import { DataService } from '../data.service'; //added
import { Observable } from 'rxjs'; //added
```

3. **export class EngineerComponent implements OnInit**

```
  engineers$: Object; //to display results, import the Observable
```

4. constructor(private data: DataService)

5. **ngOnInit()**

```
    this.data.getEngineers().subscribe(
      data => this.engineers$ = data
```

engineer.component.html

1. **Iteration over an Array - *ngFor**

```
<li *ngFor="let engineer of engineers$">
```

2. **Interpolation {{…}}**

```
<a routerLink="/engineer-details/{{engineer.id}}">{{ engineer.name }}</a>
```

engineer-details.component.ts

1. **Values used - engineer$** // singular "engineer", no "s"

2. **Import**

```
import { DataService } from '../data.service';
import { Observable } from 'rxjs';
import { ActivatedRoute } from "@angular/router"; // added for the router
```

3. **export class EngineerDetailsComponent implements OnInit**

```
engineer$: Object; // singular "engineer", no "s"
```

4. **constructor(private route: ActivatedRoute, private data: DataService)**

```
this.route.params.subscribe(params => this.engineer$ = params.id);
```

5. **ngOnInit() // loads data when person clicks name**

```
this.data.getEngineer(this.engineer$).subscribe(
    data => this.engineer$ = data
```

engineer-details.component.html

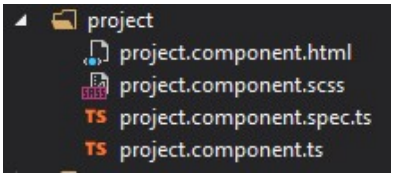
1. **Interpolation {{…}}**

```
<h3>Engineer Details: <br /></h3>
<h1>{{ engineer$.name }}</h1>

<li><strong>Email: </strong>{{ engineer$.email }}</li>
<li><strong>Phone: </strong>{{ engineer$.phone }}</li>
<li><strong>Building: </strong>{{ engineer$.address.street }}</li>
<li><strong>Room: </strong>{{ engineer$.address.suite }}</li>
<li><strong>Job: </strong>{{ engineer$.company.catchPhrase }}</li>
```

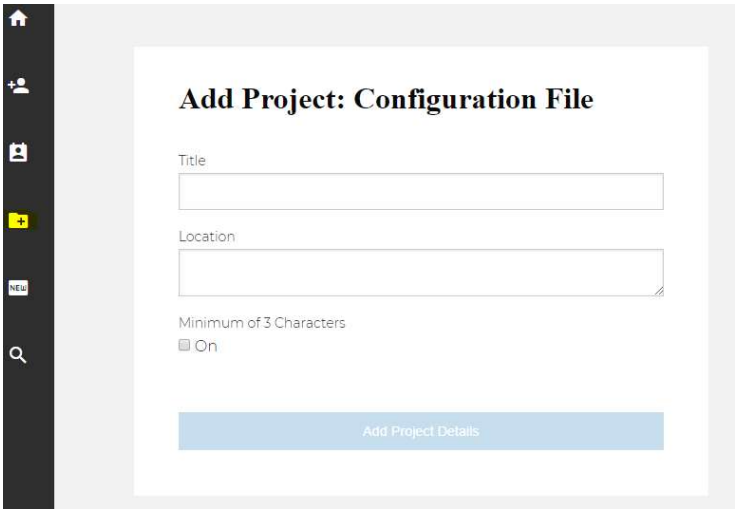
PAGE 4: ADD PROJECT CONFIGURATION FILE

- This is where the System Admin would add new projects to the database
- When they input information and click “Add Project Details” button, a synopsis page is shown
- Need to find a way to link it to a SQL database through C#
- Want to create a button that gives System Admin the ability to add more “Location” rows
- The optional checkbox can change the minimum amount of characters required to enter into a textbox

Screen Shot: Add Project Component Files



Screen View: Add Project Page



project.component.ts

1. **Values used - title, location**

2. **Import**

```
import { FormBuilder, FormGroup, Validators } from '@angular/forms'; // added
for employee configuration file reactive form
```

3. **export class ProjectComponent implements OnInit**

```
rForm: FormGroup;                //rForm is short for Reactive Form
post: any;                       // A property for our submitted form
title: string = '';               //entering the project title
titleAlert: string = 'This field is required'; //alert if nothing is entered
in textbox
location: string = '';
```

4. **constructor(private fb: FormBuilder)**

```
this.rForm = fb.group

'title': [null, Validators.required],
'validate': '',  // needed for the checkbox
'location': [null, Validators.compose([Validators.required,
Validators.minLength(30), Validators.maxLength(500)])],
```

5. **addPost(post)**

```
this.title = post.title;
this.location = post.location;
```

project.component.html - values

1. **Labels (match components above) - two types**

```
<label>
          Title
          <input type="text" formControlName="title">
        </label>
        <div class="alert" *ngIf="!rForm.controls['title'].valid &&
rForm.controls['title'].touched">{{ titleAlert }}</div>


        <label>
          Location  <!--new value-->
          <textarea formControlName="location"></textarea>
        </label>
        <div class="alert" *ngIf="!rForm.controls['location'].valid &&
rForm.controls['location'].touched">You must specify a description that's
between 30 and 500 characters.</div>
```

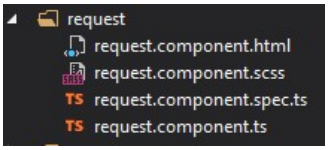
2. **After submit button is clicked `<ng-template #forminfo>`**

```
<h1>{{ title }}</h1>
<p> {{ location }}</p>
```

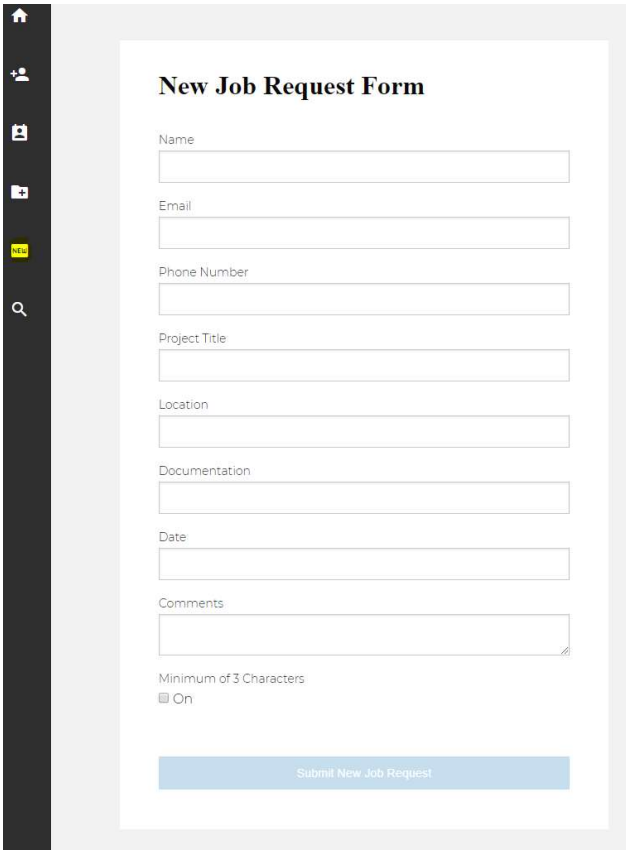
PAGE 5: NEW JOB REQUEST FORM

- This is where **Any Stakeholder** would add new job requests to the database
- When they input information and click “Submit New Job Request” button, a synopsis page is shown
- Need to find a way to link it to a SQL database through C#
- Want to create a way to upload documents in the “Documentation” box
- Need to find a way to make a calendar drop down from “Date” so the stakeholder can choose the current day or have it auto-populate today’s date
- Want the name and phone number to auto-populate when the email address is entered
- The optional checkbox can change the minimum amount of characters required to enter into a textbox

Screen Shot: New Job Request Form Component Files



Screen View: New Job Request Form Page



request.component.ts

1. **Values used - name, email, phone, project, location, documentation, date, comments**

2. **Import**

```
import { FormBuilder, FormGroup, Validators } from '@angular/forms'; // added
for job request reactive form
```

3. **`export class RequestComponent implements OnInit`**

```
        rForm: FormGroup;                   //rForm is short for Reactive Form
        post: any;                          // A property for our submitted form
        name: string = '';                  //entering the employee name
        titleAlert: string = 'This field is required'; //alert if nothing is
entered in textbox
        email: string = '';
        phone: string = '';
        project: string = '';
        location: string = ''; //needs to be a dropdown menu, populated by
locations specified in the project configuration file
        documentation: string = ''; //need to be able to upload document
        date: string = ''; //need to change from string to datetime
        comments: string = '';
```

4. **`constructor(private fb: FormBuilder)`**

```
      'name': [null, Validators.required],
      'validate': '',  // needed for the checkbox
      'email': [null, Validators.required],
      'phone': [null, Validators.required],
      //new values added
      'project': [null, Validators.required],
      'location': [null, Validators.required],
      'documentation': [null, Validators.required],
      'date': [null, Validators.required],
      'comments': [null, Validators.compose([Validators.required,
Validators.minLength(30), Validators.maxLength(500)])],
```

5. **addPost(post)**

```
    this.name = post.name;
    this.email = post.email;
    this.phone = post.phone;
    this.project = post.project;
    this.location = post.location;
    this.documentation = post.documentation;
    this.date = post.date;
    this.comments = post.comments;
```

request.component.html

1. **Labels (match values above) - name, email, phone, project, location, documentation, date, comments**

```
        <label>
          Project Title <!--new value-->
          <input type="text" formControlName="project">
        </label>
        <div class="alert" *ngIf="!rForm.controls['project'].valid &&
rForm.controls['project'].touched">{{ titleAlert }}</div>

<... other labels. . .>

       <label>
          Comments  <!--new value-->
          <textarea formControlName="comments"></textarea>
        </label>
        <div class="alert" *ngIf="!rForm.controls['comments'].valid &&
rForm.controls['comments'].touched">You must specify a description that's between
30 and 500 characters.</div>
```

2. **After submit button is clicked `<ng-template #forminfo>`**

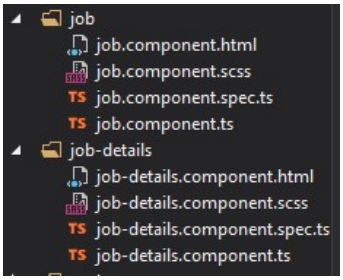
```
    <h1>{{ name }}</h1>
    <p>
      {{ email }}    <br>
      {{ phone }}   <br>
      {{ project }}<br>
      {{ location }}    <br>
      {{ documentation }}     <br>
      {{ date }} <br>
```

```
        {{ comments }}
          </p>
```

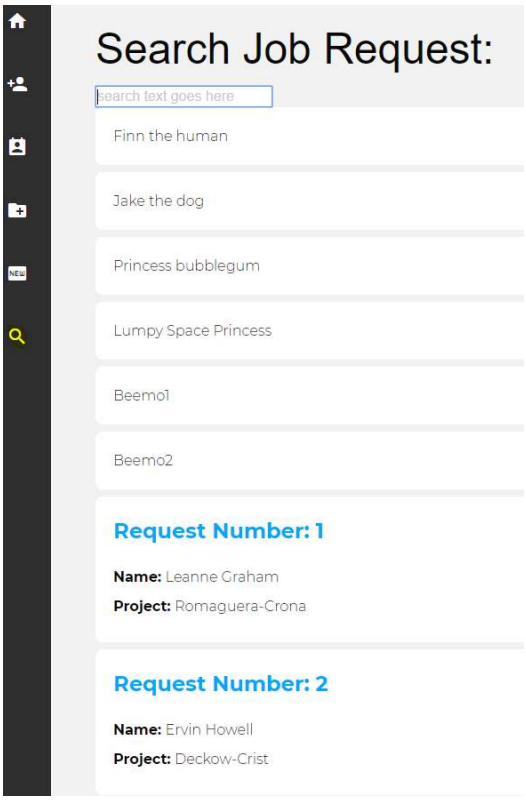
PAGE 6: SEARCH JOB REQUEST FORM

- This is where **Any Stakeholder** can search for job requests in the database
- List of current job requests in the system
- want to have a search box where the stakeholder can search by request number, name, or project
- tried to make a search box, but need to find a way to link it with an external database

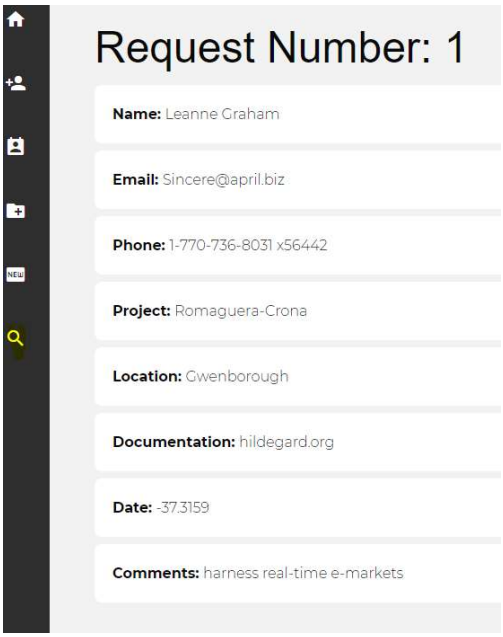
Screen Shot: Search Job Request Form Component Files



Screen View: Search Job Request Form Page



Screen View: Search Job Request Form Details Page



job.component.ts

1. **Values used - jobs$**

2. **Import**

```
import { DataService } from '../data.service'; //added
import { Observable } from 'rxjs'; //added
```

3. **export class JobComponent implements OnInit**

```
jobs$: Object; //to display results, import the Observable

//START TEST FOR SEARCH PIPE

name = 'Angular';

characters = [ // dummy data that is searchable finding a better way to
search data
    'Finn the human',
    'Jake the dog',
    'Princess bubblegum',
    'Lumpy Space Princess',
    'Beemo1',
    'Beemo2'

//END TEST FOR SEARCH PIPE
```

4. **constructor(private data: DataService)**

5. **ngOnInit()**

```
this.data.getJobs().subscribe(
  data => this.jobs$ = data
```

job.component.html

1. **Search Pipe**

```
<input [(ngModel)]="searchText" placeholder="search text goes here">

  <li *ngFor="let c of characters | filter : searchText">
    {{c}}
```

2. **Iteration over an Array - *ngFor**

```
<li *ngFor="let job of jobs$">
```

3. **Interpolation {{…}}**

```
<a routerLink="/job-details/{{job.id}}">Request Number: {{ job.id }}</a>
```

job-details.component.ts

1. **Values used - job$** `// singular "job", no "s"`

2. **Import**

```
import { DataService } from '../data.service';
import { Observable } from 'rxjs';
import { ActivatedRoute } from "@angular/router"; // added for the router
```

3. **`export class JobDetailsComponent implements OnInit`**

```
job$: Object; // singular "engineer", no "s"
```

4. **`constructor(private route: ActivatedRoute, private data: DataService)`**

```
this.route.params.subscribe(params => this.job$ = params.id);
```

5. **`ngOnInit()  // loads data when person clicks name`**

```
    this.data.getJob(this.job$).subscribe(
         data => this.job$ = data
```

job-details.component.html

1. **Interpolation {{…}}**

```
<h1>Request Number: {{ job$.id }}</h1>

 <li><strong>Name: </strong>{{ job$.name }}</li>
<li><strong>Email: </strong>{{ job$.email }}</li>
<li><strong>Phone: </strong>{{ job$.phone }}</li>
<li><strong>Project: </strong>{{ job$.company.name }}</li>
<li><strong>Location: </strong>{{ job$.address.city }}</li>
<li><strong>Documentation: </strong>{{ job$.website }}</li>
<li><strong>Date: </strong>{{ job$.address.geo.lat }}</li>
```

```
        <li><strong>Comments: </strong>{{ job$.company.bs }}</li>
```

2. COMPLETE ANGULAR SA QAS CODE BEFORE ADDING IT TO ASP.NET

Employee

employee.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms'; // added for
employee configuration file reactive form

@Component({
  selector: 'app-employee',
  templateUrl: './employee.component.html',
  styleUrls: ['./employee.component.scss']
})
export class EmployeeComponent implements OnInit {

  rForm: FormGroup;               //rForm is short for Reactive Form
  post: any;                      // A property for our submitted form
  name: string = '';              //entering the employee name
  titleAlert: string = 'This field is required'; //alert if nothing is entered in textbox
  email: string = '';
  phone: string = '';
  building: string = '';
  room: string = '';
  job: string = '';

  constructor(private fb: FormBuilder) {   //dependancy injection

    this.rForm = fb.group({
      'name': [null, Validators.required],
      'validate': ''   ,  // needed for the checkbox
      'email': [null, Validators.required],
      'phone': [null, Validators.required],
      'building': [null, Validators.required],
      'room': [null, Validators.required],
      'job': [null, Validators.compose([Validators.required, Validators.minLength(30),
Validators.maxLength(500)])],
    });

  }

  addPost(post) {  //handles the submitted form, where you would store back API
    this.name = post.name;
    this.email = post.email;
    this.phone = post.phone;
```

```
    this.building = post.building;
    this.room = post.room;
    this.job = post.job;
  }

  ngOnInit() { //lifecycle hook, used for the checkbox

    this.rForm.get('validate').valueChanges.subscribe(
      (validate) => {
        if (validate == '1') {
          this.rForm.get('name').setValidators([Validators.required,
Validators.minLength(3)])
          this.titleAlert = "You need to specify at least 3 characters";
        } else {
          this.rForm.get('name').setValidators(Validators.required);
        }
        this.rForm.get('name').updateValueAndValidity();
      }
    )
  }

}
```

employee.component.html

```
<div *ngIf="!name; else forminfo">
  <form [formGroup]="rForm" (ngSubmit)="addPost(rForm.value)">
    <div class="form-container">
      <div class="row columns">
        <!--Info for new employee form-->
        <h1>Add Employee: Configuration File</h1>

        <!--Each label has to match component name in employee.component.ts -->

        <label>
          Email <!--new value-->
          <input type="text" formControlName="email">
        </label>
        <div class="alert" *ngIf="!rForm.controls['email'].valid &&
rForm.controls['email'].touched">{{ titleAlert }}</div>

        <label>
          Name
          <input type="text" formControlName="name">
        </label>
        <div class="alert" *ngIf="!rForm.controls['name'].valid &&
rForm.controls['name'].touched">{{ titleAlert }}</div>

        <label>
          Phone Number <!--new value-->
          <input type="text" formControlName="phone">
        </label>
        <div class="alert" *ngIf="!rForm.controls['phone'].valid &&
rForm.controls['phone'].touched">{{ titleAlert }}</div>
```

```
        <label>
          Building Number <!--new value-->
          <input type="text" formControlName="building">
        </label>
        <div class="alert" *ngIf="!rForm.controls['building'].valid &&
rForm.controls['building'].touched">{{ titleAlert }}</div>

        <label>
          Room Number <!--new value-->
          <input type="text" formControlName="room">
        </label>
        <div class="alert" *ngIf="!rForm.controls['room'].valid &&
rForm.controls['room'].touched">{{ titleAlert }}</div>

        <label>
          Job Type  <!--new value-->
          <textarea formControlName="job"></textarea>
        </label>
        <div class="alert" *ngIf="!rForm.controls['job'].valid &&
rForm.controls['job'].touched">You must specify a description that's between 30 and 500
characters.</div>


        <!--checkbox-->
        <label for="validate">Minimum of 3 Characters</label>
        <input type="checkbox" name="validate" formControlName="validate" value="1"> On

        <input type="submit" class="button expanded" value="Add Engineer Details"
[disabled]="!rForm.valid">
      </div>
    </div>
  </form>
</div>

<ng-template #forminfo> <!--page that appears after submit button is clicked-->
  <div class="form-container">
    <div class="row columns">

      <h1>{{ name }}</h1>
      <p>
        {{ email }}    <br>
        {{ phone }}   <br>
        {{ building }}<br>
        {{ room }}    <br>
        {{ job }}     <br>
        {{ description }}
      </p>

    </div>
  </div>
</ng-template>
```

employee.component.scss

```
.form-container {
```

```
  display: block;
  width: 90%;
  padding: 2em;
  margin: 2em auto;
  background: #fff;
}

.alert {
  background: #f2edda;
  padding: 7px;
  font-size: .9em;
  margin-bottom: 20px;
  display: inline-block;
  animation: 2s alertAnim forwards;
}

.button {
  margin-top: 3rem;
}

h1 {
  margin-bottom: 2rem;
  font-weight: bold;
  font-family: 'Muli';
  font-size: 2em;
}


@keyframes alertAnim {
  from {
    opacity: 0;
    transform: translateY(-20px);
  }

  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

employee.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { EmployeeComponent } from './employee.component';

describe('EmployeeComponent', () => {
  let component: EmployeeComponent;
  let fixture: ComponentFixture<EmployeeComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ EmployeeComponent ]
```

```
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(EmployeeComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Engineer

engineer.component.ts

```
import { Component, OnInit } from '@angular/core';
import { DataService } from '../data.service'; //added
import { Observable } from 'rxjs'; //added

@Component({
  selector: 'app-engineer',
  templateUrl: './engineer.component.html',
  styleUrls: ['./engineer.component.scss']
})
export class EngineerComponent implements OnInit {

  engineers$: Object; //to display results, import the Observable

  constructor(private data: DataService) { }

  ngOnInit() {
    this.data.getEngineers().subscribe(
      data => this.engineers$ = data
    );
  }

}
```

engineer.component.html

```
<h1>Search Engineer:</h1>

<ul>
  <li *ngFor="let engineer of engineers$">
    <a routerLink="/engineer-details/{{engineer.id}}">{{ engineer.name }}</a>
```

```
  </li>
</ul>
```

engineer.component.scss

empty

engineer.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { EngineerComponent } from './engineer.component';

describe('EngineerComponent', () => {
  let component: EngineerComponent;
  let fixture: ComponentFixture<EngineerComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ EngineerComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(EngineerComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Engineer Details

engineer-details.component.ts

```
import { Component, OnInit } from '@angular/core';
import { DataService } from '../data.service';
import { Observable } from 'rxjs';
```

```
import { ActivatedRoute } from "@angular/router"; // added for the router

@Component({
  selector: 'app-engineer-details',
  templateUrl: './engineer-details.component.html',
  styleUrls: ['./engineer-details.component.scss']
})
export class EngineerDetailsComponent implements OnInit {

  engineer$: Object; // singular "engineer", no "s"

constructor(private route: ActivatedRoute, private data: DataService) {
  this.route.params.subscribe(params => this.engineer$ = params.id);
}

  ngOnInit() { // loads data when person clicks name
    this.data.getEngineer(this.engineer$).subscribe(
      data => this.engineer$ = data
    );
  }

}
```

engineer-details.component.html

```
<h3>Engineer Details: <br /></h3>
<h1>{{ engineer$.name }}</h1>

<ul>
    <li><strong>Email: </strong>{{ engineer$.email }}</li>
    <li><strong>Phone: </strong>{{ engineer$.phone }}</li>
    <li><strong>Building: </strong>{{ engineer$.address.street }}</li>
    <li><strong>Room: </strong>{{ engineer$.address.suite }}</li>
    <li><strong>Job: </strong>{{ engineer$.company.catchPhrase }}</li>

</ul>
```

engineer-details.component.scss

empty

engineer-details.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { EngineerDetailsComponent } from './engineer-details.component';
```

```
describe('EngineerDetailsComponent', () => {
  let component: EngineerDetailsComponent;
  let fixture: ComponentFixture<EngineerDetailsComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ EngineerDetailsComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(EngineerDetailsComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Home (Not added to ASP.NET)

home.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

home.component.html

```
<p>
  Welcome Page: This is System Administrator's view
</p>
```

home.component.scss

empty

home.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { HomeComponent } from './home.component';

describe('HomeComponent', () => {
  let component: HomeComponent;
  let fixture: ComponentFixture<HomeComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ HomeComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(HomeComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Job

job.component.ts

```
import { Component, OnInit } from '@angular/core';
import { DataService } from '../data.service'; //added
import { Observable } from 'rxjs'; //added

@Component({
  selector: 'app-job',
  templateUrl: './job.component.html',
  styleUrls: ['./job.component.scss']
})
export class JobComponent implements OnInit {
```

```
  jobs$: Object; //to display results, import the Observable

  //START TEST FOR SEARCH PIPE https://codeburst.io/create-a-search-pipe-to-dynamically-
filter-results-with-angular-4-21fd3a5bec5c
  name = 'Angular';

  characters = [
    'Finn the human',
    'Jake the dog',
    'Princess bubblegum',
    'Lumpy Space Princess',
    'Beemo1',
    'Beemo2'

  ]

  //END TEST FOR SEARCH PIPE

  constructor(private data: DataService) { }

  ngOnInit() {
    this.data.getJobs().subscribe(
      data => this.jobs$ = data
    );

    //START TEST FOR SEARCH PIPE https://codeburst.io/create-a-search-pipe-to-
dynamically-filter-results-with-angular-4-21fd3a5bec5c
    //this.jobs = ['https://jsonplaceholder.typicode.com/users/'];
      //END TEST FOR SEARCH PIPE

  }

}
```

job.component.html

```
<h1>Search Job Request:</h1>

<!--  START TEST FOR SEARCH PIPE https://codeburst.io/create-a-search-pipe-to-
dynamically-filter-results-with-angular-4-21fd3a5bec5c-->
<input [(ngModel)]="searchText" placeholder="search text goes here">
<ul>
  <li *ngFor="let c of characters | filter : searchText">
    {{c}}
  </li>
</ul>
<!--  END TEST FOR SEARCH PIPE-->
<!--<input [(ngModel)]="searchText" placeholder="search text goes here"> -->
<ul>
  <li *ngFor="let job of jobs$">
    <a routerLink="/job-details/{{job.id}}">Request Number: {{ job.id }}</a>
```

```
    <ul>
      <li><strong>Name: </strong>{{ job.name }}</li>
      <li><strong>Project: </strong>{{ job.company.name }}</li>
    </ul>
  </li>
</ul>
```

job.component.scss

empty

job.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { JobComponent } from './job.component';

describe('JobComponent', () => {
  let component: JobComponent;
  let fixture: ComponentFixture<JobComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ JobComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(JobComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Job Details

job-details.component.ts

```
import { Component, OnInit } from '@angular/core';
import { DataService } from '../data.service';
import { Observable } from 'rxjs';
```

```
import { ActivatedRoute } from "@angular/router";

@Component({
  selector: 'app-job-details',
  templateUrl: './job-details.component.html',
  styleUrls: ['./job-details.component.scss']
})
export class JobDetailsComponent implements OnInit {

  job$: Object;

  constructor(private route: ActivatedRoute, private data: DataService) {
    this.route.params.subscribe(params => this.job$ = params.id);
  }

  ngOnInit() {
    this.data.getJob(this.job$).subscribe(
      data => this.job$ = data
    );
  }

}
```

job-details.component.html

```
<h1>Request Number: {{ job$.id }}</h1>

<ul>

  <li><strong>Name: </strong>{{ job$.name }}</li>
  <li><strong>Email: </strong>{{ job$.email }}</li>
  <li><strong>Phone: </strong>{{ job$.phone }}</li>
  <li><strong>Project: </strong>{{ job$.company.name }}</li>
  <li><strong>Location: </strong>{{ job$.address.city }}</li>
  <li><strong>Documentation: </strong>{{ job$.website }}</li>
  <li><strong>Date: </strong>{{ job$.address.geo.lat }}</li>
  <li><strong>Comments: </strong>{{ job$.company.bs }}</li>

</ul>
```

job-details.component.scss

empty

job-details.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { JobDetailsComponent } from './job-details.component';
```

```
describe('JobDetailsComponent', () => {
  let component: JobDetailsComponent;
  let fixture: ComponentFixture<JobDetailsComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ JobDetailsComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(JobDetailsComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Project

project.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms'; // added for
employee configuration file reactive form


@Component({
  selector: 'app-project',
  templateUrl: './project.component.html',
  styleUrls: ['./project.component.scss']
})
export class ProjectComponent implements OnInit {

  rForm: FormGroup;               //rForm is short for Reactive Form
  post: any;                      // A property for our submitted form
  title: string = '';              //entering the project title
  titleAlert: string = 'This field is required'; //alert if nothing is entered in textbox
  location: string = '';

  constructor(private fb: FormBuilder) {   //dependency injection

    this.rForm = fb.group({
      'title': [null, Validators.required],
      'validate': '',  // needed for the checkbox
      'location': [null, Validators.compose([Validators.required,
Validators.minLength(30), Validators.maxLength(500)])],
    });
```

```
  }

  addPost(post) {  //handles the submitted form, where you would store back API

    this.title = post.title;
    this.location = post.location;
  }

  ngOnInit() { //lifecycle hook, used for the checkbox

    this.rForm.get('validate').valueChanges.subscribe(
      (validate) => {
        if (validate == '1') {
          this.rForm.get('name').setValidators([Validators.required,
Validators.minLength(3)])
          this.titleAlert = "You need to specify at least 3 characters";
        } else {
          this.rForm.get('name').setValidators(Validators.required);
        }
        this.rForm.get('name').updateValueAndValidity();
      }
    )


  }

}
```

project.component.html

```
<div *ngIf="!name; else forminfo">
  <form [formGroup]="rForm" (ngSubmit)="addPost(rForm.value)">
    <div class="form-container">
      <div class="row columns">
        <!--Info for new employee form-->
        <h1>Add Project: Configuration File</h1>

        <!--Each label has to match component name in employee.component.ts -->


        <label>
          Title
          <input type="text" formControlName="title">
        </label>
        <div class="alert" *ngIf="!rForm.controls['title'].valid &&
rForm.controls['title'].touched">{{ titleAlert }}</div>



        <label>
          Location  <!--new value-->
          <textarea formControlName="location"></textarea>
        </label>
```

```
        <div class="alert" *ngIf="!rForm.controls['location'].valid &&
rForm.controls['location'].touched">You must specify a description that's between 30 and
500 characters.</div>


        <!--checkbox-->
        <label for="validate">Minimum of 3 Characters</label>
        <input type="checkbox" name="validate" formControlName="validate" value="1"> On

        <input type="submit" class="button expanded" value="Add Project Details"
[disabled]="!rForm.valid">
      </div>
    </div>
  </form>
</div>

<ng-template #forminfo>
  <!--page that appears after submit button is clicked-->
  <div class="form-container">
    <div class="row columns">
      <h1>{{ title }}</h1>

      <p> {{ location }}</p>

    </div>
  </div>
</ng-template>
```

project.component.scss

```
.form-container {
  display: block;
  width: 90%;
  padding: 2em;
  margin: 2em auto;
  background: #fff;
}

.alert {
  background: #f2edda;
  padding: 7px;
  font-size: .9em;
  margin-bottom: 20px;
  display: inline-block;
  animation: 2s alertAnim forwards;
}

.button {
  margin-top: 3rem;
}

h1 {
  margin-bottom: 2rem;
```

```
  font-weight: bold;
  font-family: 'Muli';
  font-size: 2em;
}


@keyframes alertAnim {
  from {
    opacity: 0;
    transform: translateY(-20px);
  }

  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

project.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { ProjectComponent } from './project.component';

describe('ProjectComponent', () => {
  let component: ProjectComponent;
  let fixture: ComponentFixture<ProjectComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ ProjectComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(ProjectComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Project Details

project-details.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-project-details',
  templateUrl: './project-details.component.html',
  styleUrls: ['./project-details.component.scss']
})
export class ProjectDetailsComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

project-details.component.html

```
<p>
  project-details works!
</p>
```

project-details.component.scss

empty

project-details.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { ProjectDetailsComponent } from './project-details.component';

describe('ProjectDetailsComponent', () => {
  let component: ProjectDetailsComponent;
  let fixture: ComponentFixture<ProjectDetailsComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ ProjectDetailsComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(ProjectDetailsComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
```

```
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Request

request.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms'; // added for job
request reactive form

@Component({
  selector: 'app-request',
  templateUrl: './request.component.html',
  styleUrls: ['./request.component.scss']
})
export class RequestComponent implements OnInit {

  rForm: FormGroup;               //rForm is short for Reactive Form
  post: any;                      // A property for our submitted form
  name: string = '';              //entering the employee name
  titleAlert: string = 'This field is required'; //alert if nothing is entered in textbox
  email: string = '';
  phone: string = '';
  // new values added
  project: string = '';
  location: string = ''; //needs to be a dropdown menu, populated by locations specified
in the project configuration file
  documentation: string = ''; //need to be able to upload document
  date: string = ''; //need to change from string to datetime
  comments: string = '';



  constructor(private fb: FormBuilder) {   //dependency injection
    this.rForm = fb.group({
      'name': [null, Validators.required],
      'validate': '',  // needed for the checkbox
      'email': [null, Validators.required],
      'phone': [null, Validators.required],
     //new values added
      'project': [null, Validators.required],
      'location': [null, Validators.required],
      'documentation': [null, Validators.required],
      'date': [null, Validators.required],
      'comments': [null, Validators.compose([Validators.required,
Validators.minLength(30), Validators.maxLength(500)])],
    });
  }
```

```
  addPost(post) {  //handles the submitted form, where you would store back API

    this.name = post.name;
    this.email = post.email;
    this.phone = post.phone;
    this.project = post.project;
    this.location = post.location;
    this.documentation = post.documentation;
    this.date = post.date;
    this.comments = post.comments;
  }

  ngOnInit() { //lifecycle hook, used for the checkbox

    this.rForm.get('validate').valueChanges.subscribe(
      (validate) => {
        if (validate == '1') {
          this.rForm.get('name').setValidators([Validators.required,
Validators.minLength(3)])
          this.titleAlert = "You need to specify at least 3 characters";
        } else {
          this.rForm.get('name').setValidators(Validators.required);
        }
        this.rForm.get('name').updateValueAndValidity();
      }
    )

  }

}
```

request.component.html

```
<div *ngIf="!name; else forminfo">
  <form [formGroup]="rForm" (ngSubmit)="addPost(rForm.value)">
    <div class="form-container">
      <div class="row columns">
        <!--Info for new job request form-->
        <h1>New Job Request Form</h1>

        <!--Each label has to match component name in request.component.ts -->

        <label>
          Name
          <input type="text" formControlName="name">
        </label>
        <div class="alert" *ngIf="!rForm.controls['name'].valid &&
rForm.controls['name'].touched">{{ titleAlert }}</div>


        <label>
```

```
          Email
          <input type="text" formControlName="email">
        </label>
        <div class="alert" *ngIf="!rForm.controls['email'].valid &&
rForm.controls['email'].touched">{{ titleAlert }}</div>


        <label>
          Phone Number
          <input type="text" formControlName="phone">
        </label>
        <div class="alert" *ngIf="!rForm.controls['phone'].valid &&
rForm.controls['phone'].touched">{{ titleAlert }}</div>

        <label>
          Project Title <!--new value-->
          <input type="text" formControlName="project">
        </label>
        <div class="alert" *ngIf="!rForm.controls['project'].valid &&
rForm.controls['project'].touched">{{ titleAlert }}</div>

        <label>
          Location <!--new value-->
          <input type="text" formControlName="location">
        </label>
        <div class="alert" *ngIf="!rForm.controls['location'].valid &&
rForm.controls['location'].touched">{{ titleAlert }}</div>

        <label>
          Documentation <!--new value-->
          <input type="text" formControlName="documentation">
        </label>
        <div class="alert" *ngIf="!rForm.controls['documentation'].valid &&
rForm.controls['documentation'].touched">{{ titleAlert }}</div>

        <label>
          Date <!--new value-->
          <input type="text" formControlName="date">
        </label>
        <div class="alert" *ngIf="!rForm.controls['date'].valid &&
rForm.controls['date'].touched">{{ titleAlert }}</div>


        <label>
          Comments  <!--new value-->
          <textarea formControlName="comments"></textarea>
        </label>
        <div class="alert" *ngIf="!rForm.controls['comments'].valid &&
rForm.controls['comments'].touched">You must specify a description that's between 30 and
500 characters.</div>



        <!--checkbox-->
        <label for="validate">Minimum of 3 Characters</label>
        <input type="checkbox" name="validate" formControlName="validate" value="1"> On
```

```
        <input type="submit" class="button expanded" value="Submit New Job Request"
[disabled]="!rForm.valid">
      </div>
    </div>
  </form>
</div>

<ng-template #forminfo>
  <!--page that appears after submit button is clicked-->
  <div class="form-container">
    <div class="row columns">

      <h1>{{ name }}</h1>
      <p>
        {{ email }}    <br>
        {{ phone }}   <br>
        {{ project }}<br>
        {{ location }}    <br>
        {{ documentation }}     <br>
        {{ date }} <br>
        {{ comments }}
      </p>

    </div>
  </div>
</ng-template>
```

request.component.scss

```
.form-container {
  display: block;
  width: 90%;
  padding: 2em;
  margin: 2em auto;
  background: #fff;
}

.alert {
  background: #f2edda;
  padding: 7px;
  font-size: .9em;
  margin-bottom: 20px;
  display: inline-block;
  animation: 2s alertAnim forwards;
}

.button {
  margin-top: 3rem;
}

h1 {
  margin-bottom: 2rem;
  font-weight: bold;
  font-family: 'Muli';
```

```
  font-size: 2em;
}


@keyframes alertAnim {
  from {
    opacity: 0;
    transform: translateY(-20px);
  }

  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

request.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { RequestComponent } from './request.component';

describe('RequestComponent', () => {
  let component: RequestComponent;
  let fixture: ComponentFixture<RequestComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ RequestComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(RequestComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Sidebar (Not added to ASP.NET)

sidebar.component.ts

```
import { Component, OnInit } from '@angular/core';
```

```
@Component({
  selector: 'app-sidebar',
  templateUrl: './sidebar.component.html',
  styleUrls: ['./sidebar.component.scss']
})
export class SidebarComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

sidebar.component.html

```
<nav>
  <ul>
    <li>
      <a routerLink="" [class.activated]="currentUrl == '/'">
        <i class="material-icons">home</i> <!--Welcome Page-->
      </a>
    </li>

    <li>
      <a routerLink="employee" [class.activated]="currentUrl == '/employee'">
        <i class="material-icons">person_add</i> <!--Employee Configuration File-->
      </a>
    </li>

    <li>
      <a routerLink="engineer" [class.activated]="currentUrl == '/engineer'">
        <i class="material-icons">perm_contact_calendar</i> <!--Engineer List-->
      </a>
    </li>

    <li>
      <a routerLink="project" [class.activated]="currentUrl == '/project'">
        <i class="material-icons"> create_new_folder</i> <!--Project Configuration File--
>
      </a>
    </li>

    <li>
      <a routerLink="request" [class.activated]="currentUrl == '/request'">
        <i class="material-icons">fiber_new</i><!--Job Request-->
      </a>
    </li>

    <li>
      <a routerLink="job" [class.activated]="currentUrl == '/job'">
        <i class="material-icons">search</i><!--Job Request search-->
```

```
      </a>
    </li>

  </ul>
</nav>
```

sidebar.component.scss

```
nav {
  background: #2D2E2E;
  height: 100%;

  ul {
    list-style-type: none;
    padding: 0;
    margin: 0;

    li {

      a {
        color: #fff;
        padding: 20px;
        display: block;
      }

      .activated {
        background-color: #00a8ff;
      }
    }
  }
}
```

sidebar.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { SidebarComponent } from './sidebar.component';

describe('SidebarComponent', () => {
  let component: SidebarComponent;
  let fixture: ComponentFixture<SidebarComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ SidebarComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(SidebarComponent);
    component = fixture.componentInstance;
```

```
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Other App Files

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { SidebarComponent } from './sidebar/sidebar.component';
import { EmployeeComponent } from './employee/employee.component';
import { RequestComponent } from './request/request.component';
import { HomeComponent } from './home/home.component';

import { FormsModule, ReactiveFormsModule } from '@angular/forms'; // added for reactive
forms
import { ProjectComponent } from './project/project.component';
import { EngineerComponent } from './engineer/engineer.component';

import { HttpClientModule } from '@angular/common/http';
import { JobComponent } from './job/job.component'; //linked to service to get users from
public API

  //START TEST FOR SEARCH PIPE https://codeburst.io/create-a-search-pipe-to-dynamically-
filter-results-with-angular-4-21fd3a5bec5c
import { FilterPipe } from './filter.pipe';
import { EngineerDetailsComponent } from './engineer-details/engineer-details.component';
import { ProjectDetailsComponent } from './project-details/project-details.component';
import { JobDetailsComponent } from './job-details/job-details.component';
  //END TEST FOR SEARCH PIPE


@NgModule({
  declarations: [
    AppComponent,
    SidebarComponent,
    EmployeeComponent,
    RequestComponent,
    HomeComponent,
    ProjectComponent,
    EngineerComponent,
    JobComponent,

      //START TEST FOR SEARCH PIPE https://codeburst.io/create-a-search-pipe-to-
dynamically-filter-results-with-angular-4-21fd3a5bec5c
```

```
    FilterPipe,

      EngineerDetailsComponent,

      ProjectDetailsComponent,

      JobDetailsComponent
      //END TEST FOR SEARCH PIPE


  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule, //added
    ReactiveFormsModule, //added
    HttpClientModule, //added
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app-routing.module.ts (Not added to ASP.NET)

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { EmployeeComponent } from './employee/employee.component'; //added
import { RequestComponent } from './request/request.component'; //added
import { HomeComponent } from './home/home.component'; //added
import { ProjectComponent } from './project/project.component'; //added
import { EngineerComponent } from './engineer/engineer.component'; //added
import { JobComponent } from './job/job.component';
import { EngineerDetailsComponent } from './engineer-details/engineer-details.component';
import { ProjectDetailsComponent } from './project-details/project-details.component';
import { JobDetailsComponent } from './job-details/job-details.component';

const routes: Routes = [
  {
    path: '',
    component: HomeComponent //routes to the homepage
  },
  {
    path: 'employee',
    component: EmployeeComponent // routes to employee configuration file page
  },
  {
    path: 'request',
    component: RequestComponent //routes to job request page
  },
  {
    path: 'project',
    component: ProjectComponent //routes to project configuration file page
```

```
  },
  {
    path: 'engineer',
    component: EngineerComponent //routes to engineer list page
  },
  {
    path: 'job',
    component: JobComponent //routes to job request search page
  },
  {
    path: 'engineer-details/:id',
    component: EngineerDetailsComponent //routes to engineer details page
  },
  {
    path: 'project-details/:id',
    component: ProjectDetailsComponent //routes to project details page
  },
  {
    path: 'job-details/:id',
    component: JobDetailsComponent //routes to job details page
  },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

data.service.ts

```
//Service is used to fetch a list of users from a public API

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http'; //added

@Injectable({
  providedIn: 'root'
})
export class DataService {

  constructor(private http: HttpClient) { }

  getEngineers() {
    return this.http.get('https://jsonplaceholder.typicode.com/users/') //make sure to
have '/' at the end of the website
  }

  getEngineer(engineerId) {
    return this.http.get('https://jsonplaceholder.typicode.com/users/' + engineerId)
//make sure to have '/' at the end of the website
  }

  getJobs() {
```

```
    return this.http.get('https://jsonplaceholder.typicode.com/users/') //make sure to
have '/' at the end of the website
  }

  getJob(jobId) {
    return this.http.get('https://jsonplaceholder.typicode.com/users/' + jobId) //make
sure to have '/' at the end of the website
  }
}
```

filter.pipe.ts (Not added to ASP.NET)

```
  //START TEST FOR SEARCH PIPE https://codeburst.io/create-a-search-pipe-to-dynamically-
filter-results-with-angular-4-21fd3a5bec5c

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'filter'
})

export class FilterPipe implements PipeTransform {
  transform(items: any[], searchText: string): any[] {
    if (!items) return [];
    if (!searchText) return items;

    searchText = searchText.toLowerCase();

    return items.filter(it => {
      return it.toLowerCase().includes(searchText);
    });
  }
}

  //END TEST FOR SEARCH PIPE
```

Other SRC Files

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>SaTest1</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
        rel="stylesheet">
```

```
  <link href="https://fonts.googleapis.com/css?family=Montserrat:300,700"
rel="stylesheet">
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/foundation/6.3.1/css/foundation.min.css">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

styles.scss

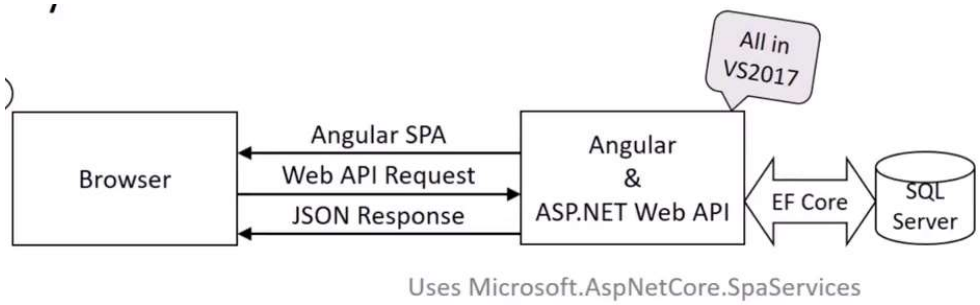
```
/* You can add global styles to this file, and also import other style files */
body {
  margin: 0;
  background: #F2F2F2;
  font-family: 'Montserrat', sans-serif;
  height: 100vh;
}

#container {
  display: grid;
  grid-template-columns: 70px auto;
  height: 100%;

  #content {
    padding: 30px 50px;

    ul {
      list-style-type: none;
      margin: 0;
      padding: 0;

      li {
        background: #fff;
        border-radius: 8px;
        padding: 20px;
        margin-bottom: 8px;

        a {
          font-size: 1.5em;
          text-decoration: none;
          font-weight: bold;
          color: #00A8FF;
        }

        ul {
          margin-top: 20px;

          li {
            padding: 0;

            a {
```

```
                font-size: 1em;
                font-weight: 300;
            }
        }
    }
}
}
}
```

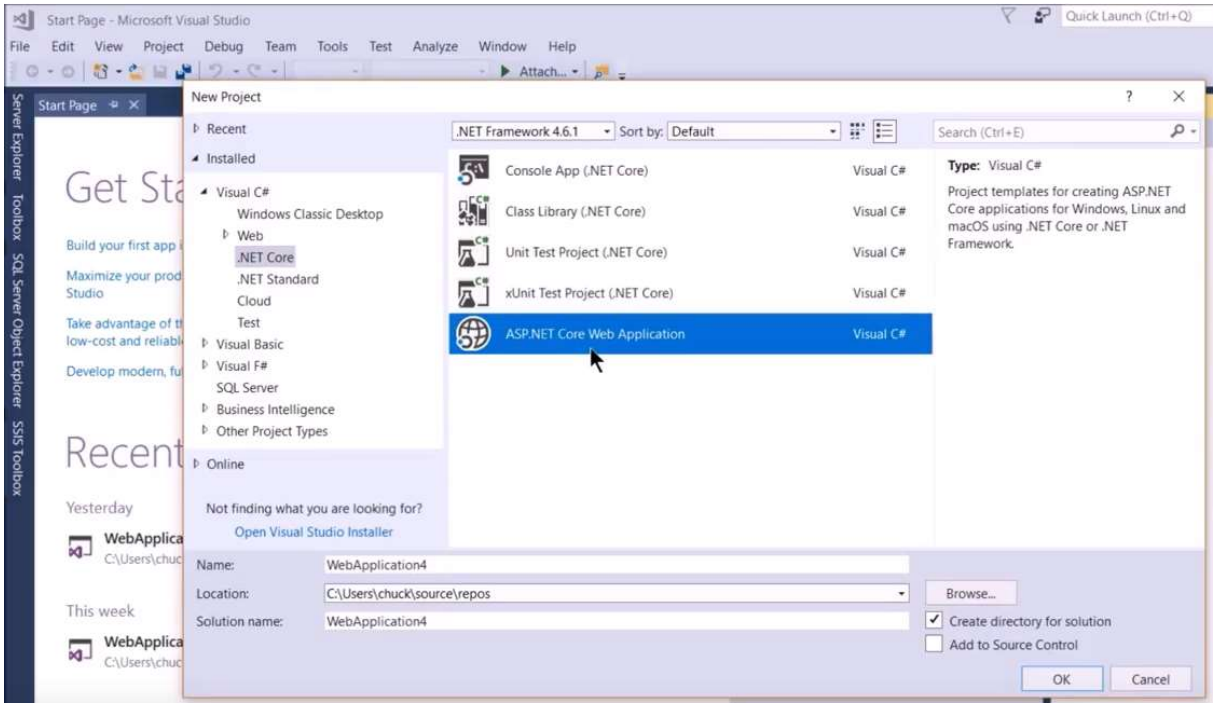
APPENDIX I – TUTORIAL: CONNECTING SQL DATABASE TO ASP.NET

- Angular, ASP.NET Core Web API & SQL Server Quickly
  URL: https://www.youtube.com/watch?v=ISOpIEOvPq4 [cited 5 October 2018]

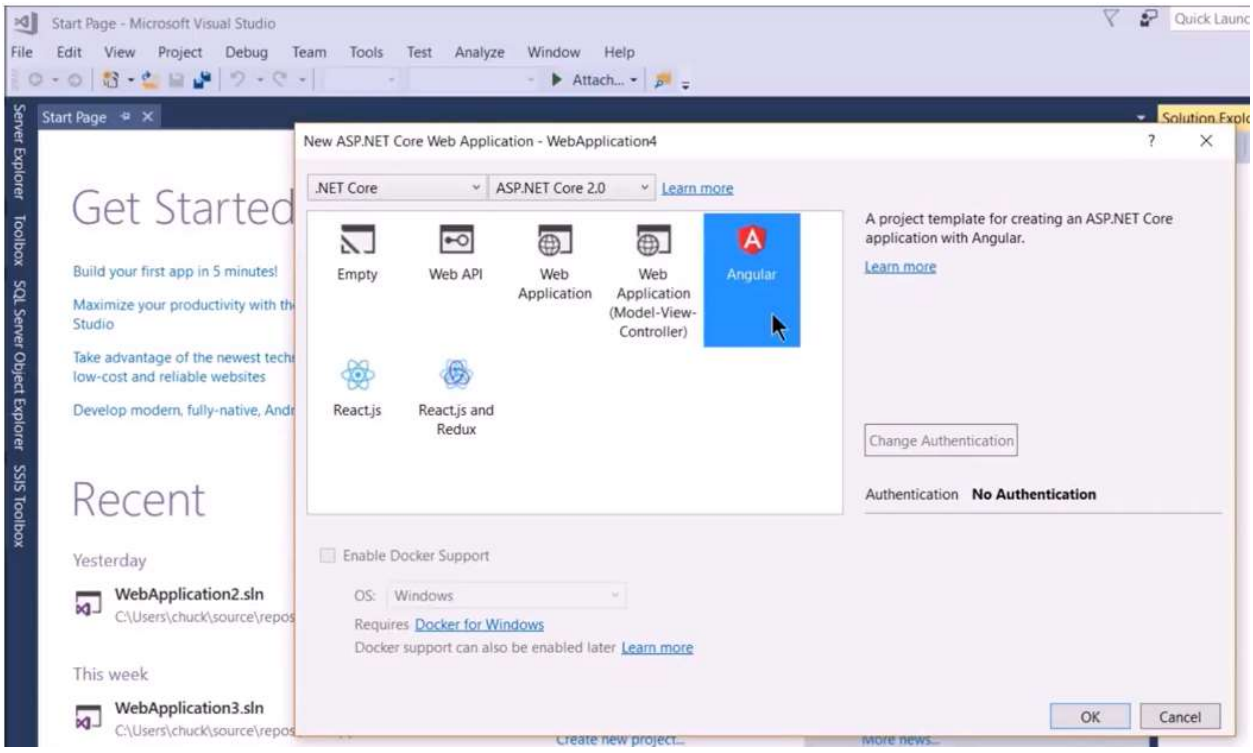
- **OVERVIEW**



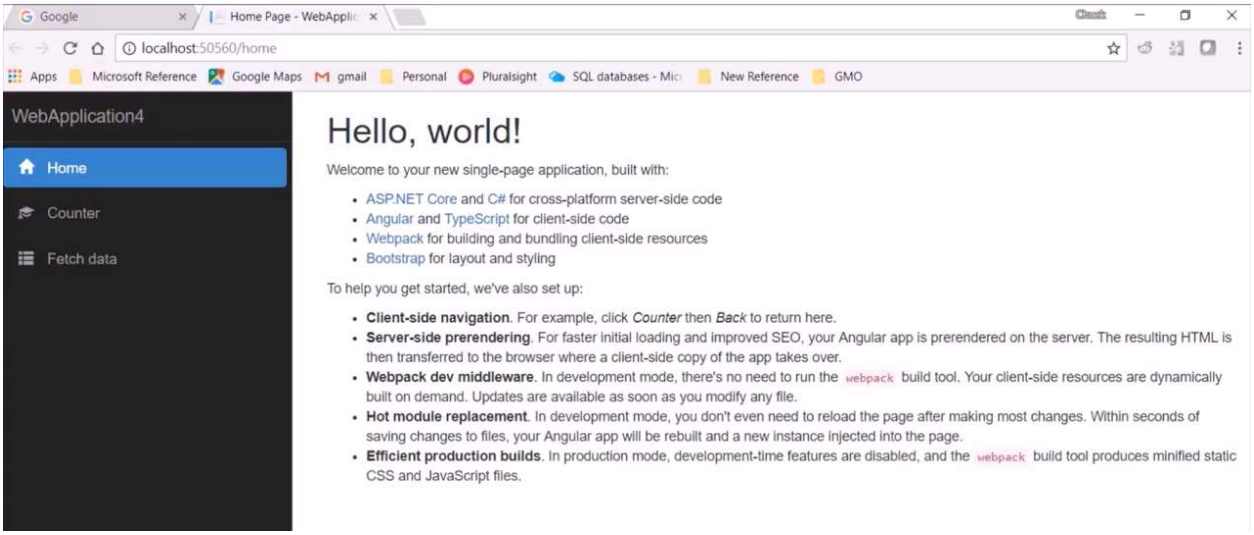
- **STEPS:**
  1. **New Visual Studio Project**
     i. Start a new Microsoft Visual Studios Project in .NET Core, choose ASP.NET Core Web Application

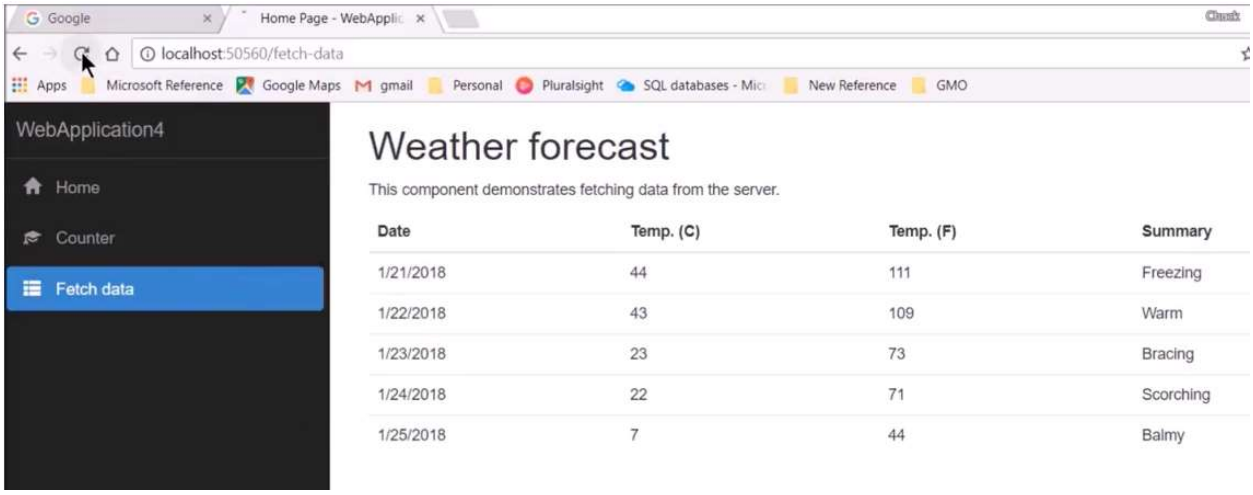


ii. Choose Angular project template. It is a Web API that follows the Model-View-Controller (MVC) design pattern.

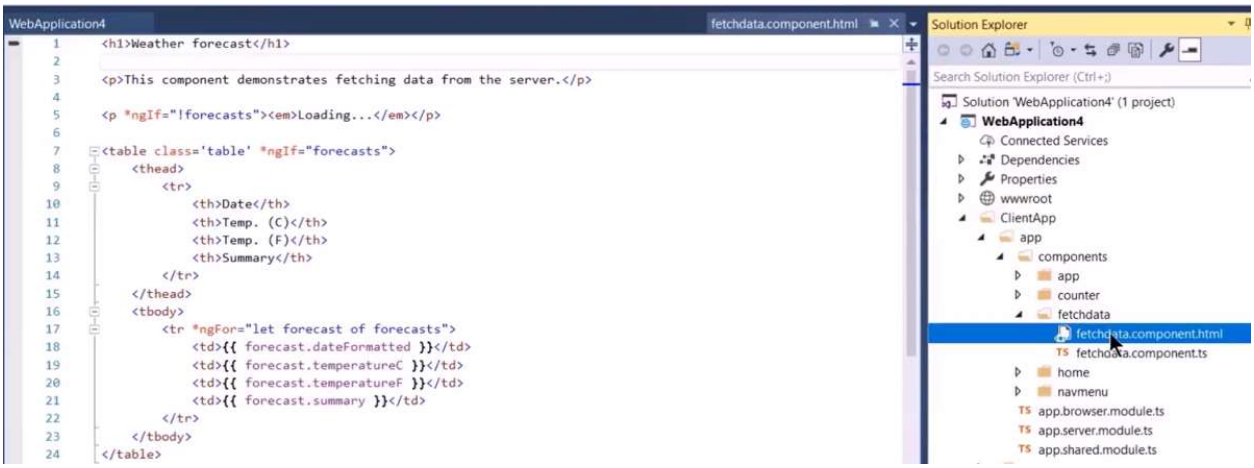


iii. Start website without debugging to see default web application. It may or may not have the Webpack bullet points.

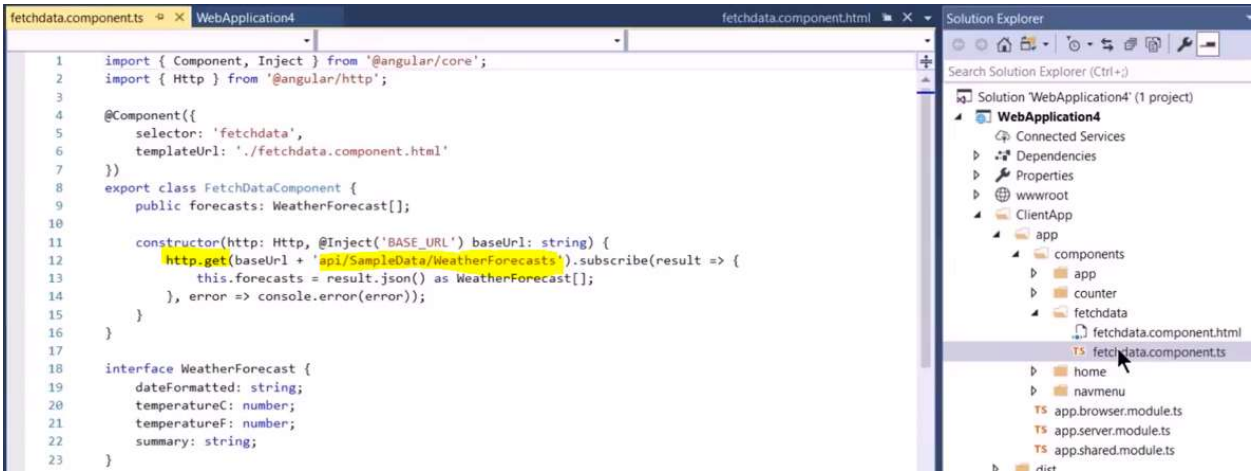




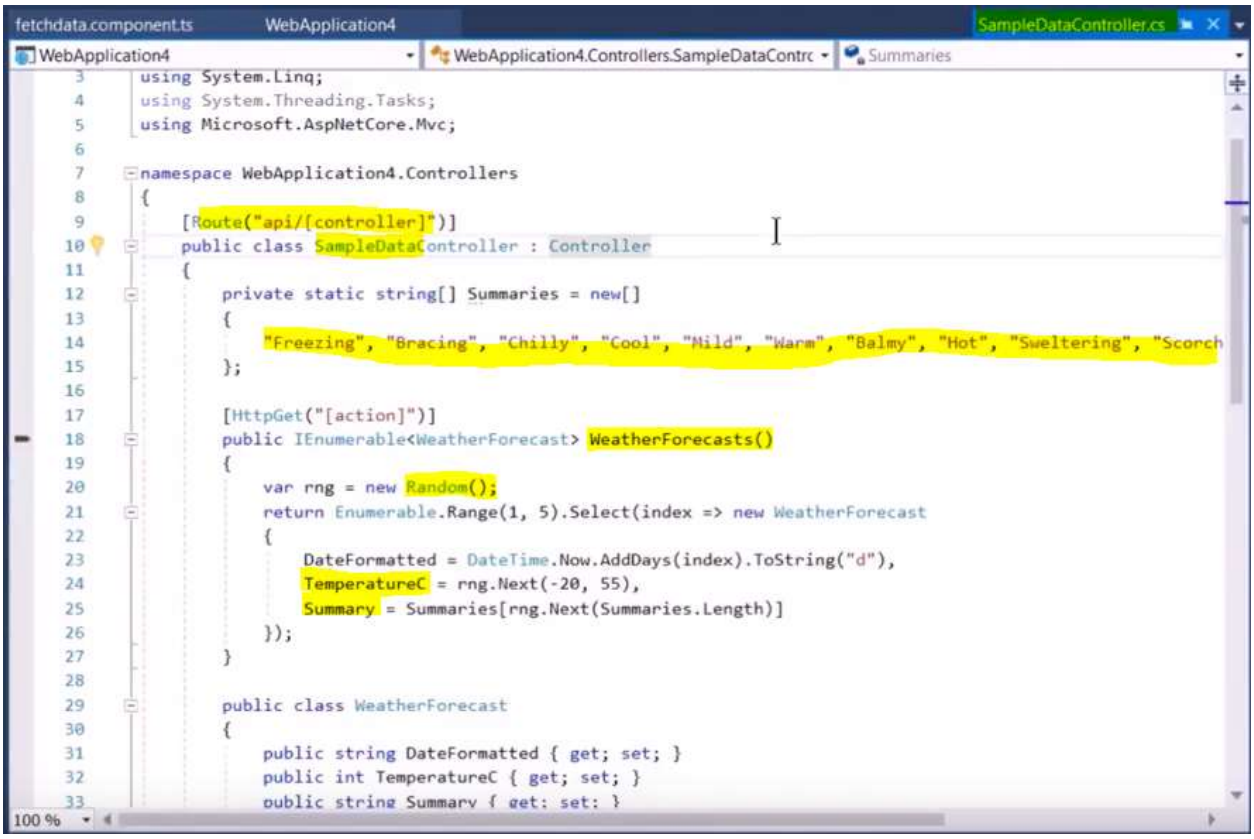
iv. Format for the Fetch data tab called "Weather forecast" is coming from *fetchdata.component.html* file.



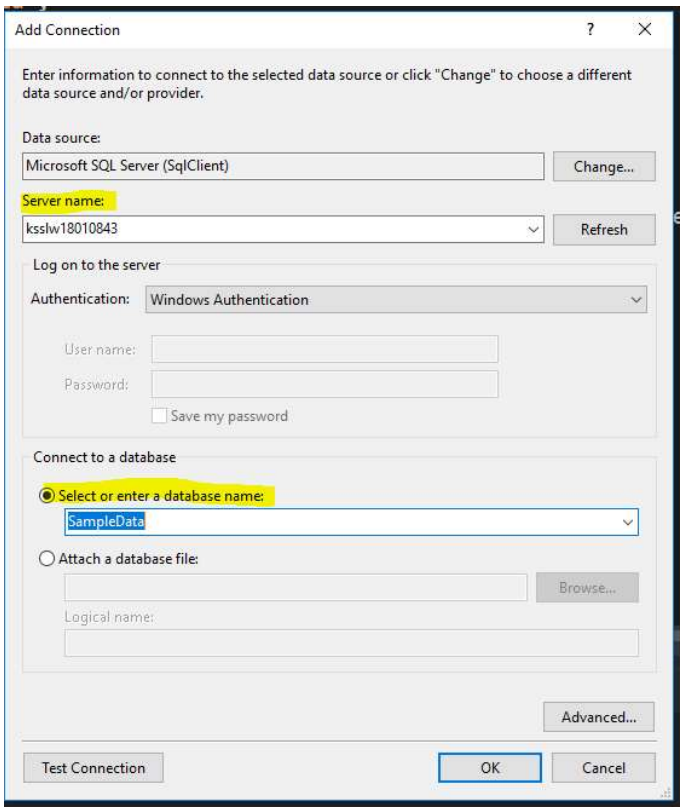
v. The HTML file is the template for the Typescript page *fetchdata.component.ts* . This is the page getting the results with *http.get()* to the Application Program Interface (API) *api/SampleData/WeatherForecasts*



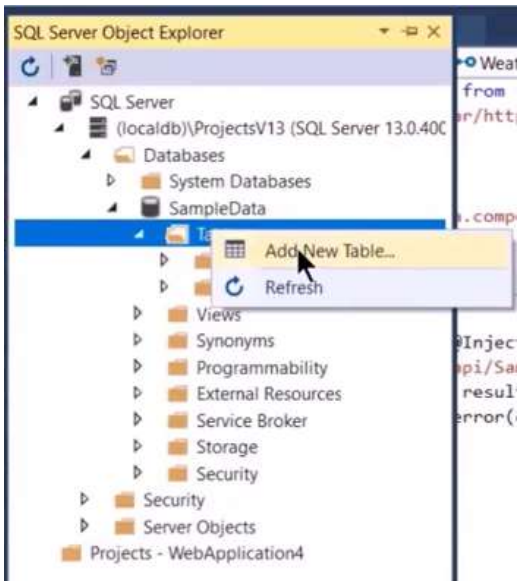
vi. The API is coming from the *SampleDataController.cs* file. It shows the Route api/[controller] as *SampleData* in class name *SampledataController* and HttpGet [action] as the method *WeatherForecast()* . It is using a random number to pick a temperature and summary.



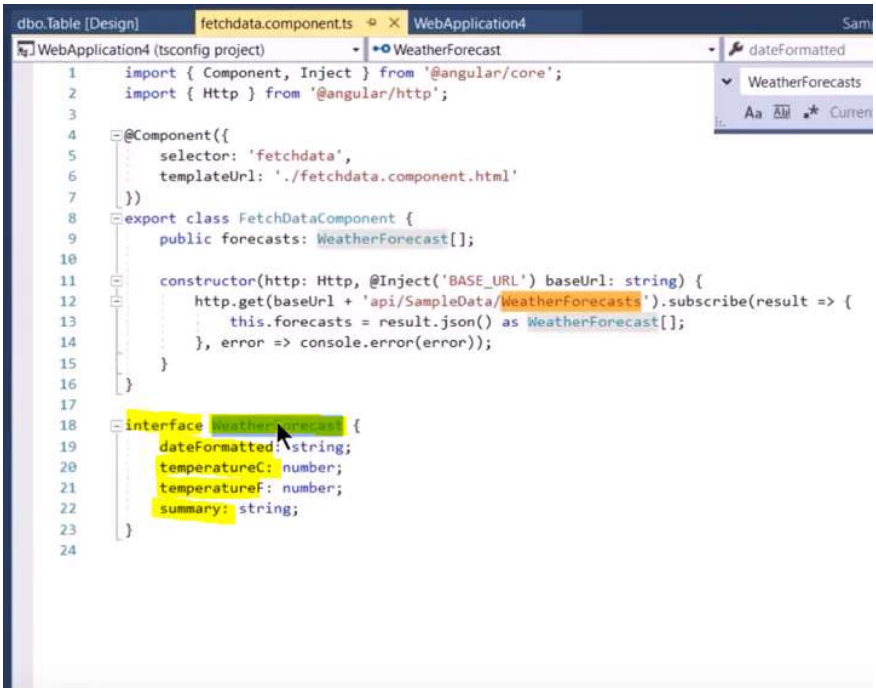
2. **Database Table**
    i. Make a sample database in Microsoft SQL Server and connect the server to Microsoft Visual Studios. In Visual Studios click *Tools > Connect To Database*. Write your server name and select the database you created in SQL from the dropdown list.
        - Additional directions on how to connect Microsoft SQL Server to Microsoft Visual Studios can be found in the cited source entitled **ASP.NET Essential Training**, specifically in the video "Connecting to the Database in ASP.NET" https://www.linkedin.com/learning/asp-dot-net-essential-training/connecting-to-the-database-in-asp-dot-net



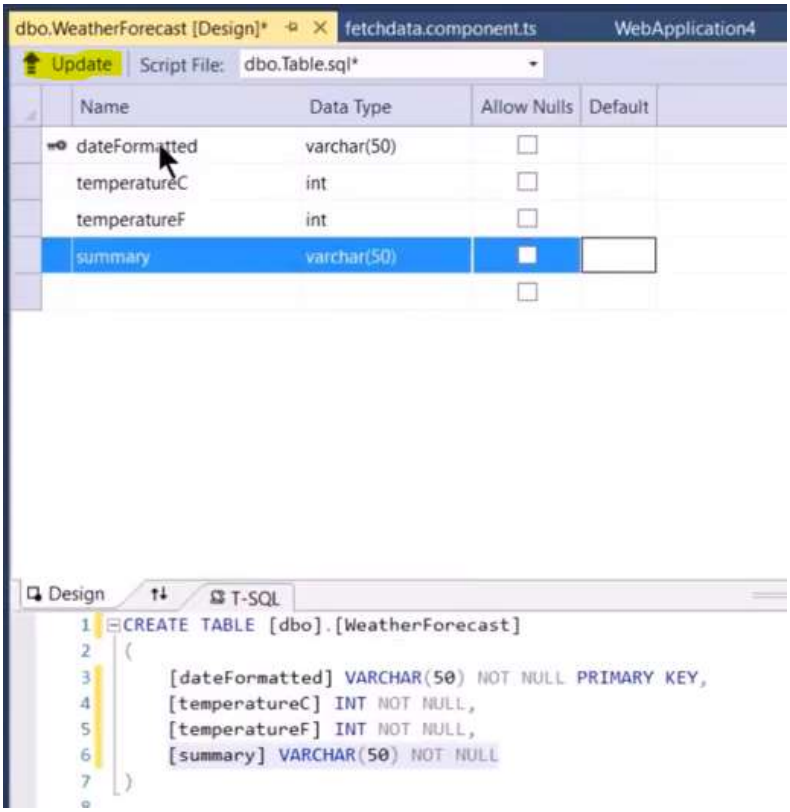
    ii. After SQL is connected, use Visual Studios to add a new table to the sample database



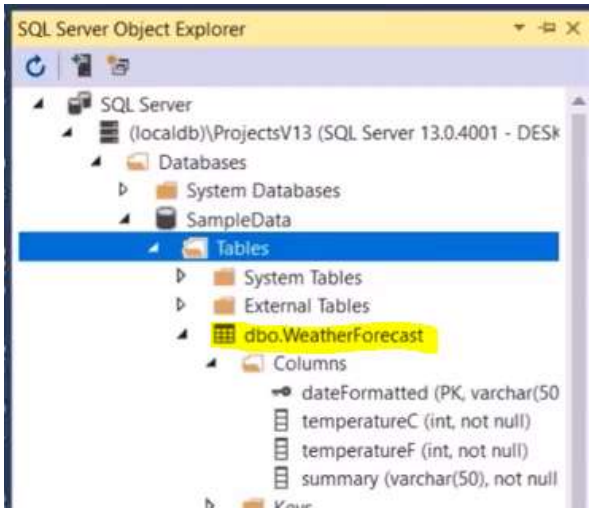
iii. The table name and column names must match the interface already created in the typescript file.



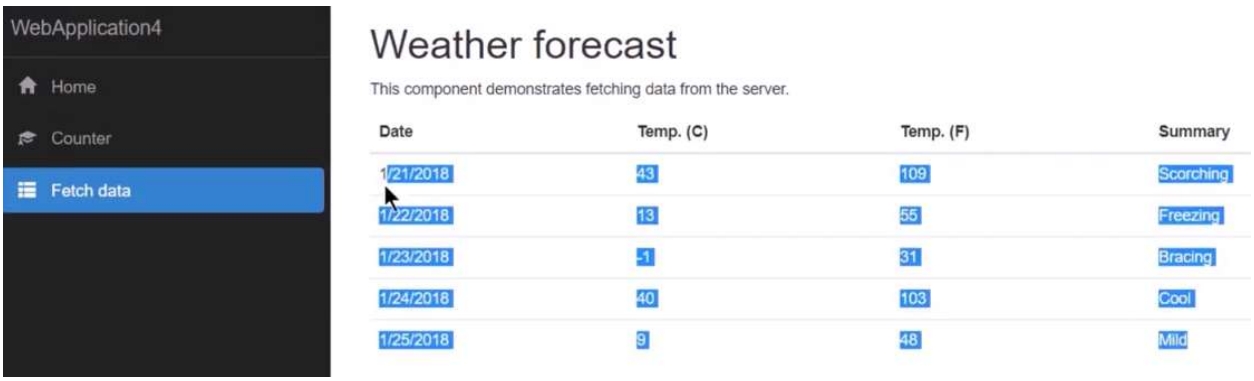
iv. Then click Update button to update the *dbo.WeatherForecast [Design]* table in the database. Check to make sure the table is in the database.



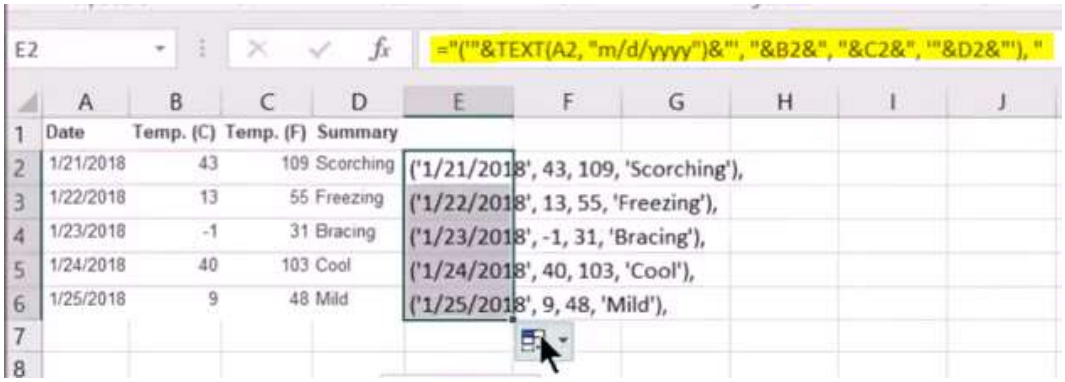
v. Check to make sure the table is in the database.



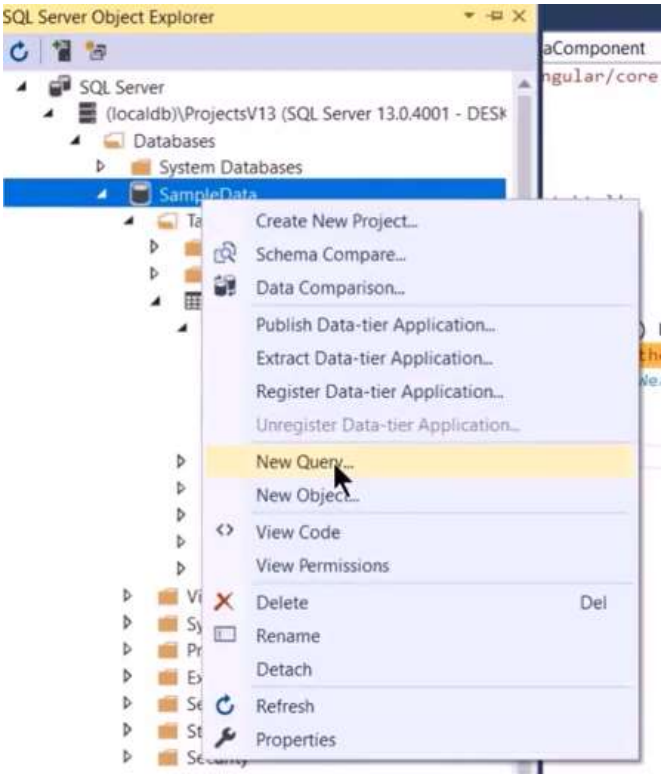
vi. Insert rows into table: Highlight info from website.



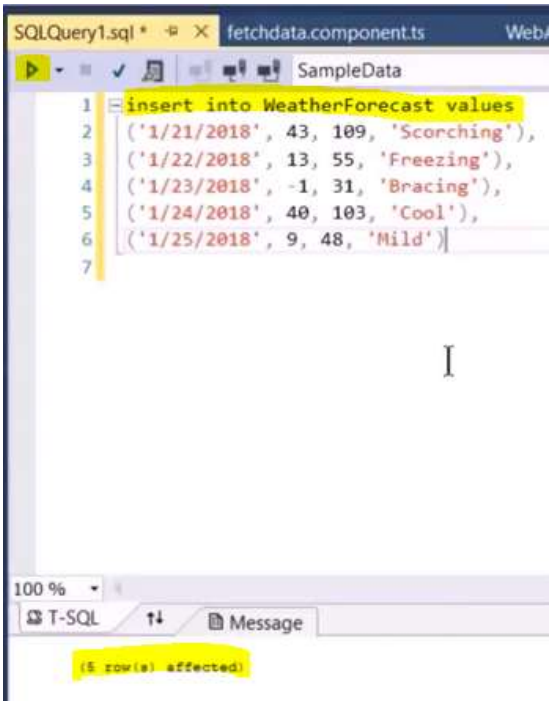
vii. Paste into Excel file and format data into rows.



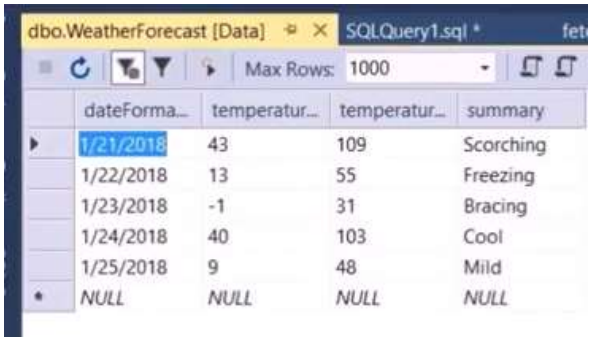
viii. In Visual Studios, right click the sample database and select New Query



ix. Use SQL syntax to insert the formatted rows into the WeatherForecast table. Then click green arrow to update table rows. Then check database.

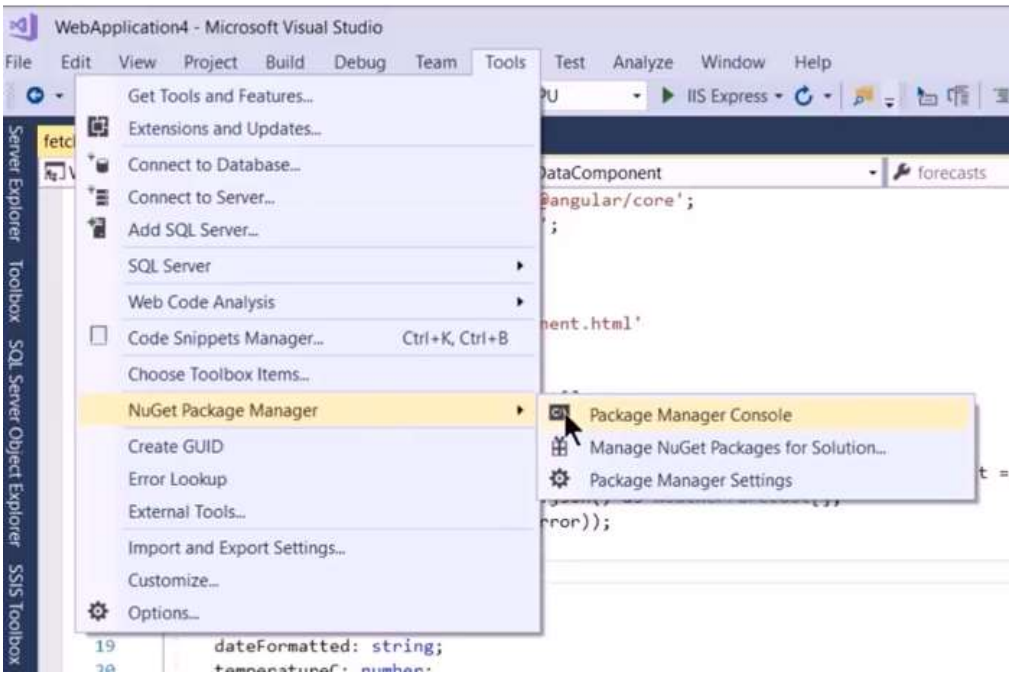


x. Check to make sure the rows are in the table.



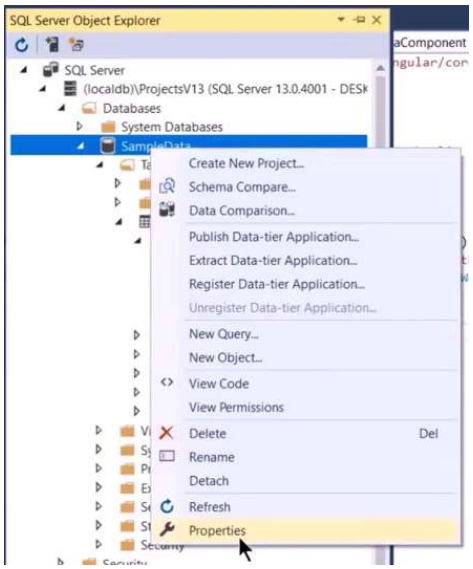
3. Model and DbContext

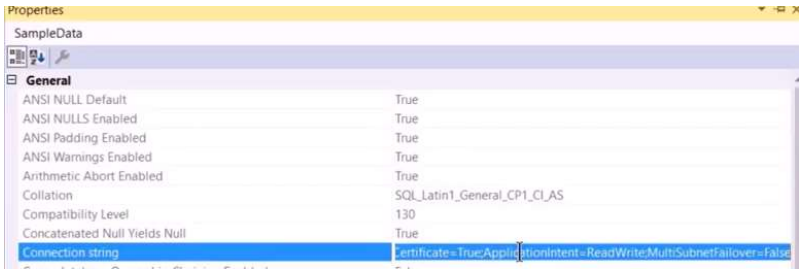
i. From that table, generate a Model and a DbContext by using a built-in console. If you want to use a command prompt outside of Visual Studio, you might have to insert different instructions.



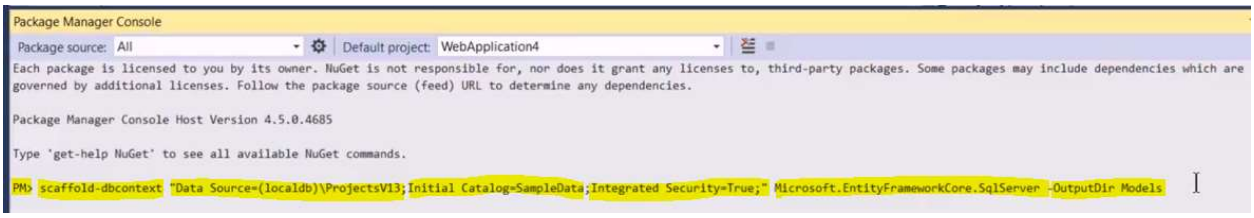
ii. Next to "PM>"

- Search *scaffold-dbcontext*
- Write the parameters of a connection string. It is found by looking at the Properties of the sample database. Copy the Connection string, then past into console. Only paste Data Source, Initial Catelog, and Integrated Security

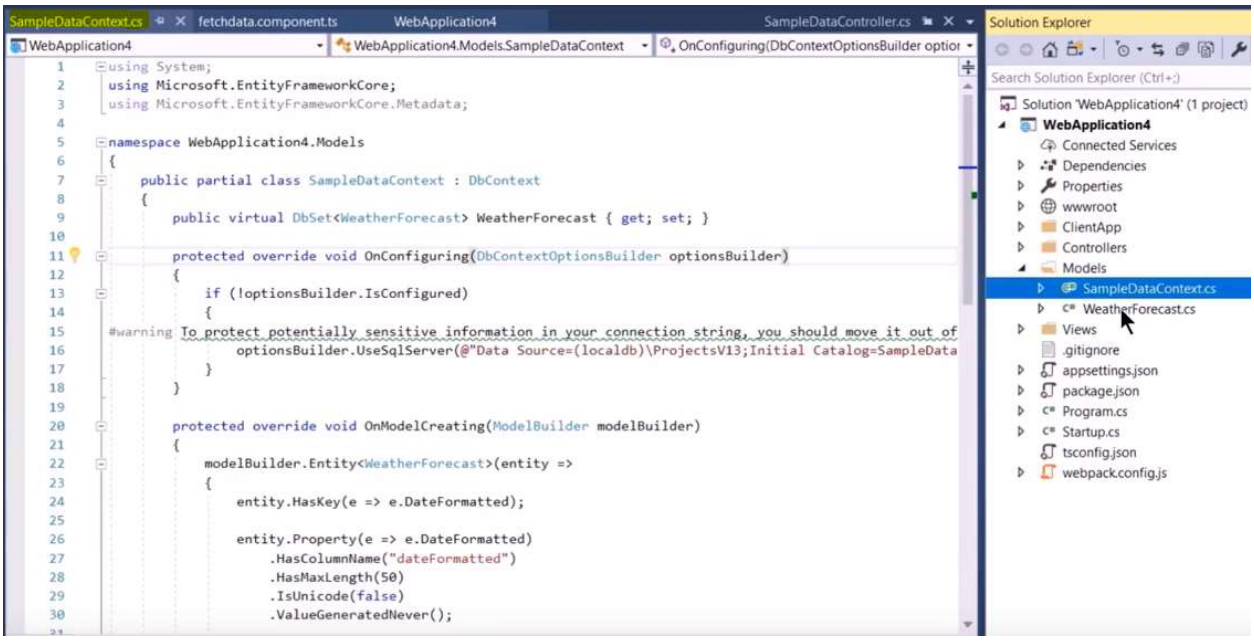




- Write the SQL Server driver that is a part of the Entity Framework Core *Microsoft.EntityFrameworkCore.SqlServer*.
- Then write that you want the results to go into the Models folder, which does not exist yet. It will create that. *-OutputDir Models*



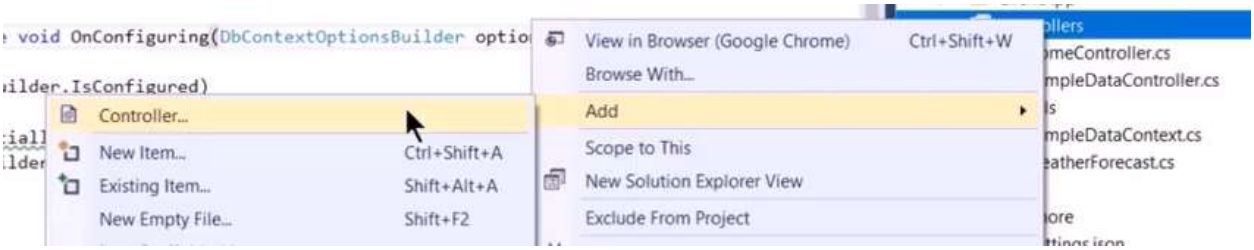
iii. When you run the search, it creates and opens a new Model called *SampleDataContext.cs* . It also creates a WeatherForecast.cs file



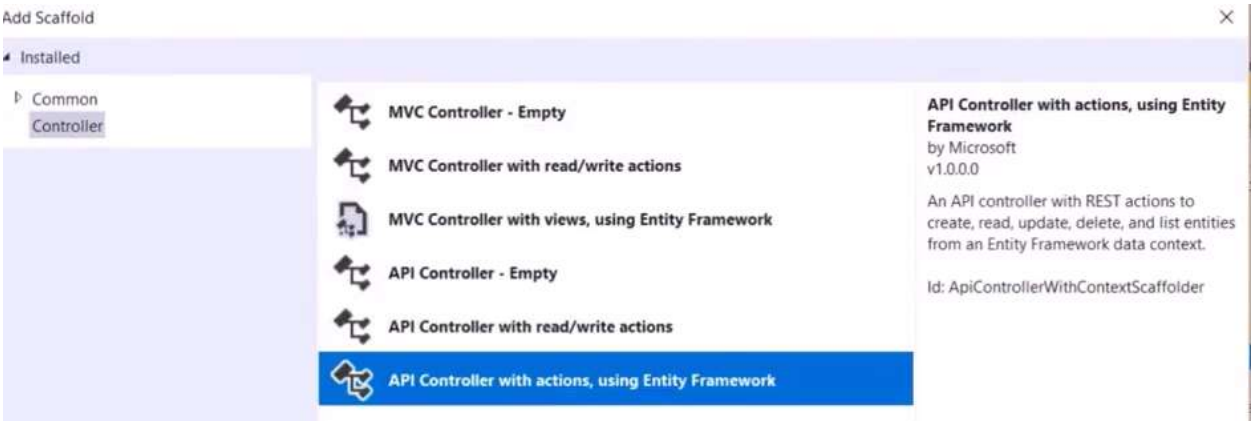
iv. It also creates a *WeatherForecast.cs* file in Models with the same set of fields.

4. Controller

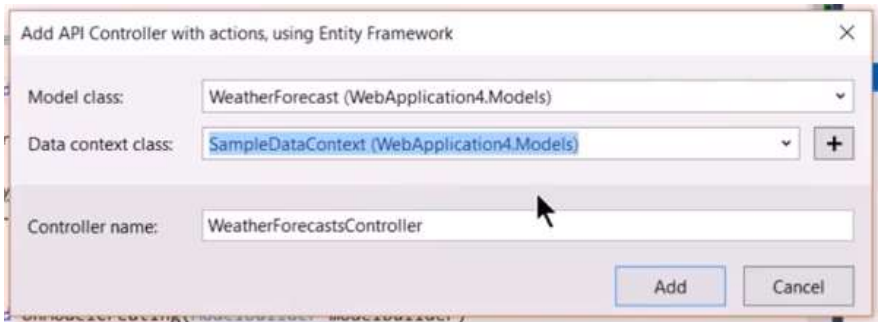
i. Before you run the Model, you need to add a Controller. Right click Controllers folder, Add > Controller



ii. Choose the API Controller with actions, using Entity Framework



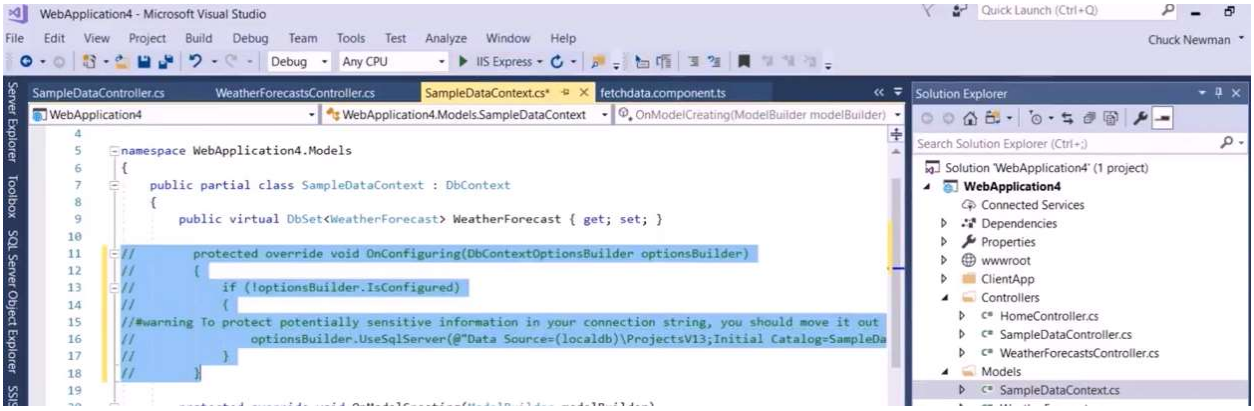
iii. From dropdown list, choose WeatherForecast for Model class and SampleDataContext for the Data context file.



iv. Creates a Controller called *WeatherForecastsController.cs* where you will get data from the database through the Route *api/WeatherForecasts*

5. DbContext service and Dependency Injection

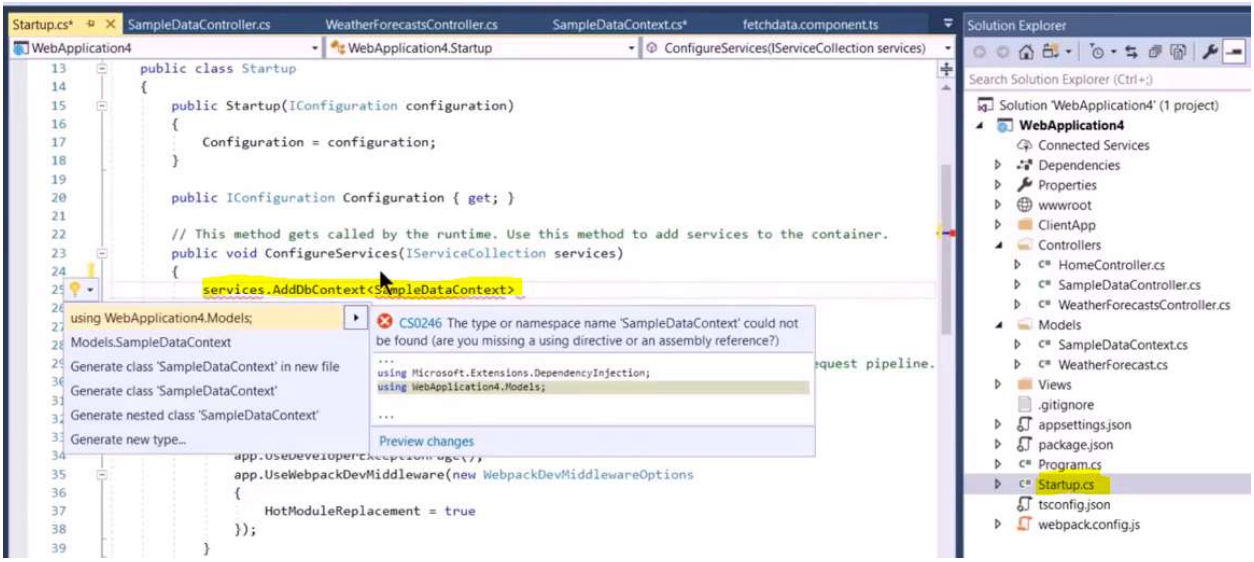
i. The DbContext needs to be injected into the class as a service. Comment out *DbContextOptionsBuilder* area in Model file *SampleDataContext.cs* .



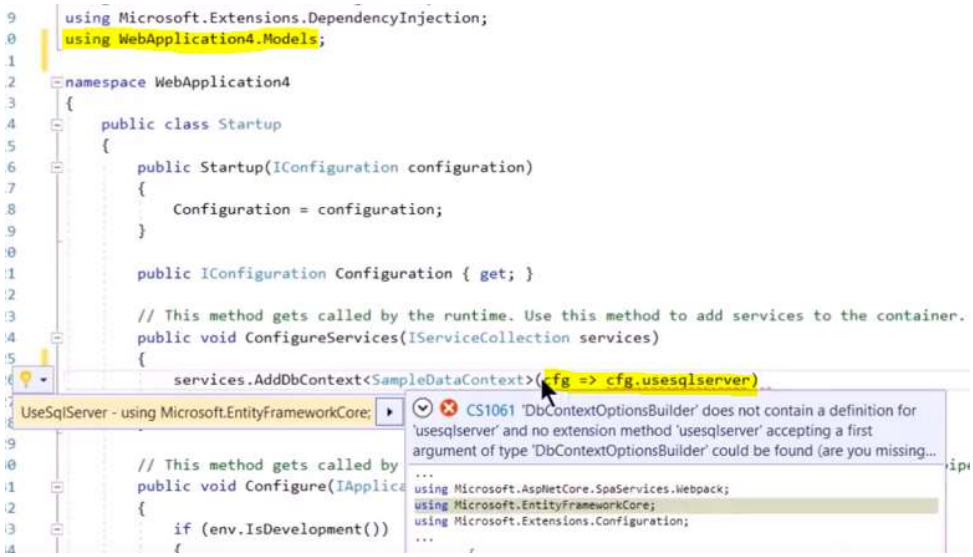
ii. Inject the service *dbContextOptions* before creating the service by adding it in the method *SampleDataContext(),* if the method is not already written, and pass the connection down to the base class

```
namespace WebApplication4.Models
{
    public partial class SampleDataContext : DbContext
    {
        public virtual DbSet<WeatherForecast> WeatherForecast { get; set; }

        public SampleDataContext(DbContextOptions<SampleDataContext> options) : base(options)
        {

        }
//      protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
//      {
```

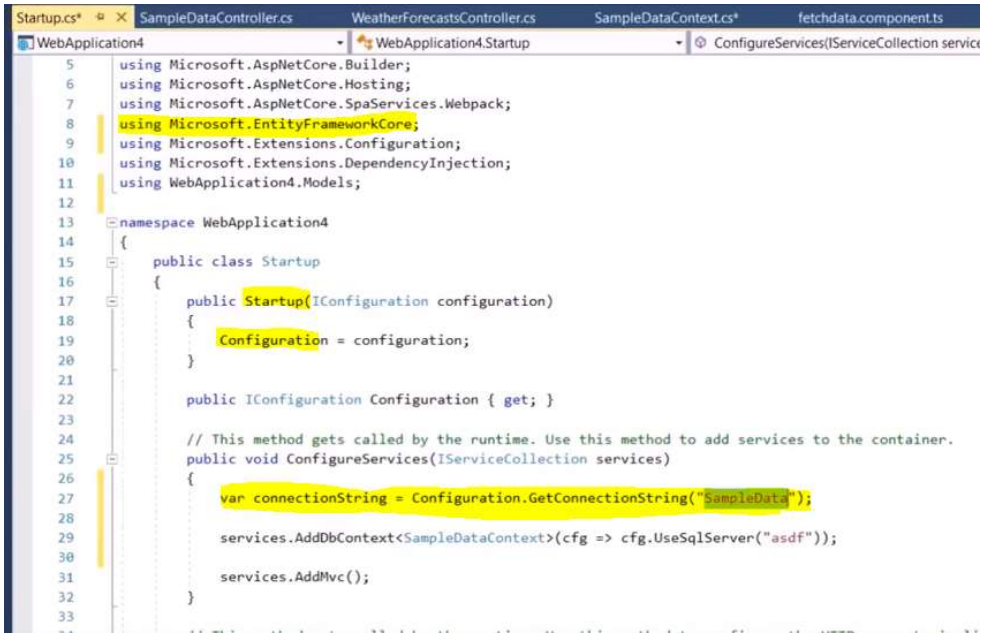
iii. Now create the service in the file *Startup.cs* by writing *services.AddDbContext<SampleDataContext>* in the *ConfigureServices()* method. The namespace is not recognized. Right-click *SampleDataContext* to get Quick Actions and choose using Models.



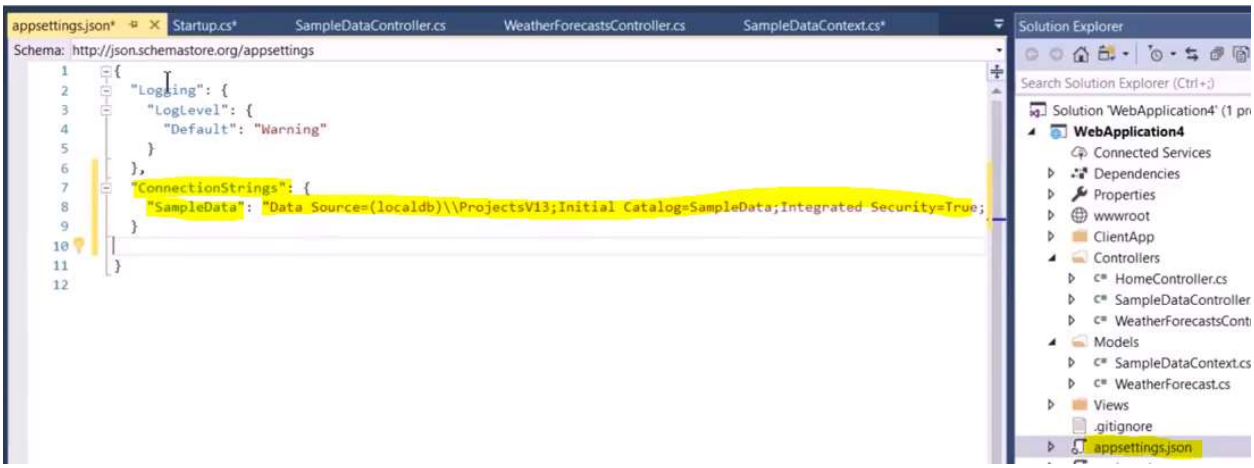
iv. Use a configuration object *cfg* to add a connection string *UseSqlServer*. Is also not recognized, so right click and add using Microsoft.EntityFrameworkCore



v. Finish *UseSqlServer("asdf")* but don't want to write (hard code) the connection string in the parentheses. It is better to use the *Configuration* object in the *Startup()* method above. Add GetConnectionString () method by writing *var connectionString = Configuration.GetConnectionString("SampleData");*



vi. Configuration gets data from *appsettings.json* file. Add a connection string to that file.

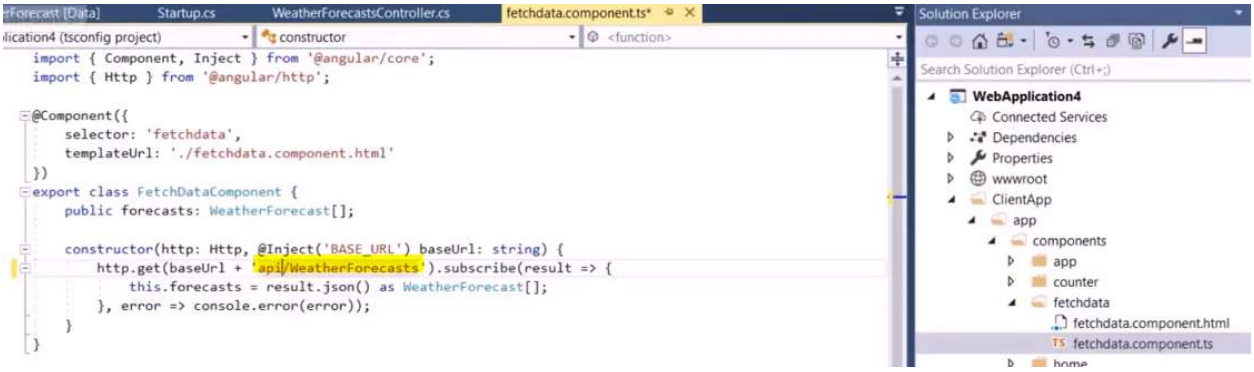


vii. Now back in Model *SampleDataContext.cs*, put *connectionString* into *UseSqlServer()* instead of "asdf"

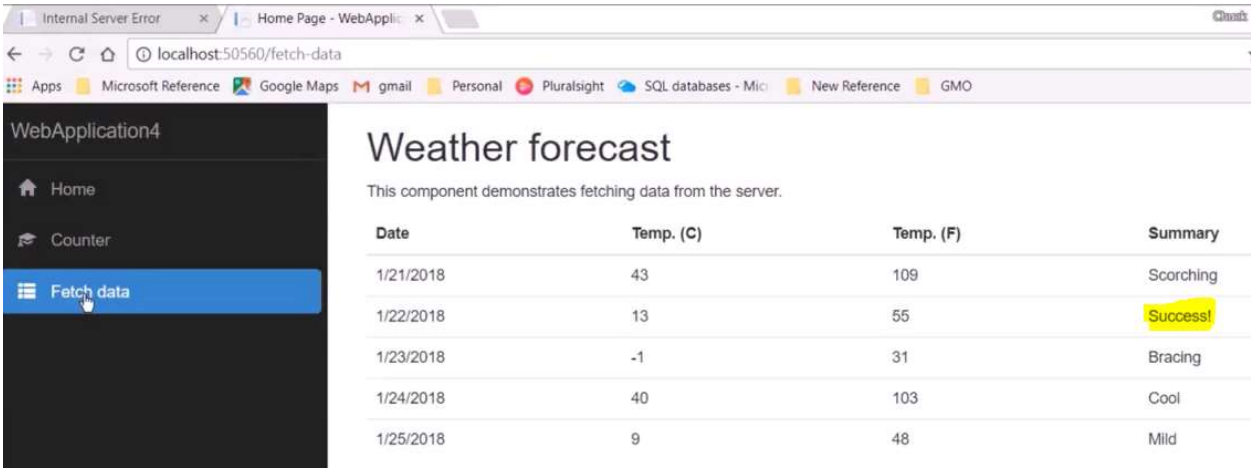
viii. Change an entry in the database to Success! to show the API is working

	dateFormatted	temperatureC	temperatureF	summary
1	9/25/2018	-9	16	Mild
2	9/26/2018	48	118	Success!
3	9/27/2018	28	82	Freezing
4	9/28/2018	31	87	Mild
5	9/29/2018	-10	15	Warm

i. Make sure API is working in Angular by going back to *fetchdata.component.ts* file and change the *http.get api/SampleData/WeatherForecasts* to *api/WeatherForecasts*



i. Build and Run the website. The SQL table with Success! can be seen



APPENDIX II - INTERVIEWING CLIENTS: REQUIREMENTS ELICITATION

- **Mississippi Board of Pharmacy, "Functional Requirements Document Template"**
  URL: http://www.mbp.ms.gov/Documents/PMP%20NextGen%20Software.pdf [cited 13 April 2018]
  Description: Shows example of a Functional Requirements Document

- [15]**LinkedIn Learning/Lynda.com, "Requirements Elicitation for Business Analysts: Interviews"**
  URL: https://www.linkedin.com/learning/requirements-elicitation-for-business-analysts-interviews/welcome [cited 6 July 2018]

- [16]**LinkedIn Learning/Lynda.com, "Requirements Gathering Course"**
  URL: https://www.linkedin.com/learning/requirements-elicitation-and-analysis/welcome [cited 22 June 2018]

- **MITRE Systems Engineering Guide, "Analyzing and Defining Requirements"**
  URL: https://www.mitre.org/publications/systems-engineering-guide/se-lifecycle-building-blocks/requirements-engineering/analyzing-and-defining-requirements [cited 13 July 2018]

- **MITRE Systems Engineering Guide, "Eliciting, Collecting, and Developing Requirements"**
  URL: https://www.mitre.org/publications/systems-engineering-guide/se-lifecycle-building-blocks/requirements-engineering/eliciting-collecting-and-developing-requirements [cited 13 July 2018]

[15] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research
[16] *Free access with library membership at Orange County Public Library https://www.ocls.info/learning-research