Setting Up An ASP.Net/React Development Environment

Adrian King

Version of November 5th, 2018

Introduction

These notes describe how to set up a development environment for the Last Call project. These notes should apply to both Windows and Mac environments although the screenshots reflect only the Windows setup.

Be warned! Before you start you will need lots of empty disk space and a good Internet connection. The main reason setting up as described here is due to the very large number of files that Visual Studio insists on creating for the project. Once everyone has their base established we can exchange updates via GitHub.

Please contact me with questions or reports of success or failure in getting yourself set up.

Downloads

Download Microsoft Visual Studio 2017 Community Edition:

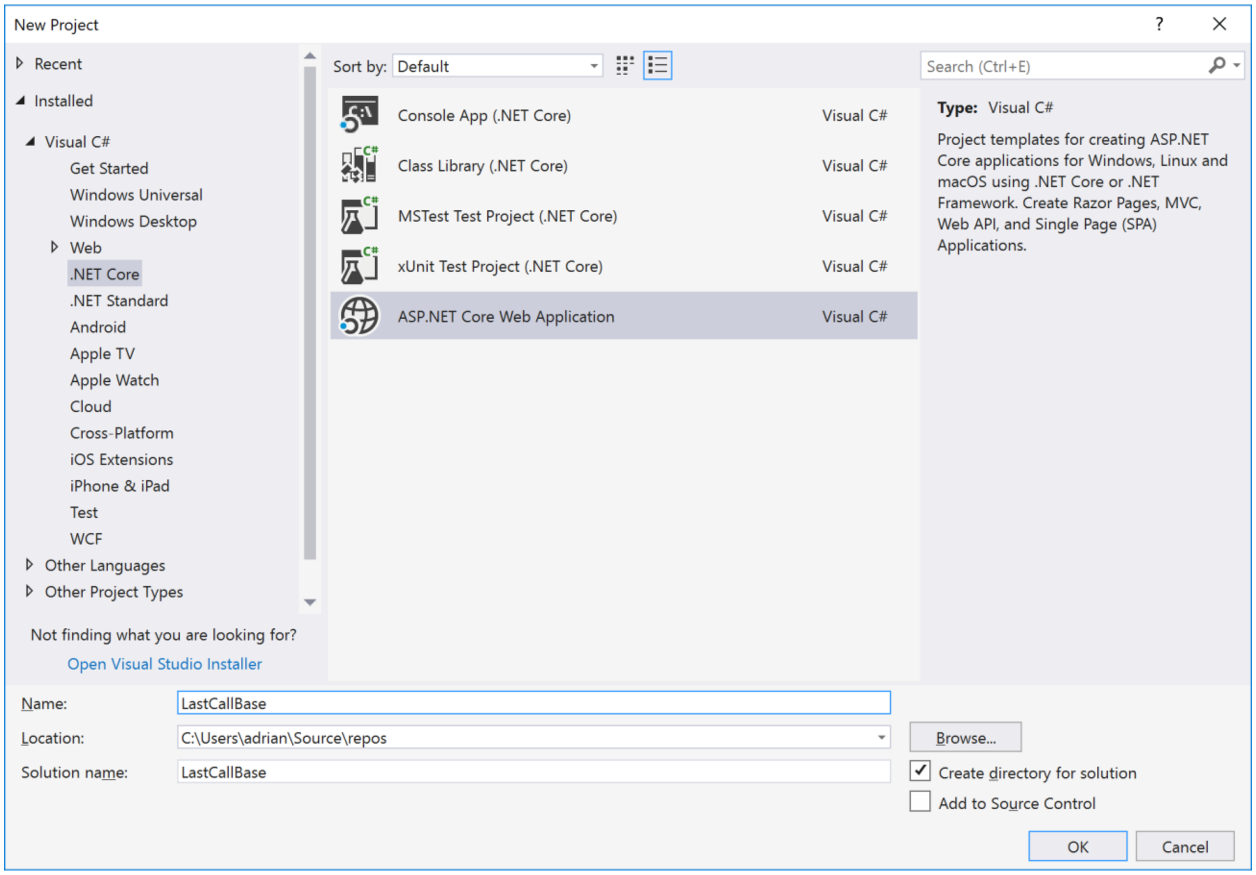
https://visualstudio.microsoft.com/downloads/ -- the link for the Mac version is down the page a little ways.

There are lots of options that the installer allows you to choose. If you have the disk space, choose most things. At the very least you will need C# language support, the Windows Universal Program SDK and the Android SDK. If you don't get these then you'll be prompted later on in the process to go get them.

First Project

Run Visual Studio, choose File->New->Project and then select a Visual C#, Web, .Net Core project type and choose ASP.Net Core Web Application.

Name the project LastCallBase. These options are shown in the screenshot below.



Console App (.NET Core)	Visual C#
Class Library (.NET Core)	Visual C#
MSTest Test Project (.NET Core)	Visual C#
xUnit Test Project (.NET Core)	Visual C#
ASP.NET Core Web Application	Visual C#

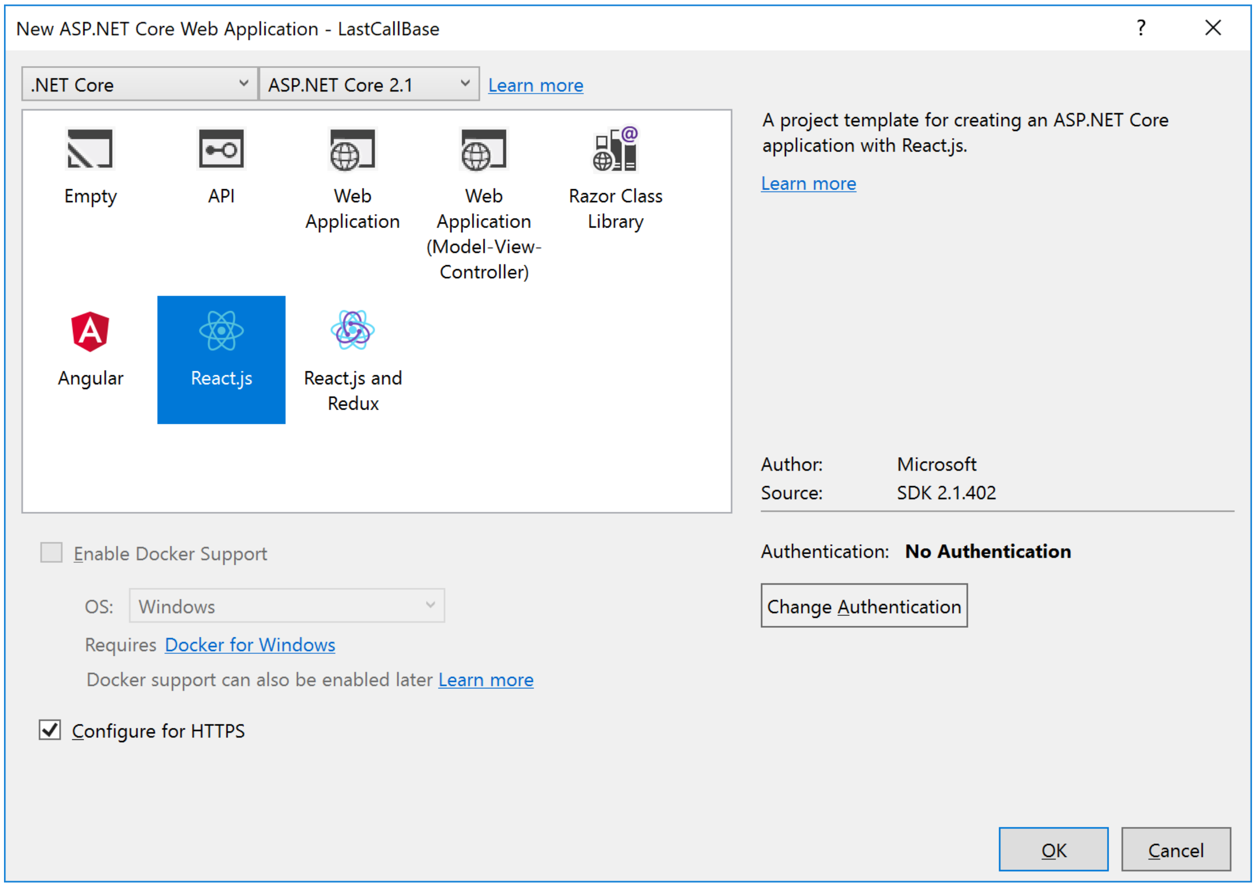
Type: Visual C#

Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create Razor Pages, MVC, Web API, and Single Page (SPA) Applications.

Name: LastCallBase
Location: C:\Users\adrian\Source\repos
Solution name: LastCallBase

[✓] Create directory for solution
[ ] Add to Source Control

Once you click OK, you will get a second screen asking you to choose a project template. Choose React.js as shown in the next screen. It doesn't matter whether or not you configure for HTTPS since this is only for local development. If you do select the HTTPS option then you will get a couple of extra dialogs asking you to confirm that you will accept the IIS generated certificate for localhost. Just accept the defaults.



New ASP.NET Core Web Application - LastCallBase

.NET Core | ASP.NET Core 2.1 | Learn more

A project template for creating an ASP.NET Core application with React.js.

Learn more

Author: Microsoft
Source: SDK 2.1.402

☐ Enable Docker Support

OS: Windows

Requires Docker for Windows

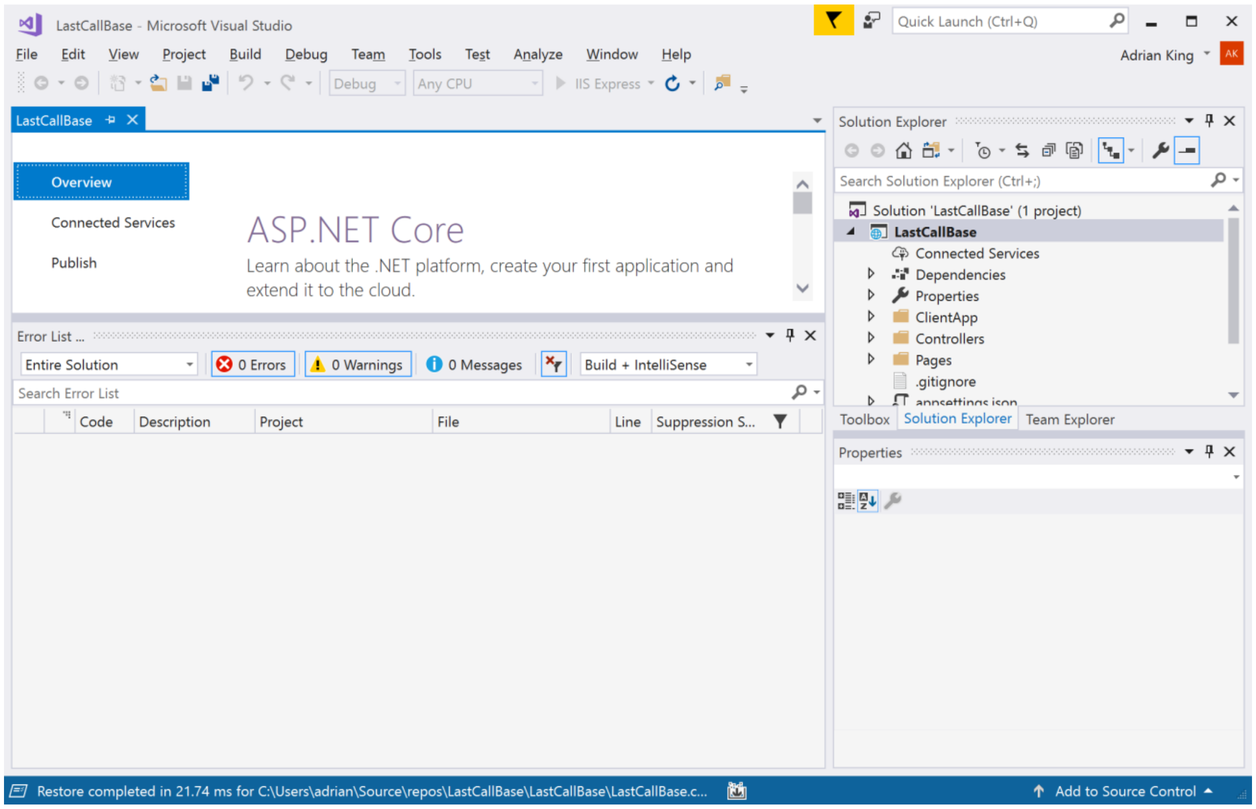
Docker support can also be enabled later Learn more

☑ Configure for HTTPS

Authentication: **No Authentication**

Click OK and the project will be prepared. You will likely want to go make tea while this happens. A lot of dependency packages get downloaded during this step and the first build step, noted next.

When you see this screen, hit F5 or Debug->Start Debugging

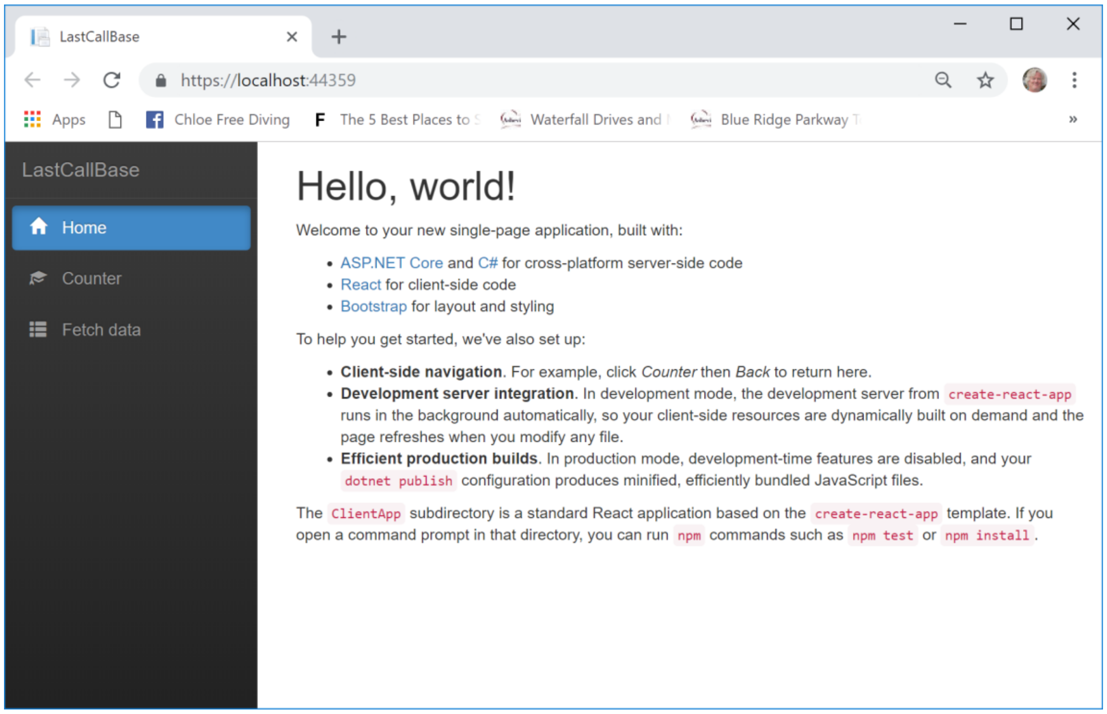


ASP.NET Core

Learn about the .NET platform, create your first application and extend it to the cloud.

Restore completed in 21.74 ms for C:\Users\adrian\Source\repos\LastCallBase\LastCallBase\LastCallBase.c...

You will eventually get to a browser screen that will look like this:



Hello, world!

Welcome to your new single-page application, built with:

- ASP.NET Core and C# for cross-platform server-side code
- React for client-side code
- Bootstrap for layout and styling

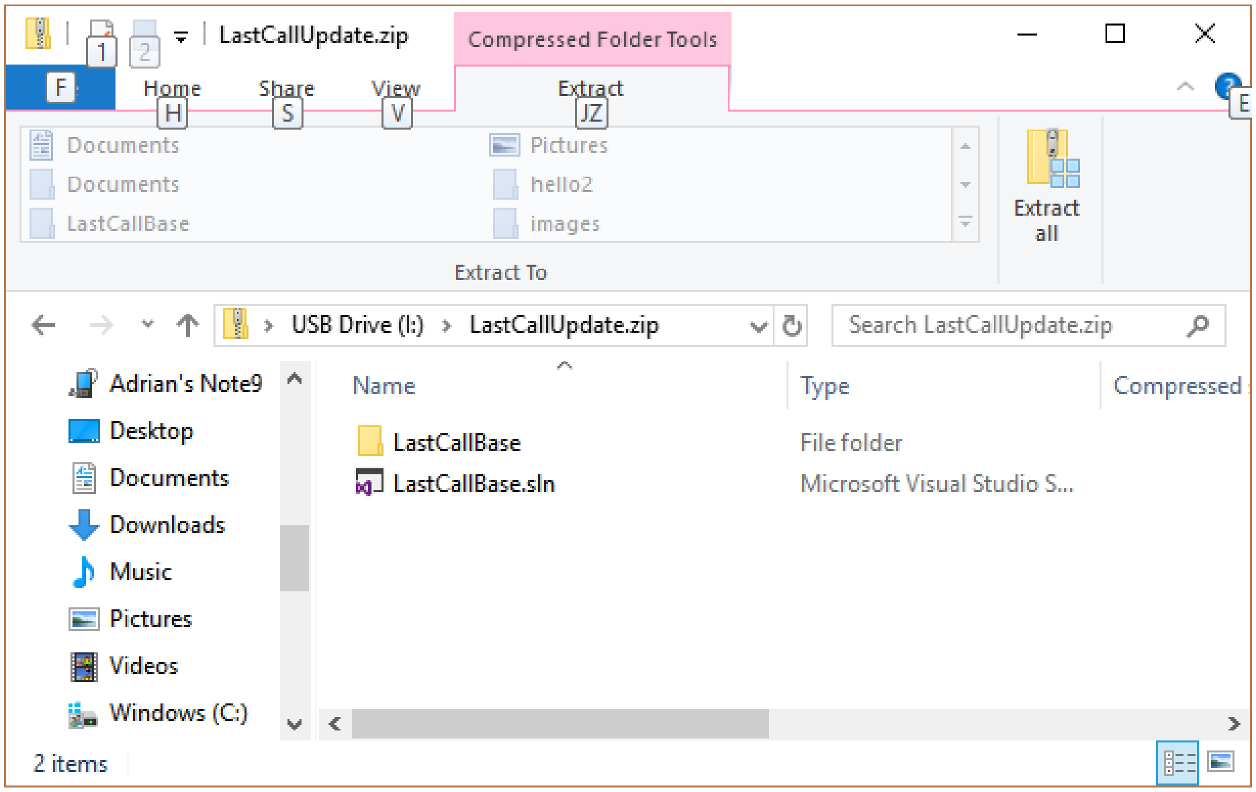
To help you get started, we've also set up:

- **Client-side navigation**. For example, click *Counter* then *Back* to return here.
- **Development server integration**. In development mode, the development server from `create-react-app` runs in the background automatically, so your client-side resources are dynamically built on demand and the page refreshes when you modify any file.
- **Efficient production builds**. In production mode, development-time features are disabled, and your `dotnet publish` configuration produces minified, efficiently bundled JavaScript files.

The `ClientApp` subdirectory is a standard React application based on the `create-react-app` template. If you open a command prompt in that directory, you can run `npm` commands such as `npm test` or `npm install`.

Close the browser window and close Visual Studio.

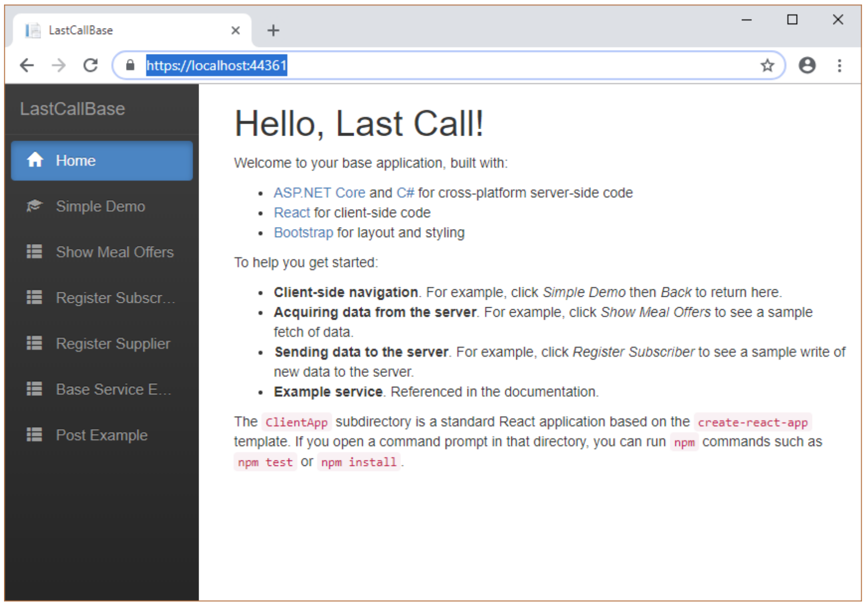
Take the ZIP file LastCallUpdate.zip that I placed on the Slack site and open it. Here's the contents:



Name	Type	Compressed
LastCallBase	File folder	
LastCallBase.sln	Microsoft Visual Studio S...	

Copy these files into the LastCallBase folder that was set up by Visual Studio. Make sure these files overwrite versions that are already there. Bad things will happen if that is not the case.

Restart Visual Studio and choose File->Recent Projects and Solutions->LastCallBase.sln. Hit F5 to restart the build. Your browser should show a screen that looks like this:



Hello, Last Call!

Welcome to your base application, built with:

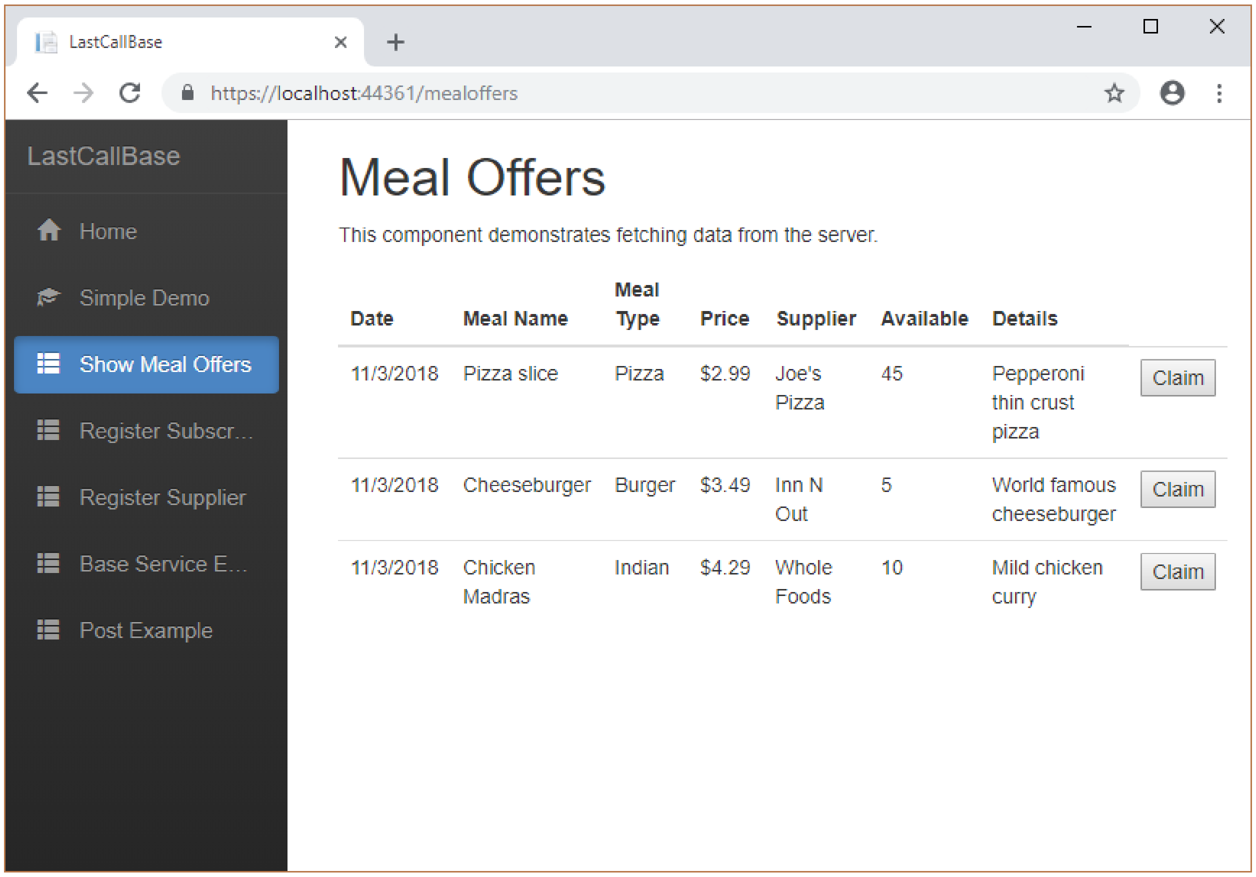
- ASP.NET Core and C# for cross-platform server-side code
- React for client-side code
- Bootstrap for layout and styling

To help you get started:

- **Client-side navigation**. For example, click *Simple Demo* then *Back* to return here.
- **Acquiring data from the server**. For example, click *Show Meal Offers* to see a sample fetch of data.
- **Sending data to the server**. For example, click *Register Subscriber* to see a sample write of new data to the server.
- **Example service**. Referenced in the documentation.

The `ClientApp` subdirectory is a standard React application based on the `create-react-app` template. If you open a command prompt in that directory, you can run `npm` commands such as `npm test` or `npm install`.

To make sure everything is really working, choose the ‘Show Meal Offers’ menu option and you should see screen like this:



Meal Offers

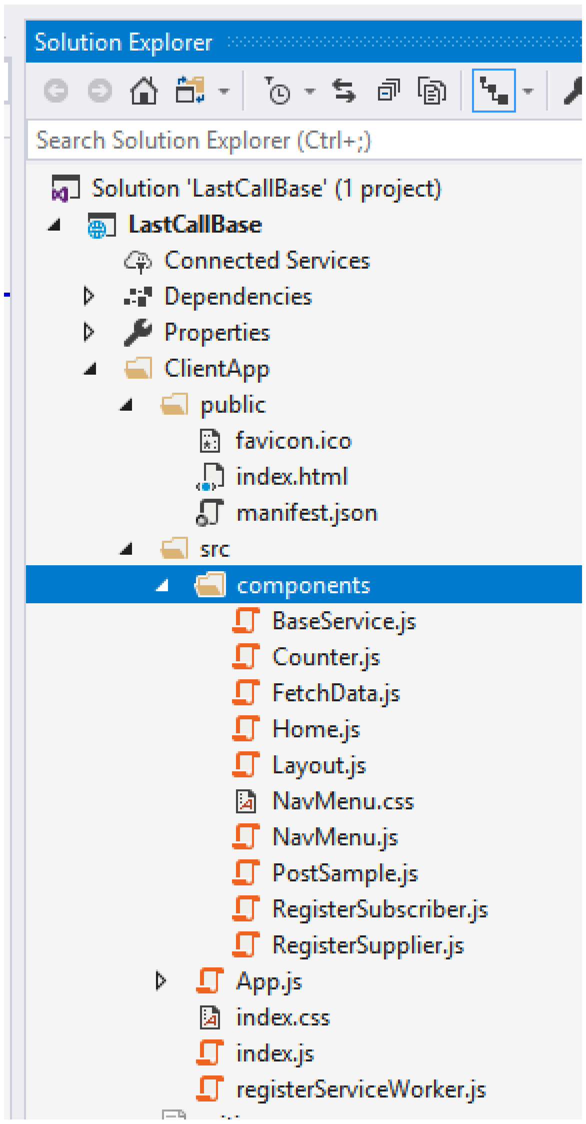
This component demonstrates fetching data from the server.

Date	Meal Name	Meal Type	Price	Supplier	Available	Details	
11/3/2018	Pizza slice	Pizza	$2.99	Joe's Pizza	45	Pepperoni thin crust pizza	Claim
11/3/2018	Cheeseburger	Burger	$3.49	Inn N Out	5	World famous cheeseburger	Claim
11/3/2018	Chicken Madras	Indian	$4.29	Whole Foods	10	Mild chicken curry	Claim

Congratulations, you now have a working ASP.Net + React development environment.

Preliminary Code Discussion

I will be writing up some more notes on this but for those who want to dive in immediately, the interesting files are to be found in the ClientApp/src/components folder.


Solution Explorer

Search Solution Explorer (Ctrl+;)

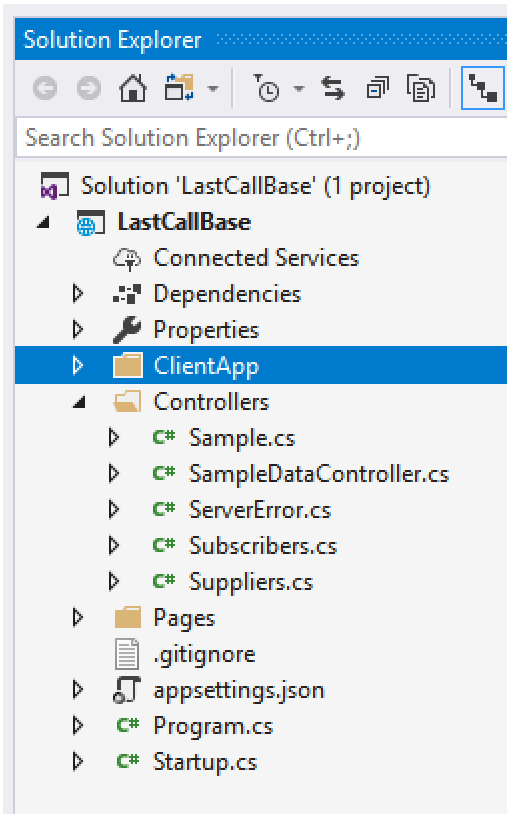
```
Solution 'LastCallBase' (1 project)
  LastCallBase
    Connected Services
    Dependencies
    Properties
    ClientApp
      public
        favicon.ico
        index.html
        manifest.json
      src
        components
          BaseService.js
          Counter.js
          FetchData.js
          Home.js
          Layout.js
          NavMenu.css
          NavMenu.js
          PostSample.js
          RegisterSubscriber.js
          RegisterSupplier.js
        App.js
        index.css
        index.js
        registerServiceWorker.js
```

The interesting ones to look at are BaseService.js, FetchData.js and PostSample.js. Each of these is a React component that includes the HTML (actually JSX) markup for the page plus the actual script code. You can think of each of these components as corresponding to a page. The process of developing a new page consists of taking a copy of one of these components and modifying it with new markup and new code. Data is exchanged with the server using standard HTTP GET, PUT and POST operations controlled by the React 'fetch()' function which we will all get to know intimately.

The React documentation is rooted here: https://reactjs.org/docs/getting-started.html and is actually pretty good.

Of the three components I mentioned, BaseService does a very simple GET of a small amount of data, FetchData illustrates more how the data model interacts with the markup for the page, and PostSample illustrates how you use HTTP POST to send data to the server.

If you are interested to look at the server side of things, look in the Controllers folder:



```
Solution 'LastCallBase' (1 project)
  LastCallBase
    Connected Services
    Dependencies
    Properties
    ClientApp
    Controllers
      Sample.cs
      SampleDataController.cs
      ServerError.cs
      Subscribers.cs
      Suppliers.cs
    Pages
    .gitignore
    appsettings.json
    Program.cs
    Startup.cs
```

Sample.cs implements the web services for the example GET and PUT (BaseService and PostSample components) and SampleDataController.cs implements the service used by the FetchData component. Currently these services are simply sending back static data. As soon as I can get the actual database hooked in then the data will be dynamic.

In order to build a new component and have it show up this sample app there are a few other steps that need to be followed. I'll send out some notes on that in the next day or two.

Initial Development Tasks

The first set of tasks for the client app (ignoring the template and graphic design for the moment) will be, as we've talked about, to figure out the different pages needed and the function of each. A single one of those will be a React component. That component will require (in most cases) a corresponding server side web service that will either accept data or deliver it back as the result of some kind of query. If the data interaction requirements are specified up front then development of the web service can proceed in parallel with the development of the component.

There will also need to be some discussion about a standardized way of handling errors.

I'd also emphasize that there will need to be a lot code sharing. Most of the React components are going to do very similar things, so the process of interacting with the server, error handling and inserting data into the markup is going to look a lot the same between different components. Lots of copying and pasting and teaching each other about new techniques will be in order.

Suggested Template For Client Pages

Here’s a suggestion about how to specify a single client page (screen) that needs to be developed. I think if we draw one of these up for each page it will make it easier for people to pick up an individual development task.

Name and Purpose

RegisterSubscriber – accepts user input and registers the new subscriber details.

Outline Description

This screen accepts user supplied information and passes it to the server with a request to register a new LastCall subscriber.

Design Template

Something that spells out which layout should be used.

Data Input Fields (With Field Names Used In Server Interaction)

Email address (email)

Password (password)

Verify password

‘Friendly’ subscriber name (friendlyname)

Telephone number (telephone)

Checkbox indicating whether subscriber wishes to be notified of food offers by email (emailnotify)

Checkbox indicating whether subscriber wishes to be notified of food offers by text (textnotify)

Checkbox indicating whether subscriber wishes to be added to the LastCall general email list (emailsubscribe)

Default delivery address, consisting of street address, apartment number and zipcode (address)

Submit button

Client Side Processing

1. Email is validated for correct format.
2. Password is checked for compliance with password rules.
3. Password and verify password fields are checked to be the same.
4. Telephone number is verified to be in correct format.
5. Address is prepared as a single, comma separated, concatenated list of fields.
6. Data is sent to server as a single form transmission (HTTP POST.)
7. Server response received and either error is displayed or congratulatory message displayed.
8. If registration succeeds, user is automatically logged in.
9. User can click on ‘See Food Offers’ button and proceed to that screen.

Possible Errors

1. Invalid email address format.
2. Passwords do not match.
3. Password does not meet compliance rules.
4. (From server) Email already registered.
5. (From server) invalid delivery address.

Server Processing

1. Email address verified as not already in use. (Return an error.)
2. Subscriber details saved to database. (Return an authentication token to indicate the subscriber is now logged in.)

Notes

Any additional notes, diagrams, comments.

Code Discussion

Here's a few notes on things I found while building the samples in the example app that you now (hopefully) have set up. These are in no particular order.

Basic Elements Of A React Component

The structure of a React component is fairly simple with a few basic things to be remembered:

1. Every component has, first, its constructor called and its render() function in order to deliver the HTML (plus embedded data) to the browser. If you have these two functions you have the basis of a component.
2. A React component does not have local property variables per se. It has a 'state' in which you must keep values that are to be shared between different functions. The syntax for doing this is illustrated in the example components and is worth studying. Essentially you do something like the code below in the component constructor. This establishes component state variables called username, password and errorInfo (which has its own fields) and sets initial values for each.

```
this.state = {
    username: '',
    password: '',
    errorInfo: {
      errorCode: 0,
      errorMessage: '',
      errorDescription: '',
      statusMessage: ''
 }  // Standard server error object
 };
```

Notes On React/JSX/Javascript

1. Initially the source code for a component looks very confusing because it is a mixture of HTML markup, good old Javascript and the more specialized JSX markup (usually identifiable because of all the curly braces { and }). It does take a while to get used to, so be patient.
2. Also, in a lot of cases, it is not clear which API calls belong to React and which belong to Javascript (in its more modern form.) For example, fetch() is a Javascript API rather than a React. The biggest downside of this is searching for information in the documentation. For our purposes I think we can just treat it all as one big happy API family.
3. There is some weird stuff going on with respect to the handling of identifiers, in that property names defined as part of a class in the server side code, for example 'ID' have to be referenced in the client code using a lowercase identifier ('id'). More strange is the fact that a mixed case identifier with a lowercase initial letter, for example 'mealName', is recognized just fine on the client side. I have not been able to figure out what 'rules' are in use. For now I suggest staying away from names with an uppercase initial letter.
4. Remember that a lot of operations in React are asynchronous, so debugging client side code can be quite frustrating. The debugger will break (best case) when the operation is initiated and has no way of understanding how to proceed while the async operation is underway. You have to resort to good old logging printouts in order to work through cases like this.

5. The simplest way to insert a breakpoint in client side code is to write the statement ‘debugger;’. This works on all browsers. Using Visual Studio to set breakpoints doesn’t work without mucking with your browser’s settings.
6. Referencing fields within an HTML page has a whole selector style capability which looks like it is drawn somewhat from jQuery. The simplest way to reference a field on a form is, for example, to include an `<input type='text' name='username' id='username' />` field. You can reference the value of this field in client side code using the identifier ‘#username’. However, this is often not the way you want to do it, the React documentation on forms shows better ways.
7. Every iterated element on a page (such as <tr> or <li>) must include a key tag, as in ‘<tr key=1>…<tr key=2> etc. Most often it is going to be the server that generates these keys and you often won’t need to use them. The FetchData.js sample I built shows you an example of when you do need to use these key fields.
8. In JSX you can only return a single element, so if you have multiple top level elements in the markup you must wrap them with a <div></div> or similar.
9. Conditionally include markup by doing something like this: `<p>Text offers? {person.textoffers > 0 && <span>Yes</span>} {person.textoffers == 0 && <span>No</span>}</p>`

Sending Data To The Server

There are several ways to format a server request (all using fetch()) and the documentation shows examples of each: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

The simplest way to do it seems to be to use a form object (this is illustrated in the PostSample.js sample I built.) Certainly it makes the server side processing super easy. Down the road we may end up needing to send JSON to the server for something more complicated but we can deal with that if and when we come across it.

Issues Needing Research And/Or Further Specification

1. We need to specify standard snippets of code for the use of the fetch() function and how to handle error processing. We should have snippets for GET and POST.
2. Someone needs to research how to best publish the app and exactly where we would publish it while we're in development mode.
3. We need to research how best to use the so called 'Error Boundaries' in React.
4. We need to research and document how to maintain a global state item (specifically the 'logged in' authentication token.) Or, someone needs to research exactly how credentials should be generated and used.