Timesheet Application Technical Guide

by Youngeun Seo and Jimmy Huang

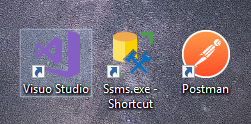
**Summary:**

The timesheet application is application designed for employees to process time reporting procedure more efficiently and effectively. The application was referred to, during the development phase, as project D where the D stands for digitize. Due the size of the project, a complete file is provided with all necessary documents. This technical guide severs to provide an explanation regarding the setup as well as the logics behind the application.

**Configuration**:

While the application should be able to run on other environments, doing so may require translation of syntaxes to that specific environment. For instance, some MS SQL syntax differs from MySQL.

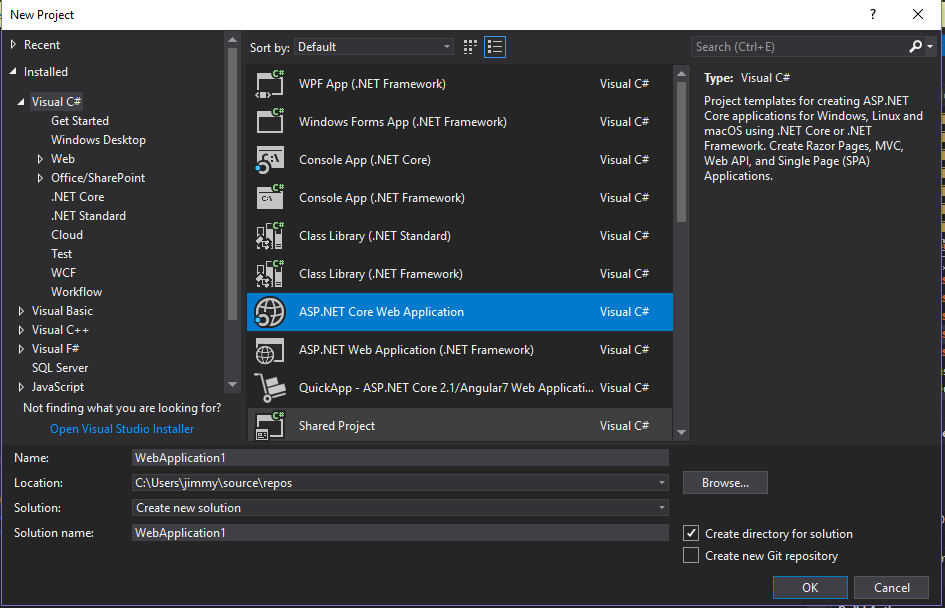
* Database: Microsoft SQL Server 2017
* While it is possible to use just the command tool, the interaction with MS SQL is primary conducted through the SQL Server Management Studio (SSMS)
* IDE: Microsoft Visual Studio 2017
* Optional: Postman to send API requests
* \* assumed installation of Node.js and npm package manager



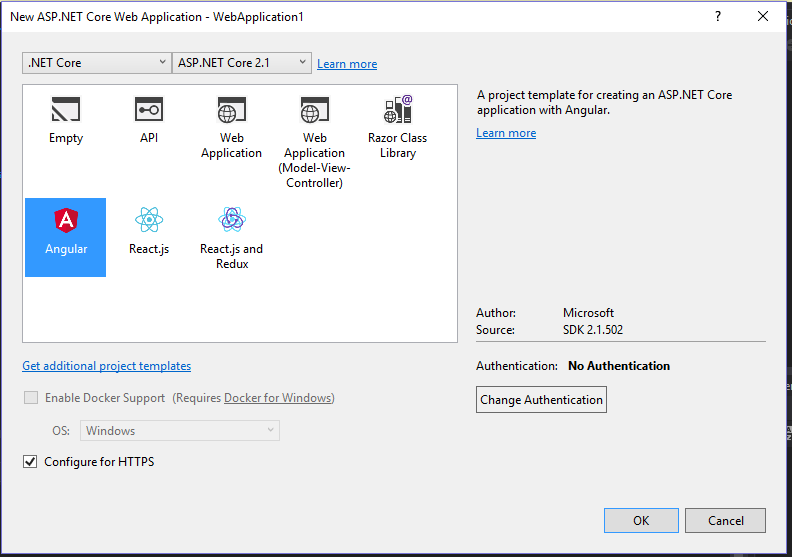
**Setup:**

The initial setup of the application is done inside Visual Studio. This is due the complete development environment Visual Studio, mainly the default IIS Sever Express which provides the sever layer for the application to operate on. Going forward, the application should be migrated out of the visual studio environment. The critical files of application mentioned below should have no dependencies with visual studio environment. Thus, porting the application is mainly consist of simply moving these critical files.

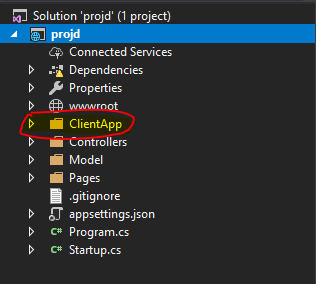
1. Create new Project inside Visio Studio
   1. Create under ASP.NET web application



* + 1. **Be aware of the name in which the project is named, the namespace reference inside the provided files are for “projd”**
  1. Configure to .NET Core, ASP.NET Core 2.1, Angular base template, Configure for HTTPS

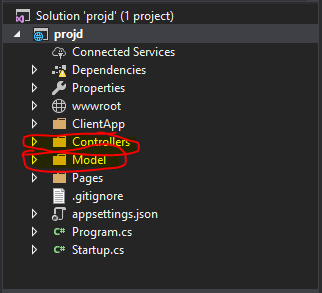


1. Since the Angular base template on Visual Studio uses Angular 5, we remove the complete “ClientApp” from the project and replace it with the provided “ClientApp”

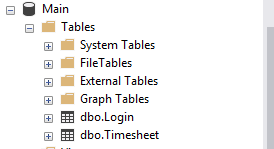
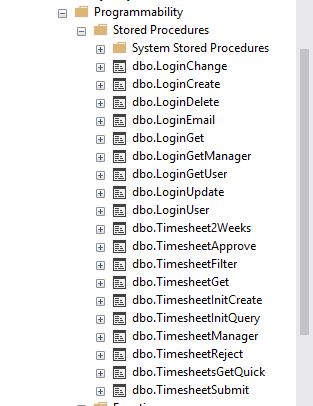


* 1. Using Commander Console, Navigate to the “ClientApp” folder, execute “npm install” to install all the dependencies.

1. Replace files inside “Controller” folder and create “Model” folder to move the files into

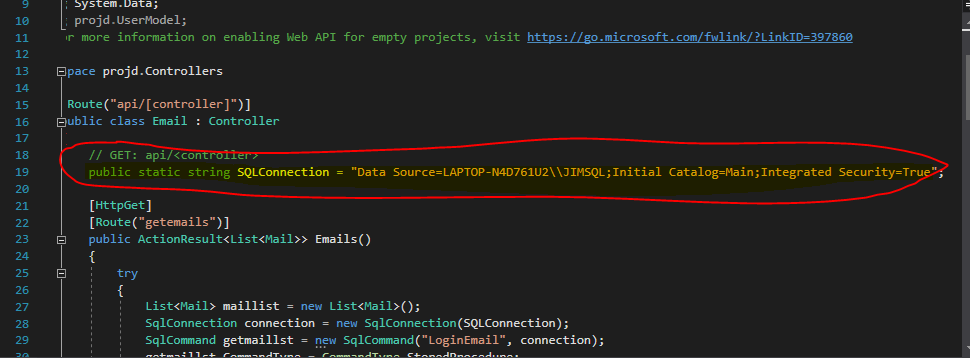


1. Create table and procedures inside the SQL database. Please refer provided files under “SQL”. All the provided SQL files are SQL CREATE statements.

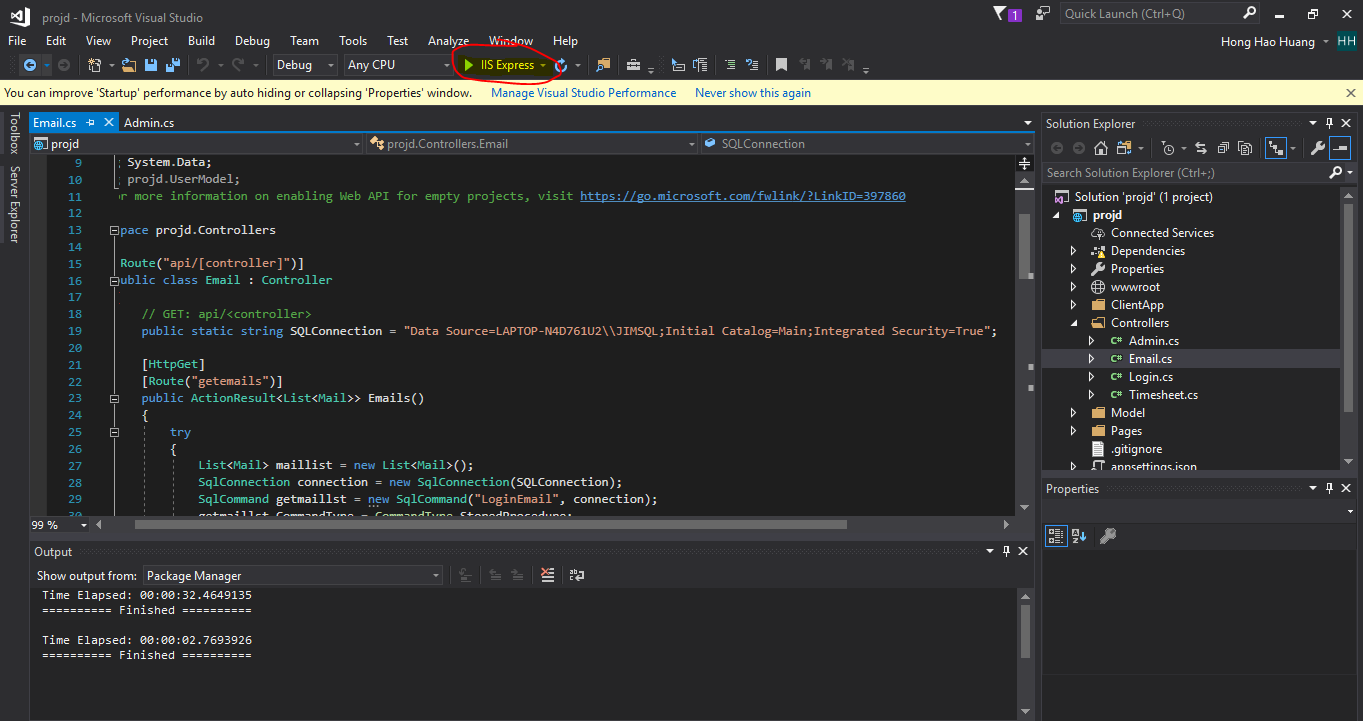
 

* 1. There are 2 data sets in which the SQL procedures are based on, “Login” and “Timesheet” Be sure to modify the table reference name inside the procedure if the table are named differently

1. Replace the “SQLConnection” string each of the files inside “Controllers”. The replacement should the SQL database string.

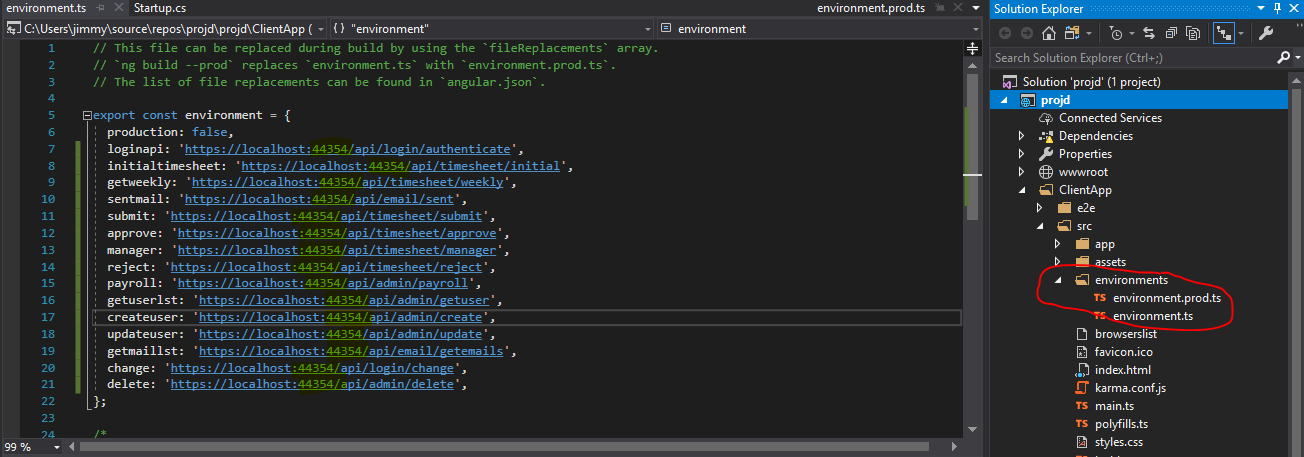


1. Execute the project by clicking on the “IIS Express”. F5 should also work as visual studio will build the solution, create IIS Sever, and open the application on your defaulted website.



* 1. Once IIS Sever is open, the application can be run from “ng serve -o” inside the “ClientApp” folder. This should make debugging the API easier.

1. Replace the API address ClientApps/environment/ environment.ts to the correct local link

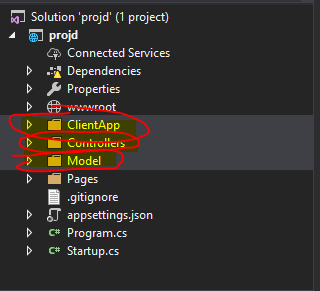


1. Inside Visual Studio, open Packet Manger Console and “Install-Package Microsoft.AspNet.WebApi.Cors”
   1. Ref <https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api#enable-cors>

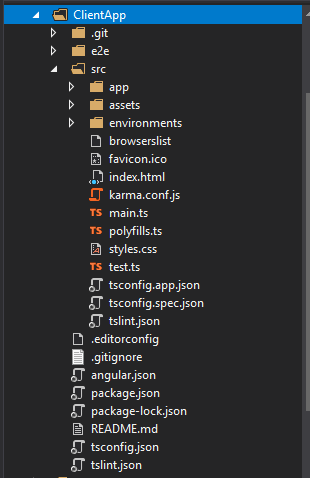
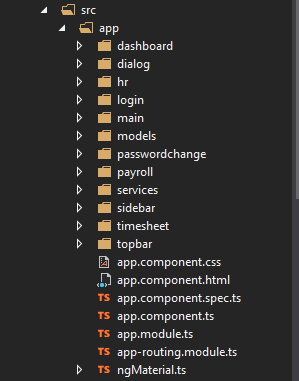
**File Structure:**

The critical files are as follows:

* ClientApp – Client-side styling and scripts
* Controllers – Sever side API
* Model – Sever side objects for APIs



The client-side of the application, the presentation of the website, is constructed using the Angular 7 framework. The file structure inside the ClientApp is based on the Angular workspace and project file structure. Please refer to <https://angular.io/guide/file-structure>.



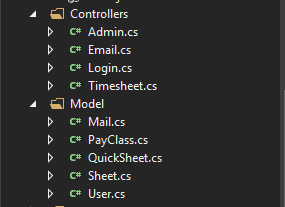
All pictures are stored inside the src/assets file

A brief breakdown of src/app file, which contains app's logic is as follows

* Dashboard – initial dashboard interface
* Dialog – dialog base on the success/error of the HTTP request
* hr – the interface for HR user accounting use to edit and create user
* login - login interface
* main – container for the UI after login
* models – Objects used in the client-side
* passwordchange – User password change interface
* payroll – payroll interface
* services – main storage for the scripts which are used to send HTTP requests
* sidebar – sidebar interface
* timesheet – timesheet interface
* topbar – topbar interface

\*interface refers to HTML, CSS styling as well as scripting specific for that component

Indi



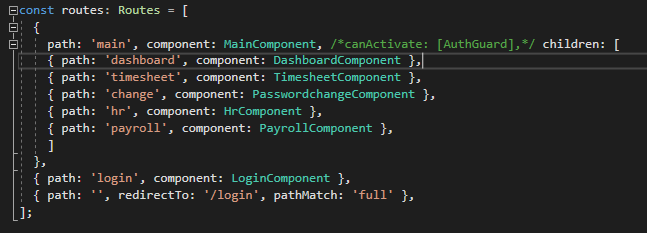
**User Type:**

There are 4 main user types:

* Employee – can view, edit timesheets. User action is only “submit”.
* Manager – can view timesheets. User actions are “reject” and “approve”.
* Payroll – can only view a summarized list of timesheets
* HR – can only view user list. User actions are “Update”, “Delete”, and “Create” users.

**Routing**

Please refer to “app-routing.module.ts”. In general, once the user logs in, they will be directed to the “main” container which decorates the “sidebar” and “topbar”. The main display is “dashboard” before the user directs to either “timesheet”, “change”, “hr” and “payroll” base on input and user type. See User Guide for the intended routing



**Security:**

Most of the security revolves around Boolean checks on client side and sever side. The client-side mainly validates the stored user information on session storage to allow a HTTP request. The sever-side then validates the requested information before sending processing and responding to the request.

Client-side AuthGuard was not implemented as there is already Java Web token validation occurring on APIs. AuthGuard can be implemented as enhancements. The JWT token only checks the “EmployeeID” and “EmployeeType” field of request. Further enhancements could be to include time stamp for the JWT token.

**Additional Libraries:**

These libraries are for client side and should install automatically once “npm install” was executed.

* Angular Material - Material Design components for UI
* SheetJS – JavaScript and Excel library, used for Payroll’s excel export

**Additional Notes:**

* As this was our first time creating a website application, the codes are messy as we experiment and applied various techniques. For instance, the use of Observables was mastered late into the project. The use of \*ngFor was also mastered late and the styling for “timesheet” could especially make more use of \*ngFor. There was also much discussion regarding the design of the application. As a result, there are many duplications of storage of data.
* Code clean up should also be conducted as many imports had become ineffective
* Feed back from end users should be gather and application should be modified based on their feedback
* Delegation for managers is a further enhancement for the application

**Data Set:**

The data set revolves around 2 data sets, Login table and Timesheet table.

For Login set, “EmployeeID” is the primary key

For Timesheet set, “Timesheet ID” is the primary key which is automatically generated. The timesheet data set also has a foreign key “EmployeeID” which is from Login set

Login Set

Timesheet Set

