**INSYS.SUITE TECHNICAL DOCUMENTATION**

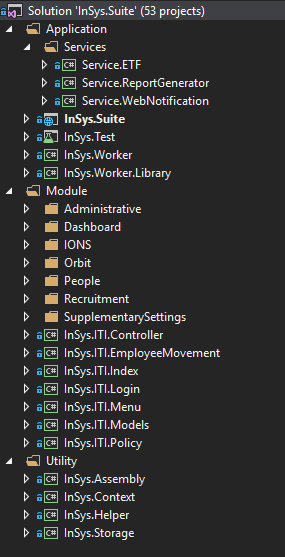
**TABLE OF CONTENTS**

1. Folder Structure
2. InSys.Suite
3. Pre – requisite libraries
4. How the system work
5. How the data flow in the system
6. How to create a module
7. How to create detail tab
8. Directives in the system
9. Form Controls
10. Development & Production Pre – requisite

**Rossu M. Belmonte**

Systems Developer

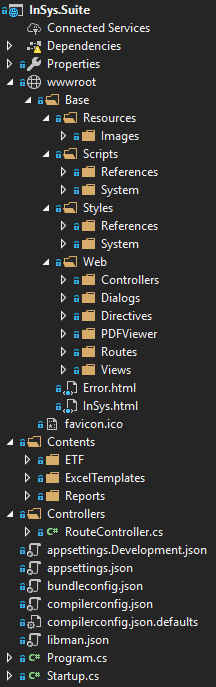
1. **FOLDER STRUCTURE**

* **Application** – The Application folder contains the main project of the solution.

**Services –** The Services folder contains Windows service for the InSys.Worker Application.

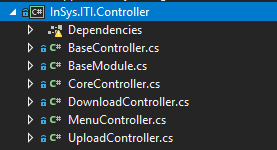
* + - * **Module –** The Module folder contains the necessary modules required for InSys.Suite. In this figure, the modules directory is organize in who is their parent module. If the module doesn’t have a parent it is place in root directory of Module. It is the tree view form of the navigation menu inside the system.
* **Utility –** The Utility folder contains libraries globally used in the main project a

1. **InSys.Suite**

* **wwwroot** – Contains the client side files, when we say client side (HTML, JS, CSS)
* **Base** – This is the folder that contains the Base files.
* **Resources** – This is the folder that contains images used in the system.
* **Scripts** – This is the folder that contains Javascript files used in the system.
* **References** – This is the folder that contains 3rd party Javascript libraries used in the system.
* **System** – This is the folder that contains Javascript files used in the system mainly made of AngularJS
* **Styles** – This is the folder that contains Cascading Style Sheets used in the system.
* **References** – This is the folder that contains 3rd party Cascading Style Sheets used in the system.
* **System** – This is the folder that contains Cascading Style Sheets designs used in various modules and templates.
* **Web** – This is the folder that contains files used to build a module.
* **Controllers** – This is the folder that contains all Angular Controllers for the modules.
* **Dialogs** – This is the folder that contains all the HTML templates used in pop-up dialogs.
* **Directives** – This is the folder that contains all the HTML templates used in custom Angular Direcives.
* **PDFViewer** – This is the folder that contains all the necessary files needed when generating report.
* **Views** – This is the folder that contains all the HTML templates used in modules.
* **Error.html** – not used file.
* **InSys.html** – This is the landing page if the browser has session.
* **Contents** – This is the folder that contains various files such as Excel Templates, Report Files and storage for uploading files.
* **Controllers** – This is the folder that contains the RouteController.cs which is the first route access on first load of the page.
* **appsettings.json** – This is the file that contains the Configuration for the system in JSON format.
* **Bundleconfig.json** – This is the file that contains all files needed to be bundled as one in JSON format.
* **Program.cs** – This is file that initializes the WebHost.
* **Startup.cs** – This is the file called when the WebHost initializes.

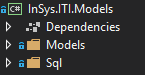
1. **PRE-REQUISITE LIBRARIES**

**InSys.ITI.Controller** – This is the library that contains various controllers used in the system.

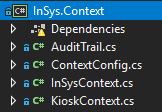


* **BaseController** – This is the controller that contains generic or widely used Api Methods for the system.
* **BaseModule** – This is an Abstract Class that implements override able methods called in BaseController methods.
* **CoreController** – This is the controller responsible for decrypting the data sent from the client-side Dataservice. This is also responsible for Instantiating a module accessed by the user.
* **DownloadController** – This is the controller responsible for downloading files from the server to the client.
* **MenuController** – This is the controller responsible for loading the list of menus, loading rights for the menus and loading the schema for the table.
* **UploadController** – This is the controller responsible for uploading files from the client to the server.

**InSys.ITI.Models** – This is the library that contains all the Table in the database in object form. This also contains all the queries used in modules.



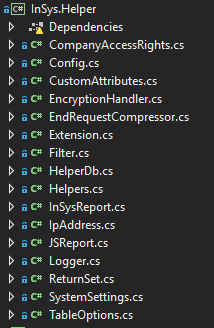
**InSys.Context** – This is the library that contains the configuration of the database connection and also the bridge between the models and the database.

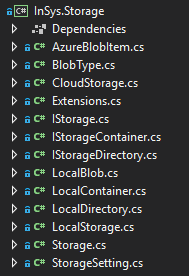


* **AuditTrail** – This is the class responsible for auditing EntitfyFramework transaction such as Add, Update and Delete. **Note**: If you use ADO.NET the insert, update and delete, the AuditTrail will not be triggered.
* **ContextConfig** – This is a class that sets the SQLConnection for the InSys.Suite and for the InSys.Kiosk.
* **InSysContext** – This is a class that connects the database and models.
* **KioskContext** – This is a class that connects the InSys.Kiosk database and models.

**PRE-REQUISITE LIBRARIES**

**InSys.Helper** – This is the library that contains classes and functions used widely in the solution.

* **CompanyAccessRights** – Not used anymore.
* **Config** – Contains all the properties defined in AppSettings.json
* **CustomAttributes** – In the name itself, it contains custom attributes used in the system.
* **EncryptionHandler** – Contains Encryption used for password and decryption of rights for the modules.
* **EndRequestCompressor** – Is a middleware for Api Request’s that compresses the data returned from the TaskResult.
* **Extension** – This contains extensions mainly used for EntityFramework queries.
* **Filter** – This is a class configuration for filter types and function that builds all the filter types in to a string “Where” query.
* **HelperDb** – This is a DbContext used in this library.
* **Helpers** – This contains various functions used widely in the solution.
* **InSysReport** – Not used anymore.
* **JSReport** – Not in used but ready.
* **Logger** – This is a class that logs error messages.
* **ReturnSet** – This is a class used when returning data to the TaskResult
* **SystemSettings** – This is a class that gets configuration values from the tSystemSettings.
* **TableOptions** – This is a class that contains property options for the DataGrid.

**InSys.Storage** – This is a library used for Read/Write of files both for physical server and azure storage. This library is made by a senior dev and it’s only purpose is for debugging or if you have any customization. Overall, this is stable.

1. **HOW THE SYSTEM WORK**

On the system’s first run, the **Program.cs** initiates first, it builds the **WebHost** then calls for the **Startup.cs** to initiate.

On the **Startup.cs**, the first to run is the **Startup constructor**, next is the **ConfigureServices**. This is where the **StorageSetting**, **MenuCollection** and **SystemSettings** are initiated. This is also where the **InSysBundles** is called.

In **InSysBundles** by default it bundles the necessary files inside the **Base** path. Later in this story telling I will explain how it bundles if you have custom files for a specific company.

Back to **ConfigureServices**, this is also where we configure the **EndRequest** **Key** and **Salt** for encryption and decryption of data. This is also where we configure the **Antiforgery** header name for the server.

Next to **ConfigureServices** is **Configure**, this is where we setup the **InSys.Storage** path, and Initiates the **JSReport** but we haven’t use this because it is hard to configure a report design. But this is ready in case you have a **JSReport** template, you just need to call the right function at the right time and place. After the configurations in this and that, we also set here what middleware’s we are going to use.

When the **Configure** is done, the first route we land by default is the **RouteController** and the first method to run is the **Index**. In this method we set the default **BasePath** value as “Base”, after that we check for any **Session**. Why? Because this **RouteController** is always called if you refresh your browser. Now if the **Session** has a value, we check what company you are using. Why again? Saying that you have a **Session** this means you have login to the system and selected a company you are going to use. If you didn’t select a company this means you are only designated to a single company. Some users are allowed to access other company such as HR. Back to the topic, after selecting what company you are using, we are going to find its code in the database. The Code itself “MUST” be always the same name of the folder you are going to create if you are going to have a customization for the system. Now if that folder exists it will now be the value of our **BasePath**. If not, the value of our **BasePath** will not change. Now let’s say our **BasePath** has change, we will call the **InSysBundles** again to generate new bundles for our custom files. What files? If we are having a customization first we need to create a folder named after the Code of the company we are going to customize. Then, we copy the contents of the **Base** folder which is our base files. Now that we are done with the bundling, this will now call the **InSys** function to return the template we are going to use.

In the **InSys** function we are going to read the file **InSys.html**, this is our default landing page. If we have a session we load the **Index.html**, if not the **Login.html**. Now inside we are going to replace specific strings in pattern and append some **Javascript** dependency for the system, after that we return it as an **IActionResult,** this is a **ContentResult** meaning it’s a web page.

Let’s proceed to the **Login** page. Upon login, if we are authenticated and we have rights to more than 1 company, the user is enforce to choose a company. After selecting a company we are now refreshing the page to be redirected to the **RouteController**, as I have explained earlier the **InSys.html** will now be loaded. When this template is loaded, the **Config.js** is now called. Inside the **Config.js** is an angular script responsible for storing important data such as **SessionID, HTMLTemplatePath,** and **X-XSRF-TOKEN.** This is also where the **Salt** and **Key** is pass to the **DataServices.js**.

**Index.js** is called from there, it will request to the server some necessary information like **BrowserSession**, **CompanyList** and **MenuList**. The **BrowserSession** contains the user’s information, while the **CompanyList** is the collection of company the user has rights to access. The **MenuList** is the collection of all the modules the user has access. Inside **Index.js**, **BaseController** and **MenuController** is inherited from the **Controllers.js**, it also connects to the SignalR server after requesting the necessary information from the server.

**NAVIGATION**, when navigating the menu the first to run when you move to a menu is the Route of the Menu. Let’s say the User Menu, the route it will call is **UsersRoute.js**. Inside the route it will request for the Rights of the user, then if the URL has a parameter id it will load the **UsersRecord.html** else it will load the **Users.html** together with the **Users.js**. After loading the **Users.js**, it will check if there is a parameter id from the URL, if none it will call the **LoadList** method else it will call the **LoadRecord** method.

1. **HOW THE DATA FLOW IN THE SYSTEM**

When calling a request from the client to the server it is always in the form of **POST** and the data in **JSON** format. The data first run through the **DataServices.js** then the data is encrypted using the salt and key and posted to the server. Now in the server the data will run through the **EndRequestCompressor,** now in the diagram don’t be confuse, the **EndRequestCompressor** will await the return from the **CoreController.** Now in the **CoreController** it will decrypt the data and check if the data contains a parameter named **MenuCode**. If it is found then it will create an **Instance** for the User module, then it will execute the method through the **Instance** created. If there is no **Instance** created it will check if the **BaseController** is **Inherited** to another **Controller**. If not, then it will execute the method else it will return an error.

After the method is executed it will return to the **CoreController** and encrypt the data, then return to the **EndRequestCompressor** and compress the data and return to the client. Now in the client side or **DataServices.js** it will unencrypt the data and return to the **Angular Controller** of the module.

1. **HOW TO CREATE A MODULE**

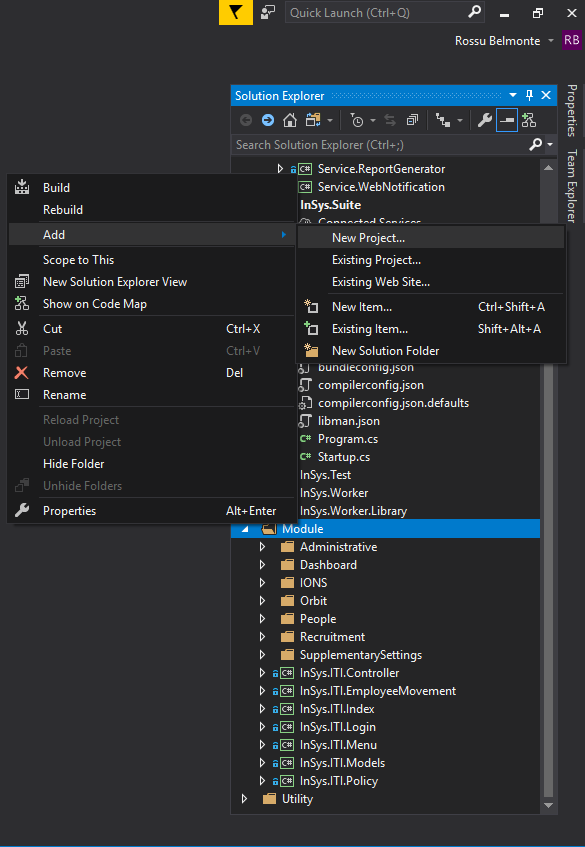


Figure 1: Create a new project, right click on the Module folder.

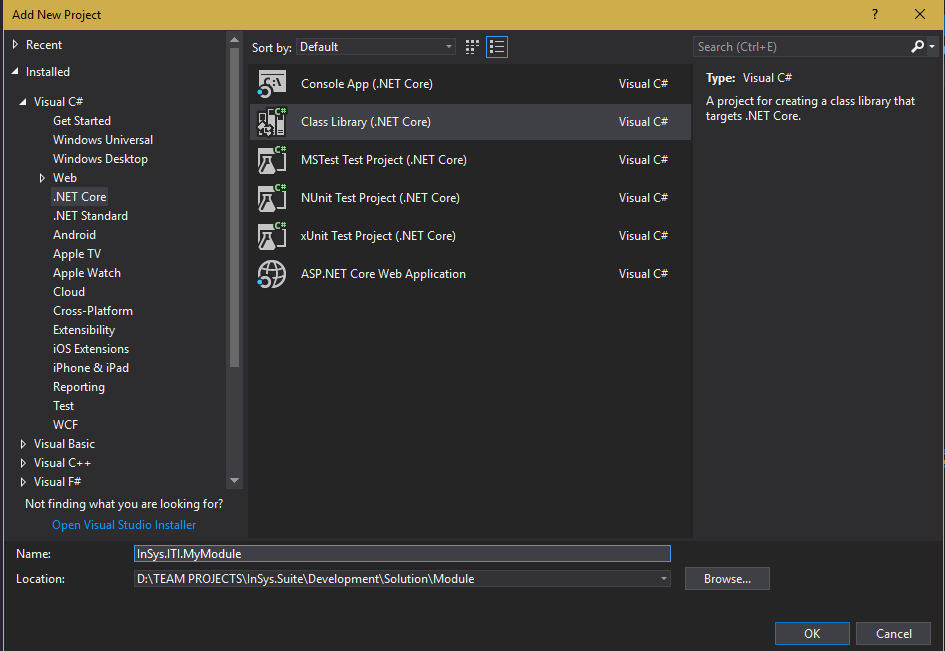


Figure 2: Select Class Library. Make sure it is a .NET Core project

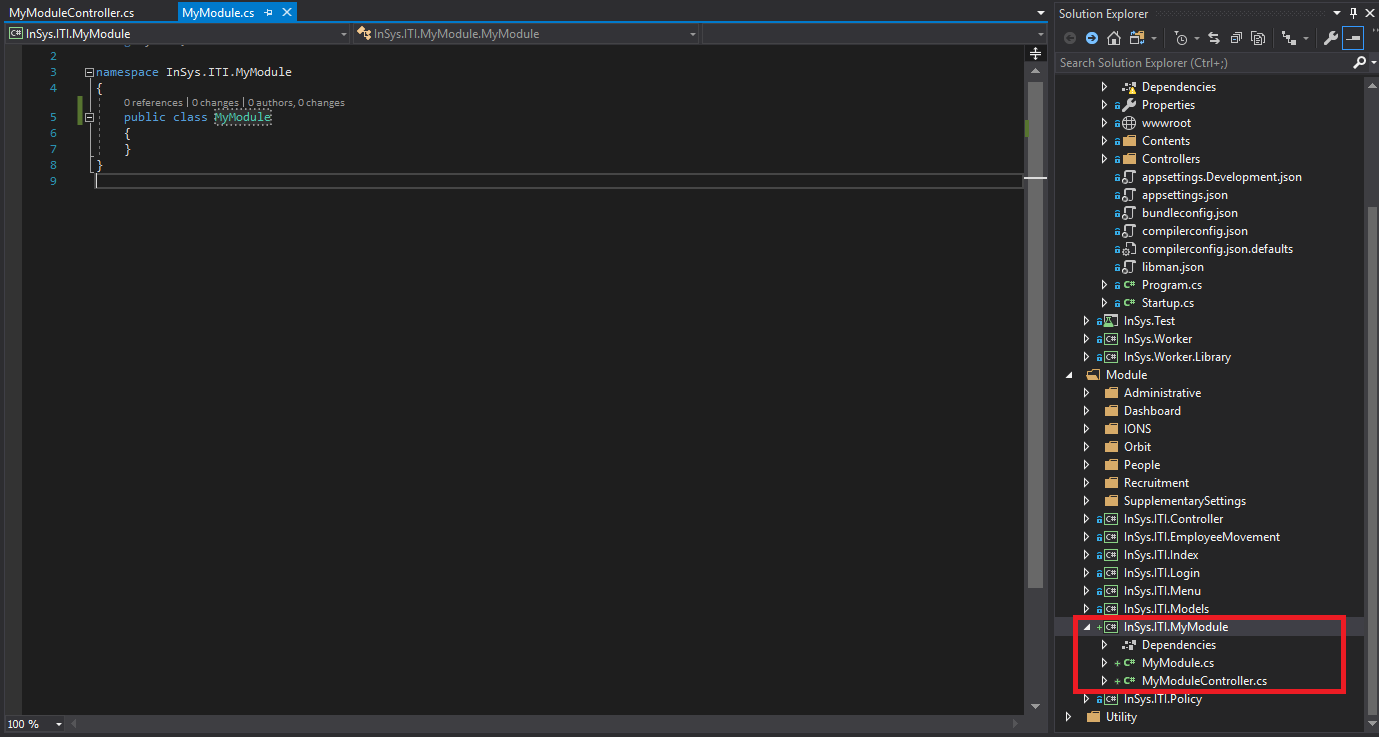


Figure 3: Rename the default class name to the name of your module. In our example I named it MyModule

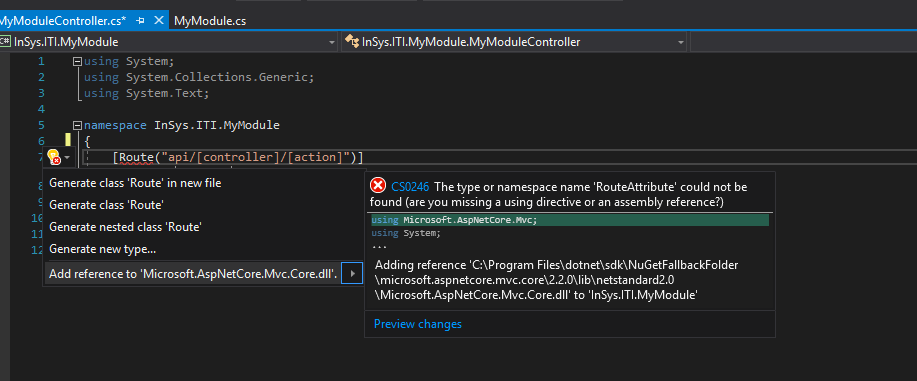


Figure 4: Add another class named your module name + Controller. This is where all our post request will run through. Then add the attribute Route just like in the figure.

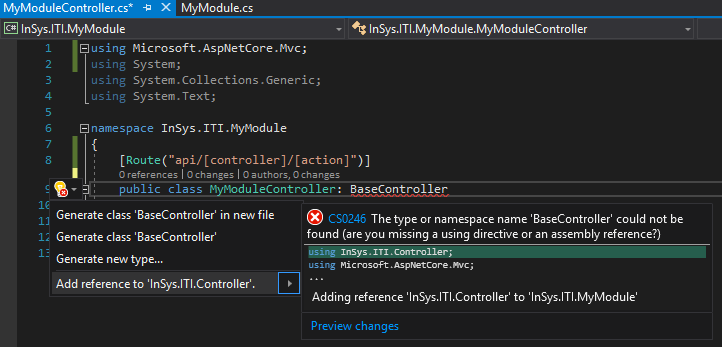


Figure 5: Now Inherit the BaseController. This is where our request will first go through before looking for the API in our defined Controller. The defined controller is created just in case our API request is customized.

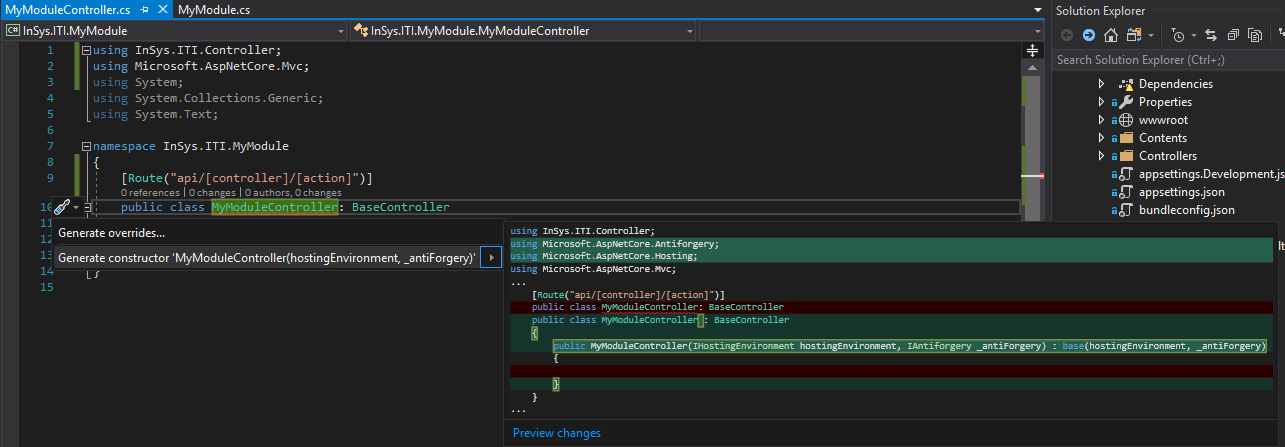


Figure 6: Generate the required Constructor for our defined Controller.

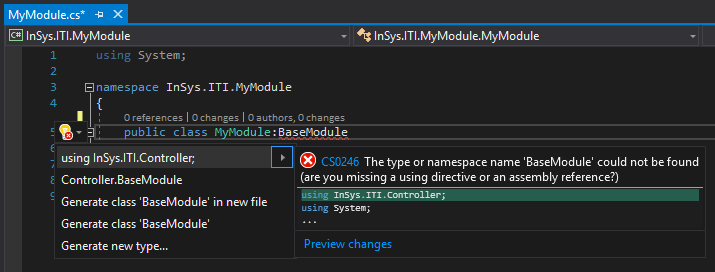


Figure 7: Now in our Module Class, inherit the BaseModule. This is where all our pre-defined method located.

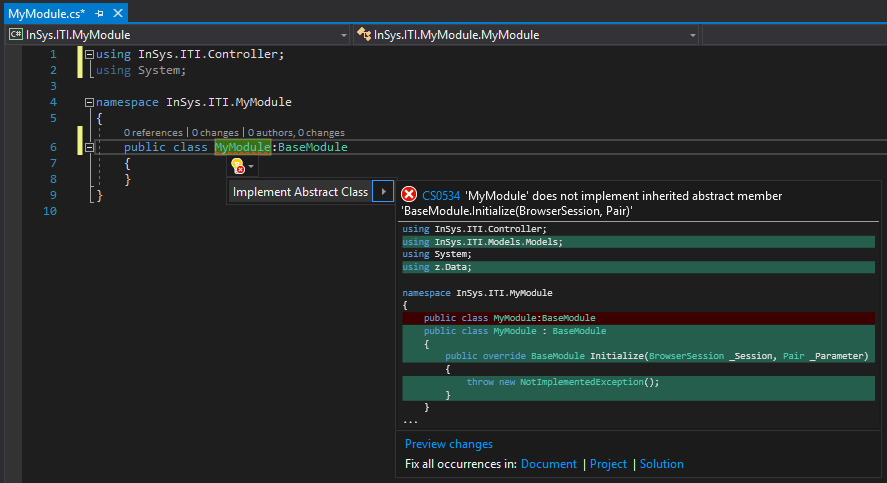


Figure 8: Now we will implement the required method for our module

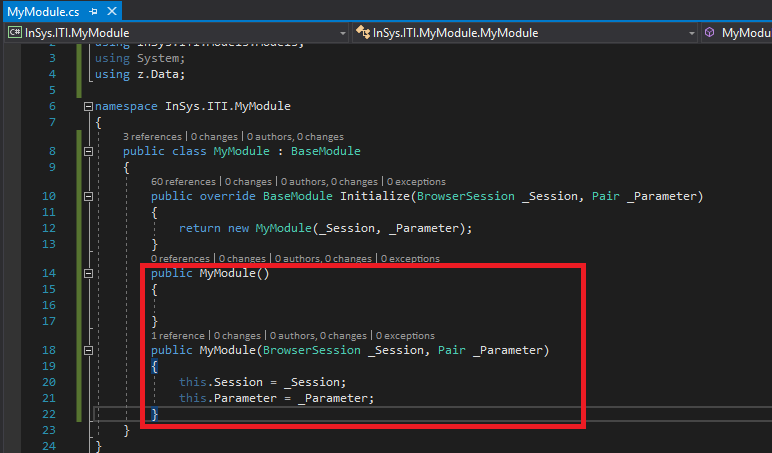


Figure 9: Now we will create a blank Constructor and an overload Constructor to be called by the Initialize

The blank constructor is required when instantiating. This project we created will be Instantiated when we call this module

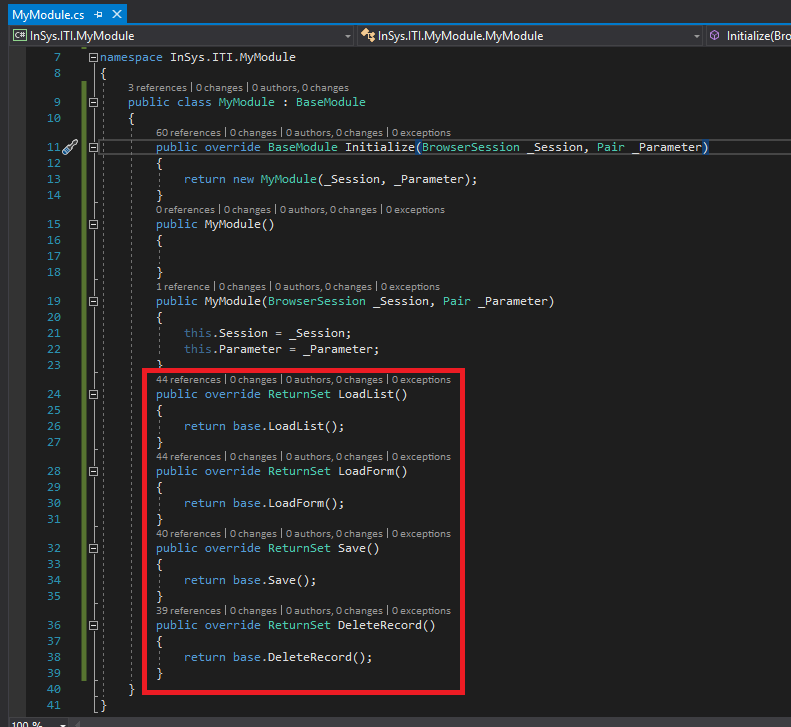


Figure 10: Now we will override all the method we are going to use. All the override able method is found in the BaseModule

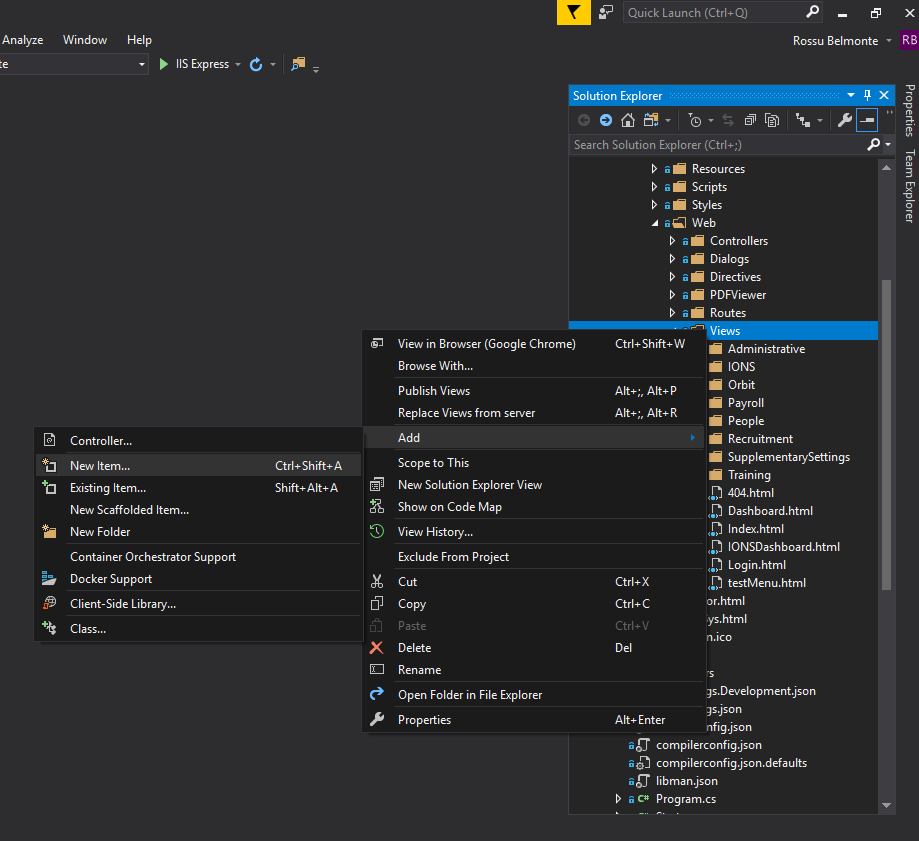


Figure 11: Now we will add a HTML file in the wwwroot > Base > Web > Views directory.



Figure 12: We will name it MyModule.html and Add another HTML File named MyModuleRecord.html

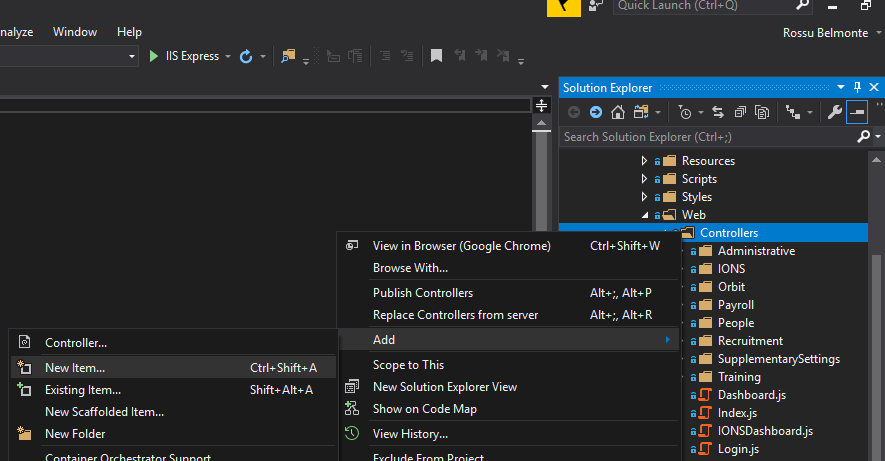


Figure 13: Next we will create a Javascript file inside wwwroot > Base > Web > Controllers director.

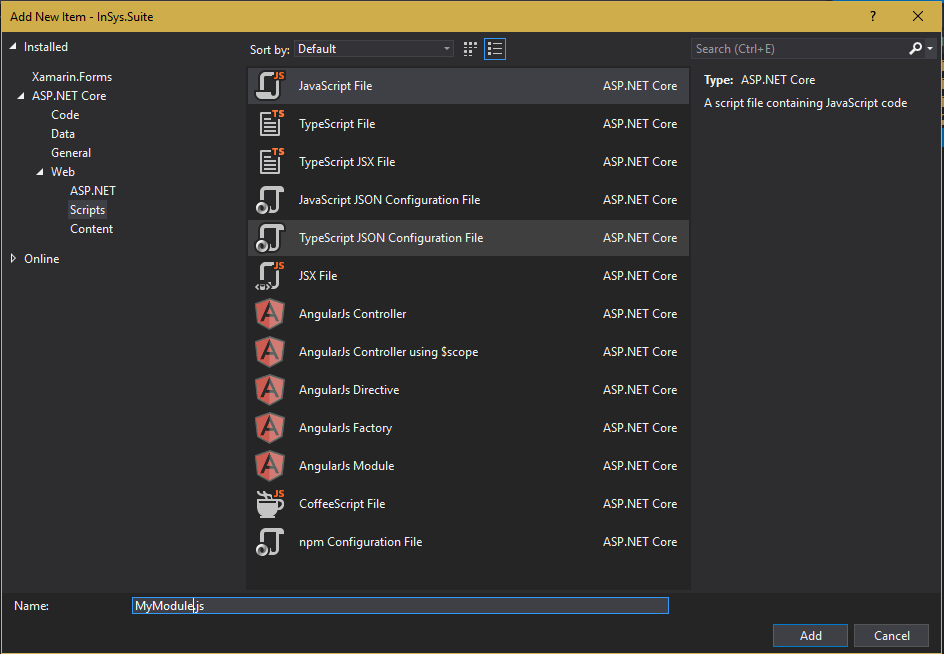


Figure 14: We will name it MyModule.js

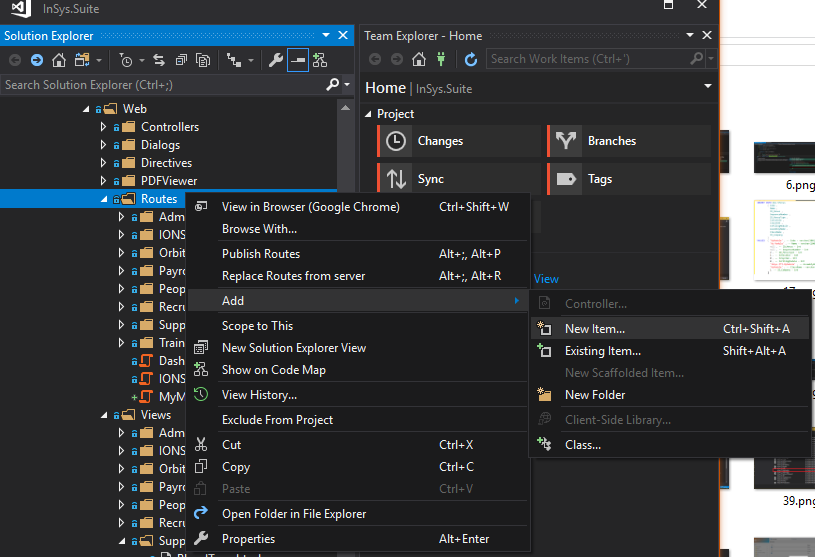


Figure 15: Now we will add another Javascript file to the wwwroot > Base > Web > Routes directory.

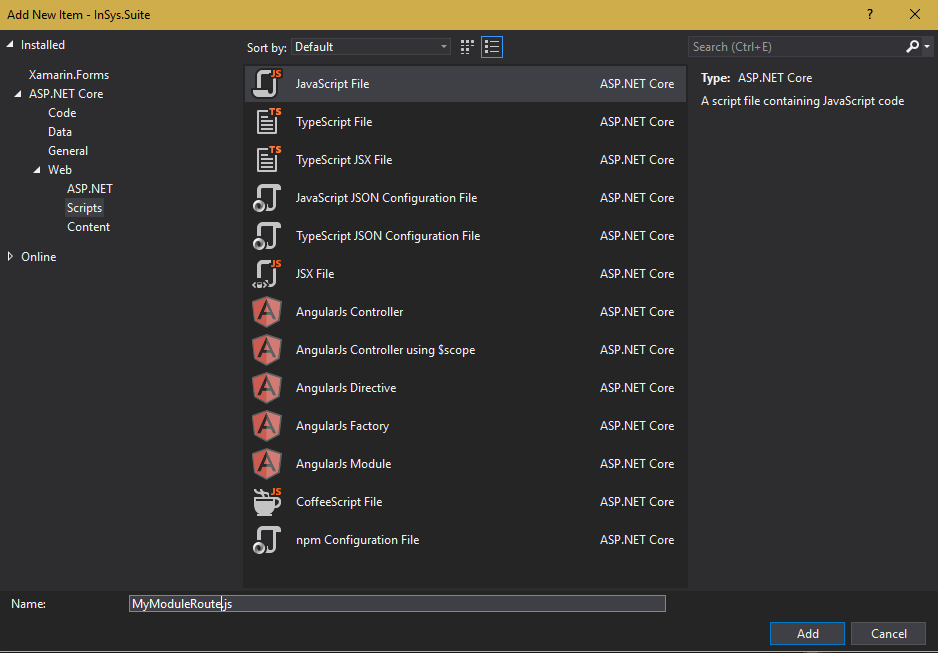


Figure 16: We will name it MyModuleRoute.js

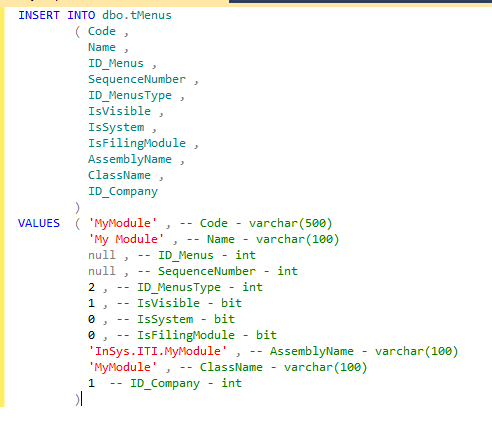


Figure 17: Now let’s insert a menu to our system.

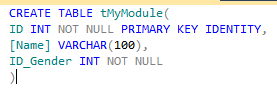


Figure 18: Create a table named tMyModule for our sample Menu

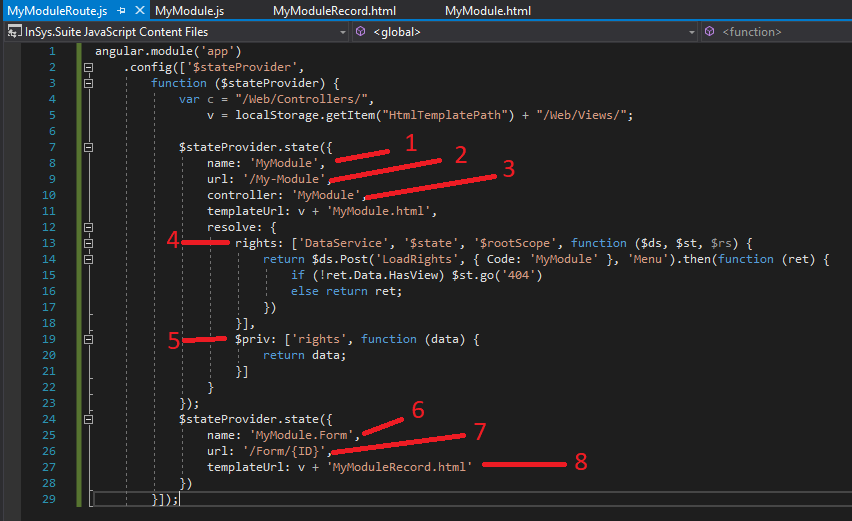


Figure 19: Now we configure our route.

1. **name** : The default name of our state will be the code of our menu that we inserted earlier.
2. **url** : This is what will appear on the address bar of our browser. Just name it whatever the name of your module.
3. **controller** : The same as the name, just name it base on our code of the menu.
4. **rights** : This is a request to the server to get the rights of our menu.
5. **$priv** : This is the responsible for returning the output of **rights**.
6. **name** : This is the sub route when we open a record.
7. **url** : This is the ourl for our sub route. The “{ID}” is where the record id will go if we open a record.
8. **templateUrl** : This is the template we are going to use when we open a record.

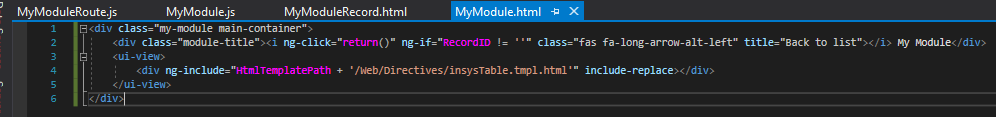


Figure 20 : This is the basic template for our listing.

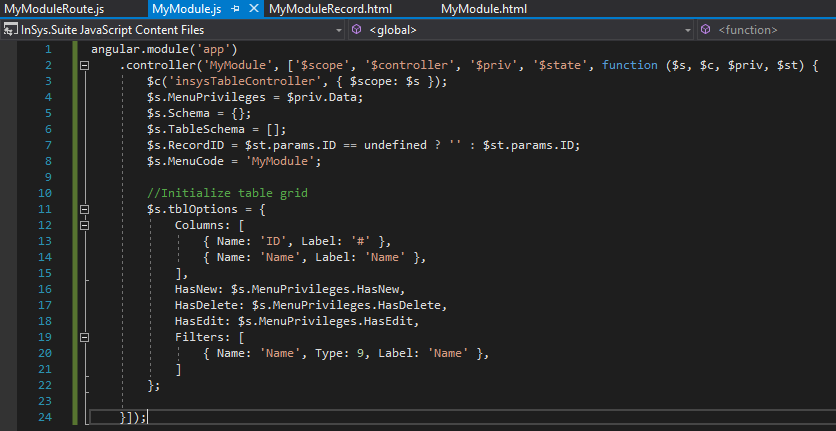


Figure 21 : This is the script for our angular Controller

First we need to inherit the “insysTableController” this is where all the the functions for our listing is stored.

Next we put the “$priv.Data” to a variable. This is the “rights” for our module that I have explained earlier on “MyModuleRoute.js”

We also declared some variables that we are going to use later. Next is the setup for our listing which is the “tblOptions”. Note that the variable tblOptions must not be changed, because this variable is used in the “insysTableController” when we call the LoadList later.

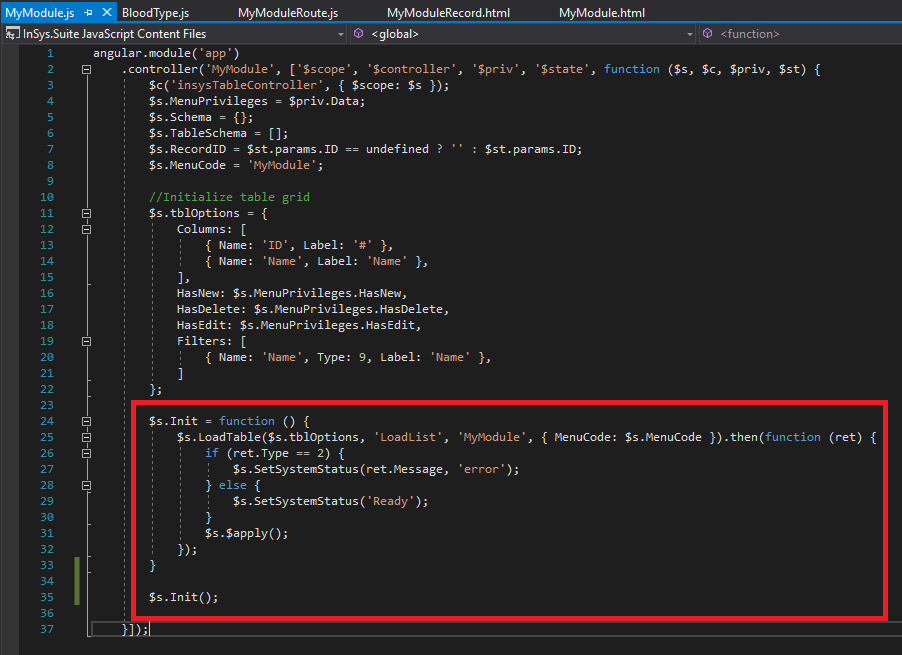


Figure 22 : Now we add the Init function to initialize the LoadList.

The parameters for LoadTable are the

Table Options, Method, ControllerName and extra parameters.

Now in the extra parameters we have added the MenuCode, now this is ABSOLUTE. You must not forget about this, else your code will not work or if you have already made a custom method inside the MyModuleController.cs. In our case we are relying to the BaseController because the LoadList is a generic method for the system. You can add more properties to the extra parameters if you need more data to pass to the LoadList method.

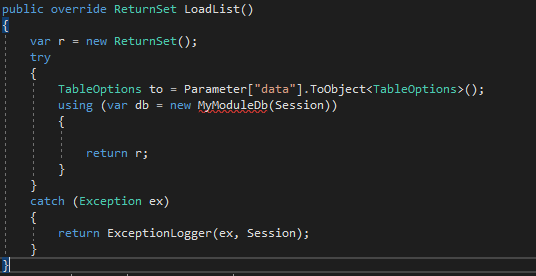


Figure 23 : Now we override the LoadList function that we have already implemented earlier.

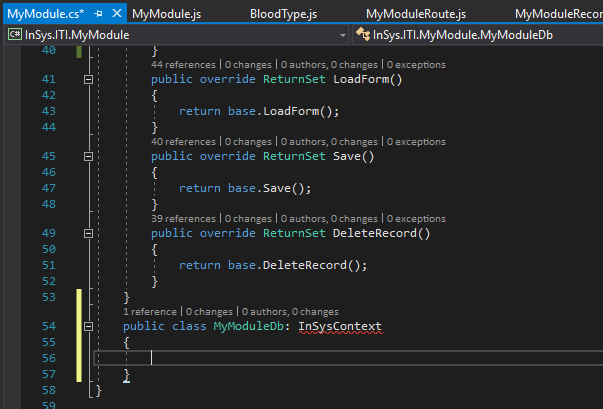


Figure 24 : Now at the end of MyModule Class, we create the MyModuleDb Class and inherit the InSysContext.

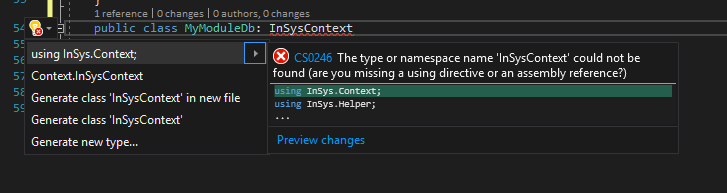


Figure 25 : Now we Import the InSys.Context

The “InSys.Context” is what will connect us to the database.

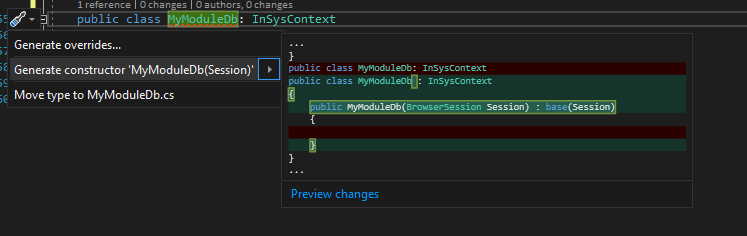


Figure 26 : Now we implement the required constructor when using the InSysContext

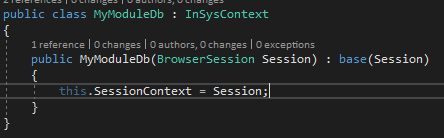


Figure 27 : Next we put the required variable for inheriting the InSysContext which is the SessionContext

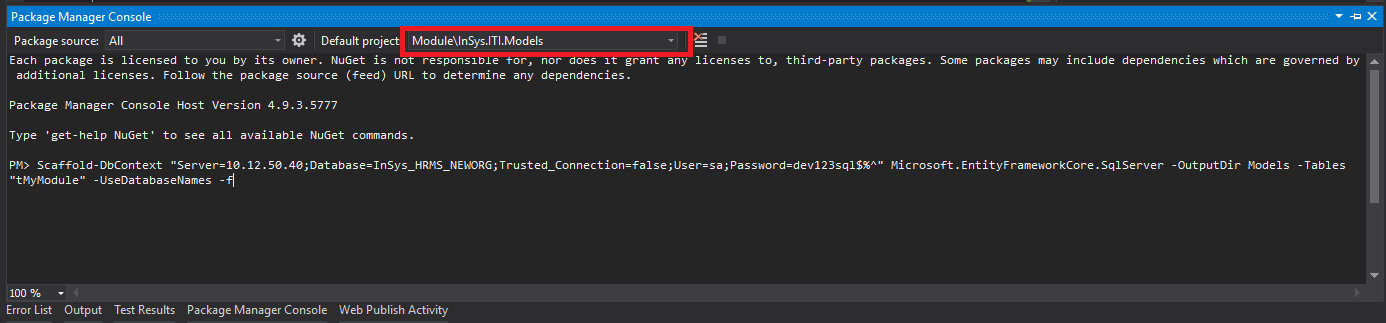


Figure 28 : Next is we create the model table that we created in the database which is the tMyModule. Make sure the target project is the InSys.ITI.Models

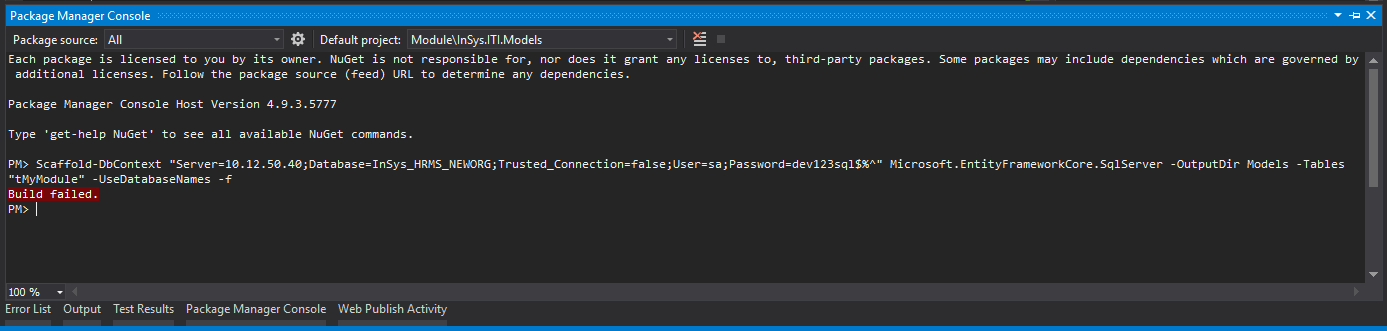


Figure 29 : Now our command failed. Go to the error window to see what's causing the error.

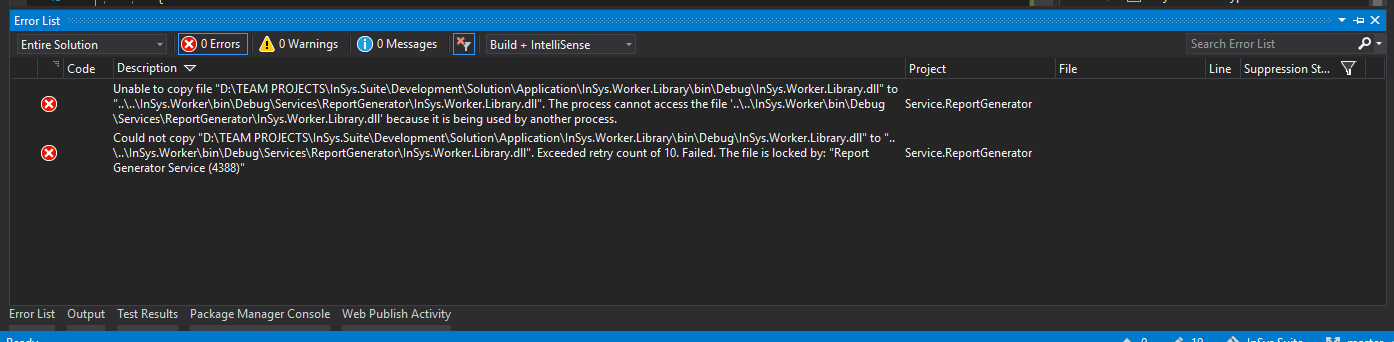


Figure 30 : Now in my case, it's the services that's causing the error.

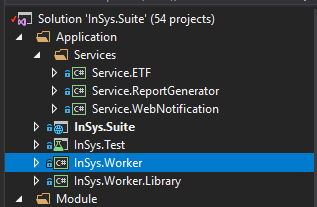


Figure 31 : If you have the same error, locate the InSys.Worker project

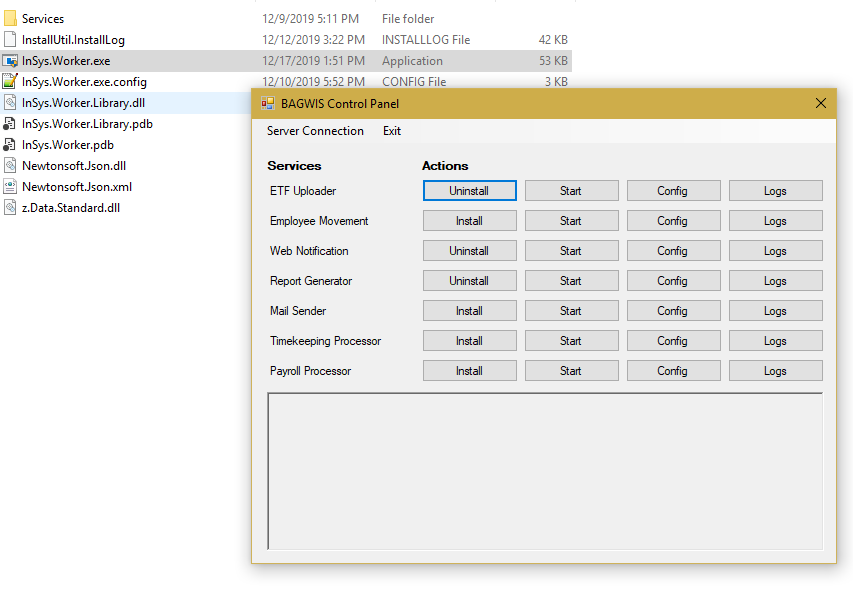


Figure 32 : Open its containing folder and launch the InSys.Worker.exe in the Bin > Debug folder and stop all the running services.

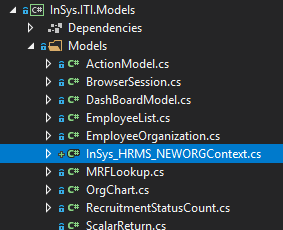


Figure 33 : Now after rerunning the command. It will generate this Context file, but we don't need this so just delete it.

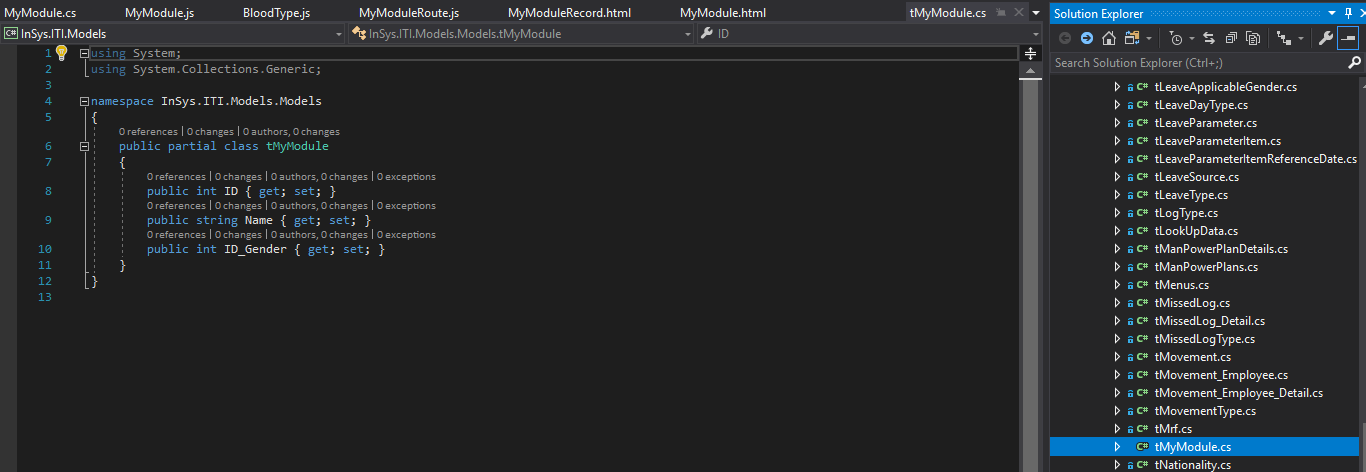


Figure 34 : Next is look for the tMyModule.cs class inside the InSys.ITI.Models > Models folder.

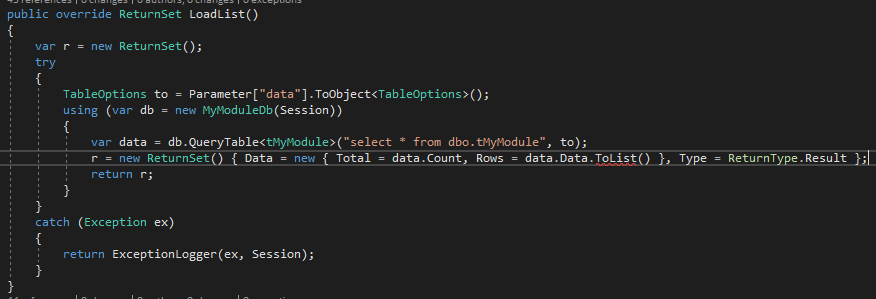


Figure 35 : Now we put our code to fetch for the list. This is always the code pattern when using the insysTable or the listing and also when using the lookup control.

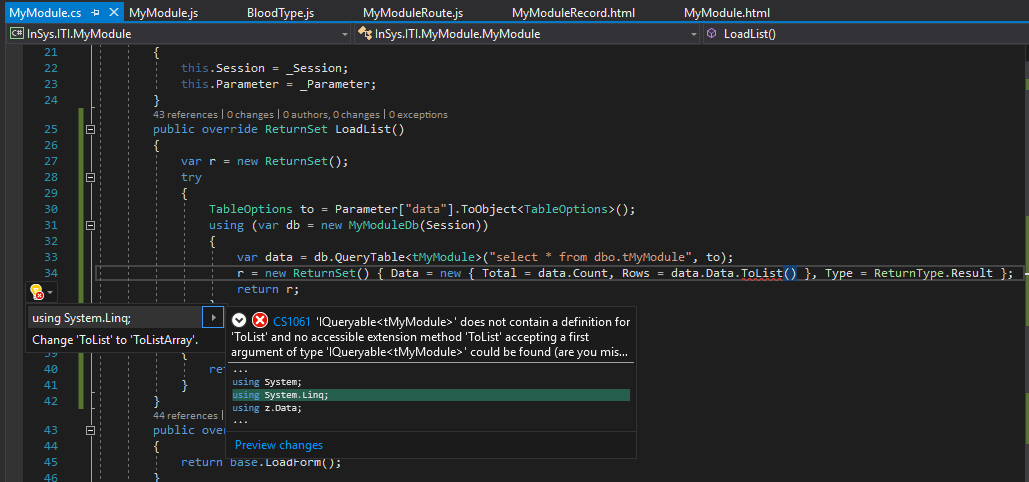


Figure 36 : If the .ToLIst() extension is missing, just import the System.Linq

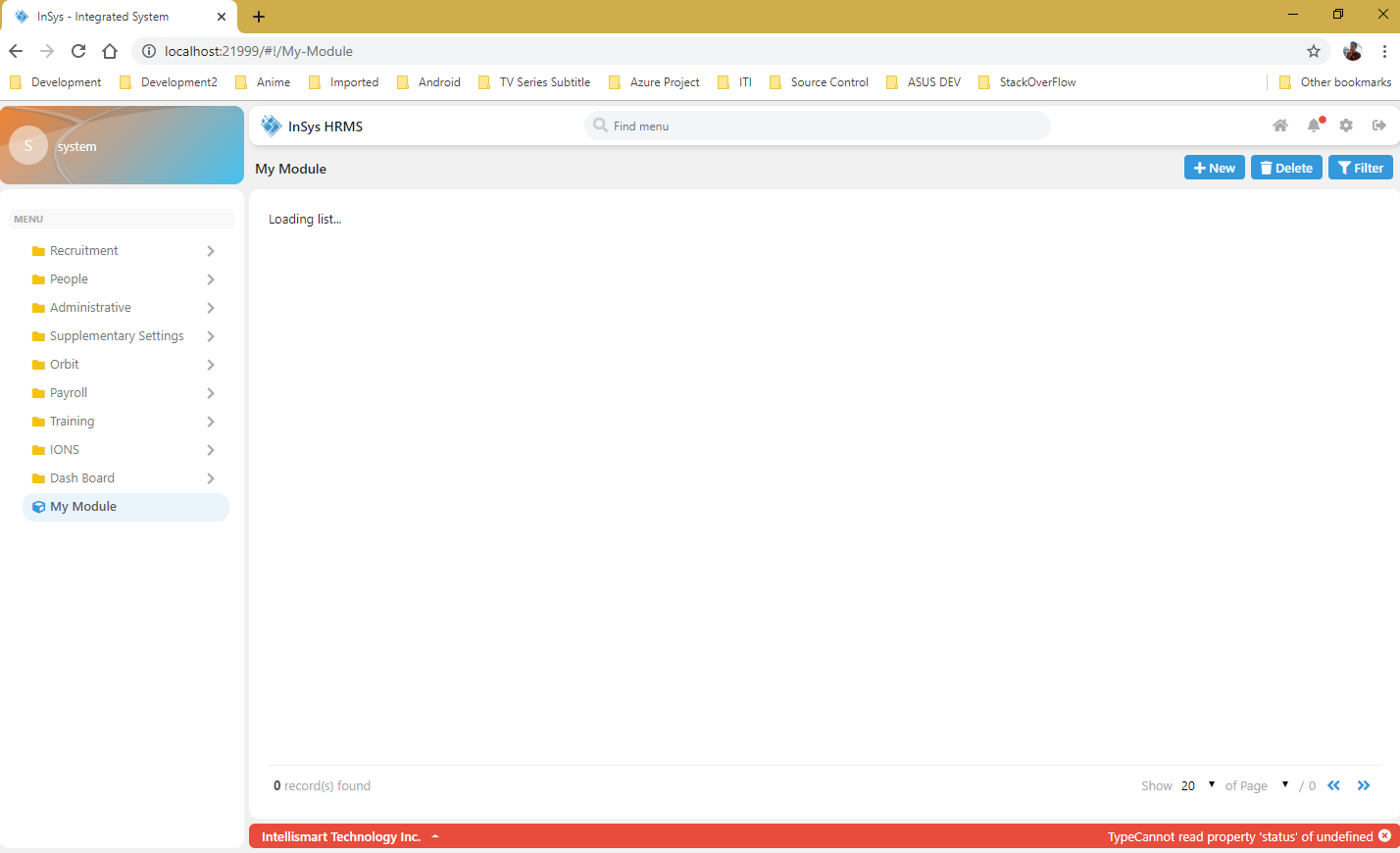


Figure 37 : Now we can visit our created module. As you can see it has an error.

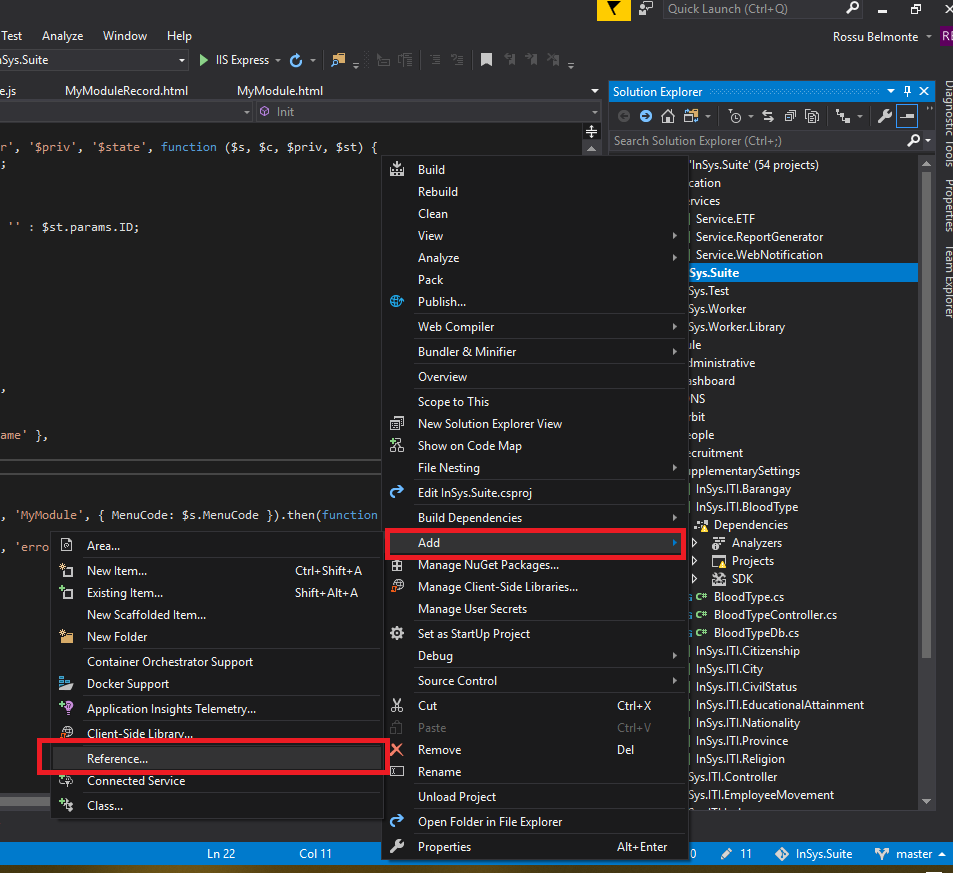


Figure 38 : Right click the InSys.Suite project then Add > Reference.

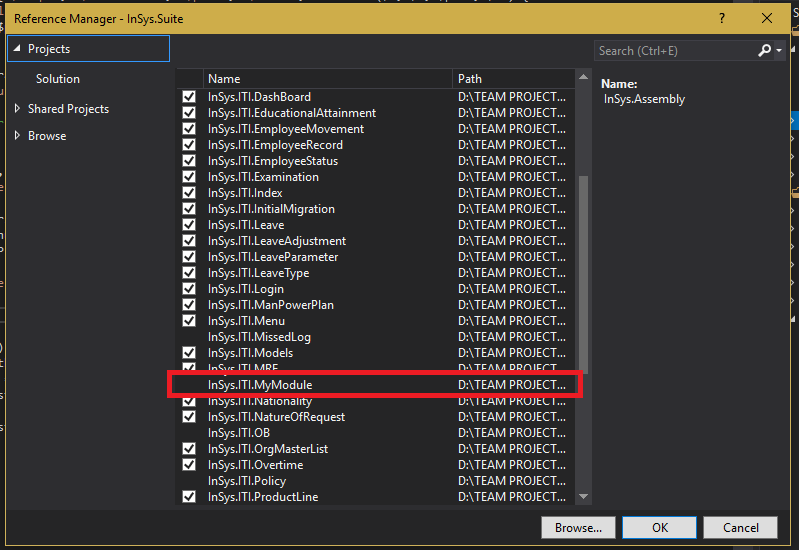


Figure 39 : We need to reference the project we created so the API request can find the MyModuleController that we created earlier.

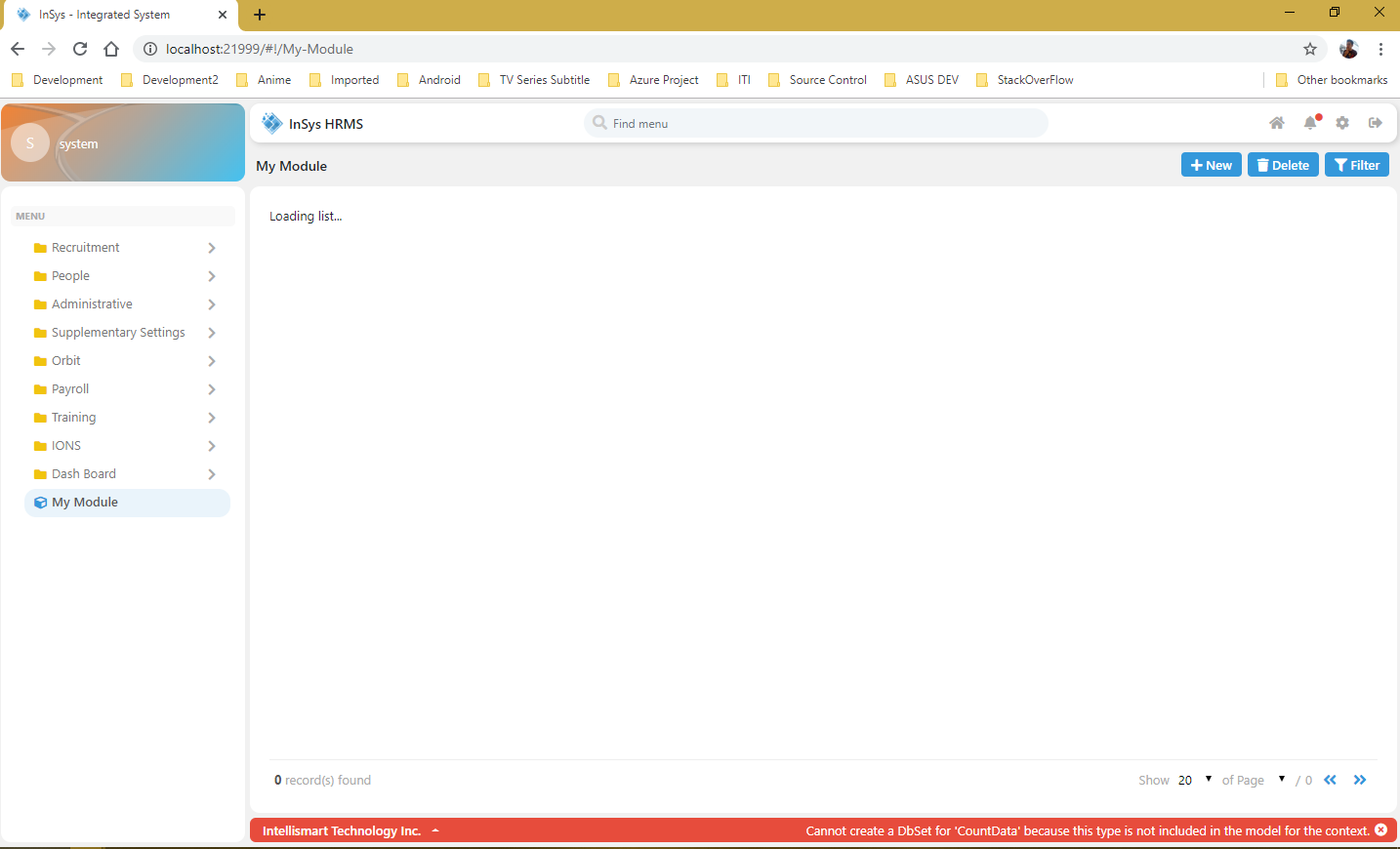


Figure 40 : Now that we have reference the project, it still has some error, which is looking for the CountData.

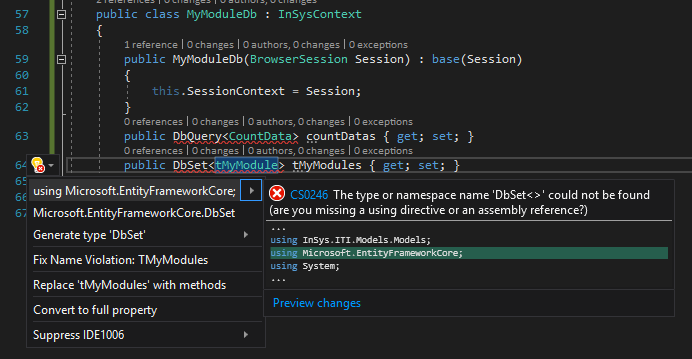


Figure 41 : In the MyModuleDB we declared some important variables. I will explain it as much as I can.

**DbSet** – This is used when querying in EF Core. As you can see we pass on the tMyModule class to the DbSet. Meaning the class tMyModule is mapped as a table in the database. It requires a primary key when using the DbSet, so make sure that the table you are mapping with the DbSet has a primary key. Also when using insert, update and delete, you need to make sure that it is declared as a DbSet.

**DbQuery** – This is also used when querying in EF Core. It does not require a primary key. The difference in DbSet is that in DbQuery you don’t need an existing table in the database and the class you are going to pass does not need to be mapped. Meaning you can query here a function or a select statement with multiple joinings.

Now we set CountData and tMyModules because this are going to be used in our LoadList. You may not see them used in the code because we are only telling the MyModuleDb to register this classes. The tMyModule class is visible in the code but you are wondering where the CountData class is used. It is used in the QueryTable extension if you view its source using F12.

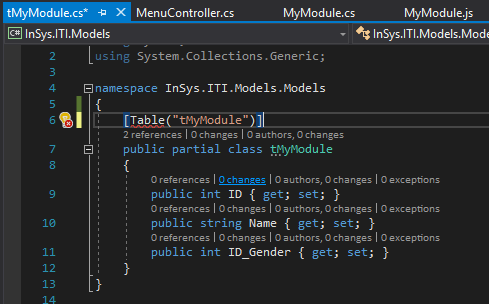


Figure 42 : In this figure we are going to map the tMyModule class to the database. Meaning this class is existing as Table in the database.

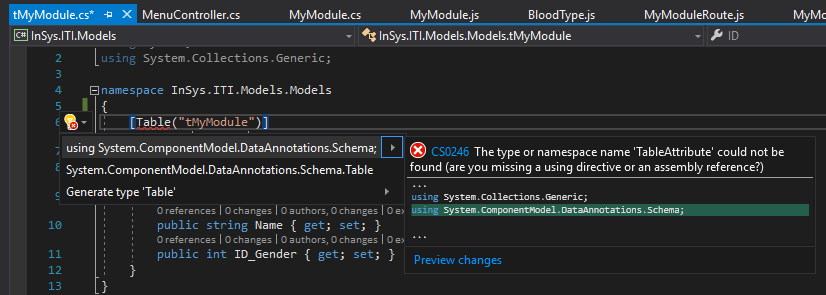


Figure 43 : Import the System.ComponentModel.DataAnnotations.Schema

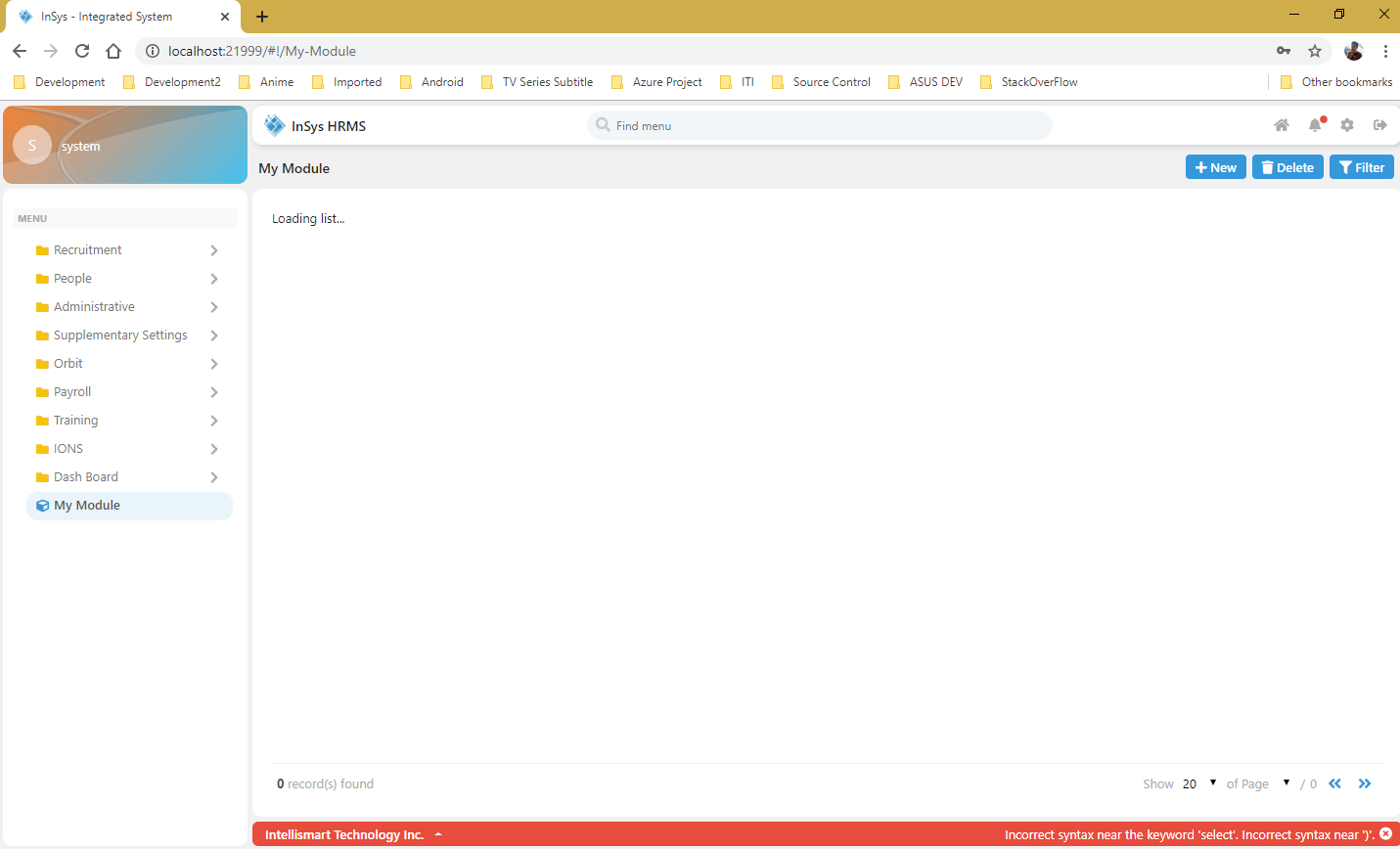


Figure 44 : Now we have another error.

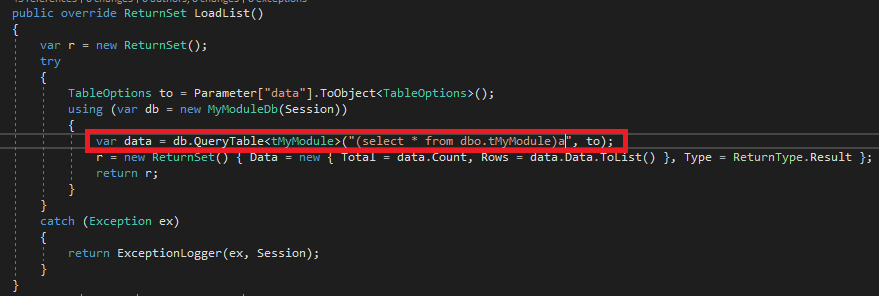


Figure 45 : We forgot to put the query inside a parenthesis and an (a) alias. Note that the (a) alias is important. Don't change it to whatever letter you have in mind.

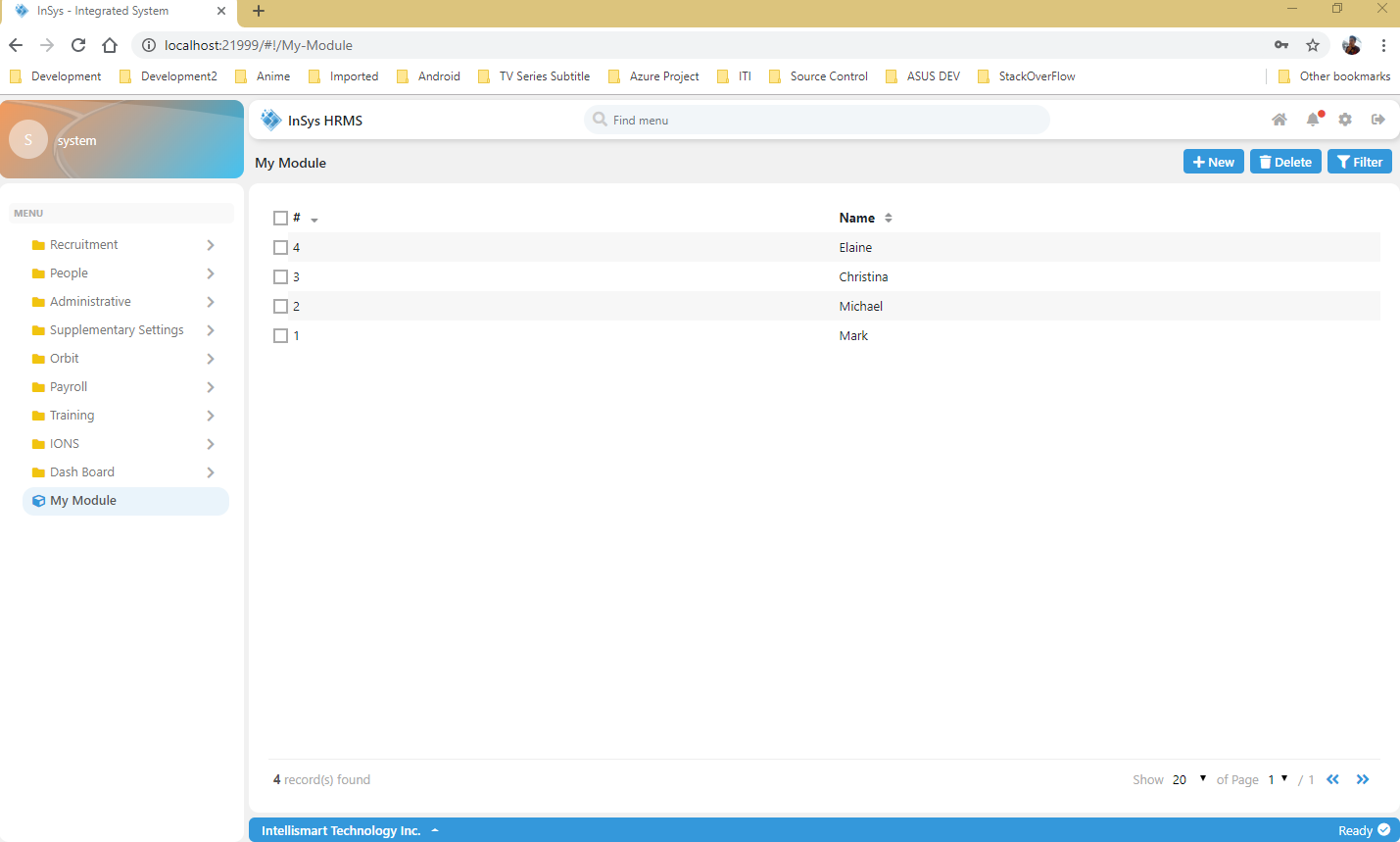


Figure 46 : Now our LoadList is now working. I have alread inserted some sample data.

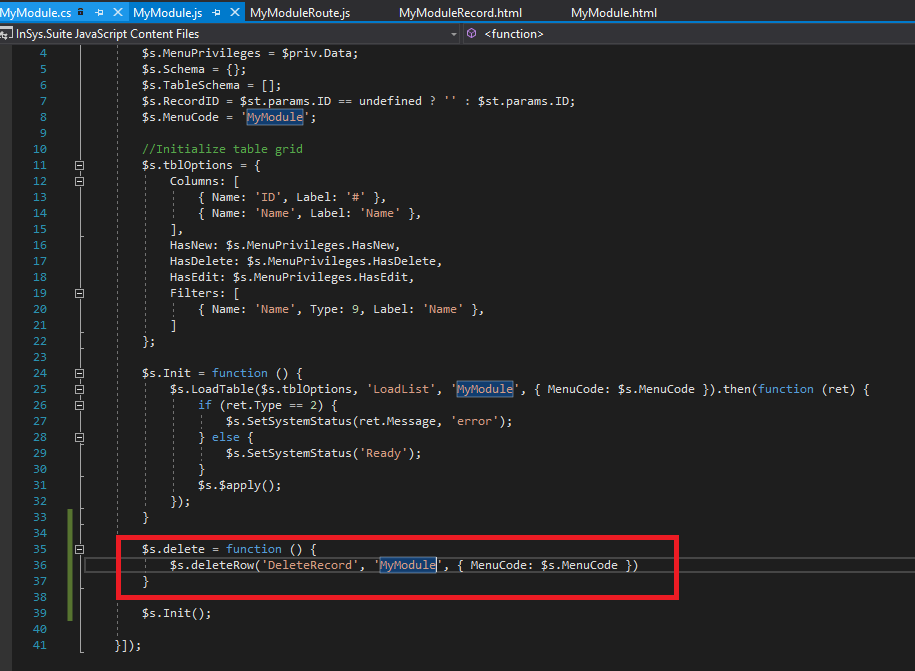


Figure 47 : Now let’s add some delete code.

The parameters for the deleteRow are:

Method, ControllerName and extra parameters.

The same in LoadList that I have explained earlier.

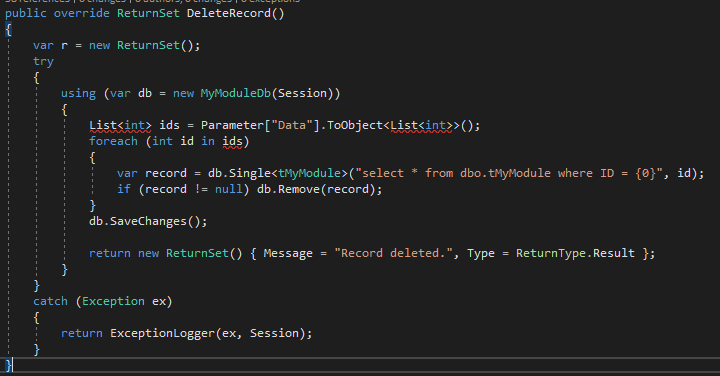


Figure 48 : Now we override the DeleteRecord method.

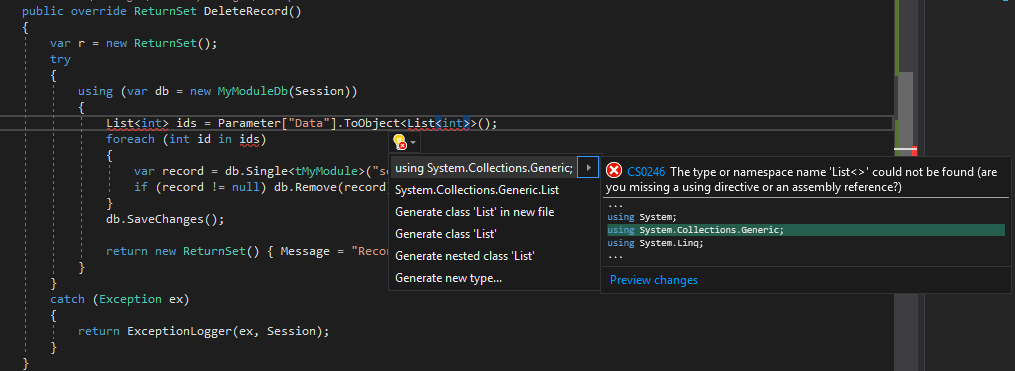


Figure 49 : Import the System.Collections.Generic

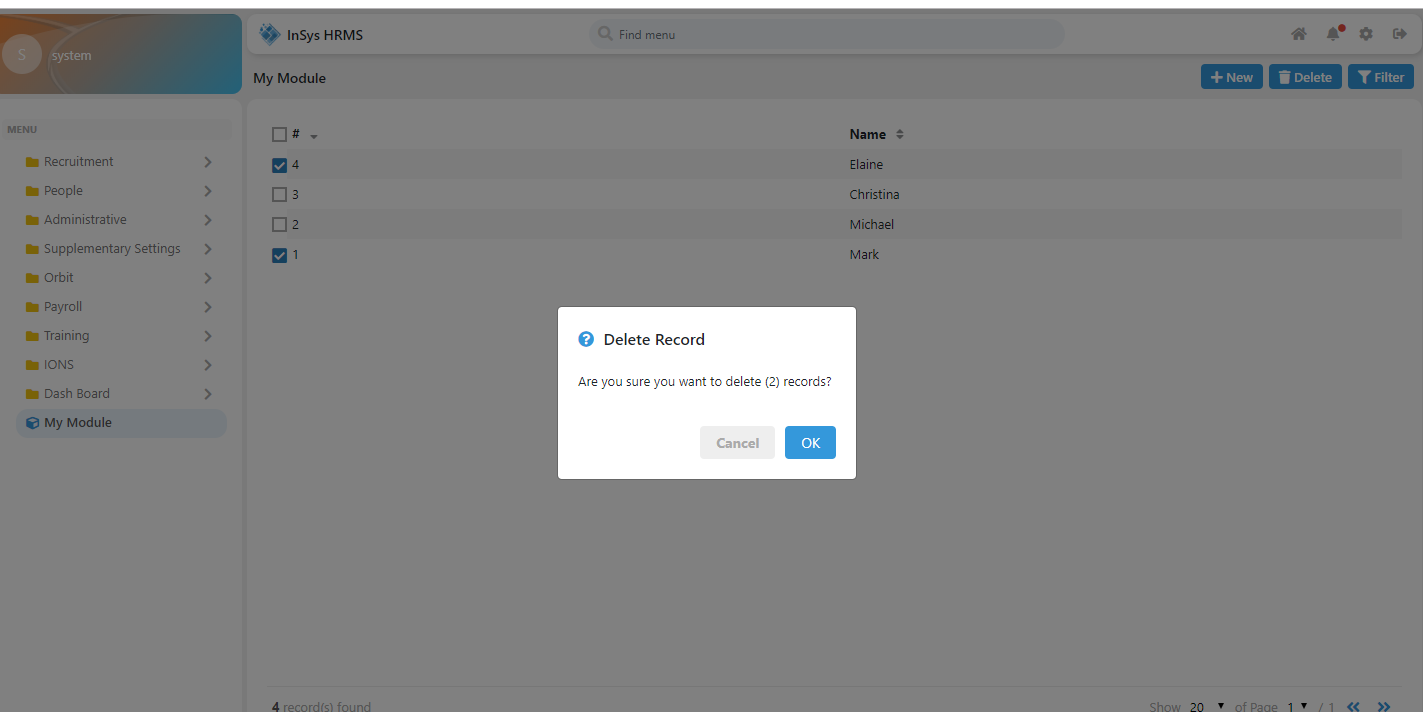


Figure 50 : Now let's try to delete some data.

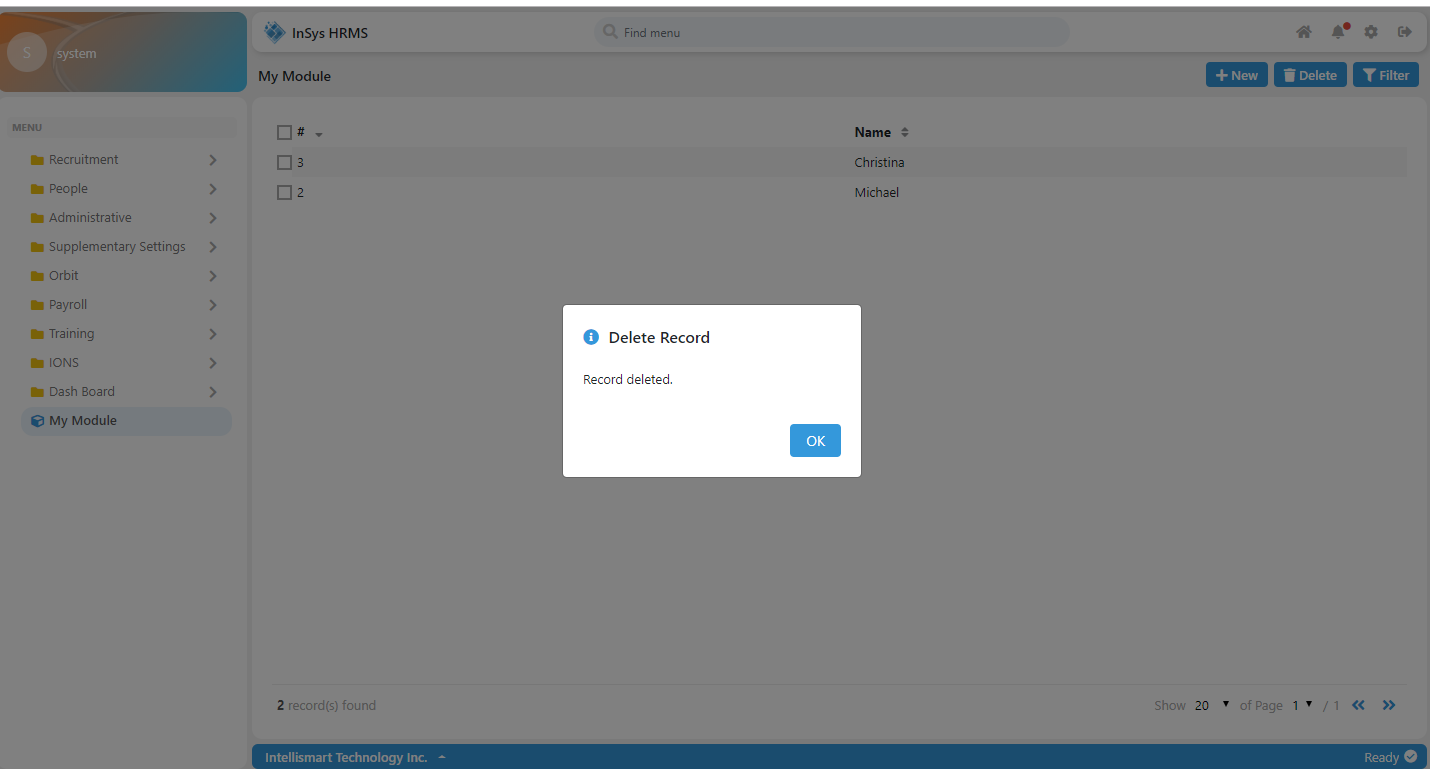


Figure 51 : As you can see the code worked perfectly.

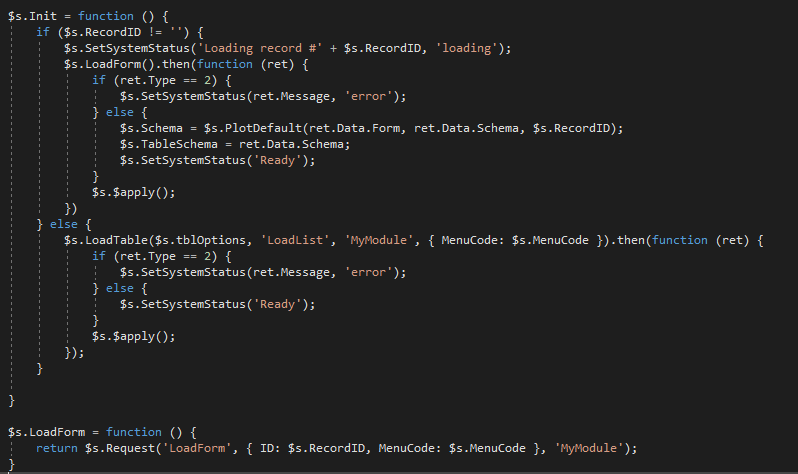


Figure 52 : Now let's add an open record functionality

As you can see we have added the LoadForm function. The parameters are:

Method, parameter and ControllerName

In the parameter object the ID property is important. We need the id of the record we opened and also the MenuCode. The MenuCode is an important Parameter especially if you are calling a method from the BaseController. If you have not provided this, it will not go to the BaseController.

Next is we change the behavior of the Init function. As you can see, we are going to check if RecordID has a value. So if it doesn’t have a value it will only use the LoadList but if it has a value it will use the LoadForm function and it will load the template I have explained earlier in the MyModuleRoute.js. You can see the RecordID getting its value in the figure 21.

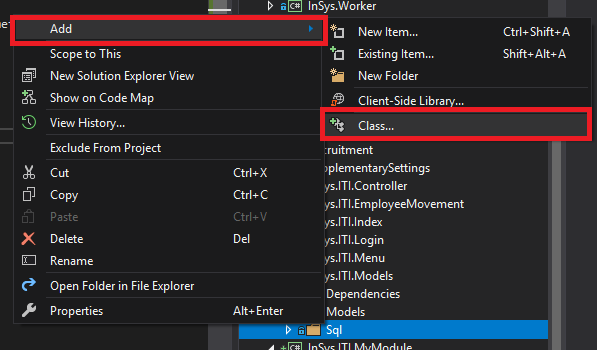


Figure 53 : Next is we are going to create a query using a .sql file

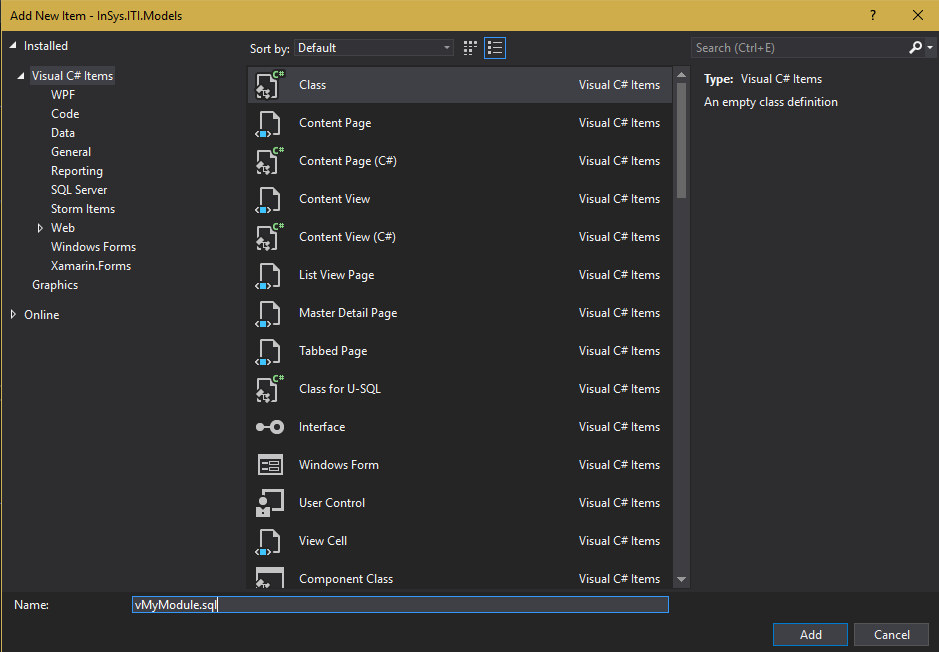


Figure 54 : Now you can see that I have changed the .cs extension to .sql file and named it vMyModule.sql

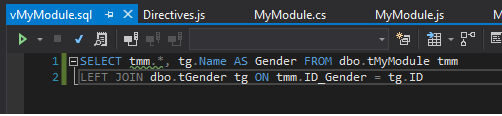


Figure 55 : Here is our query for the vMyModule.sql. We joined the tGender to the tMyModule table.

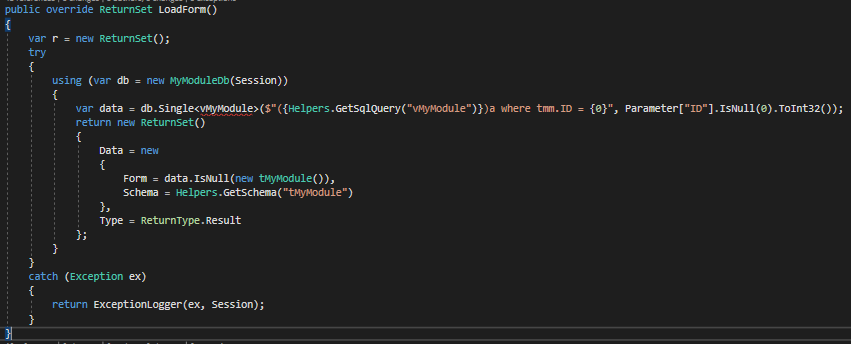


Figure 56 : Now we override the LoadForm method.

As you can see I have use the Helpers.GetSqlQuery method to read the vMyModule Sql file.

In the return statement I have set that if the data is null it’s going to declare a new tMyModule object. Meaning if the record we are looking for it is only going to treat as New Record.

The Schema property in the return statement is we are going to fetch the Table Schema of the tMyModule. We are going to use this later.

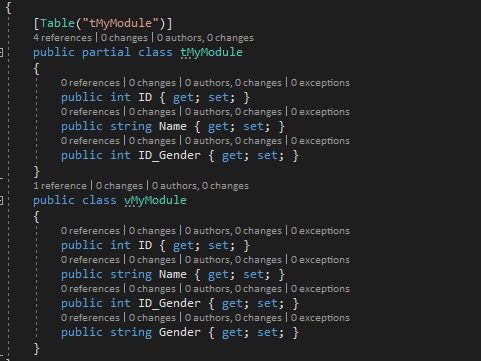


Figure 57 : Now we create the vMyModule class. We have added the Gender in our query, so we will also add it to our view.

It’s not always needed to add all the properties that you have declared in your query. You can just add the properties you are going to need. BUT if you have declared a property in the class make sure it exist in your query. Otherwise it’s going to throw an error.

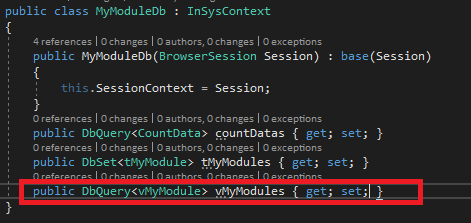


Figure 58 : Next we add the vMyModule to the MyModuleDb

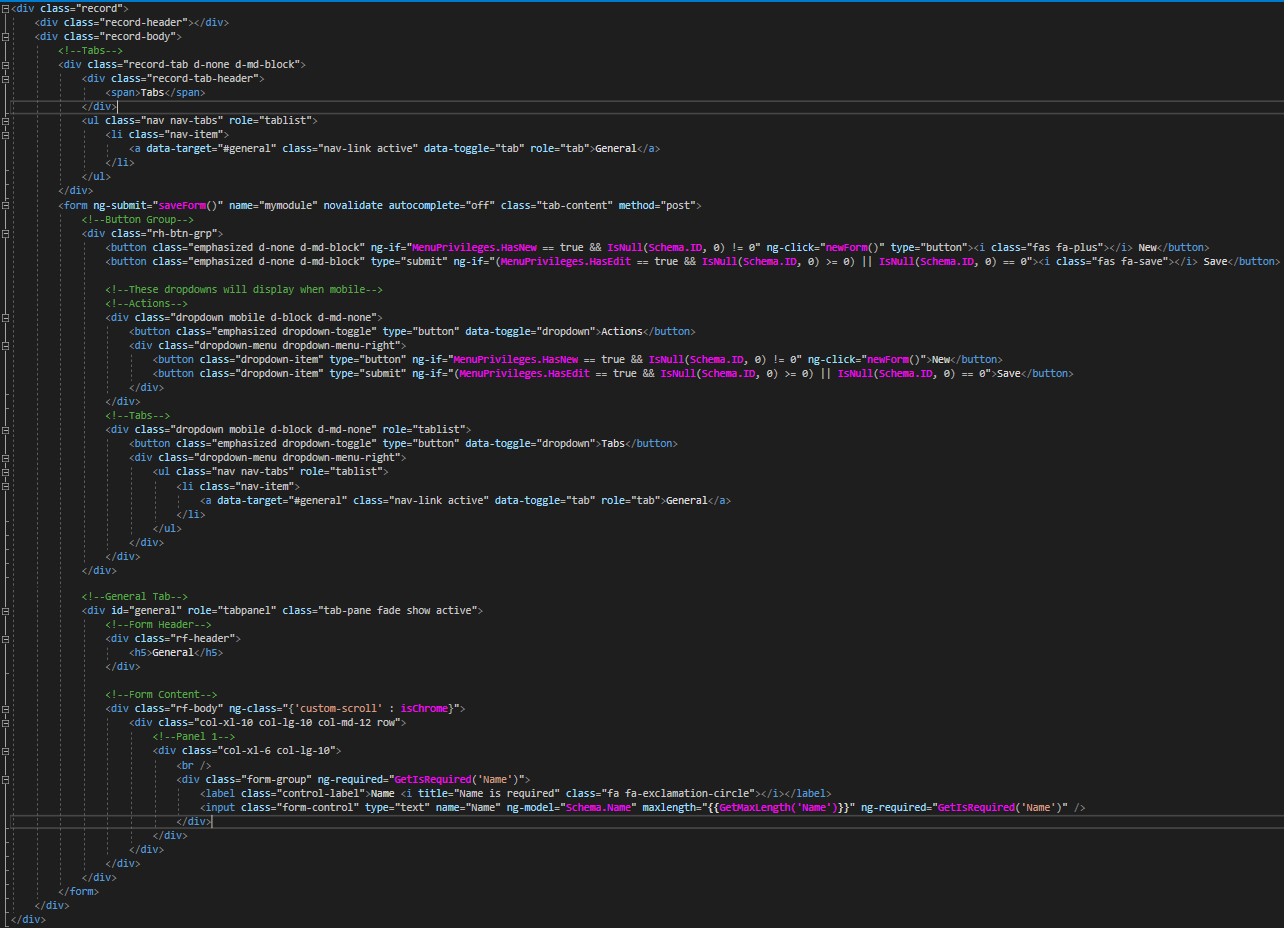


Figure 59 : Now we add the template for our Form when we open the record. Just locate the MyModuleRecord.html and add this template.

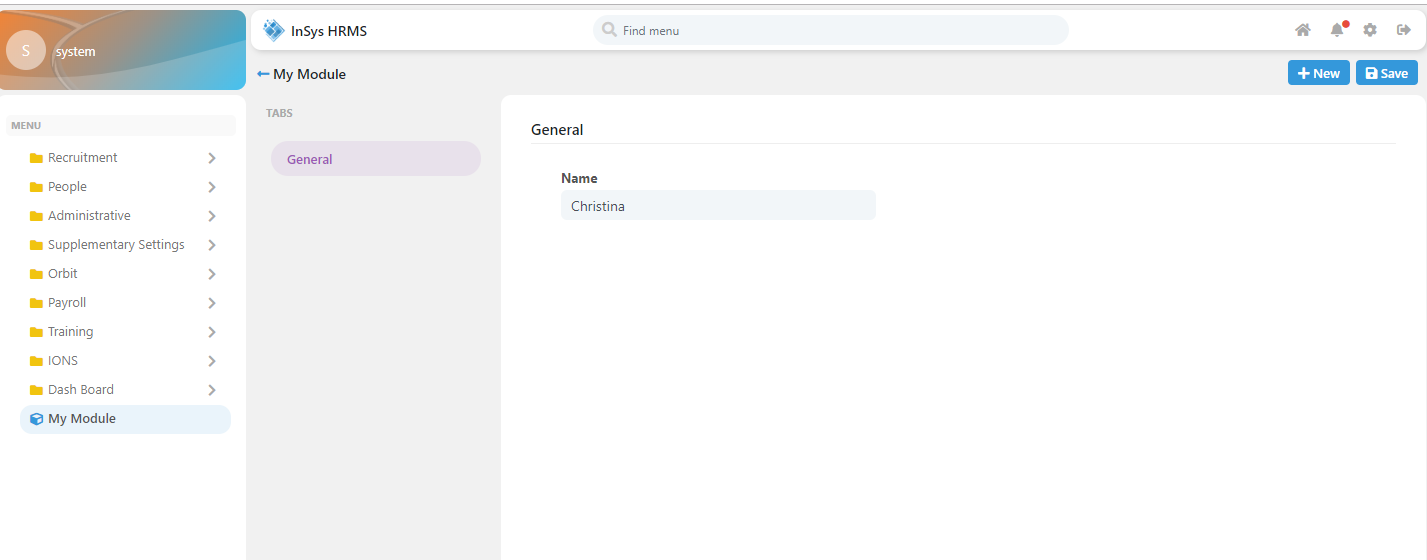


Figure 60 : Now we can see it as editable if you open the record.

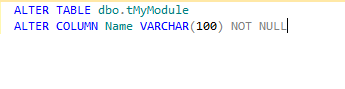


Figure 61 : Now let's make the column Name required in the form.

As you can see we altered the column Name as NOT NULL. When we first created the table it is by default NULL.

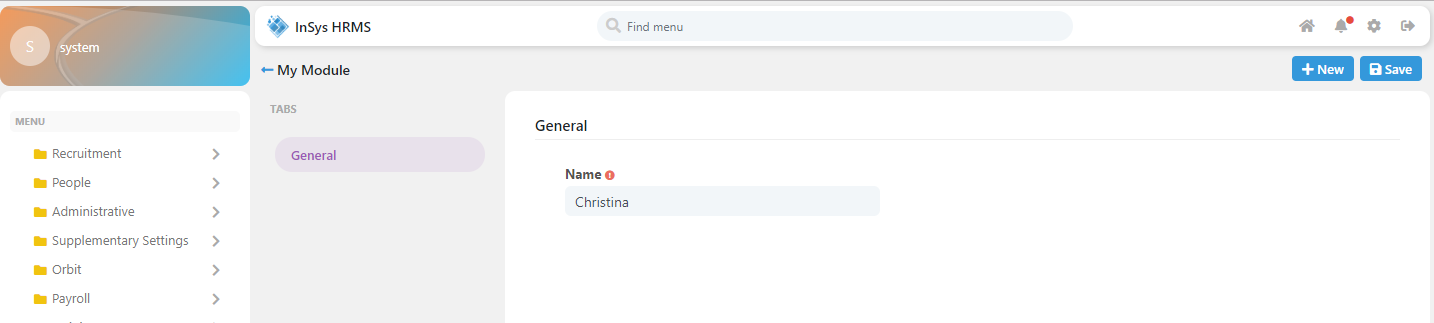


Figure 62 : Now it is automatically required. Why? This is the work of the Schema that we got from the figure 56 and we put the schema data into the TableSchema variable in figure 52.

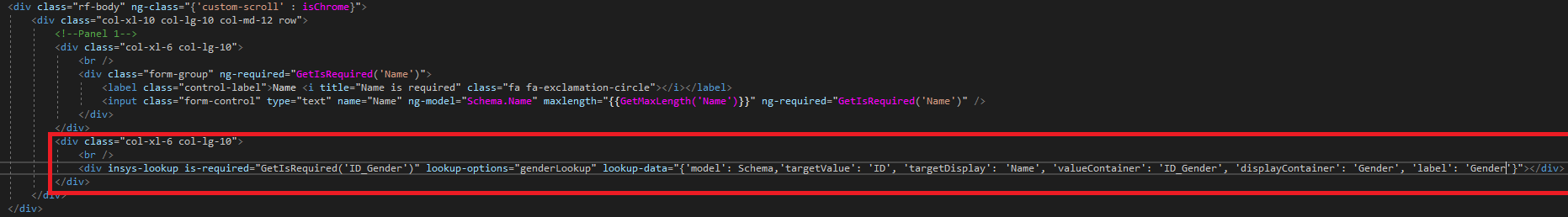


Figure 63 : Now let's add a lookup control for gender.

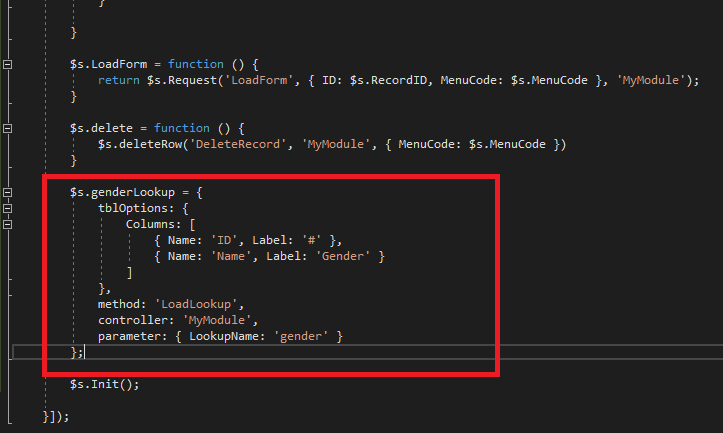


Figure 64 : Here is the property setup for the lookup. As you can see the $s.genderLookup has been passed to the lookup-options attribute in figure 63.

As you can see the setup for the tblOptions is also the same setup when we used the insysLookup for the listing.



Figure 65 : Now let's override the LoadLookup method from the BaseModule.

We will be switching the value of the parameter LookupName from the figure 64. If you have multiple lookup in your form, just add it in the swtich statement.

Now let’s add a virtual method LoadGender. It will be responsible for returning the data, just like the LoadList.

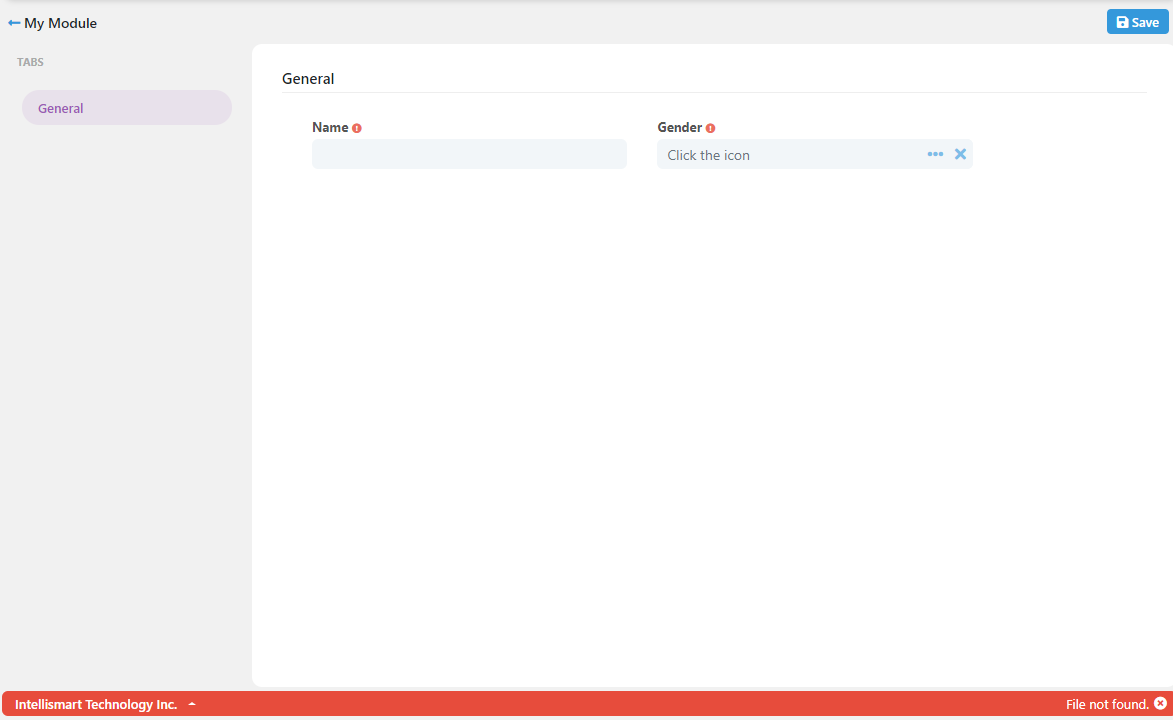


Figure 66 : Now we have an error that say's File not found.

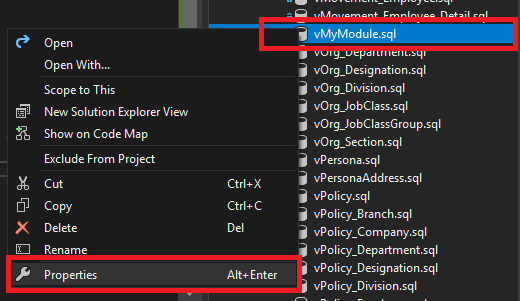


Figure 67 : To fix the error in figure 66, go to the InSys.ITI.Models > Sql and find vMyModule.sql

Right click the file and click properties.

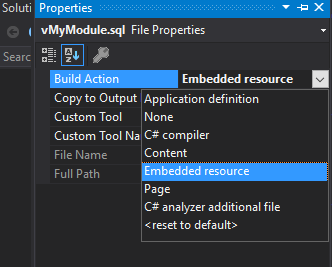


Figure 68 : Set the Build Action to Embedded resource.

The Helpers.GetSqlQuery can now read the file. It works on reading files that are embedded to the .dll file.

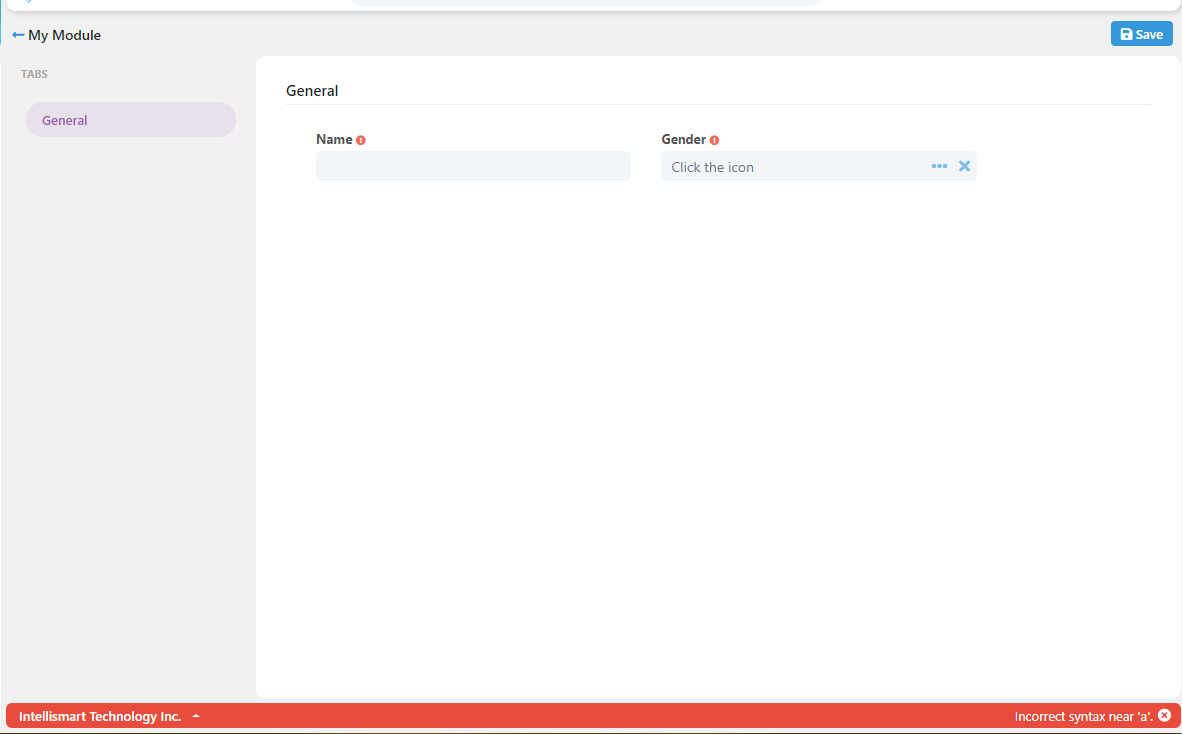


Figure 69 : Now we still have an error.

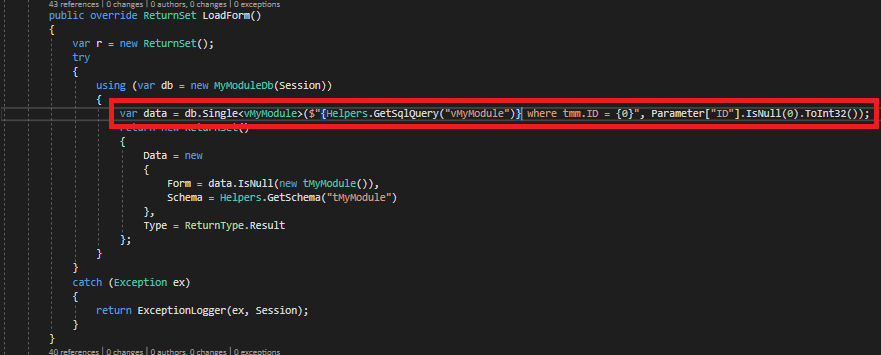


Figure 70 : If you look at the figure 56 you can see that we put the Query reader inside the parenthesis and an (a) alias. That was a mistake. When using the .Single extension you don’t need to put the query inside the parenthesis and an (a) alias. As you can see above we alread removed it.

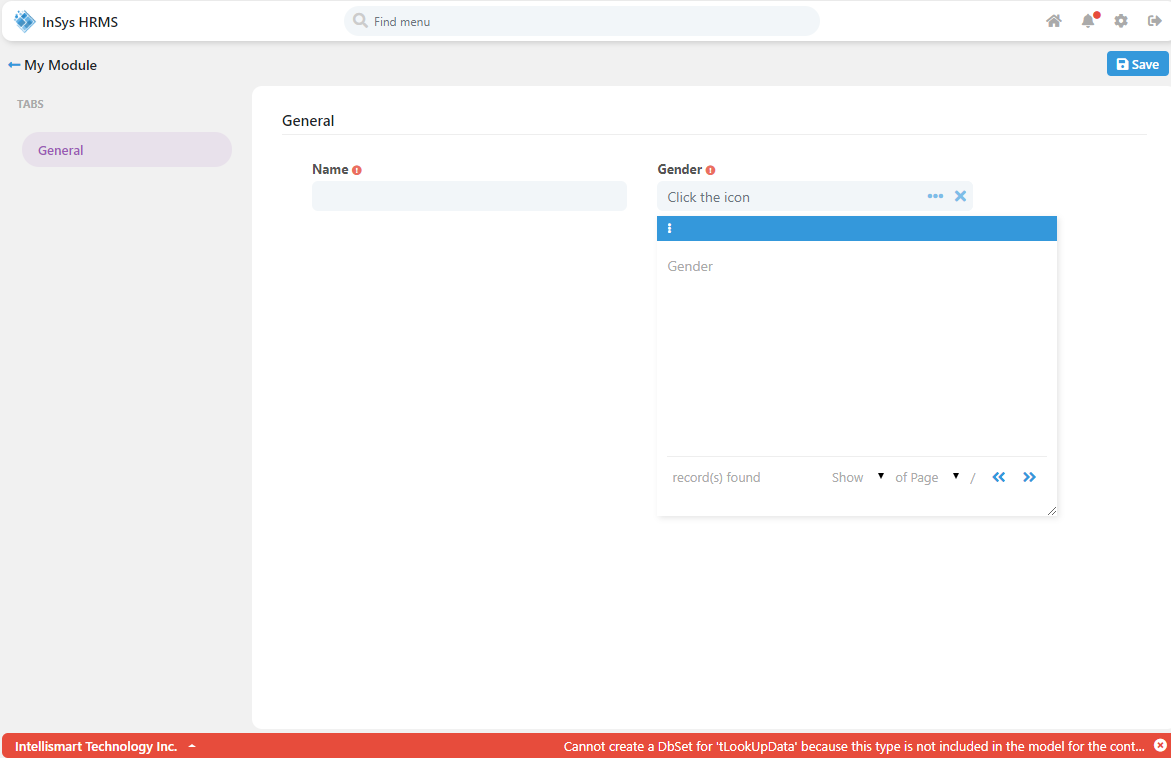


Figure 71 : Now we used the tLookupData in figure 65 which we didn't put in the MyModuleDb, resulting to an error.

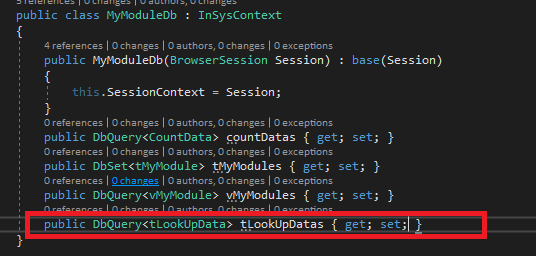


Figure 72 : Just add the tLookupData to the MyModuleDb.

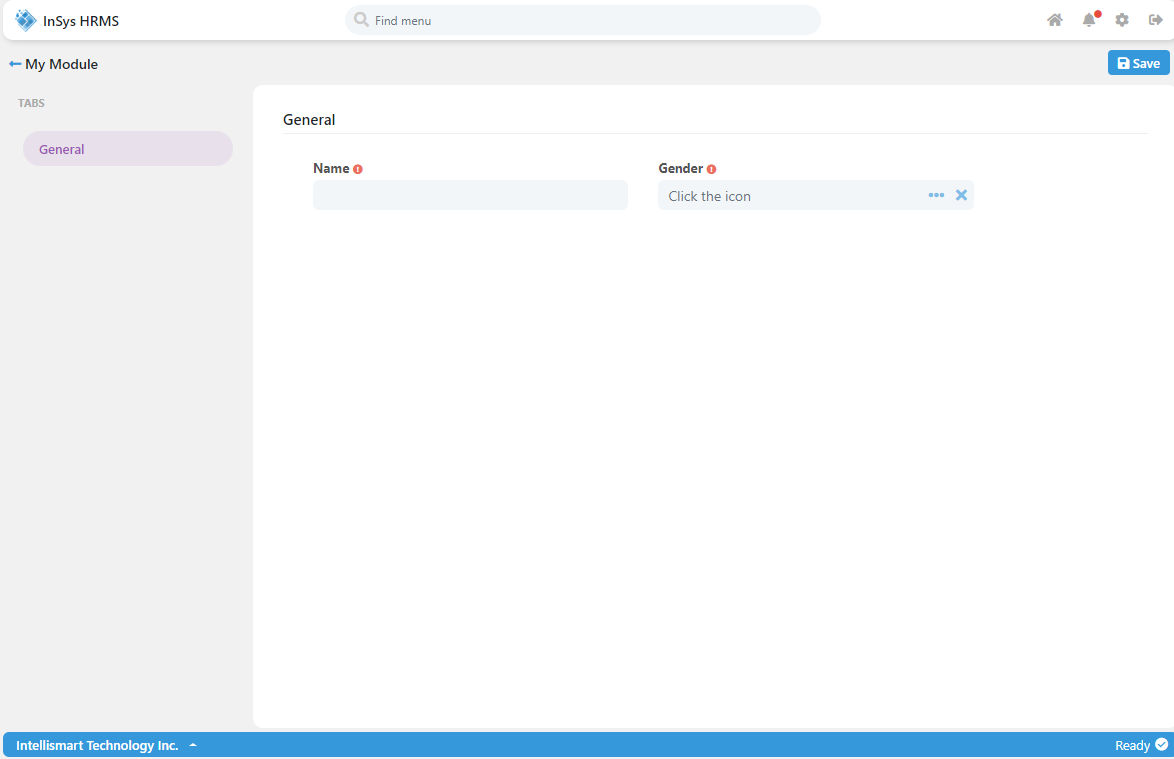


Figure 73 : Now the error is removed. But we are not getting any data even the record exist.

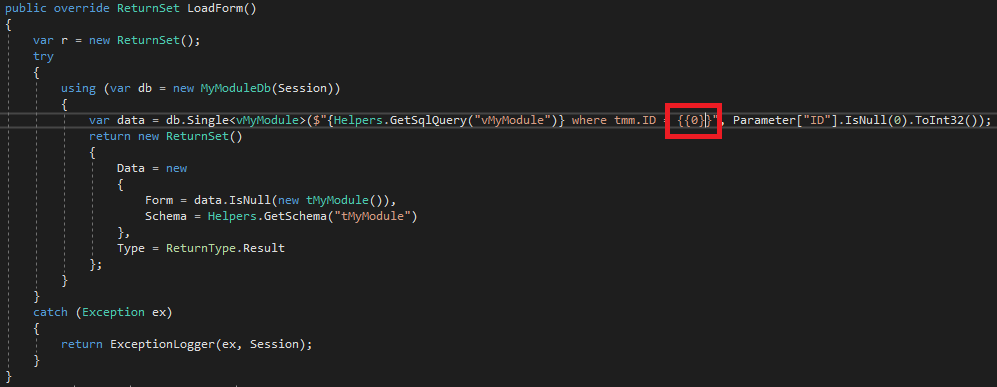


Figure 74 : In figure 56 you can notice that we are only using a single curly brace. But we need to escape it in order for the query to read it as a Sql parameter. It’s the same as using @ sign. But in our case we use the indexing parameter. Why do we need to escape it? That’s because we use the $ sign at the start of the query. Now in the figure above you can see that we escaped it using double curly brace.

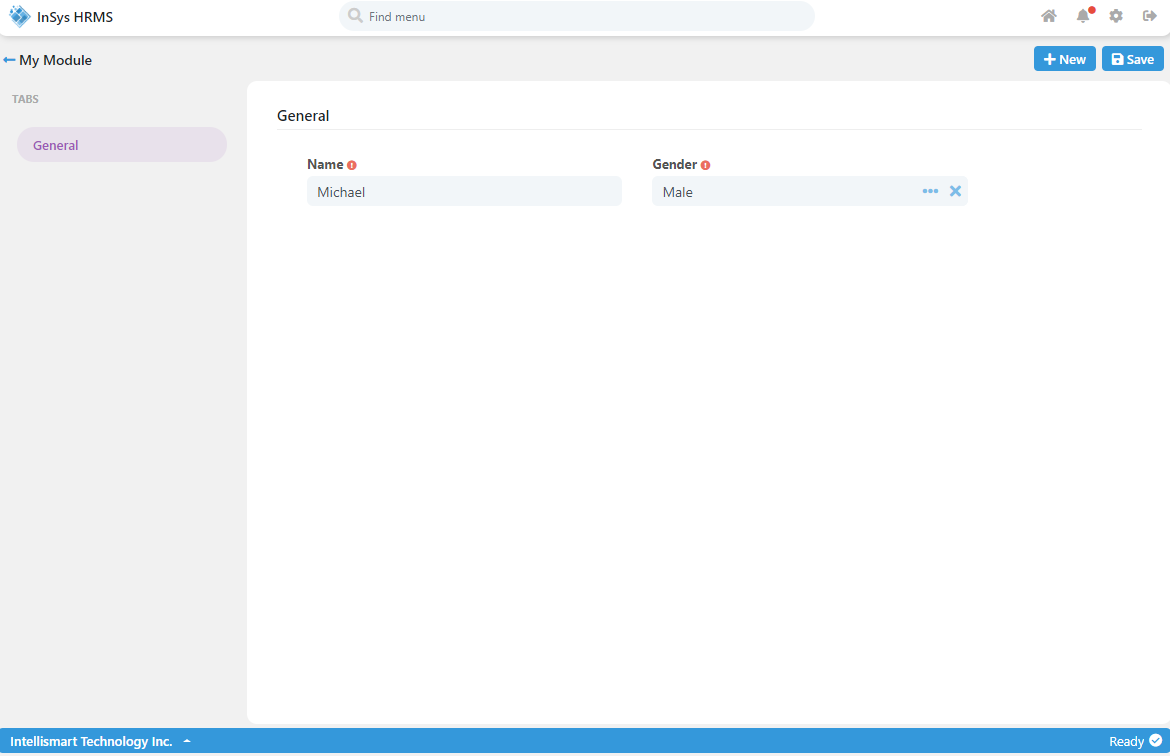


Figure 75 : Now we can edit the record.

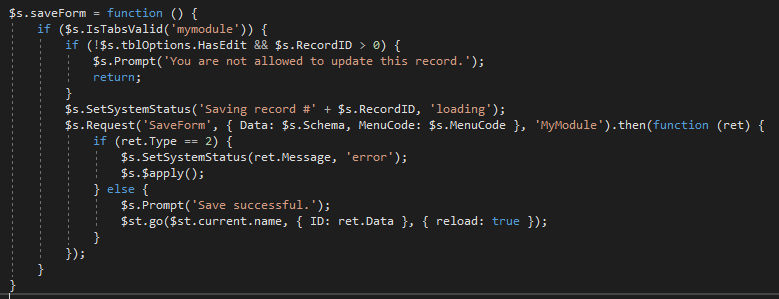


Figure 76 : Now let's add the saveForm function.

As you can see on the condition it is checking if the $s.IsTabsValid will return true. $s.IsTabsValid is made to validate the form. On the parameter, we pass the “mymodule”. What is that? That is the name of the form. If you look closely in the MyModuleRecord.html <Form> tag, it is named “mymodule”. Next is we call for the $s.Request function. The parameters are:

Method, parameter, controllerName, disableInterceptor

This is the first time you heard the “disableInterceptor”. This is the loading you can see on the screen when requesting for data or sending data to the server. Set it to “true” if you don’t want to see the loading when you make the request.

Now in the parameter we add the property “Data” and the value is “$s.Schema”. When did we have the variable “$s.Schema”? Look at figure 52, we set the value of “$s.Schema” after the LoadForm request. Now the “$s.Schema” is binded to our MyModuleRecord.html as Angular Model. Next in the parameter is the property “MenuCode” as I have explained multiple times this is essential when making request to the generic Api’s of the system. Now if the save request is successful we reload the route of the page along with the parameter ID that the Save method returned.

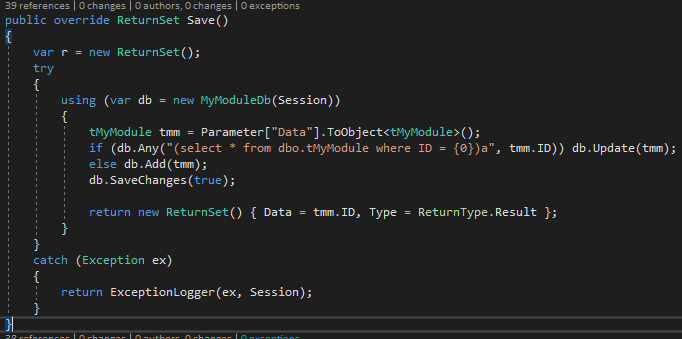


Figure 77 : Now we override the Save method. As you can see, the parameter Data that we passed from the client side, is casted to tMyModule class. If the record does not exist, meaning it's a new record and the extension called is .Add if the record does exist the extension called is .Update then .SaveChanges extension is called and the saved ID is returned to the client side.

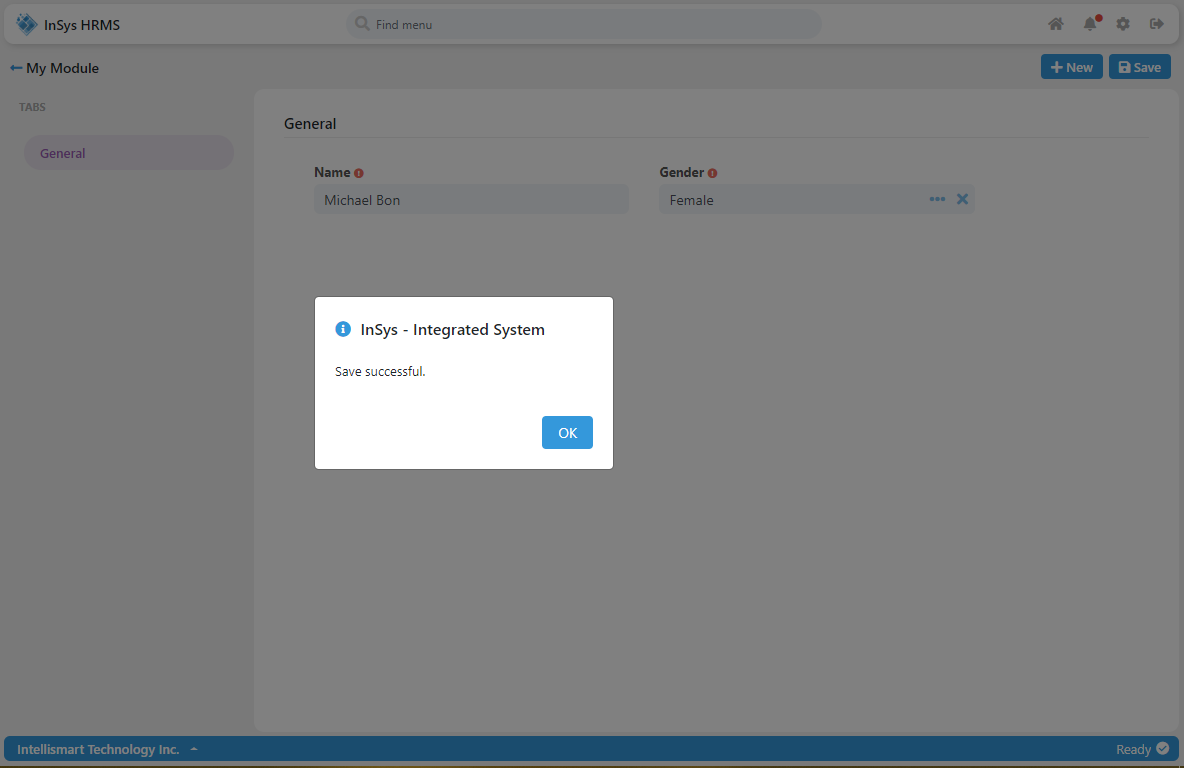


Figure 78 : Now I have added "Bon" to Michael's name and changed his gender to Female. As a result the save is successful.

1. **HOW TO CREATE DETAIL TAB**

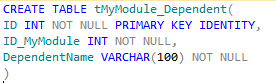


Figure 79 : Now let's create a table for our detail.

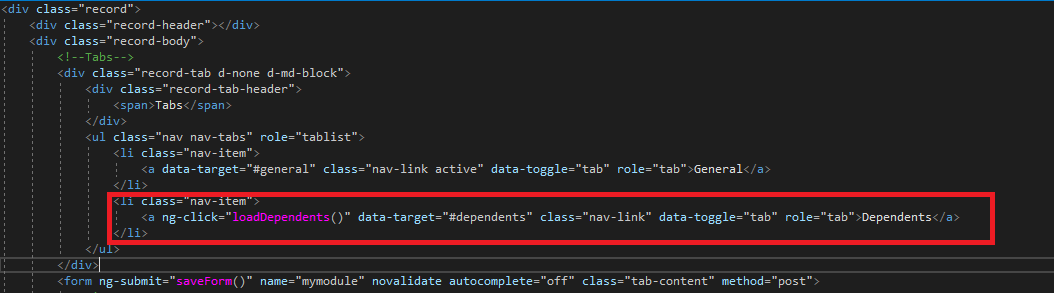


Figure 80 : Add another tab to our form. I also added a ng-click function named "loadDependents". Don't forget the to set the name of the data-target or else the tab Dependents will not be click able.

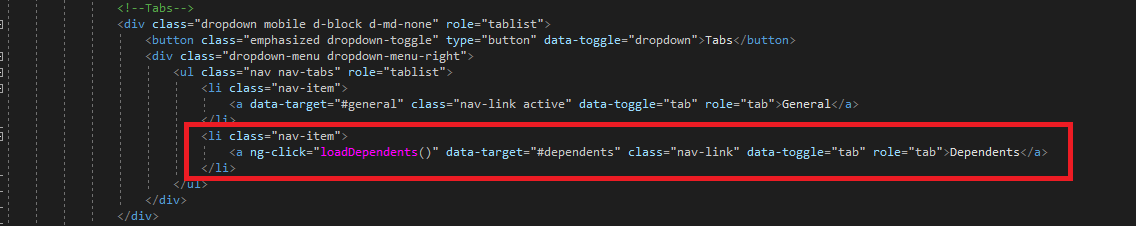


Figure 81 : Add another tab to the dropdown button. This is for the mobile view, but same functionality on the same button we added earlier.

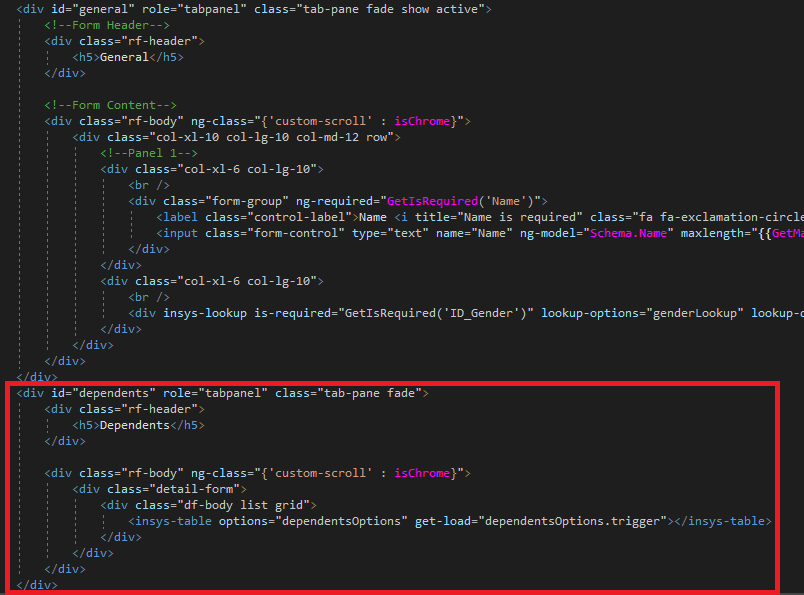


Figure 82 : Add this template to our form. This will be our detail grid.

We are reusing the insysTable like the grid on our listing. But this time it is in Angular Directive form, and we have 2 attributes to configure. First is the “options” attribute. We are passing an angular scope named “dependentsOptions”. Next is the “get-load” attribute. This “dependentsOptions.trigger” is an empty function which the insysTable directive will be using as a container. This trigger is called later on when you click the Dependents tab.

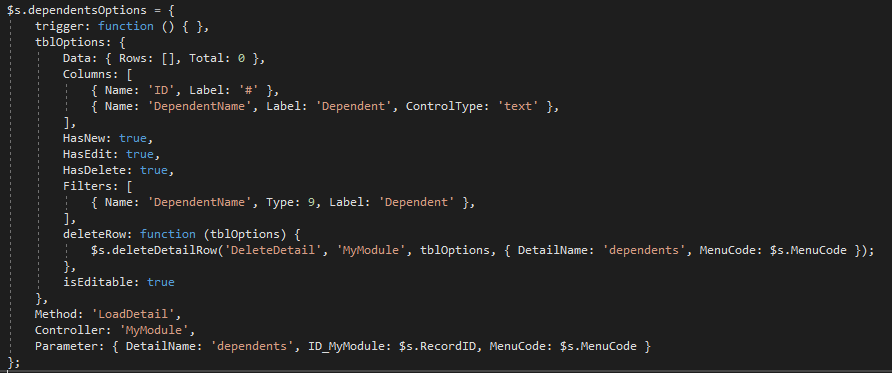


Figure 83 : This is the setup for the dependentsOptions. There is the trigger that I’m talking about earlier. Next is the tblOptions which is the same setup as the listing or grid. The other properties are:

Method, Controller and Paramater. The parameter has 3 properties. The detail name which will be inside our switch statement later on. This is just like the steps when creating a lookup, where you use the generic method “LoadLookup”. This time in our case it’s the “LoadDetail”. Next is the “ID\_MyModule” which is our parent record and the MenuCode property.

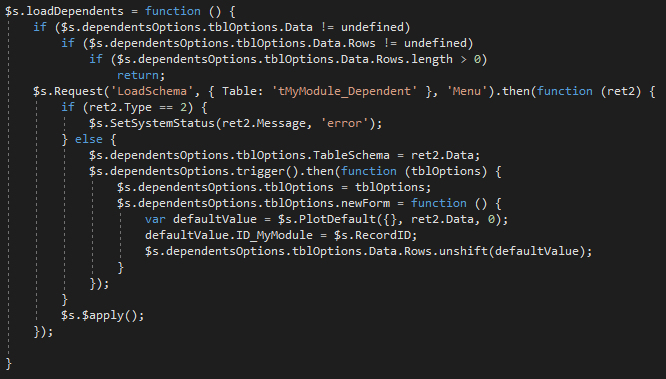


Figure 84 : Now here is our function when clicking the tab. In the start of our function we validate if the detail has already some records to avoid redundant requesting of data to the server. If none then it will request to the server. Next is we request for the “LoadSchema” function. This is to get the schema of the table “tMyModule\_Dependent” and get its default values and check if the columns are required. This is the same in our form. The only difference is we already got the schema from the server when we called the “LoadForm”. After that we now called the “trigger” which I have explained that it is only an empty container. But when the directive “insys-table” initiated it already put a function inside it. Next is we override the “.newForm” function of the tblOptions, why override? Because when we called the “.trigger” the return of that function is the default properties of the insysTable which included the “.newForm”. Don’t be confuse on the name, it just simply add record to the “Rows” property of the “Data” of the tblOptions.

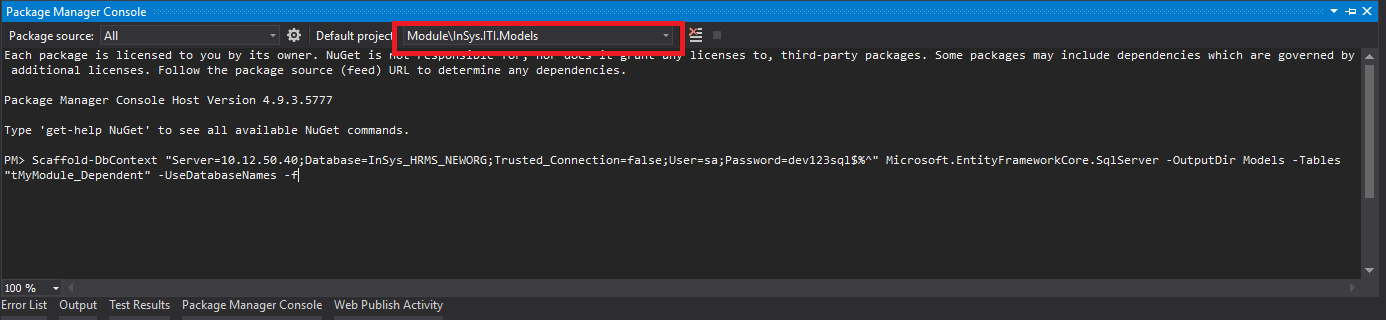


Figure 85 : Next is we run again the scaffold command to create the class tMyModule\_Dependent.cs. Remember that it will create 2 files, first is the context of the database which we don’t need that I have explained earlier on figure 33. Don’t forget to set your default project to InSys.ITI.Models.

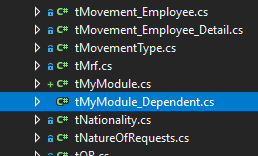


Figure 86 : Look for our tMyModule\_Dependent.cs

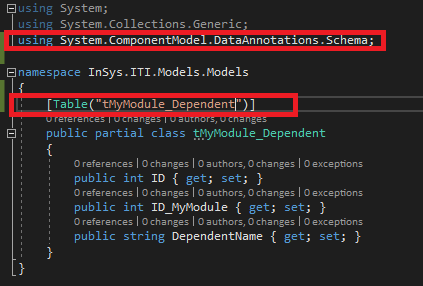


Figure 87 : Add the Table attribute and import the componentModel

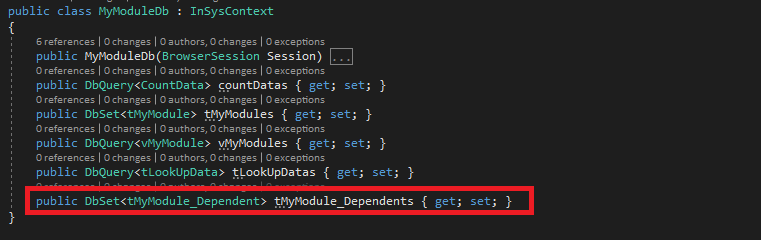


Figure 88 : Add the DbSet<tMyModule\_Dependent> to our MyModuleDb.

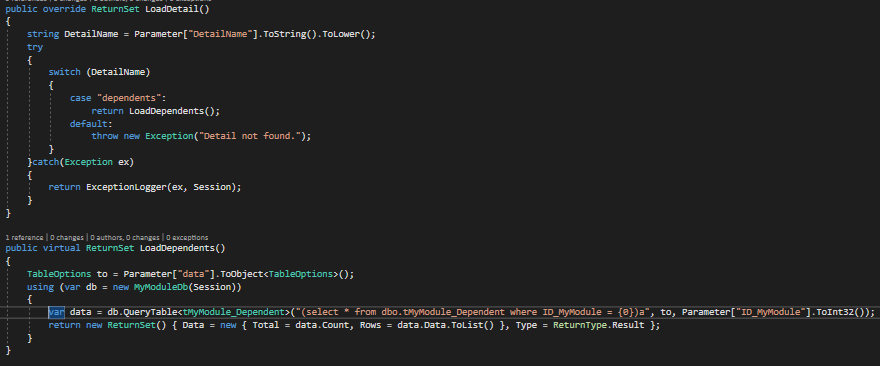


Figure 89 : Now let's override the LoadDetail function and add the method LoadDependent. As you can see we added the filter ID\_MyModule and passed the parameter ID\_MyModule which are present in the configuration in figure 83.

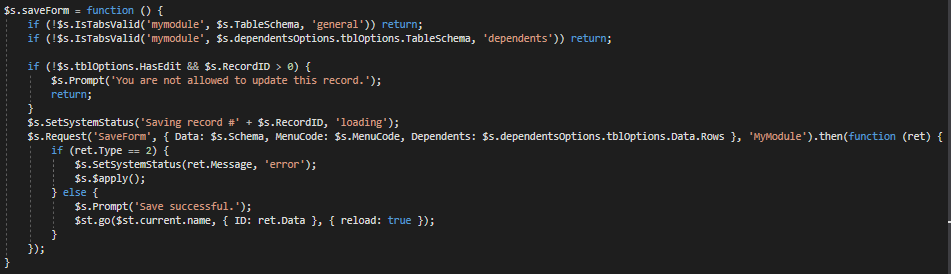


Figure 90 : Next is we transform our saveForm to this. As you can see I have separated the validation of the form and added the parameter value “general” and “$s.TableSchema” to the “IsTabsValid” function. If you look at figure 76 we only used 1 parameter which is the name of the form. Now I have added another “IsTabsValid” validation to validate the detail tab “dependents”. If you look closely at the MyModuleRecord.html the tabs have ID’s. Now the 3rd parameter of the “IsTabsValid” is the ID of the tab. I also added the “Dependents” property to the data that will be sent to the server.

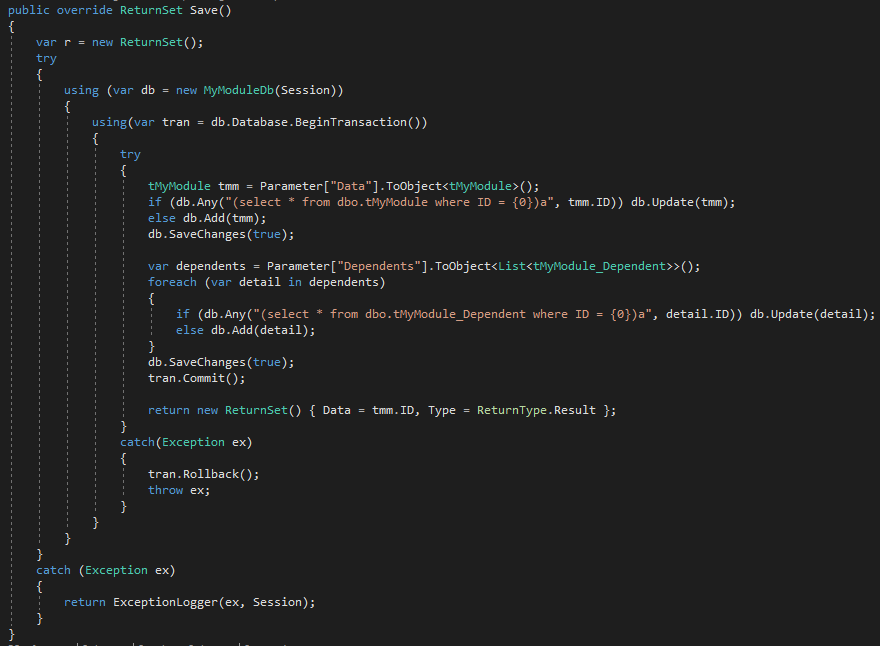


Figure 91 : Next is we transform the Save method. In this image I have added “BeginTransaction” this is to avoid saving the Header which is the “tMyModule” in case the detail fail, because in our Save method, we are saving more than 1 table. Next is we transform the parameter “Dependents” to “List<tMyModule\_Dependent>” because the data we are passing is in array form.

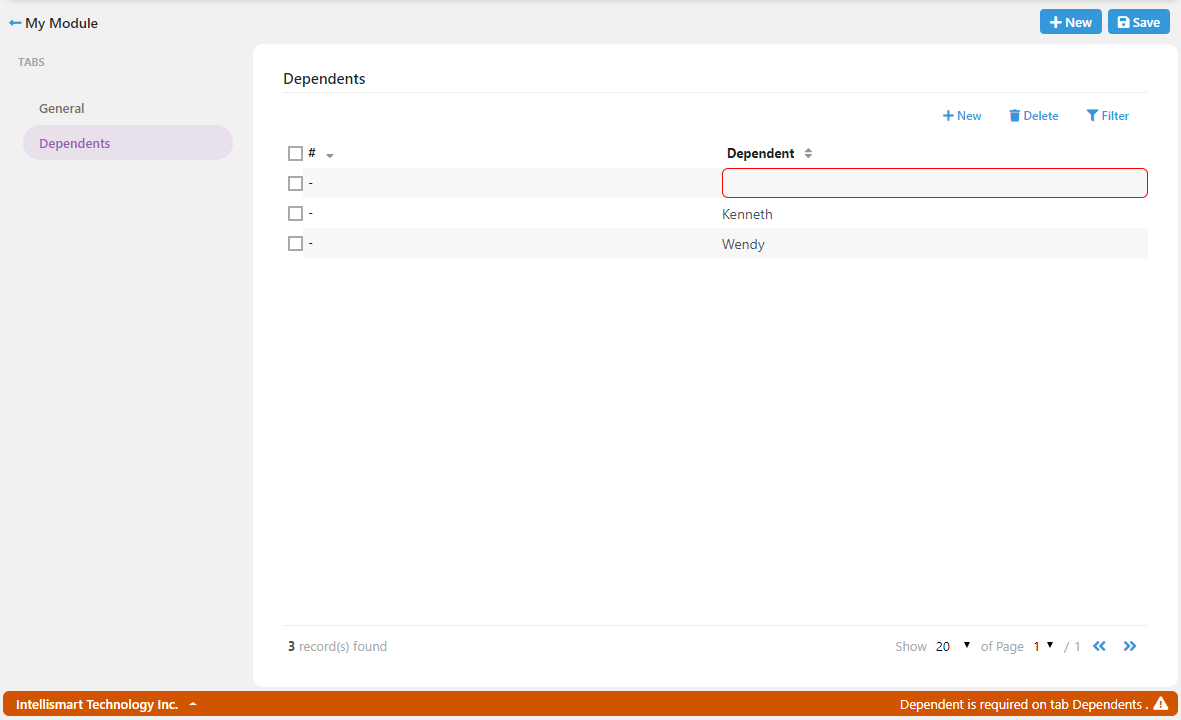


Figure 92 : Now I have added 3 record. I intentionally leave the first record Dependent field blank to check if our “IsTabsValid” is working properly.

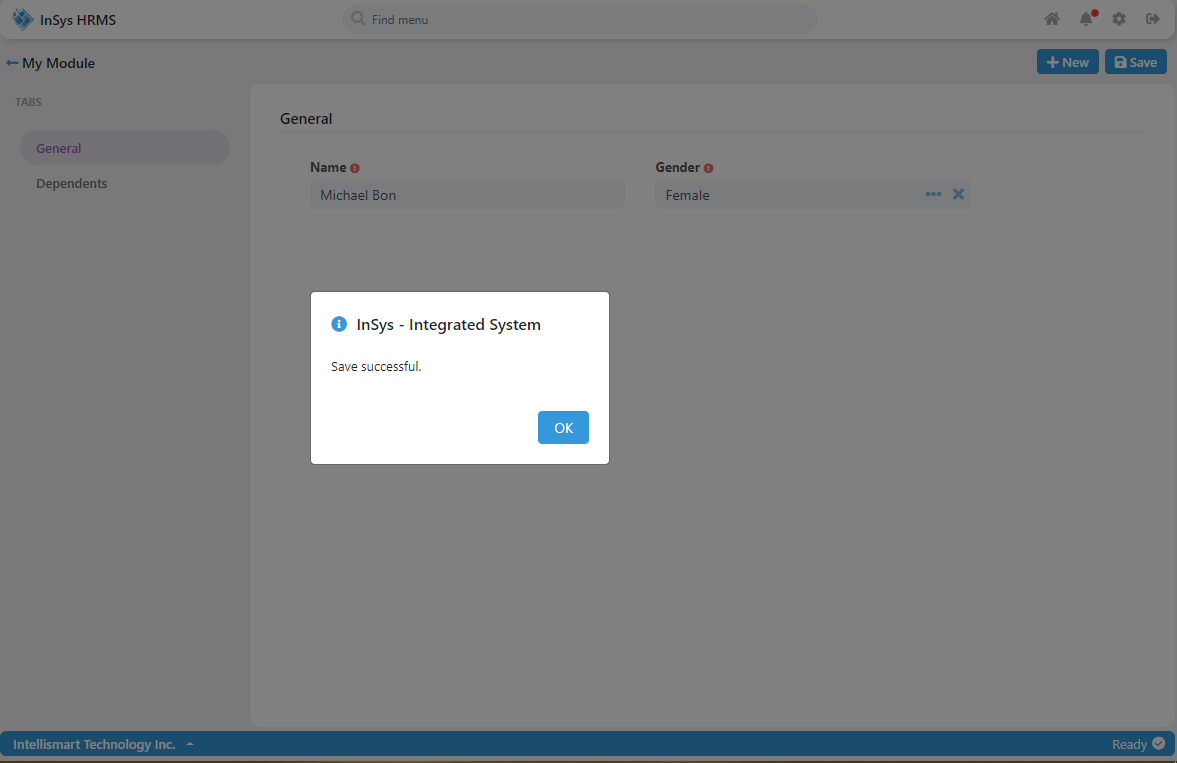
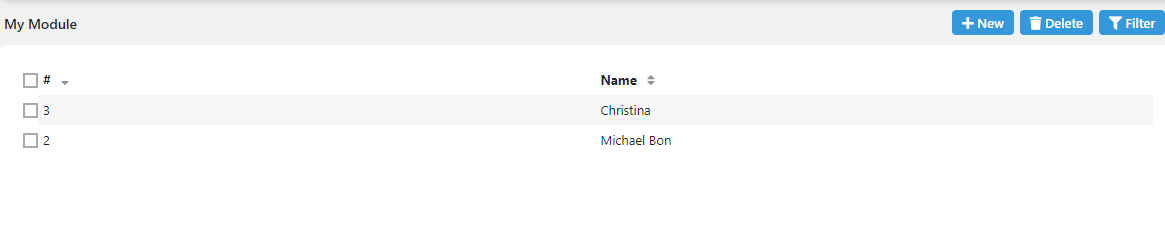


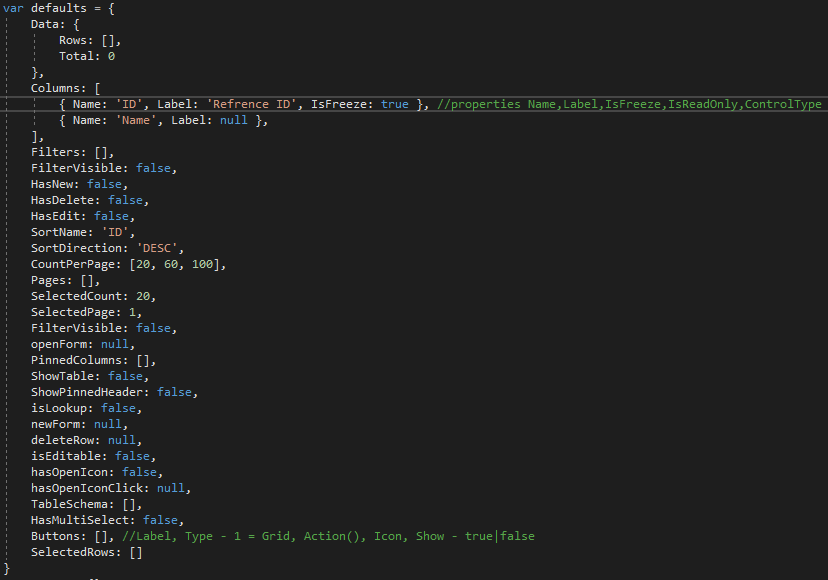
Figure 93 : Congratulations to us, it is working.

1. **DIRECTIVES IN THE SYSTEM**

* **insys-table** – This is the grid used the system, such as the Listing, Editable Detail and Lookup. See figure 82 how to use the **insys-table**.

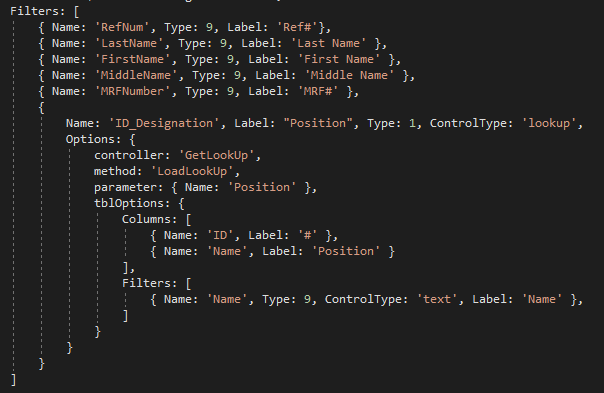


**PROPERTIES**



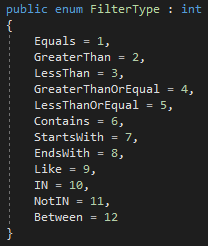
This are the properties to set when using the insys-table

1. **Data** – This is where the data is stored after a request from the server. You can control the data if you need it.
2. **Columns** – This is where you define the columns that are going to appear in the listing.
   1. **Name** – This is the name of the column base on your query or data from the server.
   2. **Label** – This is a masking to the Name if you ever need to change it. E.g. ID\_Gender -> Gender
   3. **IsFreeze** – Set to true if you want this column to be pinned when horizontal scrolling.
   4. **IsReadOnly** – This is only used if you are using the insys-table as editable table.
   5. **DisplayContainer** – This is only used if you are using the ControlType = lookup.
   6. **Options** – This is only used if you are using the ControlType = lookup.
   7. **cascade** – This is only used if you are using the ControlType = select.
   8. **ControlType** – This is only used if you are using the insysTable as editable table. This are the following control types:
      * text
      * number
      * decimal
      * alphabet
      * alphanumeric
      * date
      * lookup
      * select
      * radio
      * time
      * checkbox
3. **Filters** – This is where you defined what columns we are going to use as filter. Note that if you are going to filter a certain column, make sure that the column you are filtering is also included in the select statement of your datasource. The value for filter is only the same as the Columns the only difference is the **Type.**



This is an example of the Filters property used in the MRF module. As you can see the “Type” propery is used.

When this filters are sent to the server this is transformed into Sql filter statement.



This are the classification of types used in the filter.

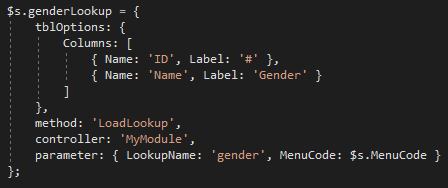
In case you are going the use the “Between” Type, you must initialize the “Value” property of the filter and the value as [null, null] e.g. 

1. **FilterVisible** – The value in this property is set when toggling the filter button.
2. **HasNew** – Set to “true” if you want the New/Add button visible.
3. **HasDelete** – Set to “true” if you want the Delete button visible.
4. **HasEdit** – Set to “true” if you want the record editable. Note that if the **HasNew** is “true” and **HasEdit** is “false” then you can only add record. You cannot update a record if **HasEdit** is “false”.
5. **SortName** – By default column “ID” is sorted. Set the column name you want sorted when the listing / grid first loaded.
6. **SortDirection** – Set the direction of the sorting whether “DESC” or “ASC”. By default it is set to “DESC”.
7. **CountPerPage** – By default [20, 60, 100]. Set your preferred number items to display per page.
8. **Pages** – The value of this property is automatically created when the listing / grid is loaded. By default [].
9. **SelectedCount** – By default it is set to 20 it is the first value in **CountPerPage**.
10. **SelectedPage** – By default this is set to 1.
11. **openForm** – By default this is null. When you add a function in this property it will override the function when opening a record.
12. **PinnedColumns** – This property is automatically populated by all the columns that has **IsFreeze** property that is set to “true”.
13. **ShowTable** – This property toggles the insys-table. It is automatically set to “true” when the **LoadTable** is called.
14. **ShowPinnedHeader** – This property is auto fill by the system. This is the property that pins the header when vertically scrolling.
15. **isLookup** – This property is set to true when using the **insys-table** as **lokup**
16. **newForm** - By default this is null. When you add a function in this property it will override the function of new / add button.
17. **deleteRow** - By default this is null. When you add a function in this property it will override the function of delete button.
18. **isEditable** – By default this is false. Set to “true” and the table will be editable. This is commonly used when the **insys-table** is a detail.
19. **hasOpenIcon** – Set to “true” if you want to show the folder icon.
20. **hasOpenIconClick** - By default this is null. When the table loaded, this is filled with **openForm** function. Meaning it has the same behavior. But if you add a function to this property it will be executed when you click the folder icon.
21. **TableSchema** – By default this is []. This is used when you have multiple detail tables. The **TableSchema** is used on the **IsTabsValid** validation.
22. **HasMultiSelect** – Not use anymore.
23. **Buttons** – If you want to add additional buttons to the table. By default the system buttons are New, Save, Delete and Show Filter. This is a collection of the follow properties of an object.
    * + **Label**
      + **Type** – Set the value to 1.
      + **Action** – Should be a function.
      + **Show** – Set to “true” if you want the button visible.
24. **SelectedRows** – Not use anymore.

* **insys-lookup** – This is a control unlike the dropdown it shows a detailed view of the value selection.



1. **lookup-options** – This is where the insys-table will be looking for its configuration. The properties are:
   1. **tblOptions** – This like the setup of the listing / grid.
   2. **method** - This is the method that will be called in the ApiController provided.
   3. **controller** – This is the ApiController where the method will be called.
   4. **parameter** – This is the values that you will be sending to the server.

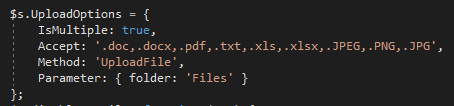


1. **lookup-data** – This is where the lookup control will look for the data. The properties are:
   1. **model** – This is the scope where the lookup will set / get it’s values.
   2. **targetValue** – This is the column name which the lookup will get its value.
   3. **targetDisplay** – This is the column name which the lookup will get what to display in the textbox.
   4. **valueContainer** – This is the property what the lookup will look for in the **model** and set its value.
   5. **displayContainer** – This is the property what the lookup will look for in the **model** and set its display value.
   6. **label** – The value of this property is what will display in the form.

* **insys-upload** – This is the control that upload files to the server.



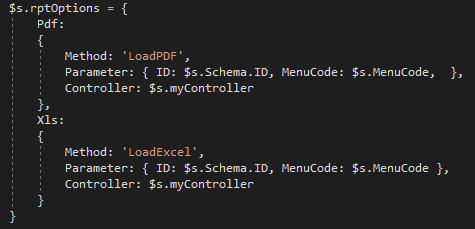
* 1. **readonly** – If you want the upload control as download file only.
  2. **required** – If you want the control to require a file attachment.
  3. **target**-**model** – This is where the filename of the attachment will be stored and save to the server.
  4. **options** – This is the configuration of the **insys-upload**. Note that after selecting files it is already uploaded on the server and it will only return the file names generated in the server that will be saved in the database. If you are looking for what ApiController is used here, it is the UploadController and DownloadController for downloading files.

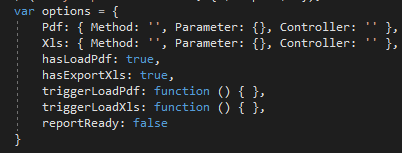


* + - 1. **IsMultiple** – If you want the upload control to have multiple files. Make sure the data type varchar length is high enough to hold the multiple file names it will save to the server.
      2. **Accept** – These are the extensions of the files it will accept when uploading files.
      3. **Method** – This is the default method in the server when uploading files.
      4. **Parameter** – The **folder** property in this example is where files will be stored when uploading.
* **insys-report** – This is the report viewer of the system.



1. **options** – This is the configuration of the **insys-report**. The **options** has 2 properties, the **Pdf** and **Xls**.

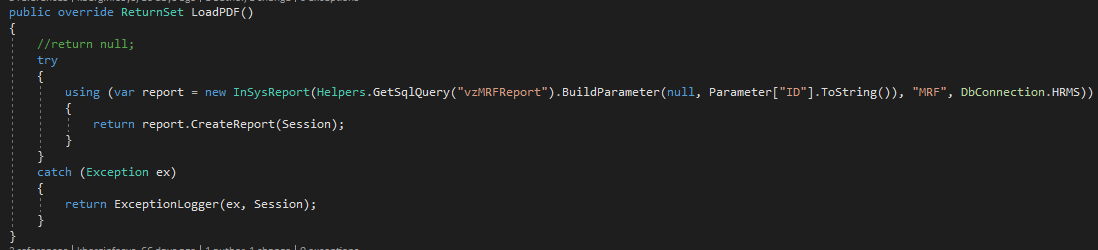




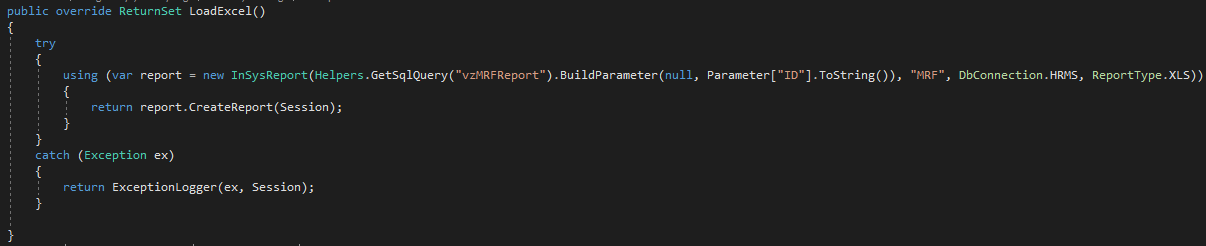
This are the other properties of the **insys-report**

1. **hasLoadPdf** – Set to “true” if you want the **Load** button visible.
2. **hasExportXls** – Set to “true” if you want the **Export to Excel** button visible.
3. **triggerLoadPdf** – This property is auto filled inside the directive. So if you don’t want to show the button **Load** you can call this to execute the **Load** function.
4. **triggerLoadXls** – The same as **triggerLoadPdf**.
5. **reportReady** – This is to indicate that the report template is done loading. If this value is set to “true” you can call the **triggerLoadPdf** or **triggetLoadXls** smoothly.

If you want to create a pdf file you need to override the **LoadPDF** method in your class and add this



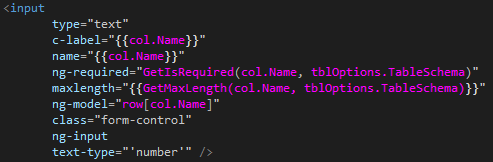
If you want to create an xls file you need to override the **LoadExcel** method in your class and add this



1. **FORM CONTROLS**

There are many form controls in the system

* **Input** – This control has many uses but may vary when it comes to the values.



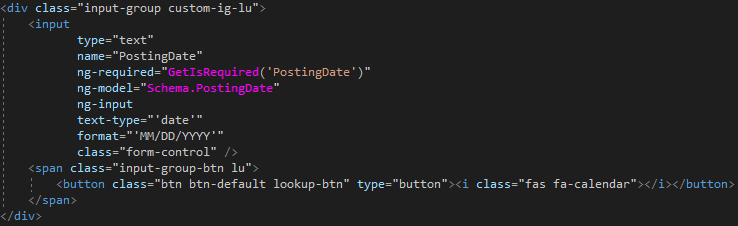
For example: You are requiring the **Input** to accept only numbers, you will need the attribute **ng-input** and **text-type** attribute. In the **text-type** attribute, the value must be inside a (‘) single quote.

Types of value the **Input** can validate:

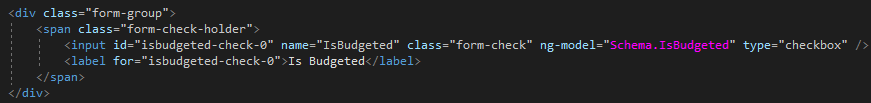
* + **number**
  + **alphabet**
  + **alphanumeric**
  + **decimal** – In case you are going to limit the decimal places it can accept, just add the attribute **decimal-place** and the length of how many places it will accept.

E.g. decimal-place=”2” Expected Output: 12345.12

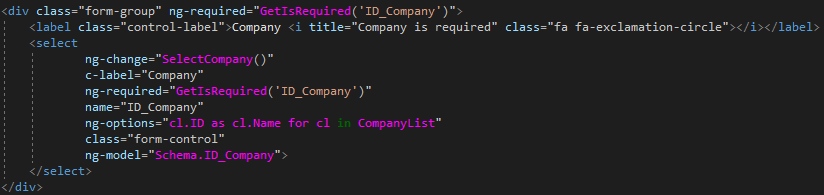
* + **date** – When using the date picker **DON’T ADD THE MAXLENGTH ATTRIBUTE** this is something you don’t want to forget. Always remember this. Also, if you want to change the format of the date just add **text-format** attribute, the value must be inside a (‘) single quote. E.g. text-format=”’MM/DD/YYYY’”



* + **time** – Same as the date picker. Just change the text-format to time format.
* **checkbox** – This control is simple to use.



* **select** – This control is simple to use just the datasource to the **ng-options** attribute.



* **insys-upload** – I have already explained this at the **DIRECTIVES IN THE SYSTEM**
* **insys-lookup** – I have already explained this at the **DIRECTIVES IN THE SYSTEM**

1. **DEVELOPMENT & PRODUCTION PRE-REQUISITE**

Note: Strictly abide to versions listed below. **Upgrade at your own risk**.

* **DEVELOPMENT**
  + **VS2017 ENTERPRISE** (Key NJVYC-BMHX2-G77MM-4XJMR-6Q8QF)
  + **SQL SERVER 2012 ENTERPRISE SERVER OR HIGHER** (Key 748RB-X4T6B-MRM7V-RTVFF-CHC8H)
  + **CRforVS13SP25\_0-10010309** (CRforVS13 Service Pack 25)

<https://origin.softwaredownloads.sap.com/public/file/0020000000898892019>

* + **Bundler & Minifier Extension for VS2017** (In VS2017 click Tools > Extensions and Updates)
  + **Web Compiler Extension for VS2017** (In VS2017 click Tools > Extensions and Updates)
  + **Microsoft .NET Core Runtime – 2.2.3**

<https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-2.2.3-windows-x86-installer>

<https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-2.2.3-windows-x64-installer>

* + **Microsoft .NET SDK 2.2.105**

<https://dotnet.microsoft.com/download/dotnet-core/thank-you/sdk-2.2.105-windows-x64-installer>  
<https://dotnet.microsoft.com/download/dotnet-core/thank-you/sdk-2.2.105-windows-x86-installer>

* + **Microsoft .NET Core 2.2.3 – Windows Server Hosting**

<https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-aspnetcore-2.2.3-windows-hosting-bundle-installer>

* **PRODUCTION**
  + **CR13SP25Redist32\_0-10010309** (CRRuntime 13.0.25)

<https://origin.softwaredownloads.sap.com/public/file/0020000000898842019> (32bit)

<https://origin.softwaredownloads.sap.com/public/file/0020000000898832019> (64bit)

* + **IIS**
  + **SQL SERVER 2012 ENTERPRISE SERVER OR HIGHER** (Key 748RB-X4T6B-MRM7V-RTVFF-CHC8H)
  + **Microsoft .NET Core Runtime – 2.2.3**

<https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-2.2.3-windows-x86-installer>

<https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-2.2.3-windows-x64-installer>

* + **Microsoft .NET Core 2.2.3 – Windows Server Hosting**

<https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-aspnetcore-2.2.3-windows-hosting-bundle-installer>