**Developer’s Guide**

* If you get “XMLHttpRequest cannot load <Your API URL>. Response for preflight has invalid HTTP status code 405”, then add the following code in Global.asax.cs in your WebApi layer.

protected void Application\_BeginRequest()

{

if (Request.Headers.AllKeys.Contains("Origin") && Request.HttpMethod == "OPTIONS")

{

Response.Flush();

}

}

* When the Api layer is hosted on the Local IIS with the **Default application pool,** you may get a SQL Exception stating ‘**Log in failed for the user XXXXXXX’**. The same error might also occur when your Windows credentials are changed. To counter this,
* Create a new application pool in the IIS.
* Select Advanced Settings 🡪 Process Model 🡪 Identity
* Select Custom Account instead of Built-in account
* Select Set and provide your Windows credentials
* Change the application pool of your hosted application from the Default application pool to the newly created application pool.
* Passing parameters from Angular Service to a Web-Api controller.

Note: **Angular Controller** code might change when Component based approach is followed Angular.

1. **Http GET**
   1. Passing an object through a GET request

**Angular Constants:**

(function () {

//This file contains all the Web-Api constants

angular

.module('YourApp')

.constant('apiurls', {

apiDomain: 'http://localhost/REST-WebApi/api/',

//Web-Api Controller LogIn, Users

users: 'Users',

login: 'LogIn/',

})

})();

**Angular Controller:**

//Search the internal user by name or email

function GetUsersBySearchParameters(user) {

//Compose the object from the input. Keep the property names of your UI object same as that of the Api object

var userObject = {

SearchNameText: user.userName,

SearchEmailAddress: user.emailID,

PageSize: 10,

PageNumber: 1,

};

//Call the Angular Service and pass the composed object

ReservationService. GetUsersBySearchParameters(userObject)

.success(function (data) {

//Success Function })

.error(function (error) {

//Error Function });

};

**Angular Service:**

//Search the internal user by name or email – Http Get

function GetUsersBySearchParameters(userObject) {

return $http({

method: 'GET',

url: apiurls.apiDomain**+**apiurls.users**+**'/'**+**' GetUsersBySearchParameters',

params: finalObject

});

};

**Web-Api Controller:**

//Web-Api Action Method for searching the internal user name or email

[HttpGet]

public IEnumerable<User> GetUsersBySearchParameters ([FromUri] UserSearchQueryModel query)

{

//Web-Api Code

}

**Note:** You may create a model class (UserSearchQueryModel) in the backend and use its object(query) to map the received object’s parameters from the Http request. Make sure that you match the parameter names.

**e.g.,**

public class UserSearchQueryModel

{

public string SearchNameText { get; set; }

public string SearchEmailAddress { get; set; }

public int PageSize { get; set; }

public int PageNumber { get; set; }

}

* 1. Passing an id through a GET request

**Angular Controller:**

//Get the details of a single Customer

$scope.GetCustomerById = function (form) {

//Perform form validation

if (form.$valid) {

var Id = $scope.custId;

//Pass the Id to Angular Service

CustomerService.GetSingle(Id)

.success(function (data) { //Success Function })

.error(function () {//Error Function});

}

};

**Angular Service:**

//Get the details of a single Customer - Http Get

function GetCustomerById (id) {

return $http({

method: 'GET',

url: apiurls.apiDomain + apiurls.customer + '/' + id });

};

**Note:** Name of the parameter(id) should match the parameter name present in the Web-Api’s route configuration.

**Web-Api Controller:**

public Customer GetCustomerById (int id)

{

//Web-Api Code

}

* 1. GET request with no parameter

**Angular Controller:**

// Get All Customers details

$scope.GetAllCustomers = function () {

CustomerService.GetAllCustomers()

.success(function (data) {

//Success Function

})

.error(function (error) {

//Error Function

});

};

**Angular Service:**

//View all books – Http Get

function GetAllCustomers () {

return $http({

method: 'GET',

url: apiurls.apiDomain + apiurls.customer

});

};

**Web-Api Controller:**

[HttpGet]

public IEnumerable<Customer> GetAllCustomers()

{

//Web-Api Code

}

* 1. GET request with two parameters

**Angular Controller:**

//Validating the Log-In

$scope.SubmitForm = function (form) {

$rootScope.isLoggedIn = false;

if(form.$valid){

AppService.SubmitForm($scope.user.username, $scope.user.password)

.success(function (response) {

$rootScope.isLoggedIn = response;

//Success Function

})

.error(function (error) {

//Error Function

});

}

};

**Angular Constants:**

(function () {

//This file contains all the Web-Api constants

angular

.module('YourApp')

.constant('apiurls', {

apiDomain: 'http://localhost/REST-WebApi/api/',

//Web-Api Controller LogIn

login: 'LogIn/',

//Action method **Authenticate** using an alias **Happy**

loginMethod: 'Happy',

//Input parameters for the Api Action Method **Authenticate**

userName: '?userName=',

password: '&password='

})

})();

**Angular Service:**

//Validating the Log-In – Http Get

function SubmitForm(userName, password) {

return $http({

method: 'GET',

url: apiurls.apiDomain + apiurls.login + apiurls.loginMethod + apiurls.userName + userName + apiurls.password + password

});

};

**Web-Api Controller:**

[RoutePrefix("api/Happy/{action}")]

public class LogInController : ApiController {

[HttpGet]

public bool Authenticate(string userName, string password)

{

//Web-Api Code

}

}

**Note:**

1**.**The Web-Api Controller is decorated with [RoutePrefix("api/Happy/{action}")]. This is to mask the actual Action Method **Authenticate** from the outside world. An alias **Happy** is used instead and the same is defined as a constant in the Angular(apiurls.loginMethod).

2. Name of the parameter(id) should match the parameter name present in the Web-Api’s route configuration.

**Always use an object to send the parameters to a Web-Api controller if the number of parameters is more than 2.**

**2. Http POST**

**Angular Controller:**

//Add a user

$scope.AddUser = function (form) {

if(form.$valid)

{

//Compose the object from the input. Keep the property names of your UI object same as that of the Api object

var userObj = {

Name: $scope.user.name,

Email: $scope.user.email,

Mobile: $scope.user.mobile,

Address: $scope.user.address

};

//Call the Angular Service and pass the composed object

UserService. AddUser (userObj)

.success(function (response) {

//Success Function

})

.error(function (error) {

//Error Function

});

}

};

**Angular Service:**

//Add a user

function AddUser(userObj) {

return $http({

method: 'POST',

url: apiurls.apiDomain + apiurls.user,

data: JSON.stringify(userObj),

dataType: "json"

});

};

**Web-Api Controller:**

public HttpResponseMessage Post(User value)

{

//Web-Api Code

}

**3. Http PUT**

**Angular Controller:**

//Update the user’s details

$scope.UpdateUserDetails = function (id, name) {

var userObj = {

BookName: name,

Id: id

};

UserService.UpdateUserDetails (userObj)

.success(function (response) {

//Success Function

})

.error(function (error) {

//Error Function

});

};

**Angular Service:**

//Update the user’s details

function UpdateUserDetails (userObj) {

id = userObj.Id;

return $http({

method: 'PUT',

url: apiurls.apiDomain + apiurls.users + '/' + id,

data: JSON.stringify(userObj),

dataType: "json"

});

};

**Web-Api Controller:**

public HttpResponseMessage Put(int id, Book value)

{

//Web-Api Code

}

**Note:**

1. Always Use Http PUT action verb to perform any update/edit operations.
2. It is a good practice to pass two parameters to a PUT operation (an id and an object).

**4. Http DELETE**

**Angular Controller:**

//Delete a user

$scope.DeleteUser = function (id) {

UserService.DeleteUser (Id)

.success(function (response) {

//Success Function

})

.error(function (error) {

//Error Function

});

};

**Angular Service:**

//Delete a user

function DeleteUser (id) {

return $http({

method: 'DELETE',

url: apiurls.apiDomain + apiurls.user + '/' + id

});

};

**Web-Api Controller:**

public HttpResponseMessage Delete(int id)

{

//Web-Api Code

}

**Note:**

1. No need to decorate the Web-Api controller action methods with [HttpGet], [HttpPost], [HttpPut] and [HttpDelete] if the names of your action method start with Get, Post, Put or Delete respectively.

e.g., GetUserDetailsById, DeleteUser

1. If there are multiple Get, Post, Put or Delete action methods in the same Web-Api controller, it is required to decorate the action methods with the appropriate Http action verb and also to include the appropriate [Route("{cntrl}/{action}")].

* **Issues with HTTP-PUT and HTTP-DELETE**

Using the put and delete http verbs without adding the proper configurations in the Web.config file may lead to "**HTTP 405 - Method Not Allowed**" error. To counter the same, make sure that the following configuration is present in the Web.config file of your Web-Api layer.

<system.webServer>

<handlers>

<remove name="ExtensionlessUrlHandler-Integrated-4.0" />

<remove name="OPTIONSVerbHandler" />

<remove name="TRACEVerbHandler" />

<add name="ExtensionlessUrlHandler-Integrated-4.0" path="\*." verb="\*" type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersionv4.0" />

<remove name="WebDAV" />

<remove name="ExtensionlessUrlHandler-ISAPI-4.0\_32bit" />

<remove name="ExtensionlessUrlHandler-ISAPI-4.0\_64bit" />

<remove name="ExtensionlessUrlHandler-Integrated-4.0" />

<add name="ExtensionlessUrlHandler-ISAPI-4.0\_32bit" path="\*." verb="GET,HEAD,POST,DEBUG,PUT,DELETE,PATCH,OPTIONS" modules="IsapiModule" scriptProcessor="%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet\_isapi.dll" preCondition="classicMode,runtimeVersionv4.0,bitness32" responseBufferLimit="0" />

<add name="ExtensionlessUrlHandler-ISAPI-4.0\_64bit" path="\*." verb="GET,HEAD,POST,DEBUG,PUT,DELETE,PATCH,OPTIONS" modules="IsapiModule" scriptProcessor="%windir%\Microsoft.NET\Framework64\v4.0.30319\aspnet\_isapi.dll" preCondition="classicMode,runtimeVersionv4.0,bitness64" responseBufferLimit="0" />

<add name="ExtensionlessUrlHandler-Integrated-4.0" path="\*." verb="GET,HEAD,POST,DEBUG,PUT,DELETE,PATCH,OPTIONS" type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersionv4.0" />

</handlers>

<validation validateIntegratedModeConfiguration="false" />

<modules>

<remove name="WebDAVModule" />

</modules>

<httpProtocol>

<customHeaders>

<add name="Access-Control-Allow-Origin" value="\*" />

<add name="Access-Control-Allow-Methods" value="GET,PUT,POST,DELETE,OPTIONS" />

<add name="Access-Control-Allow-Headers" value="Content-Type" />

</customHeaders>

</httpProtocol>

</system.webServer>

**Note:**1. Custom headers are added to counter the CORS issue.

2. ‘\*’ in the Access-Control-Allow-Origin header has to be replaced with the appropriate values in the production environment.

* **Passing a complex object to Web-Api**

**Angular Controller:**

$scope.AddCustomerAndProduct = function (addform) {

//Perform form validation

if (addform.$valid) {

//Compose the customer object

var customerObj = {

Name: $scope.cust.name,

Email: $scope.cust.email,

Mobile: $scope.cust.mobile,

};

//Compose the product object

var productObj = {

Name: $scope.product.name,

Description: $scope.product.description,

Price: $scope.product.price,

};

//Compose the final object

var finalObj = {};

finalObj.customerData = customerObj;

finalObj.productData = productObj;

//Call the Angular Service and pass the complex object

AddMultipleService.AddCustomerAndProduct (finalObj)

.success(function (response) {

//Success Function })

.error(function (error) {

//Error Function });

}

};

**Angular Service:**

//Add multiple

function AddCustomerAndProduct (finalObj) {

return $http({

method: 'POST',

url: apiurls.apiDomain + apiurls.mutipleAdd,

data: JSON.stringify(finalObj),

dataType: "json"

});

};

**Web-Api Controller:**

Install the NuGet package Newtonsoft.Json and include using Newtonsoft.Json.Linq;

public class AddMultiple : ApiController {

[HttpPost]

public bool HttpResponseMessage Post(CustomerProduct value)

{

Customer customer = value["customerData"].ToObject<Customer>();

Product product = value["productData"].ToObject<Product>();

//Web-Api Code

}

}