Securing ASP.NET Web API using Custom Token Based Authentication

Saineshwar Bageri, 21 Apr 2017

★★★★★ 4.85 (56 votes) Rate:

In this article, we are going to learn how to secure asp.net web API using custom token based authentication.

Download DBScripts.zip - 1.8 KB

Download Music API Store - 32 MB

In modern era of development we use web API for various purpose for sharing data, or for binding grid, drop-down list, and other controls, but if we do not secure this API then other people who are going access your web application or service can misuse it in some or other way and also we are into era of client-side framework (JavaScript, Angular js, react js, express js, common js..Etc) if you are using one of these client-side frameworks then you are using web service or web API it is true for getting or posting data to server and been on client side is less secure you need to add extra efforts to secure it.

In this article we are going to learn that extra part, the process of securing Web API begins with registering process in this part we are first going to register a user, after user registration next user who is registered is going to login into application, after login into application User need to register a company which is going to use this service, after company registration the next step we are going to get ClientID and ClientSecert keys.

After getting keys next we are going use these keys for authentication the first request to access API must come with valid ClientID and ClientSecert next it will validate keys and then it is going to provide Token in response, this token you need to use in every request to authenticate that you are valid user and this Token expires in 30 min but if you want to provide custom time according to your need you can do it.

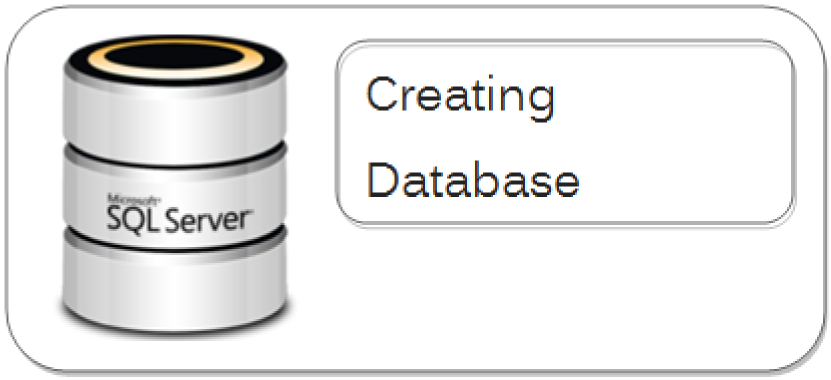
Also, this token is secured using AES 256 encryption algorithm.

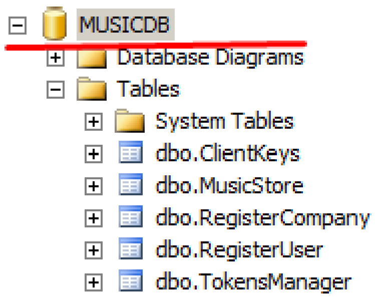
Process

1. Register User
2. Login
3. Register Company
4. Get ClientID and ClientSecert
5. Authenticate to get Token
6. Use Token for Authenticating and accessing Data.

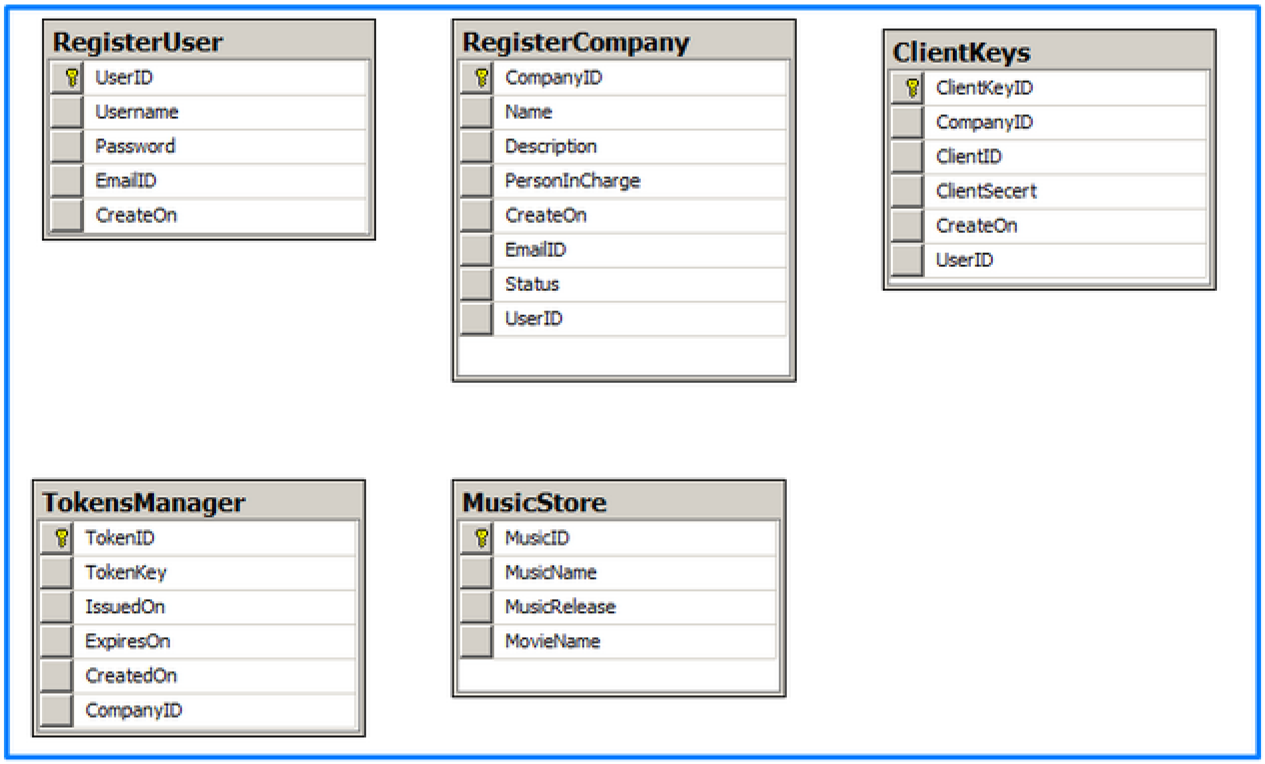
Database parts

In this part, we have created the database with name "MusicDB" and it has 5 tables which we are going to use in the application.



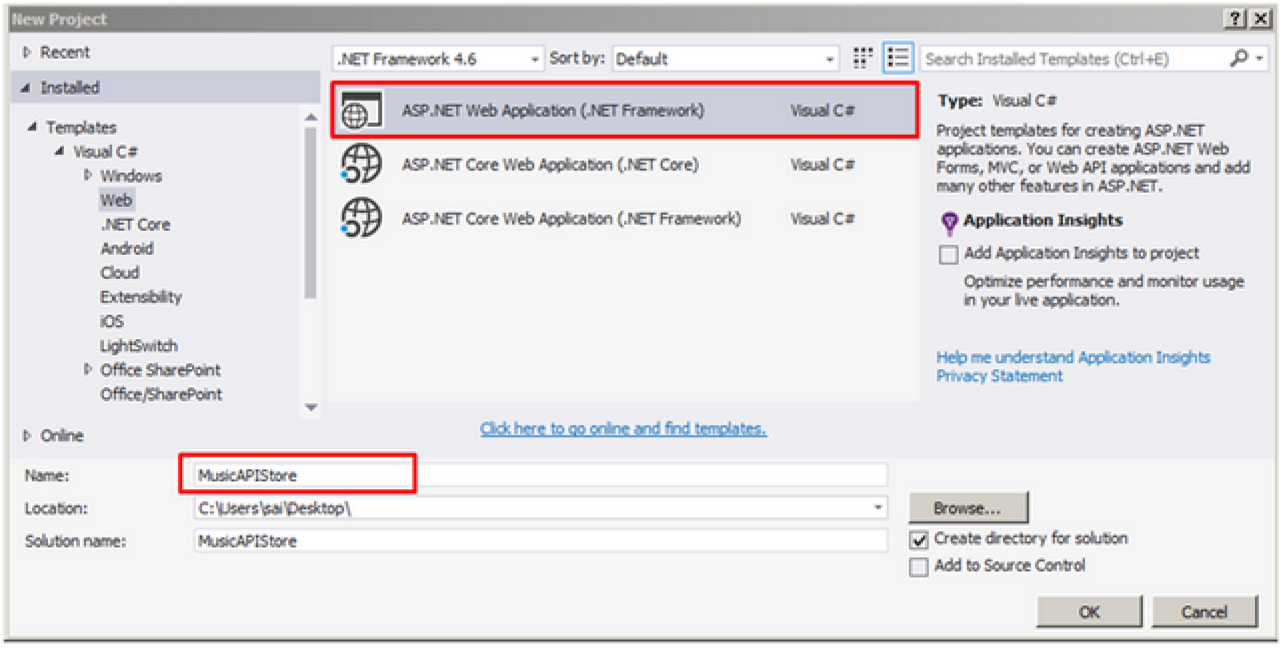


1. Register User: - Stores user information
2. Register Company: - Stores Companies information
3. Client Keys: - Stores ClientID and ClientSecert.
4. TokensManager: - Stores all token information
5. Music Store: - Stores all Music information which is access by Clients.

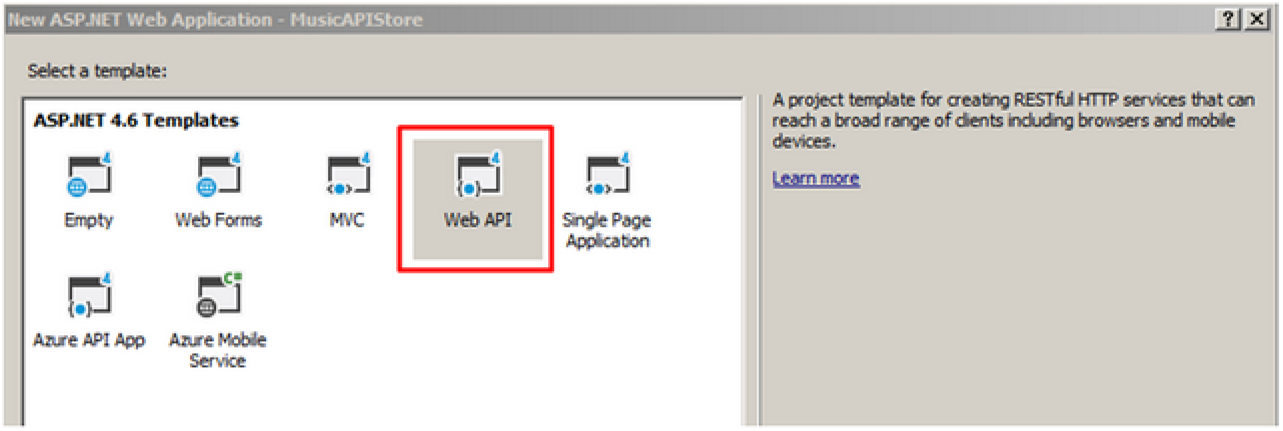


Creating WEB API Application

In this part, we are going to create simple Web API application for creating that I have chosen "**ASP.NET Web Application (.NET Framework)**" template and named the project as "**MusicAPIStore**" and next we are going to choose a template as "Web API" to create a project.

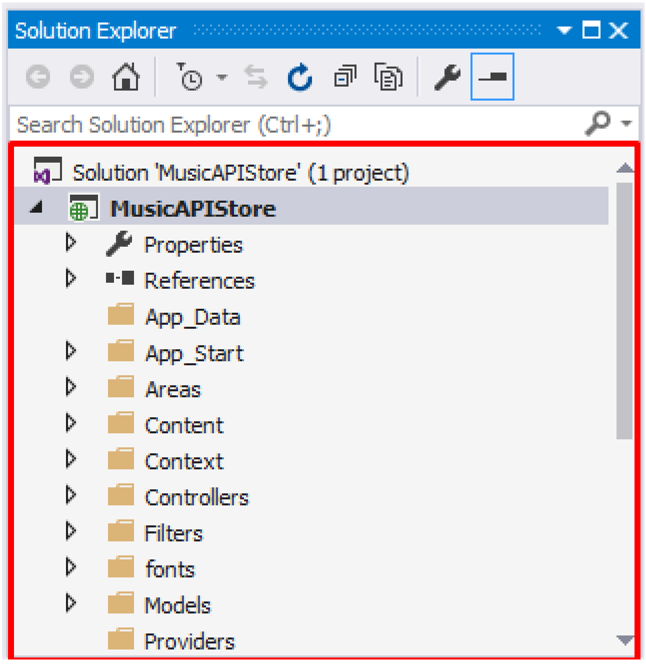

New Project
Recent
Installed
Templates
Visual C#
Windows
Web
.NET Core
Android
Cloud
Extensibility
iOS
LightSwitch
Office SharePoint
Office/SharePoint
Online
.NET Framework 4.6
Sort by: Default
Search Installed Templates (Ctrl+E)
ASP.NET Web Application (.NET Framework) Visual C#
ASP.NET Core Web Application (.NET Core) Visual C#
ASP.NET Core Web Application (.NET Framework) Visual C#
Type: Visual C#
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
Application Insights
Add Application Insights to project
Optimize performance and monitor usage in your live application.
Help me understand Application Insights
Privacy Statement
Click here to go online and find templates.
Browse...
Create directory for solution
Add to Source Control
OK
Cancel

Name: MusicAPIStore
Location: C:\Users\sai\Desktop\
Solution name: MusicAPIStore



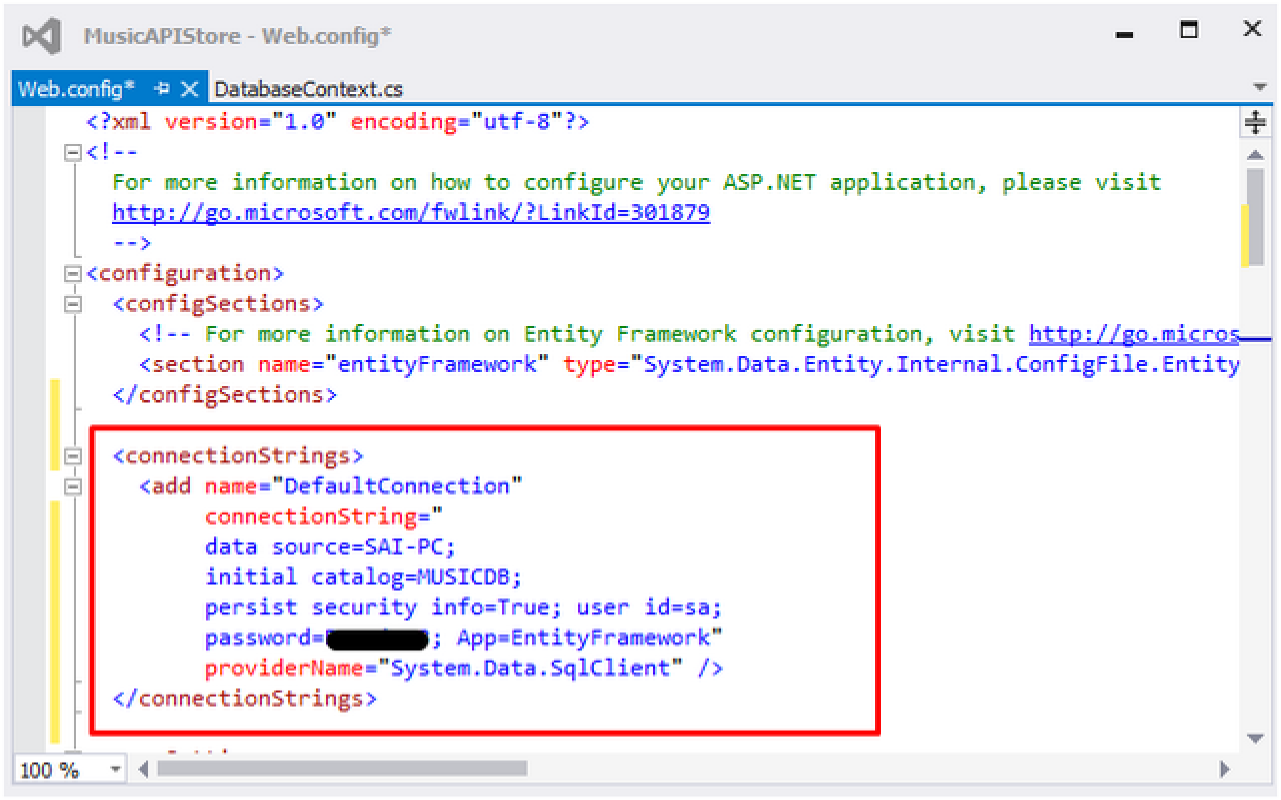
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

After creating project below is complete View of the project.



Using Entity framework Code first approach to connecting application to Database

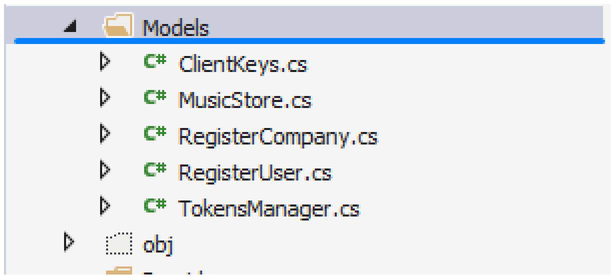
The first step we are going to add connection string of database in Web.config.



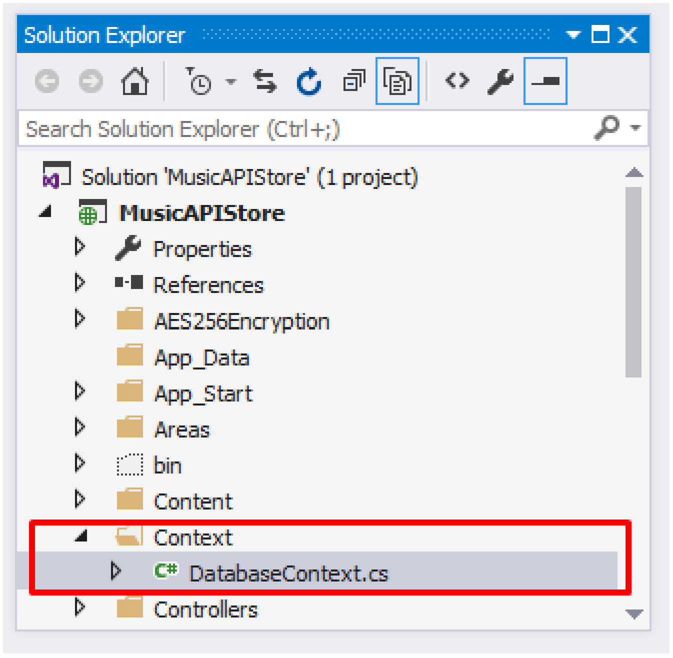
```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=301879
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.micros
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.Entity
  </configSections>

  <connectionStrings>
    <add name="DefaultConnection"
         connectionString="
         data source=SAI-PC;
         initial catalog=MUSICDB;
         persist security info=True; user id=sa;
         password=[redacted]; App=EntityFramework"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
```

The second step we are going to add Models according to tables in "**MusicDB**" database.

In below snapshot, we have created Model of all Tables with the same name as Table name.



The third step creating a class with name DatabaseContext which is going to inherit `DbContext` class than inside this class we are going add a DbSet object.



Code snippet of DatabaseContext class

Hide Copy Code

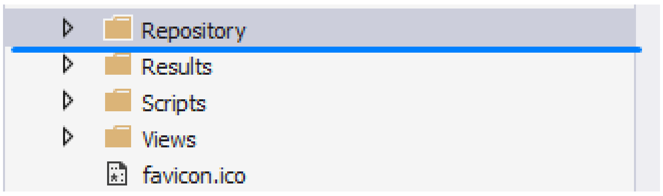
```
using MusicAPIStore.Models;
using System.Data.Entity;

namespace MusicAPIStore.Context
{
    public class DatabaseContext : DbContext
    {
        public DatabaseContext() : base("DefaultConnection")
        {

        }

        public DbSet<RegisterUser> RegisterUser { get; set; }
        public DbSet<RegisterCompany> RegisterCompany { get; set; }
        public DbSet<TokensManager> TokensManager { get; set; }
        public DbSet<ClientKeys> ClientKeys { get; set; }
        public DbSet<MusicStore> MusicStore { get; set; }
    }
}
```

After completing with adding `DatabaseContext` class next we are going to add folder with name Repository to Project.

Adding Repository Folder to Application

In this application, we are going to use repository pattern.

For storing interfaces and concrete class we have to add Repository folder to the application.



We have completed adding Repository folder to the application next we are going to add RegisterUser, Model, Interface, Concrete class, Controller, and Views.

Register User

1. RegisterUser Model
2. Adding IRegisterUser Interface
3. Adding RegisterUserConcrete
4. RegisterUserConcrete will inherit IRegisterUser
5. Adding Controller
6. Adding View.

Let’s see RegisterUser Model

1. RegisterUser Model

Hide Shrink ▲ Copy Code

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MusicAPIStore.Models
{
    [Table("RegisterUser")]
    public class RegisterUser
    {
        [Key]
        public int UserID { get; set; }

        [Required(ErrorMessage = "Required Username")]
        [StringLength(30, MinimumLength = 2, ErrorMessage = "Username Must be Minimum 2 Charaters")]
        public string Username { get; set; }

        [DataType(DataType.Password)]
        [Required(ErrorMessage = "Required Password")]
        [MaxLength(30,ErrorMessage = "Password cannot be Greater than 30 Charaters")]
        [StringLength(31, MinimumLength = 7 , ErrorMessage ="Password Must be Minimum 7 Charaters")]
        public string Password { get; set; }
        public DateTime CreateOn { get; set; }

    [Required(ErrorMessage = "Required EmailID")]
    [RegularExpression(@"[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}",ErrorMessage = "Please enter Valid Email
ID")]
        public string EmailID { get; set; }
    }
}
```

After adding RegisterUser Model next we are going to add Interface IRegisterUser in which we are going to declare methods which we require.

2. IRegisterUser Interface

We are going to add IRegisterUser interface in Repository folder which we have created.

This interface contains 4 methods

1. `Add` (Inserting User Data into database)
2. `ValidateRegisteredUser`

   (Validating User already exists in database or not returns Boolean value)
3. `ValidateUsername`

   (Validating Username is already existed in database or not returns Boolean value)
4. `GetLoggedUserID` (Get's UserID by Username and Password)

Code snippet

Hide Copy Code

```
using MusicAPIStore.Models;

namespace MusicAPIStore.Repository
{
    public interface IRegisterUser
    {
        void Add(RegisterUser registeruser);
        bool ValidateRegisteredUser(RegisterUser registeruser);
        bool ValidateUsername(RegisterUser registeruser);
        int GetLoggedUserID(RegisterUser registeruser);
    }
}
```

After adding `IRegisterUser` Interface next we are going add `RegisterUserConcrete` Class in Repository folder.

3. RegisterUserConcrete

We are going to add `RegisterUserConcrete` Class in Repository folder which we have created.

After adding `RegisterUserConcrete` class next we are going to inherit it from IRegisterUser Interface and implement all method inside of IRegisterUser Interface

Code snippet

Hide Shrink Copy Code

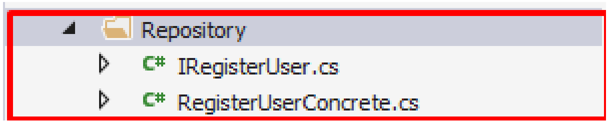
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using MusicAPIStore.Models;
using MusicAPIStore.Context;

namespace MusicAPIStore.Repository
{
    public class RegisterUserConcrete : IRegisterUser
    {
        DatabaseContext _context;
        public RegisterUserConcrete()
        {
            _context = new DatabaseContext();
        }

        public void Add(RegisterUser registeruser)
        {
            _context.RegisterUser.Add(registeruser);
            _context.SaveChanges();
        }

        public int GetLoggedUserID(RegisterUser registeruser)
        {
            var usercount = (from User in _context.RegisterUser
                             where User.Username == registeruser.Username &&
                                   User.Password == registeruser.Password
                             select User.UserID).FirstOrDefault();

            return usercount;
        }

        public bool ValidateRegisteredUser(RegisterUser registeruser)
        {
            var usercount = (from User in _context.RegisterUser
                              where User.Username == registeruser.Username &&
                              User.Password == registeruser.Password
                         select User).Count();
            if (usercount > 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }


        public bool ValidateUsername(RegisterUser registeruser)
        {
            var usercount = (from User in _context.RegisterUser
                             where User.Username == registeruser.Username
                             select User).Count();
            if (usercount > 0)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

After adding `RegisterUserConcrete` class next we are going to Add `RegisterUserController`.

4. Adding RegisterUser Controller

For adding controller just right on Controllers folder inside that select Add a Inside that select Controller after selecting Controller a new dialog with name "Add Scaffold" will Pop up for choosing type of controller to add in that we are just going to select "**MVC5Controller - Empty**" and click on Add button, after that a new dialog with name "**Add Controller**" will pop up asking for Controller name here we are going to name controller as RegisterUserController and click on Add button.

Note: Next we are going to initialize an object in Constructor of RegisterUser Controller.



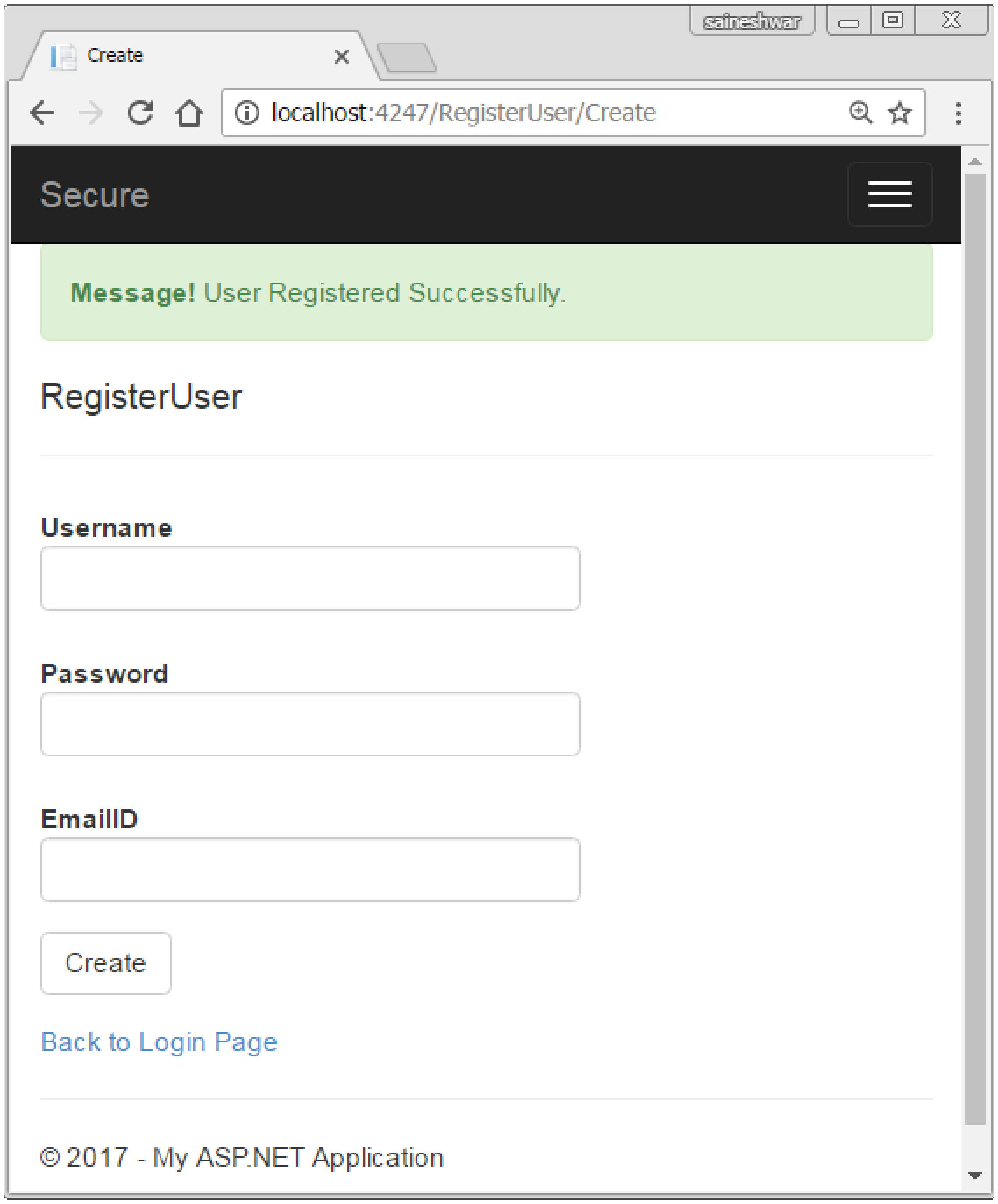
Code snippet

Hide Shrink Copy Code

```
using MusicAPIStore.AES256Encryption;
using MusicAPIStore.Models;
using MusicAPIStore.Repository;
using System;
using System.Web.Mvc;

namespace MusicAPIStore.Controllers
{
    public class RegisterUserController : Controller
    {
        IRegisterUser repository;
        public RegisterUserController()
        {
            repository = new RegisterUserConcrete();
        }
        // GET: RegisterUser/Create
        public ActionResult Create()
        {
            return View(new RegisterUser());
        }

        // POST: RegisterUser/Create
        [HttpPost]
        public ActionResult Create(RegisterUser RegisterUser)
        {
            try
            {
                if (!ModelState.IsValid)
                {
                    return View("Create", RegisterUser);
                }

                // Validating Username
                if (repository.ValidateUsername(RegisterUser))
                {
                    ModelState.AddModelError("", "User is Already Registered");
                    return View("Create", RegisterUser);
                }
                RegisterUser.CreateOn = DateTime.Now;

                // Encrypting Password with AES 256 Algorithm
                RegisterUser.Password = EncryptionLibrary.EncryptText(RegisterUser.Password);

                // Saving User Details in Database
                repository.Add(RegisterUser);
                TempData["UserMessage"] = "User Registered Successfully";
                ModelState.Clear();
                return View("Create", new RegisterUser());
            }
            catch
            {
                return View();
            }
        }
    }
}
```

In this Controller, we have added 2 Action Methods of Create one for handling `[HttpGet]` request and other for handling `[HttpPost]` request. In `[HttpPost]` request we are going to first Validate Is Username already exists or not if not then we are going to Create User, next we are also taking Password as input which we cannot store in database as clear text we need to store it in encrypted format for doing that we are going to Use AES 256 algorithm .

5. RegisterUser View



Message! User Registered Successfully.

RegisterUser

Username

Password

EmailID

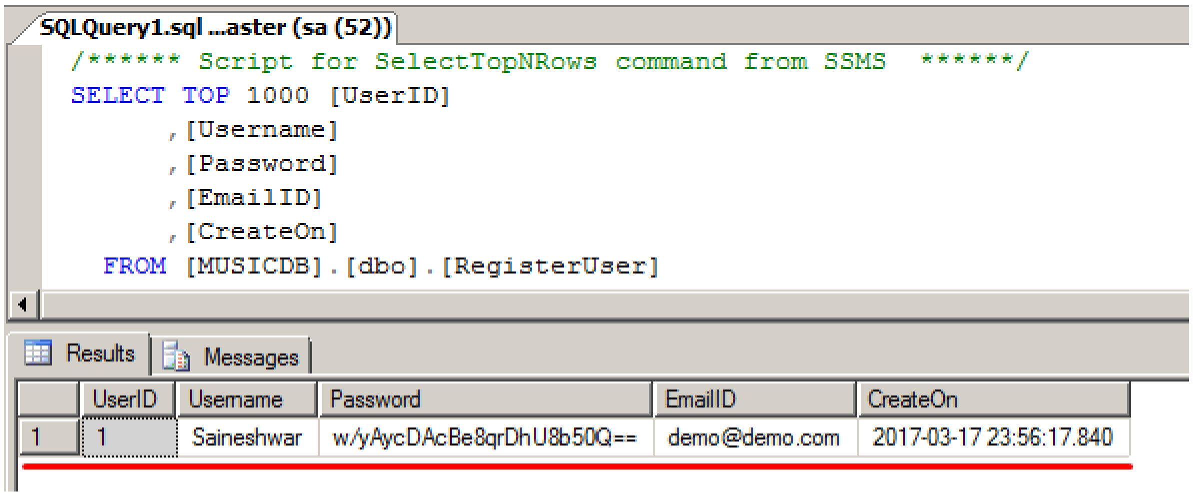
Create

Back to Login Page

© 2017 - My ASP.NET Application

After Registering a User Below are details which get stored in RegisterUser Table.

6. RegisterUser Table

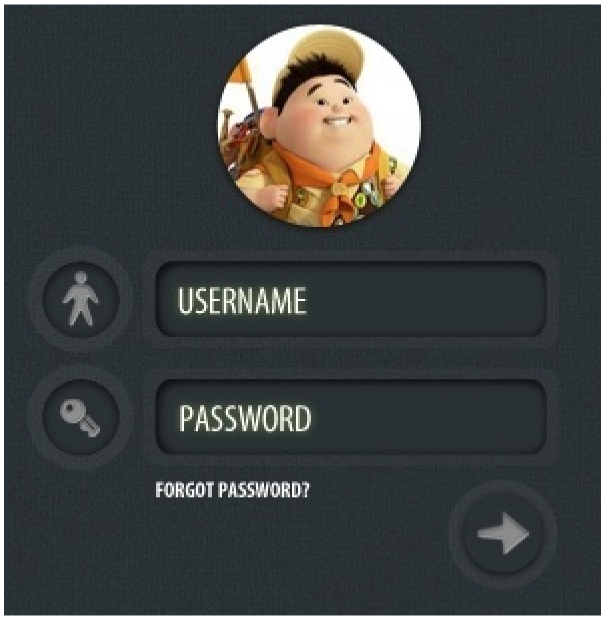


```sql
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [UserID]
      ,[Username]
      ,[Password]
      ,[EmailID]
      ,[CreateOn]
  FROM [MUSICDB].[dbo].[RegisterUser]
```

	UserID	Username	Password	EmailID	CreateOn
1	1	Saineshwar	w/yAycDAcBe8qrDhU8b50Q==	demo@demo.com	2017-03-17 23:56:17.840

After completing with Registration part next we are going create a Login page.

2. Login



In this part, we are going to add Login Controller with Login and Logout Action Method in it.

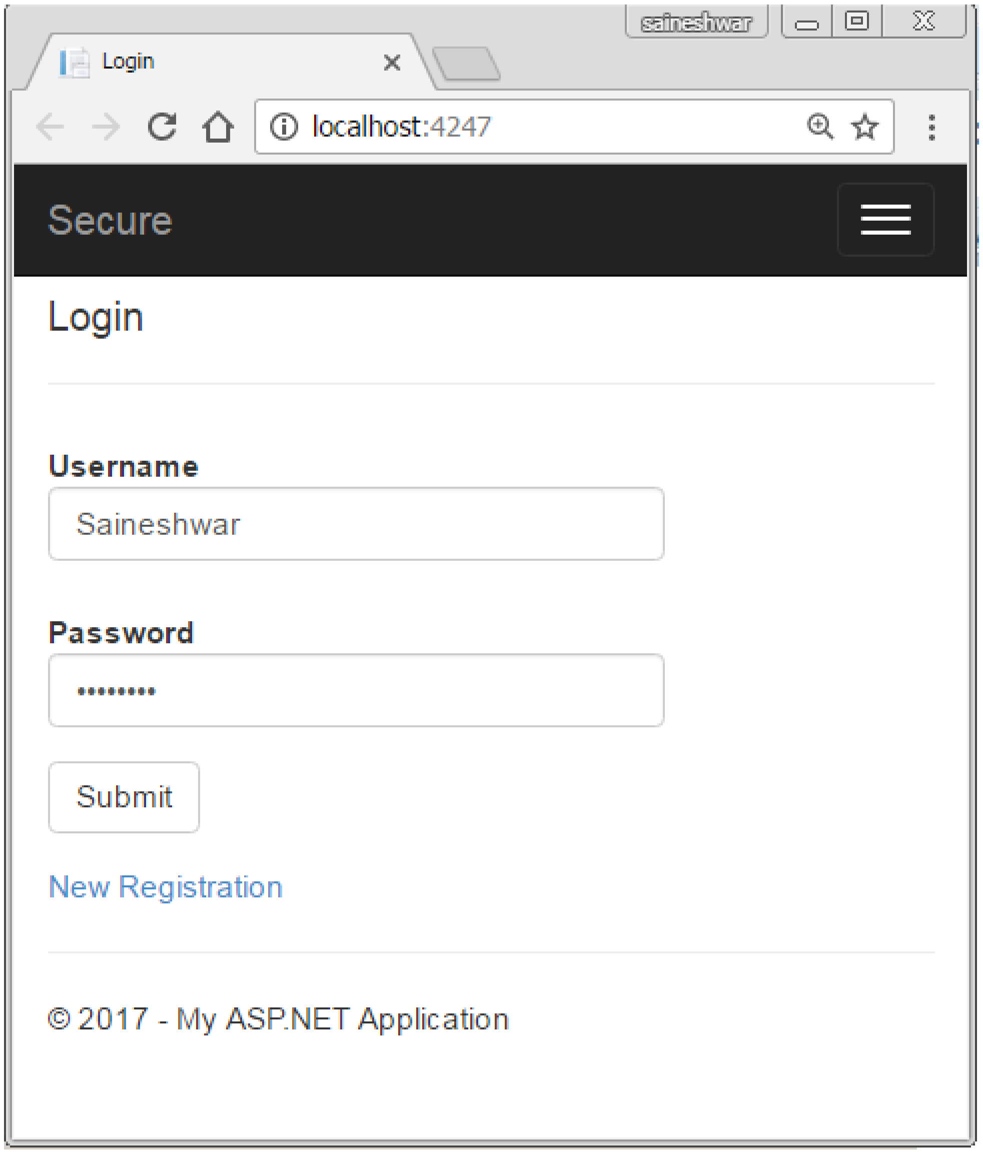
Here we do not need to add a new interface or concrete class because we have already created a `RegisterUserConcrete` method which has a method which is required by Login Controller.

Code snippet

Hide Shrink ▲ Copy Code

```csharp
using MusicAPIStore.AES256Encryption;
using MusicAPIStore.Models;
using MusicAPIStore.Repository;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
```

```
namespace MusicAPIStore.Controllers
{
    public class LoginController : Controller
    {
        IRegisterUser _IRegisterUser;
        public LoginController()
        {
            _IRegisterUser = new RegisterUserConcrete();
        }

        public ActionResult Login()
        {
            return View(new RegisterUser());
        }

        [HttpPost]
        public ActionResult Login(RegisterUser RegisterUser)
        {
            try
            {

                if (string.IsNullOrEmpty(RegisterUser.Username) &&
(string.IsNullOrEmpty(RegisterUser.Password)))
                {
                    ModelState.AddModelError("", "Enter Username and Password");
                }
                else if (string.IsNullOrEmpty(RegisterUser.Username))
                {
                    ModelState.AddModelError("", "Enter Username");
                }
                else if (string.IsNullOrEmpty(RegisterUser.Password))
                {
                    ModelState.AddModelError("", "Enter Password");
                }
                else
                {

                    RegisterUser.Password = EncryptionLibrary.EncryptText(RegisterUser.Password);

                    if (_IRegisterUser.ValidateRegisteredUser(RegisterUser))
                    {
                        var UserID = _IRegisterUser.GetLoggedUserID(RegisterUser);
                        Session["UserID"] = UserID;
                        return RedirectToAction("Create", "RegisterCompany");
                    }
                    else
                    {
                        ModelState.AddModelError("", "User is Already Registered");
                        return View("Create", RegisterUser);
                    }
                }

                return View("Login", RegisterUser);
            }
            catch
            {
                return View();
            }
        }

        public ActionResult Logout()
        {
            Session.Abandon();
            return RedirectToAction("Login", "Login");
        }
    }
}
```



Login

Username

Saineshwar

Password

••••••••

Submit

New Registration

© 2017 - My ASP.NET Application

After completing with login next we going to create Register Company Interface and Concrete in Repository folder and after that, we are going add `RegisterCompanyController` and Action Method along with View.

Note: We are using AES 256 algorithm with salt for encryption and decryption.

3. Register Company

In this part, we are going to register a company and to this company, we are going to generate ClientID and ClientSecert.

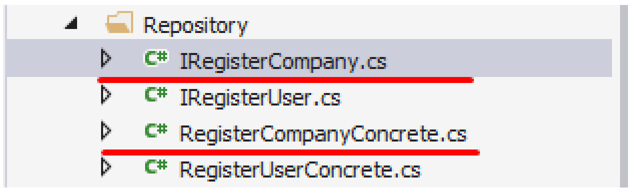
Let's start with adding an interface with name `IRegisterCompany` which will contain all methods which need to be implemented by `Concrete` class.

Code snippet of IRegisterCompany

Hide Copy Code

```
using MusicAPIStore.Models;
using System.Collections.Generic;

namespace MusicAPIStore.Repository
{
    public interface IRegisterCompany
    {
        IEnumerable<RegisterCompany> ListofCompanies(int UserID);
        void Add(RegisterCompany entity);
        void Delete(RegisterCompany entity);
        void Update(RegisterCompany entity);
        RegisterCompany FindCompanyByUserId(int UserID);
        bool ValidateCompanyName(RegisterCompany registercompany);
        bool CheckIsCompanyRegistered(int UserID);
    }
}
```

We have declared all method which is required by Register Company Controller next we are going to add `Concrete` class with name `RegisterCompanyConcrete` which is going to implement the `IRegisterCompany` interface.



Code snippet

Hide Shrink ▲ Copy Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using MusicAPIStore.Models;
using MusicAPIStore.Context;

namespace MusicAPIStore.Repository
{
    public class RegisterCompanyConcrete : IRegisterCompany
    {
        DatabaseContext _context;
        public RegisterCompanyConcrete()
        {
            _context = new DatabaseContext();
        }

        public IEnumerable<RegisterCompany> ListofCompanies(int UserID)
        {
            try
            {
                var CompanyList = (from companies in _context.RegisterCompany
                                   where companies.UserID == UserID
                                   select companies).ToList();
                return CompanyList;
            }
            catch (Exception)
            {
                throw;
            }
        }

        public void Add(RegisterCompany entity)
        {
            try
            {
                _context.RegisterCompany.Add(entity);
                _context.SaveChanges();
            }
            catch (Exception)
            {

                throw;
            }
        }

        public void Delete(RegisterCompany entity)
        {
            try
            {
                var itemToRemove = _context.RegisterCompany.SingleOrDefault(x => x.CompanyID ==
entity.CompanyID);
                _context.RegisterCompany.Remove(itemToRemove);
                _context.SaveChanges();
            }
            catch (Exception)
            {
                throw;
            }
        }

        public RegisterCompany FindCompanyByUserId(int UserID)
        {
            try
            {
                var Company = _context.RegisterCompany.SingleOrDefault(x => x.UserID == UserID);
                return Company;
            }
            catch (Exception)
            {

                throw;
            }
        }


        public bool ValidateCompanyName(RegisterCompany registercompany)
        {
            try
            {
                var result = (from company in _context.RegisterCompany
                              where company.Name == registercompany.Name &&
                                    company.EmailID == registercompany.EmailID
                              select company).Count();
                if (result > 0)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
            catch (Exception)
            {

                throw;
            }
        }


        public bool CheckIsCompanyRegistered(int UserID)
        {
            try
            {
                var companyExists = _context.RegisterCompany.Any(x => x.UserID == UserID);

                if (companyExists)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
            catch (Exception)
            {
                throw;
            }
        }

    }
}
```

After completing with implementing all methods of interface next we are going to Add RegisterCompany Controller.

Adding RegisterCompany Controller

In this part, we are going to add RegisterCompany Controller with Create and index Action Method in it.

In index Action Method we are going to get list of Companies and in creating `[HttpGet]` Action Method we are going get list of companies on basis of UserID, and in creating `[HttpPost]` Action Method we are going save data of company in database and before that we are going to check is company name already exists or not if company name exist then we are going to show error message "**Company is Already Registered**".

For adding Controller follow the same step which we have used for adding RegisterUser Controller after adding controller we have just manually added a constructor and 3 Action Method in it as shown below.

Code snippet

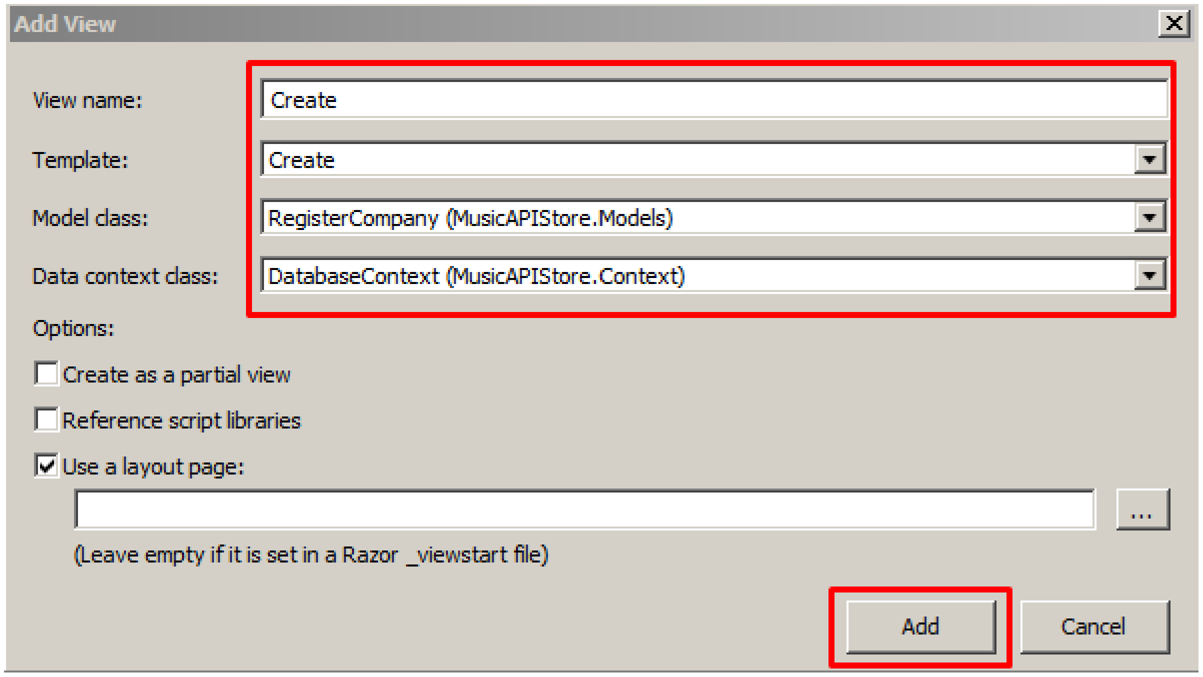
Hide Shrink Copy Code

```
using MusicAPIStore.Context;
using MusicAPIStore.Models;
using MusicAPIStore.Repository;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MusicAPIStore.Filters;
namespace MusicAPIStore.Controllers
{
    [ValidateSessionAttribute]
    public class RegisterCompanyController : Controller
    {
        IRegisterCompany _IRegister;
        public RegisterCompanyController()
        {
            _IRegister = new RegisterCompanyConcrete();
        }
        // GET: Register
       public ActionResult Index()
        {
            var RegisterList = _IRegister.ListofCompanies(Convert.ToInt32(Session["UserID"]));
            return View(RegisterList);
        }

        // GET: Register/Create
        public ActionResult Create()
        {
            var Company = _IRegister.CheckIsCompanyRegistered(Convert.ToInt32(Session["UserID"]));
            if (Company)
            {
                return RedirectToAction("Index");
            }
            return View();
        }

        // POST: Register/Create
        [HttpPost]
        public ActionResult Create(RegisterCompany RegisterCompany)
        {
            try
            {
                if (!ModelState.IsValid)
                {
                    return View("Create", RegisterCompany);
                }

                if (_IRegister.ValidateCompanyName(RegisterCompany))
                {
                    ModelState.AddModelError("", "Company is Already Registered");
                    return View("Create", RegisterCompany);
                }
                RegisterCompany.UserID = Convert.ToInt32(Session["UserID"]);
                RegisterCompany.CreateOn = DateTime.Now;
                _IRegister.Add(RegisterCompany);

                return RedirectToAction("Index");
            }
            catch
            {
                return View();
            }
        }


    }
}
```

After completing with adding Controller and its Action Method next we are going to add View to Create and index Action Method.

Adding Index and Company View

In this part we are going to add Views for adding View just right click inside Action Method then choose Add View from Menu list a new dialog will pop up with name "**Add View**" next we do not require to provide name to view it is set to default which is Action Method name in Template choose template to depend on upon which view you want to create (Create, Index) and long with that choose Model (**RegisterCompany**) Click on Add Button.



View name: Create

Template: Create

Model class: RegisterCompany (MusicAPIStore.Models)

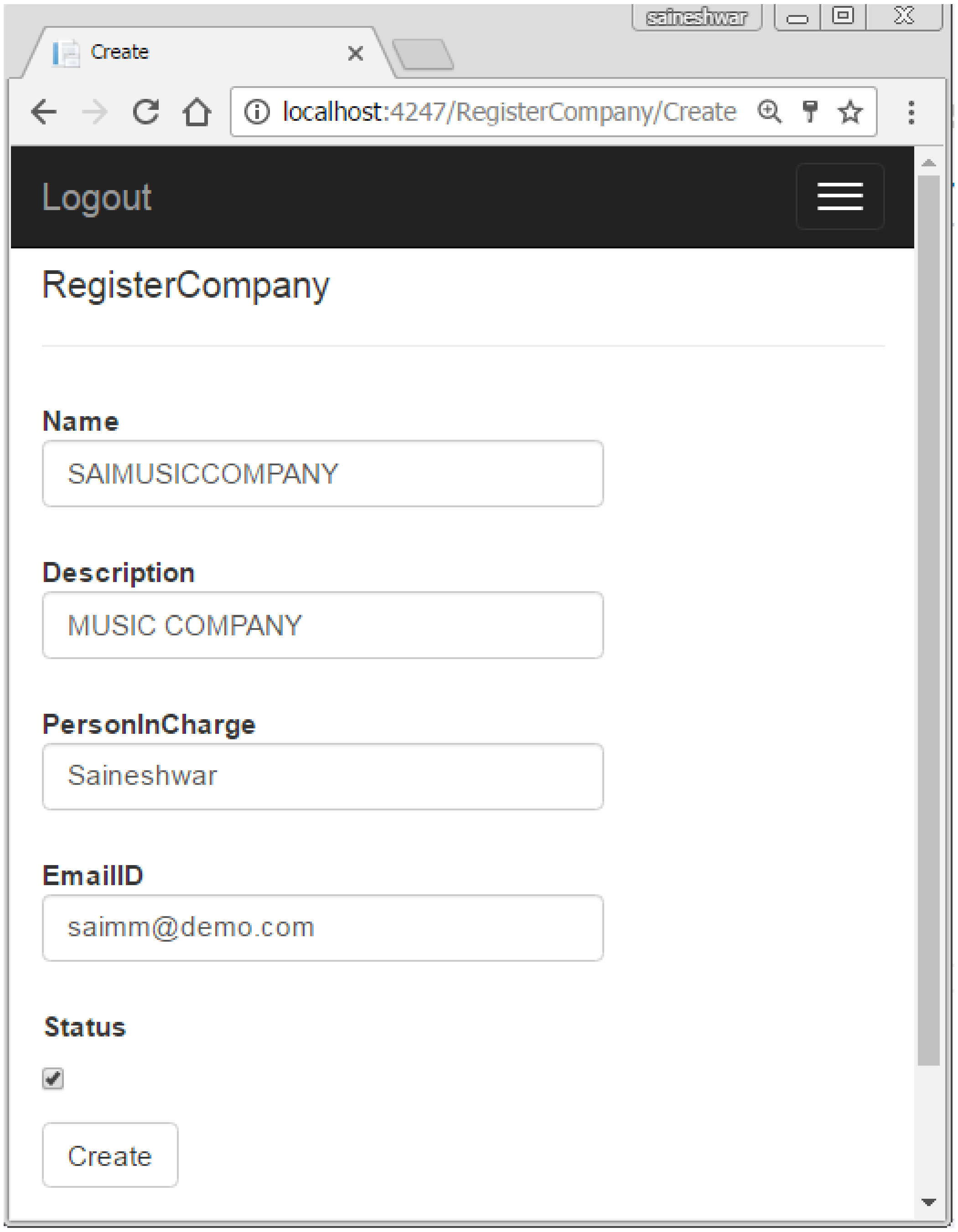
Data context class: DatabaseContext (MusicAPIStore.Context)

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

After adding View just save the application and run, then the first step is to login into the application as you log in you will see **RegisterCompany** View just register your Company.



Logout

RegisterCompany

Name: SAIMUSICCOMPANY

Description: MUSIC COMPANY

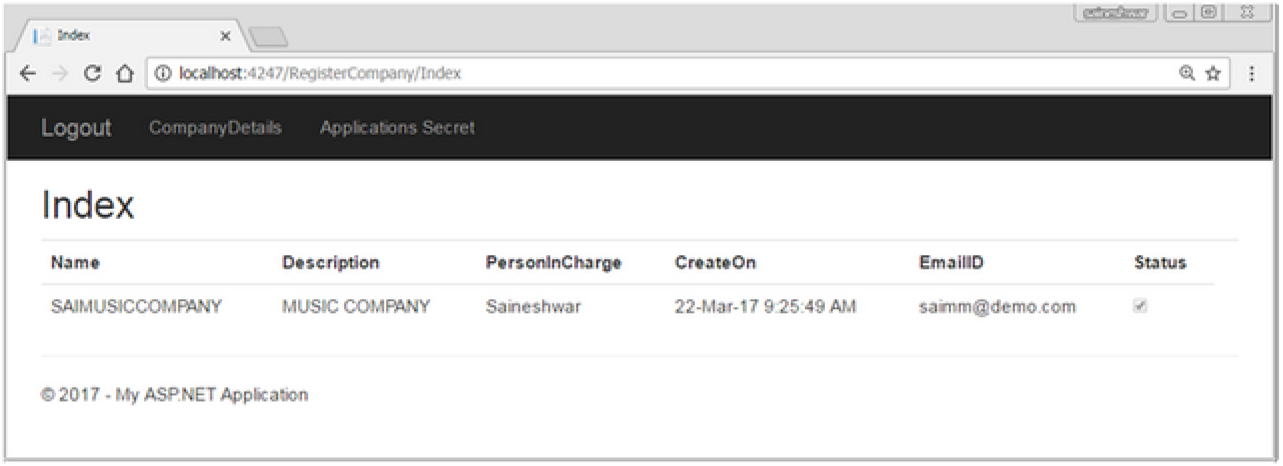
PersonInCharge: Saineshwar

EmailID: saimm@demo.com

Status: ✓

Create

After creating a company you will able to see Index View as shown in below snapshot with company details which you have filled.



Logout CompanyDetails Applications Secret

Index

Name	Description	PersonInCharge	CreateOn	EmailID	Status
SAIMUSICCOMPANY	MUSIC COMPANY	Saineshwar	22-Mar-17 9:25:49 AM	saimm@demo.com	✓

© 2017 - My ASP.NET Application

Now we have Registered Company next step is to Get Application ClientID and Client Secret.

Generate ClientID and Client Secret Keys

clipartfest.com

In this part, we are going to Generate Unique ClientID and Client Secret keys for each company which is registered.

We generate this keys using `RNGCryptoServiceProvider` algorithm.

Code snippet of Key Generator Class

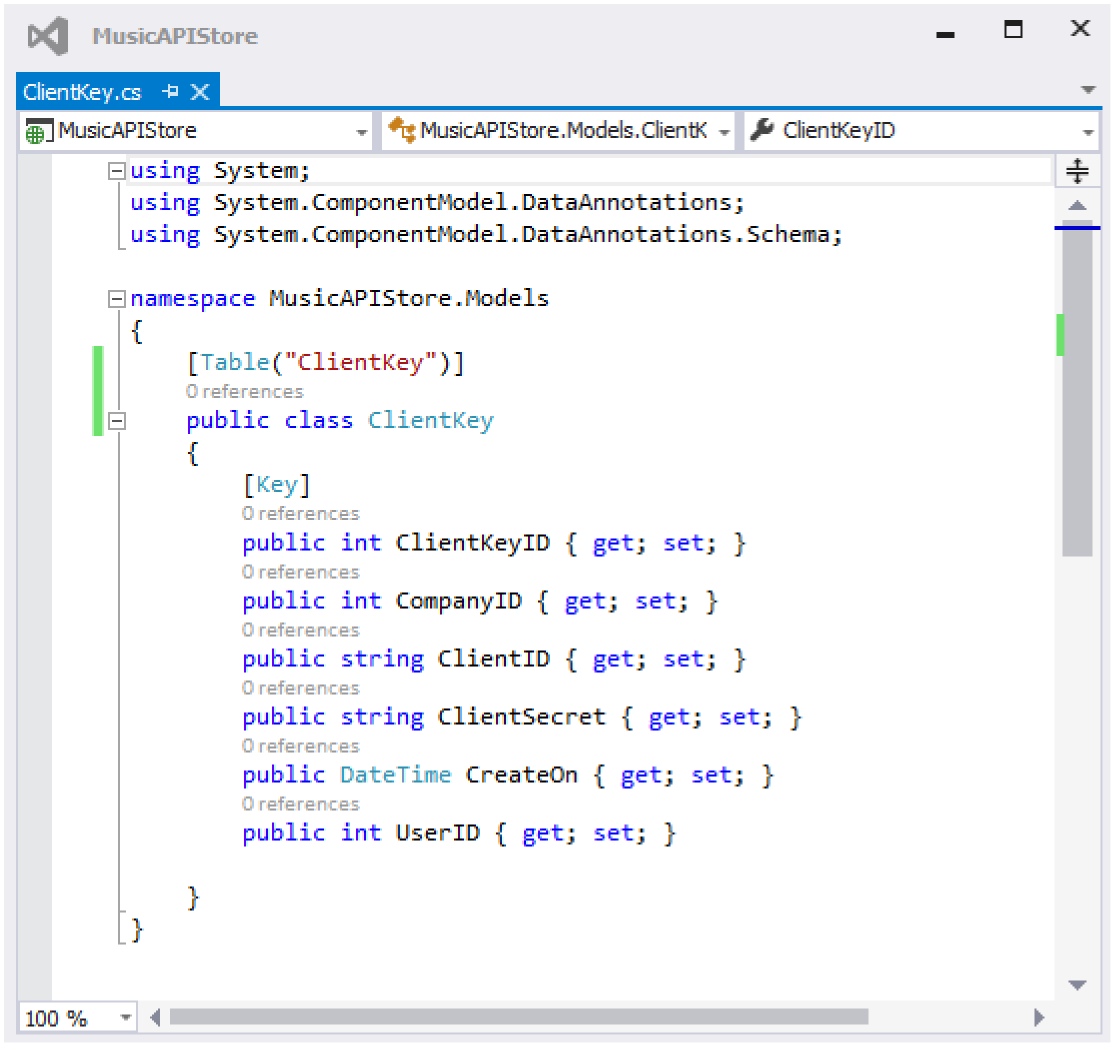
Hide Copy Code

```
public static class KeyGenerator
{
    public static string GetUniqueKey(int maxSize = 15)
    {
        char[] chars = new char[62];
        chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();
        byte[] data = new byte[1];
        using (RNGCryptoServiceProvider crypto = new RNGCryptoServiceProvider())
        {
            crypto.GetNonZeroBytes(data);
            Data = new byte[maxSize];
            crypto.GetNonZeroBytes(data);
        }
        StringBuilder result = new StringBuilder(maxSize);
        foreach (byte b in data)
        {
            result.Append(chars[b % (chars.Length)]);
        }
        return result.ToString();
    }
}
```

After generating ClientID and Client Secret keys we are going to insert these Keys with UserID in ClientKey table and display to User for using it.

And we also have provided **ReGenerate** Keys button on GenerateKeys View such that if the user wants he can generate a new pair of Keys he can do by click on this button again these values are updated to the database according to UserID.

Let’s begin with having a look at Model ClientKey first.



```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MusicAPIStore.Models
{
    [Table("ClientKey")]
    0 references
    public class ClientKey
    {
        [Key]
        0 references
        public int ClientKeyID { get; set; }
        0 references
        public int CompanyID { get; set; }
        0 references
        public string ClientID { get; set; }
        0 references
        public string ClientSecret { get; set; }
        0 references
        public DateTime CreateOn { get; set; }
        0 references
        public int UserID { get; set; }

    }
}
```

After adding ClientKey Model in Model folder next we are going to move forward by adding Interface with name IClientKeys and in this interface, we are going to declare 5 methods. The methods names inside interface are self-explanatory.

Adding Interface IClientKeys

Hide Copy Code

```
using MusicAPIStore.Models;

namespace MusicAPIStore.Repository
{
    public interface IClientKeys
    {
        bool IsUniqueKeyAlreadyGenerate(int UserID);
        void GenerateUniqueKey(out string ClientID, out string ClientSecert);
        int SaveClientIDandClientSecert(ClientKey ClientKeys);
        int UpdateClientIDandClientSecert(ClientKey ClientKeys);
        ClientKey GetGenerateUniqueKeyByUserID(int UserID);
    }
}
```

After adding Interface and declaring methods inside it next we are going to add Concrete class which is going to inherit this IClientKeys interface.

Adding Class ClientKeysConcrete

In this part, we are going to add `Concrete` class with name `ClientKeysConcrete` which is going to inherit IClientKeys interface and implement all methods in it.

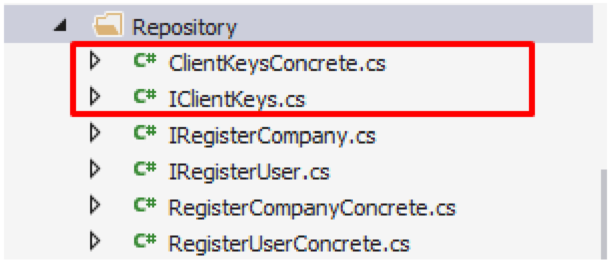
Hide Shrink ▲ Copy Code

```
using System.Linq;
using MusicAPIStore.Models;
using MusicAPIStore.Context;
using MusicAPIStore.AES256Encryption;
using System.Data.Entity;

namespace MusicAPIStore.Repository
{
    public class ClientKeysConcrete : IClientKeys
    {
        DatabaseContext _context;
        public ClientKeysConcrete()
        {
            _context = new DatabaseContext();
        }

        public void GenerateUniqueKey(out string ClientID, out string ClientSecert)
        {
            ClientID = EncryptionLibrary.KeyGenerator.GetUniqueKey();
            ClientSecert = EncryptionLibrary.KeyGenerator.GetUniqueKey();
        }

        public bool IsUniqueKeyAlreadyGenerate(int UserID)
        {
            bool keyExists = _context.ClientKeys.Any(clientkeys => clientkeys.UserID.Equals(UserID));

            if (keyExists)
            {
                return true;
            }
            else
            {
                return false;
            }

        }

        public int SaveClientIDandClientSecert(ClientKey ClientKeys)
        {
            _context.ClientKeys.Add(ClientKeys);
            return _context.SaveChanges();
        }

        public ClientKey GetGenerateUniqueKeyByUserID(int UserID)
        {
            var clientkey = (from ckey in _context.ClientKeys
                             where ckey.UserID  == UserID
                             select ckey).FirstOrDefault();
            return clientkey;
        }


        public int UpdateClientIDandClientSecert(ClientKey ClientKeys)
        {
            _context.Entry(ClientKeys).State = EntityState.Modified;
            _context.SaveChanges();
            return _context.SaveChanges();
        }

    }
}
```

If you had view on `ClientKeysConcrete` class there is "`GenerateUniqueKey`" method which generates unique ClientID and Client Secret keys and returns, next we are going to have look on method "`IsUniqueKeyAlreadyGenerate`" this method checks whether ClientID and Client Secret keys is already generated for this user or not, next we are going to have look on method "`SaveClientIDandClientSecert`" as this method name says it is going to save ClientID and Client Secret keys details in database.

If a user logs out from the portal and if he visits again to the portal to see his ClientID and Client Secret keys for getting that keys from the database we have created "`GetGenerateUniqueKeyByUserID`" method.

The last method which we are going see is "`UpdateClientIDandClientSecert`" this method is used to update ClientID and Client Secret keys when User clicks on ReGenerate Keys button on View which we are going to add in meanwhile.



Adding ApplicationKeys Controller

In this part, we are going to add ApplicationKeys Controller with2 `GenerateKeys` Action Method in it.

One Action method handles `[HttpGet]` part and another is going to handle `[HttpPost]` part.

In `[HttpGet] GenerateKeys` Action Method we are going the first check is Keys Already Generate or not if not then we are going to Generate new Keys and save that keys in Database and Display to Users.

The `[HttpPost] GenerateKeys` Action Method is called when User click on "**ReGenerate Keys"** button and this we are going to Generate new Keys and save that keys in Database and Display to Users.

For adding Controller follow the same step which we have used for adding RegisterUser Controller after adding controller we have just manually added a constructor and 2 Action Method in it as shown below.

Code snippet of ApplicationKeys Controller

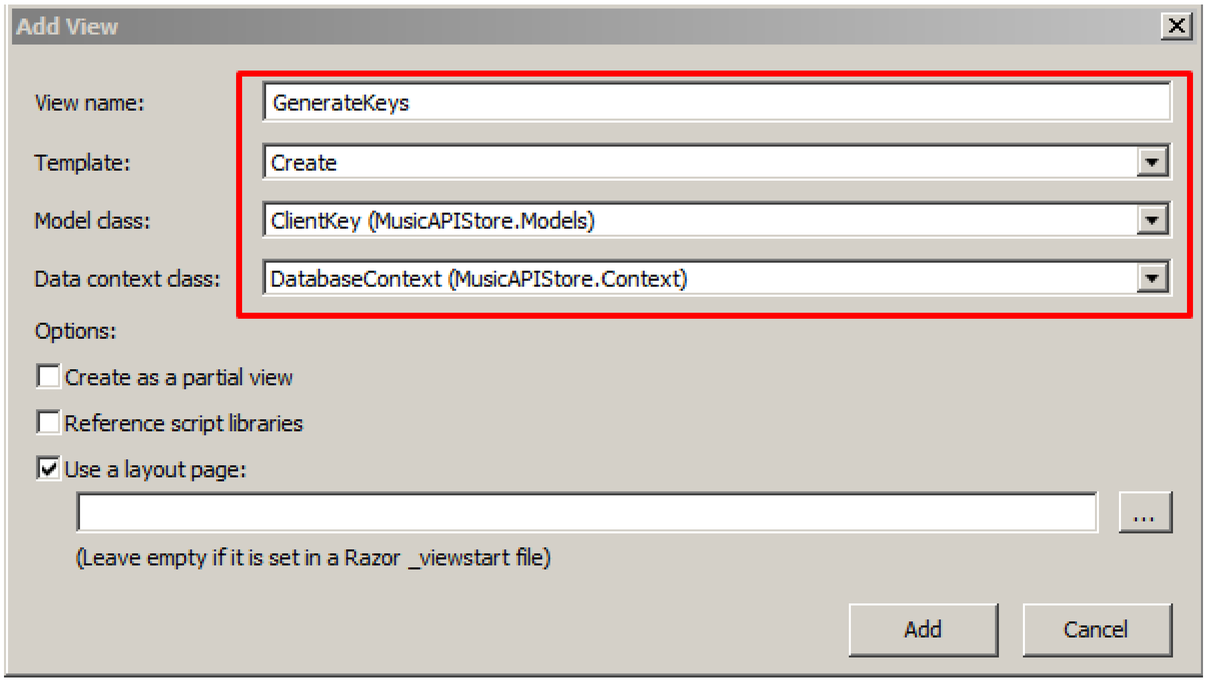
Hide Shrink ▲ Copy Code

```
using MusicAPIStore.Filters;
using MusicAPIStore.Models;
using MusicAPIStore.Repository;
using System;
using System.Web.Mvc;

namespace MusicAPIStore.Controllers
{
    [ValidateSessionAttribute]
    public class ApplicationKeysController : Controller
    {
        IClientKeys _IClientKeys;
        IRegisterCompany _IRegisterCompany;
        public ApplicationKeysController()
        {
            _IClientKeys = new ClientKeysConcrete();
            _IRegisterCompany = new RegisterCompanyConcrete();
        }

        // GET: ApplicationKeys/GenerateKeys
        [HttpGet]
        public ActionResult GenerateKeys()
        {
            try
            {
                ClientKey clientkeys = new ClientKey();

                // Validating ClientID and ClientSecert already Exists
                var keyExists =
_IClientKeys.IsUniqueKeyAlreadyGenerate(Convert.ToInt32(Session["UserID"]));

                if (keyExists)
                {
                    // Getting Generate ClientID and ClientSecert Key By UserID
                    clientkeys =
_IClientKeys.GetGenerateUniqueKeyByUserID(Convert.ToInt32(Session["UserID"]));
                }
                else
                {
                    string clientID=string.Empty;
                    string clientSecert = string.Empty;
                    int companyId = 0;

                    var company =
_IRegisterCompany.FindCompanyByUserId(Convert.ToInt32(Session["UserID"]));
                    companyId = company.CompanyID;

                    //Generate Keys
                    _IClientKeys.GenerateUniqueKey(out clientID, out clientSecert);

                    //Saving Keys Details in Database
                    clientkeys.ClientKeyID = 0;
                    clientkeys.CompanyID = companyId;
                    clientkeys.CreateOn = DateTime.Now;
                    clientkeys.ClientID = clientID;
                    clientkeys.ClientSecret = clientSecert;
                    clientkeys.UserID = Convert.ToInt32(Session["UserID"]);
                    _IClientKeys.SaveClientIDandClientSecert(clientkeys);

                }

                return View(clientkeys);
            }
            catch (Exception)
            {
                throw;
            }
        }


        // POST: ApplicationKeys/GenerateKeys
        [HttpPost]
        public ActionResult GenerateKeys(ClientKey clientkeys)
        {
            try
            {
                string clientID = string.Empty;
                string clientSecert = string.Empty;

                //Generate Keys
                _IClientKeys.GenerateUniqueKey(out clientID, out clientSecert);

                //Updating ClientID and ClientSecert
                var company = _IRegisterCompany.FindCompanyByUserId(Convert.ToInt32(Session["UserID"]));
                clientkeys.CompanyID = company.CompanyID;
                clientkeys.CreateOn = DateTime.Now;
                clientkeys.ClientID = clientID;
                clientkeys.ClientSecret = clientSecert;
                clientkeys.UserID = Convert.ToInt32(Session["UserID"]);
                _IClientKeys.UpdateClientIDandClientSecert(clientkeys);

                return RedirectToAction("GenerateKeys");
            }
            catch (Exception ex)
            {
                return View();
            }
        }
    }
}
```

After completing with adding Controller and its Action Method next we are going to add View to GenerateKeys Action Method.

Adding GenerateKeys View

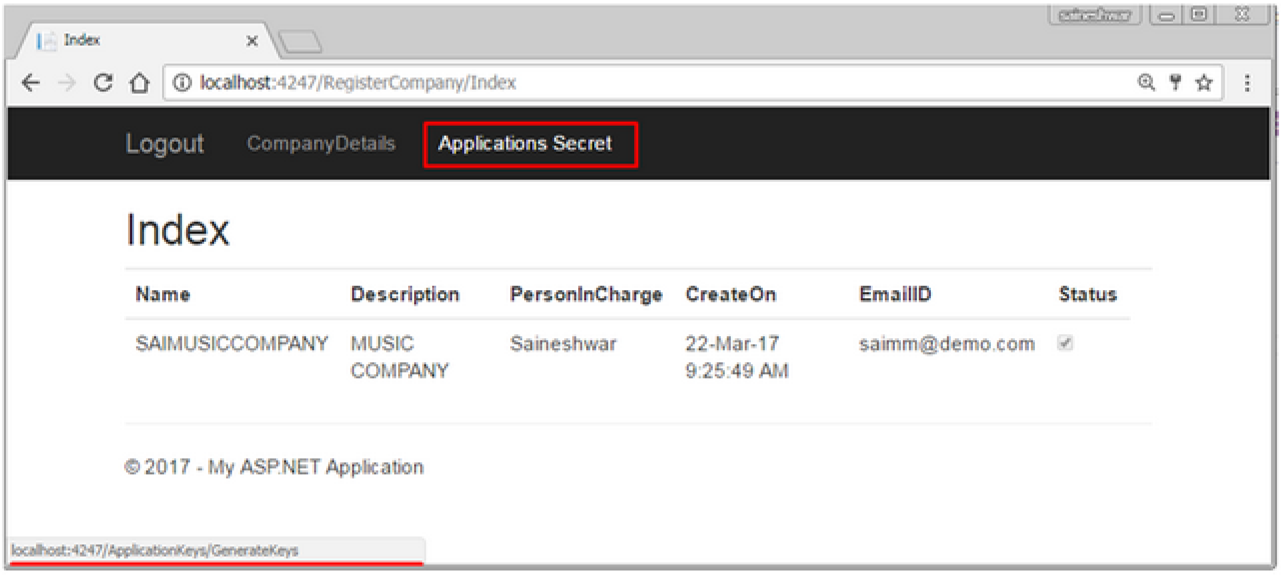
In this part we are going to add Views for adding View just right click inside Action Method then choose Add View from Menu list a new dialog will pop up with name "**Add View**" next we do not require to provide name to view it is set to default which is Action Method name in Template choose template to depend on upon which view you want to create (Create) and long with that choose Model (**ClientKey**) Click on Add Button.


Add View

View name: GenerateKeys
Template: Create
Model class: ClientKey (MusicAPIStore.Models)
Data context class: DatabaseContext (MusicAPIStore.Context)
Options:
☐ Create as a partial view
☐ Reference script libraries
☑ Use a layout page:
(Leave empty if it is set in a Razor _viewstart file)

After adding View just save the application and run, then the first step is to login into the application as you log in you will see **RegisterCompany** Index View.

On the Master page, we have added a link to show GenerateKeys View.



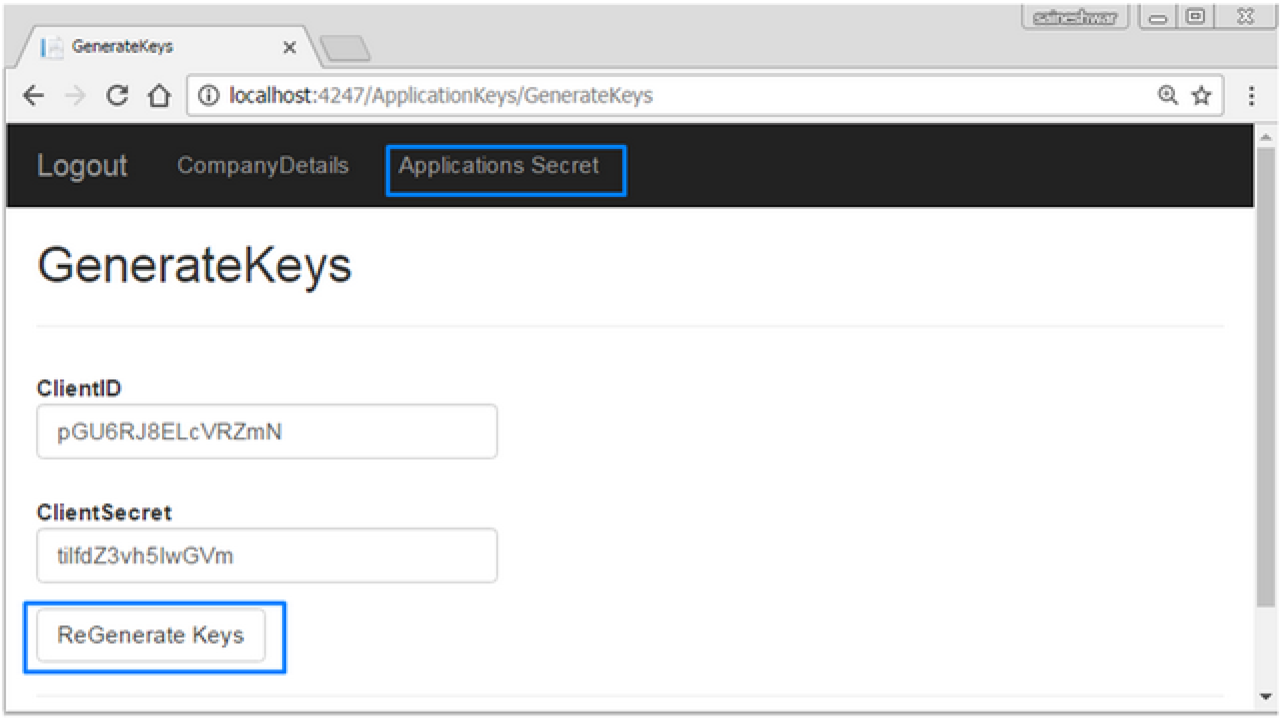
Index

Name	Description	PersonInCharge	CreateOn	EmailID	Status
SAIMUSICCOMPANY	MUSIC COMPANY	Saineshwar	22-Mar-17 9:25:49 AM	saimm@demo.com	✓

© 2017 - My ASP.NET Application

localhost:4247/ApplicationKeys/GenerateKeys

Now on clicking on Application secret link, it will show GenerateKeys View.



GenerateKeys

ClientID

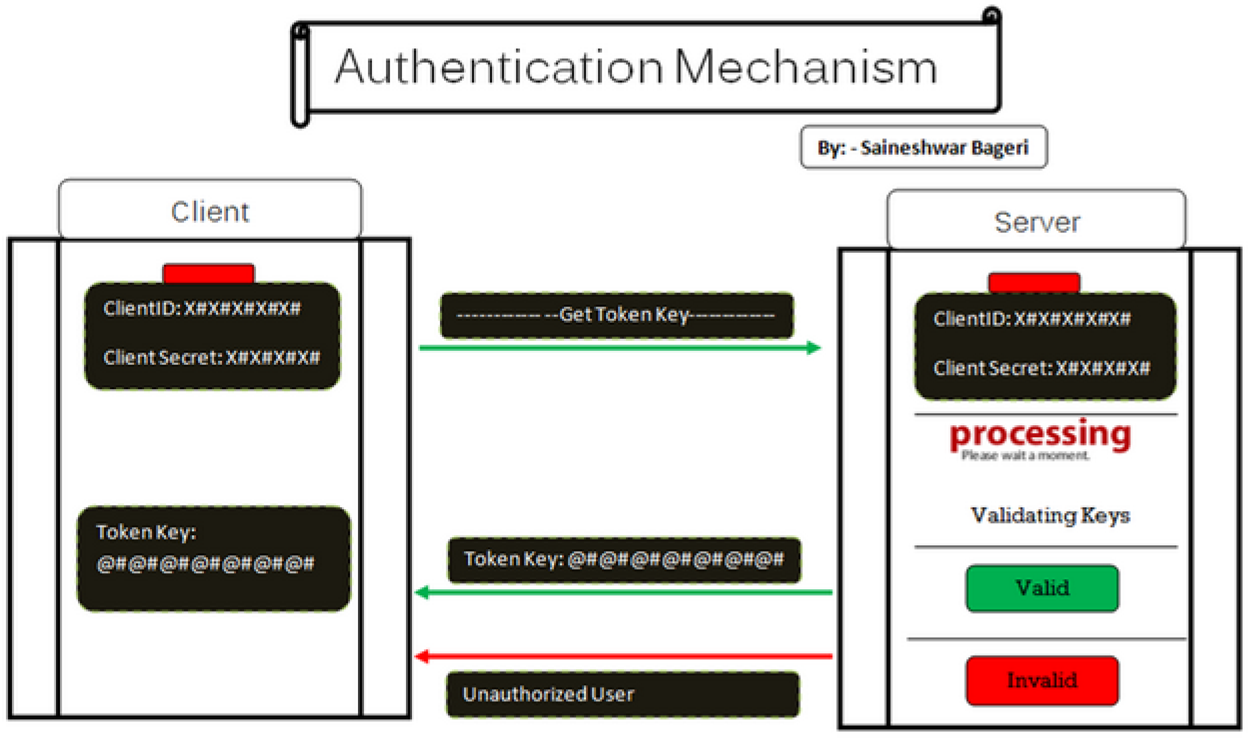
pGU6RJ8ELcVRZmN

ClientSecret

tilfdZ3vh5lwGVm

Now we have completed with generating ClientID and Client Secret next we are going to add Authenticate Controller for authenticating ClientID and Client Secret and return Token key in Response.

Authentication Mechanism



In this process we have provided ClientID and Client Secret to Client and now we need to develop authenticating mechanism where user will send this ClientID and Client Secret to Server, then we are going to validate this keys with database and after that we are going to return token to User in response if keys are valid then only else we are going to return Error Message.

Let's start with adding interface with name `IAuthenticate` and declaring methods in it.

Adding IAuthenticate Interface

Hide Copy Code

```
using MusicAPIStore.Models;
using System;

namespace MusicAPIStore.Repository
{
    public interface IAuthenticate
    {
        ClientKey GetClientKeysDetailsbyCLientIDandClientSecert(string clientID , string clientSecert);
        bool ValidateKeys(ClientKey ClientKeys);
        bool IsTokenAlreadyExists(int CompanyID);
        int DeleteGenerateToken(int CompanyID);
        int InsertToken(TokensManager token);
        string GenerateToken(ClientKey ClientKeys, DateTime IssuedOn);
    }
}
```

We have declared 5 methods in the IAuthenticate interface.

1. `GetClientKeysDetailsbyCLientIDandClientSecert`

   This method takes ClientID and Client Secret as input and gets data from the database on basis of it.

2. `ValidateKeys`

   This method takes ClientKey Model as input in which it checks ClientID and Client Secret passed by users is valid or not.

3. `IsTokenAlreadyExists`

   This method takes CompanyID as input and check are Token already generate for it.

4. `DeleteGenerateToken`

   This method takes CompanyID as input and Deletes token which is already generated on basis of the CompanyID parameter.

5. `InsertToken`

   This method takes **TokensManager** model as input parameter for saving Token values in

   Database.

6. `GenerateToken`

   This method Generate and return Token.

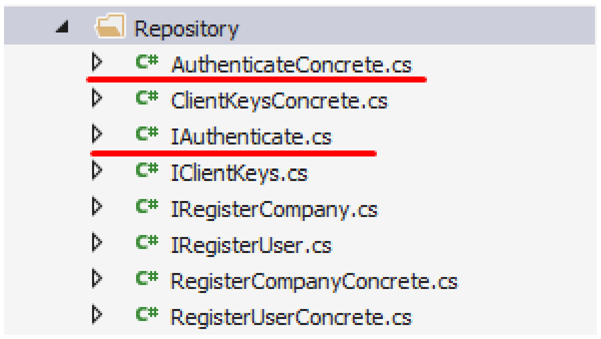
Adding Authenticate Concrete Class

In this part, we are going to add `Concrete` class with name `AuthenticateConcrete` which is going to inherit `IAuthenticate` interface and implement all methods in it.

Hide Shrink ▲ Copy Code

```
using System;
using System.Linq;
using MusicAPIStore.Models;
using MusicAPIStore.Context;
using MusicAPIStore.AES256Encryption;
using static MusicAPIStore.AES256Encryption.EncryptionLibrary;

namespace MusicAPIStore.Repository
{
    public class AuthenticateConcrete : IAuthenticate
    {
        DatabaseContext _context;

        public AuthenticateConcrete()
        {
            _context = new DatabaseContext();
        }

        public ClientKey GetClientKeysDetailsbyCLientIDandClientSecert(string clientID, string
clientSecert)
        {
            try
            {
                var result = (from clientkeys in _context.ClientKeys
                              where clientkeys.ClientID == clientID && clientkeys.ClientSecret ==
clientSecert
                              select clientkeys).FirstOrDefault();
                return result;
            }
            catch (Exception)
            {

                throw;
            }
        }

        public bool ValidateKeys(ClientKey ClientKeys)
        {
            try
            {
                var result = (from clientkeys in _context.ClientKeys
                              where clientkeys.ClientID == ClientKeys.ClientID && clientkeys.ClientSecret
== ClientKeys.ClientSecret
                              select clientkeys).Count();
                if (result > 0)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
            catch (Exception)
            {

                throw;
            }
        }

        public bool IsTokenAlreadyExists(int CompanyID)
        {
            try
            {
                var result = (from token in _context.TokensManager
                              where token.CompanyID == CompanyID
                              select token).Count();
                if (result > 0)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
            catch (Exception)
            {

                throw;
            }
        }

        public int DeleteGenerateToken(int CompanyID)
        {
            try
            {
                var token = _context.TokensManager.SingleOrDefault(x => x.CompanyID == CompanyID);
                _context.TokensManager.Remove(token);
                return _context.SaveChanges();
            }
            catch (Exception)
            {
                throw;
            }
        }

        public string GenerateToken(ClientKey ClientKeys, DateTime IssuedOn)
        {
            try
            {
                string randomnumber =
                   string.Join(":", new string[]
                   {   Convert.ToString(ClientKeys.UserID),
                KeyGenerator.GetUniqueKey(),
                Convert.ToString(ClientKeys.CompanyID),
                Convert.ToString(IssuedOn.Ticks),
                ClientKeys.ClientID
                   });

                return EncryptionLibrary.EncryptText(randomnumber);
            }
            catch (Exception)
            {

                throw;
            }
        }

        public int InsertToken(TokensManager token)
        {
            try
            {
                _context.TokensManager.Add(token);
                return _context.SaveChanges();
            }
            catch (Exception)
            {

                throw;
            }
        }

    }
}
```

Snapshot after Adding Interface IAuthenticate and AuthenticateConcrete



After completing with adding Interface IAuthenticate and AuthenticateConcrete next we are going to add ApiController with name Authenticate.

Adding Authenticate Controller

In this part, we are going to add ApiController with name Authenticate and it will have a single Action Method in it with name `Authenticate` which takes ClientKey Model as input from [FromBody].

Hide Shrink Copy Code

```
using MusicAPIStore.Models;
using MusicAPIStore.Repository;
using System;
using System.Configuration;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace MusicAPIStore.Controllers
{
    public class AuthenticateController : ApiController
    {
        IAuthenticate _IAuthenticate;
        public AuthenticateController()
        {
            _IAuthenticate = new AuthenticateConcrete();
        }

        // POST: api/Authenticate
        public HttpResponseMessage Authenticate([FromBody]ClientKey ClientKeys)
        {
            if (string.IsNullOrEmpty(ClientKeys.ClientID) && string.IsNullOrEmpty(ClientKeys.ClientSecret))
            {
                var message = new HttpResponseMessage(HttpStatusCode.NotAcceptable);
                message.Content = new StringContent("Not Valid Request");
                return message;
            }
            else
            {
                if (_IAuthenticate.ValidateKeys(ClientKeys))
                {
                    var clientkeys =
_IAuthenticate.GetClientKeysDetailsbyCLientIDandClientSecert(ClientKeys.ClientID, ClientKeys.ClientSecret);

                    if (clientkeys == null)
                    {
                        var message = new HttpResponseMessage(HttpStatusCode.NotFound);
                        message.Content = new StringContent("InValid Keys");
                        return message;
                    }
                    else
                    {
                        if (_IAuthenticate.IsTokenAlreadyExists(clientkeys.CompanyID))
                        {
                            _IAuthenticate.DeleteGenerateToken(clientkeys.CompanyID);

                            return GenerateandSaveToken(clientkeys);
                        }
                        else
                        {
                            return GenerateandSaveToken(clientkeys);
                        }
                    }
                }
                else
                {
                    var message = new HttpResponseMessage(HttpStatusCode.NotFound);
                    message.Content = new StringContent("InValid Keys");
                    return new HttpResponseMessage { StatusCode = HttpStatusCode.NotAcceptable };
                }
            }
        }


        [NonAction]
        private HttpResponseMessage GenerateandSaveToken(ClientKey clientkeys)
        {
            var IssuedOn = DateTime.Now;
            var newToken = _IAuthenticate.GenerateToken(clientkeys, IssuedOn);
            TokensManager token = new TokensManager();
            token.TokenID = 0;
            token.TokenKey = newToken;
            token.CompanyID = clientkeys.CompanyID;
            token.IssuedOn = IssuedOn;
            token.ExpiresOn =
DateTime.Now.AddMinutes(Convert.ToInt32(ConfigurationManager.AppSettings["TokenExpiry"]));
            token.CreatedOn = DateTime.Now;
            var result = _IAuthenticate.InsertToken(token);

            if (result == 1)
            {
                HttpResponseMessage response = new HttpResponseMessage();
                response = Request.CreateResponse(HttpStatusCode.OK, "Authorized");
                response.Headers.Add("Token", newToken);
                response.Headers.Add("TokenExpiry", ConfigurationManager.AppSettings["TokenExpiry"]);
                response.Headers.Add("Access-Control-Expose-Headers", "Token,TokenExpiry");
                return response;
            }
            else
            {
                var message = new HttpResponseMessage(HttpStatusCode.NotAcceptable);
                message.Content = new StringContent("Error in Creating Token");
                return message;
            }
        }
    }
}
```

Let’s understand code

Hide Shrink Copy Code

```
public HttpResponseMessage Authenticate([FromBody]ClientKey ClientKeys)
        {
            if (string.IsNullOrEmpty(ClientKeys.ClientID) && string.IsNullOrEmpty(ClientKeys.ClientSecret))
            {
                var message = new HttpResponseMessage(HttpStatusCode.NotAcceptable);
                message.Content = new StringContent("Not Valid Request");
                return message;
            }
            else
            {
                if (_IAuthenticate.ValidateKeys(ClientKeys))
                {
                    var clientkeys = _IAuthenticate.GetClientKeysDetailsbyCLientIDandClientSecert
                                                    (ClientKeys.ClientID, ClientKeys.ClientSecret);

                    if (clientkeys == null)
                    {
                        var message = new HttpResponseMessage(HttpStatusCode.NotFound);
                        message.Content = new StringContent("InValid Keys");
                        return message;
                    }
                    else
                    {
                        if (_IAuthenticate.IsTokenAlreadyExists(clientkeys.CompanyID))
                        {
                            _IAuthenticate.DeleteGenerateToken(clientkeys.CompanyID);

                            return GenerateandSaveToken(clientkeys);
                        }
                        else
                        {
                            return GenerateandSaveToken(clientkeys);
                        }
                    }
                }
                else
                {
                    var message = new HttpResponseMessage(HttpStatusCode.NotFound);
                    message.Content = new StringContent("InValid Keys");
                    return new HttpResponseMessage { StatusCode = HttpStatusCode.NotAcceptable };
                }
            }
        }
```

The first step in this part is Authenticate Method takes ClientKey Model as input and from this Model we are going to use only 2 parameters ClientID and Client Secret, next we are first going to check this is parameters null or not if it is Null this we are going to send **HttpResponseMessage as** "Not Valid Request" in response.

If ClientID and Client Secret are not null then we are going to send ClientID and Client Secret to ValidateKeys method to check is this Values passed are already exists in the database or not.

If ClientID and Client Secret value exists in database then we pass both values to "`GetClientKeysDetailsbyCLientIDandClientSecert`" Method to get all ClientKey Details, here again, we check Keys are Valid or not, if not then we are going to send **HttpResponseMessage as** "**InValid Keys**" in response.

If ClientID and Client Secret are Valid then we get ClientKey Details from the database and from this Model we pass CompanyID to "`IsTokenAlreadyExists`" Method to check is Token already exists in the database or not.

If Token already exists in database then we are going we are going to delete old token and Generate New token and Insert Newly Token in the database and we are also going send Token in response to Client who has sent the request.

If Token does not exist in database then we are going to generate token and Insert Newly Token in the database and we are also going send Token in response to Client who has sent the request.

Now we have understood how to process work let’s try real time example.

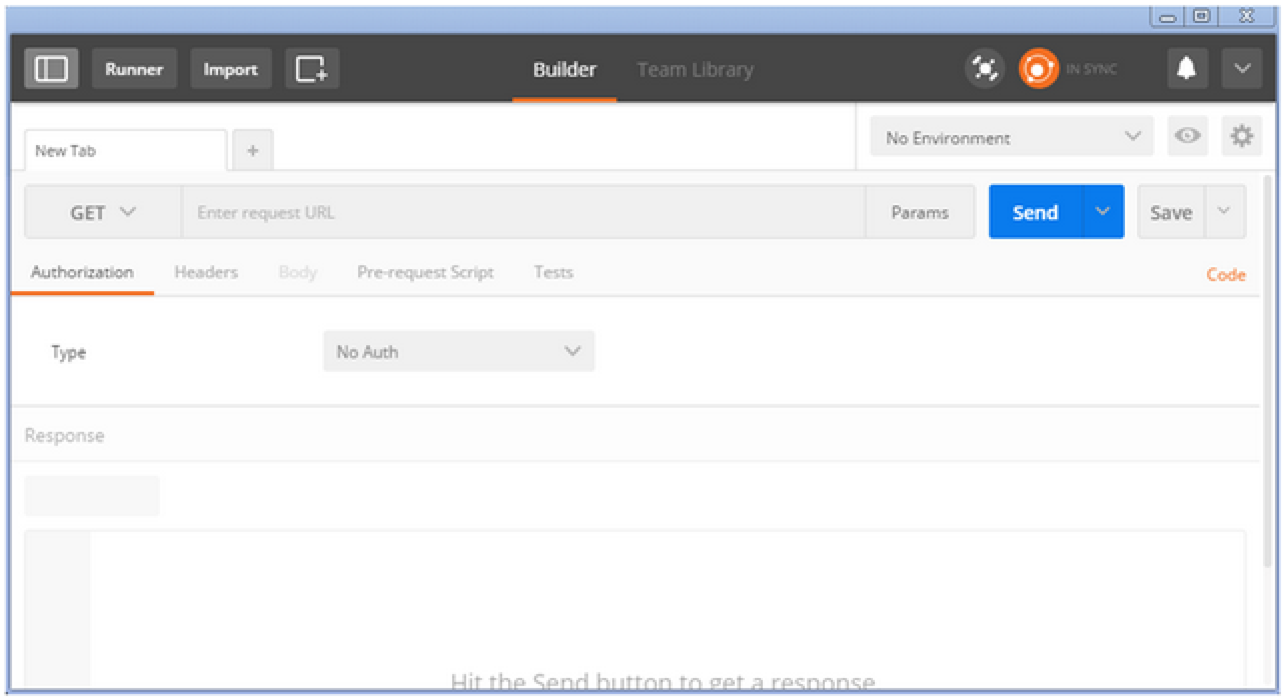
First to call Web API **api/Authenticate** method we are going to Use postman web debugger.

For downloading Postman Chrome App

Installing the Postman Chrome App

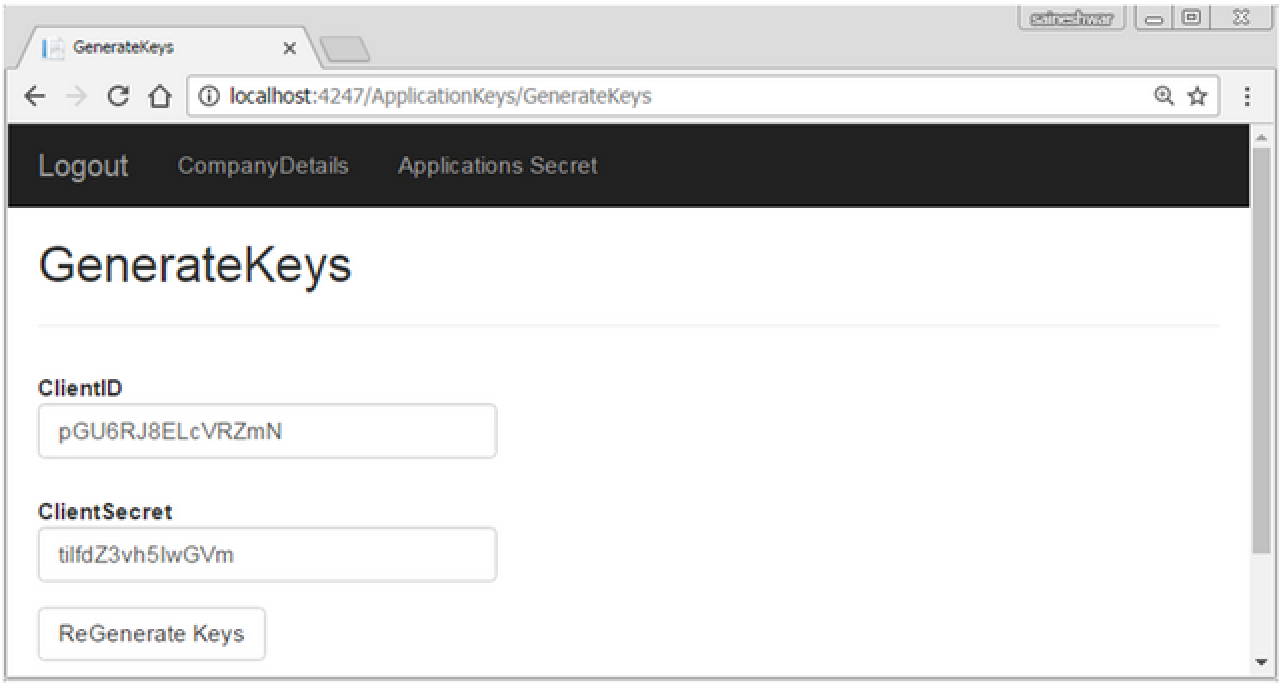
https://www.getpostman.com/docs/introduction

After installing Postman Chrome App now you can open Postman Chrome App below is a snapshot of Postman Chrome App.



Next step we are going have a Look at ClientID and ClientSecret because we need to send this Keys to Get Token from Server.

Getting keys



GenerateKeys

ClientID

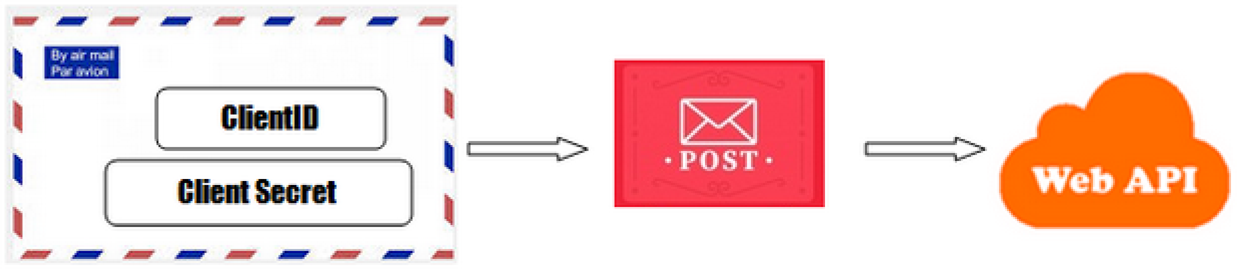
pGU6RJ8ELcVRZmN

ClientSecret

tilfdZ3vh5IwGVm

Next, we are going to set keys (ClientID and ClientSecret) in a **FromBody** request to Post.

Setting Values for Post Request in POSTMAN

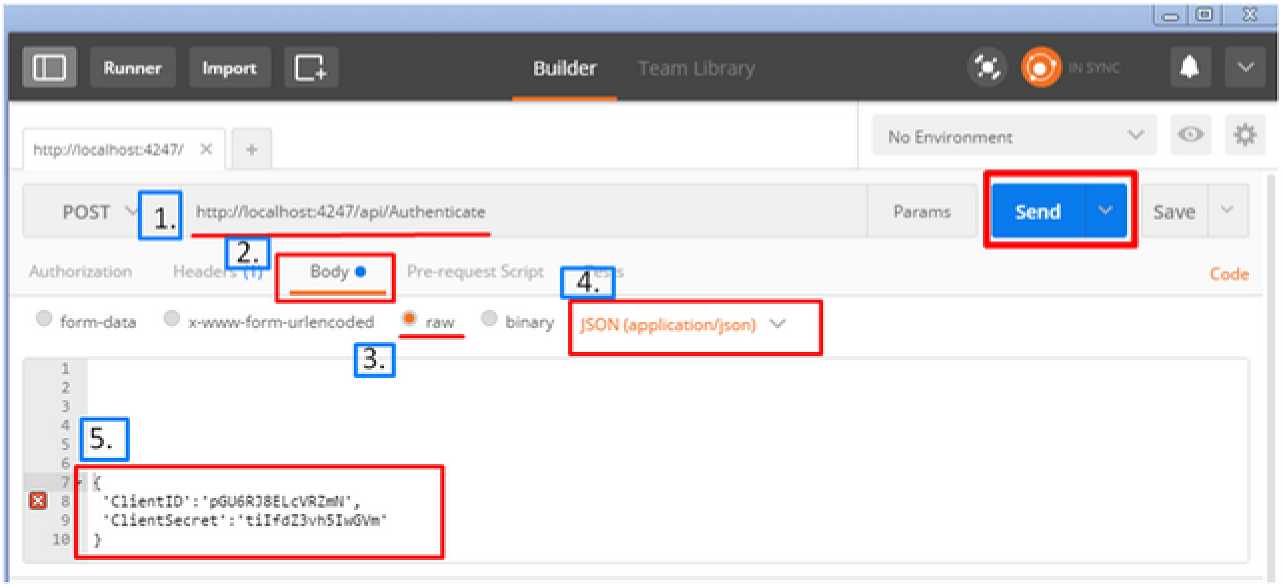


1. Choose Post Request From Dropdownlist
2. Set URL:- http://localhost:4247/api/Authenticate
3. We are going to send this Keys FromBody of Request.
4. It will be raw data with Content Type as **application/json**
5. **Set** 'ClientID' and 'ClientSecret'

Hide Copy Code

```
{
 'ClientID':'pGU6RJ8ELcVRZmN',
 'ClientSecret':'tiIfdZ3vh5IwGVm'
}
```

6. Click on Send Button to Send Request.



```
{
 'ClientID':'pGU6RJ8ELcVRZmN',
 'ClientSecret':'tiIfdZ3vh5IwGVm'
}
```

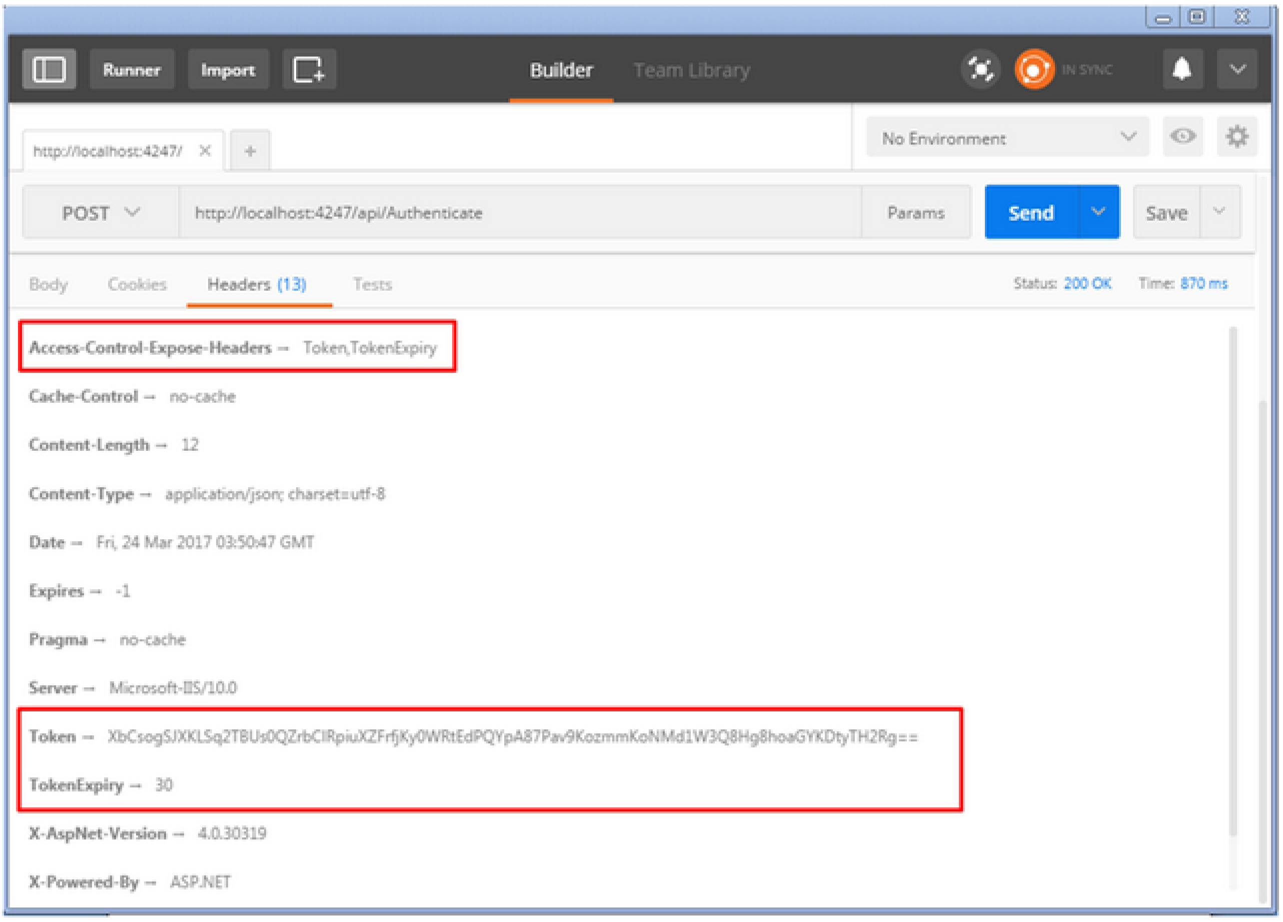
Response of Request

In Response we get Token and TokenExpiry.

Token: - XbCsogSJXKLSq2TBUs0QZrbClRpiuXZFrfjKy0WRtEdPQYpA87Pav9KozmmKoNMd1W3Q8Hg8hoaGYKDtyTH2Rg==

TokenExpiry: - 30

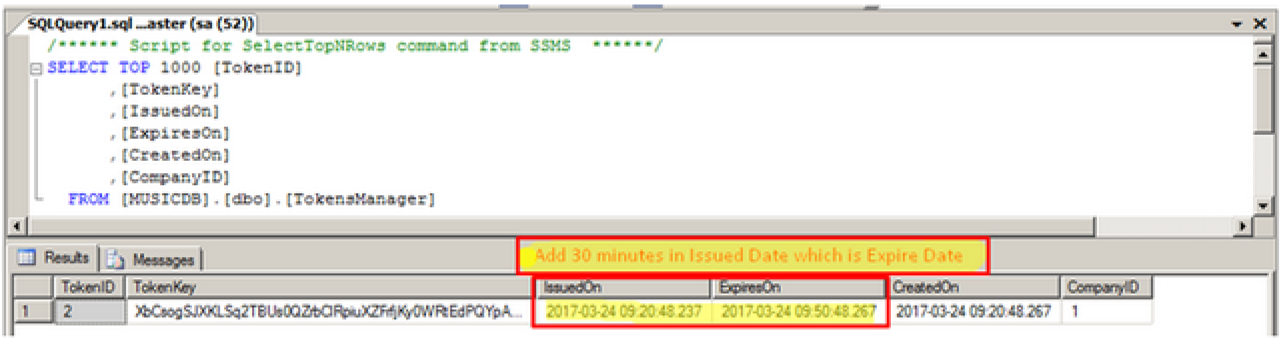
Token is only valid for 30 Minutes.



http://localhost:4247/api/Authenticate

```
Access-Control-Expose-Headers → Token,TokenExpiry
Cache-Control → no-cache
Content-Length → 12
Content-Type → application/json; charset=utf-8
Date → Fri, 24 Mar 2017 03:50:47 GMT
Expires → -1
Pragma → no-cache
Server → Microsoft-IIS/10.0
Token → XbCsogSJXKLSq2TBUs0QZrbCIRpiuXZFrfjKy0WRtEdPQYpA87Pav9KozmmKoNMd1W3Q8Hg8hoaGYKDtyTH2Rg==
TokenExpiry → 30
X-AspNet-Version → 4.0.30319
X-Powered-By → ASP.NET
```

After getting a response let's have a view of TokenManager table what are fields got inserted in it.


SQLQuery1.sql ...aster (sa (52))
Results
Messages
Add 30 minutes in Issued Date which is Expire Date

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [TokenID]
      ,[TokenKey]
      ,[IssuedOn]
      ,[ExpiresOn]
      ,[CreatedOn]
      ,[CompanyID]
  FROM [MUSICDB].[dbo].[TokensManager]
```

	TokenID	TokenKey	IssuedOn	ExpiresOn	CreatedOn	CompanyID
1	2	XbCsogSJXKLSq2TBUs0QZrbCIRpiuXZFrfjKy0WRtEdPQYpA...	2017-03-24 09:20:48.237	2017-03-24 09:50:48.267	2017-03-24 09:20:48.267	1

Next step we are going to add AuthorizeAttribute with name APIAuthorizeAttribute.

Adding AuthorizeAttribute (APIAuthorizeAttribute)

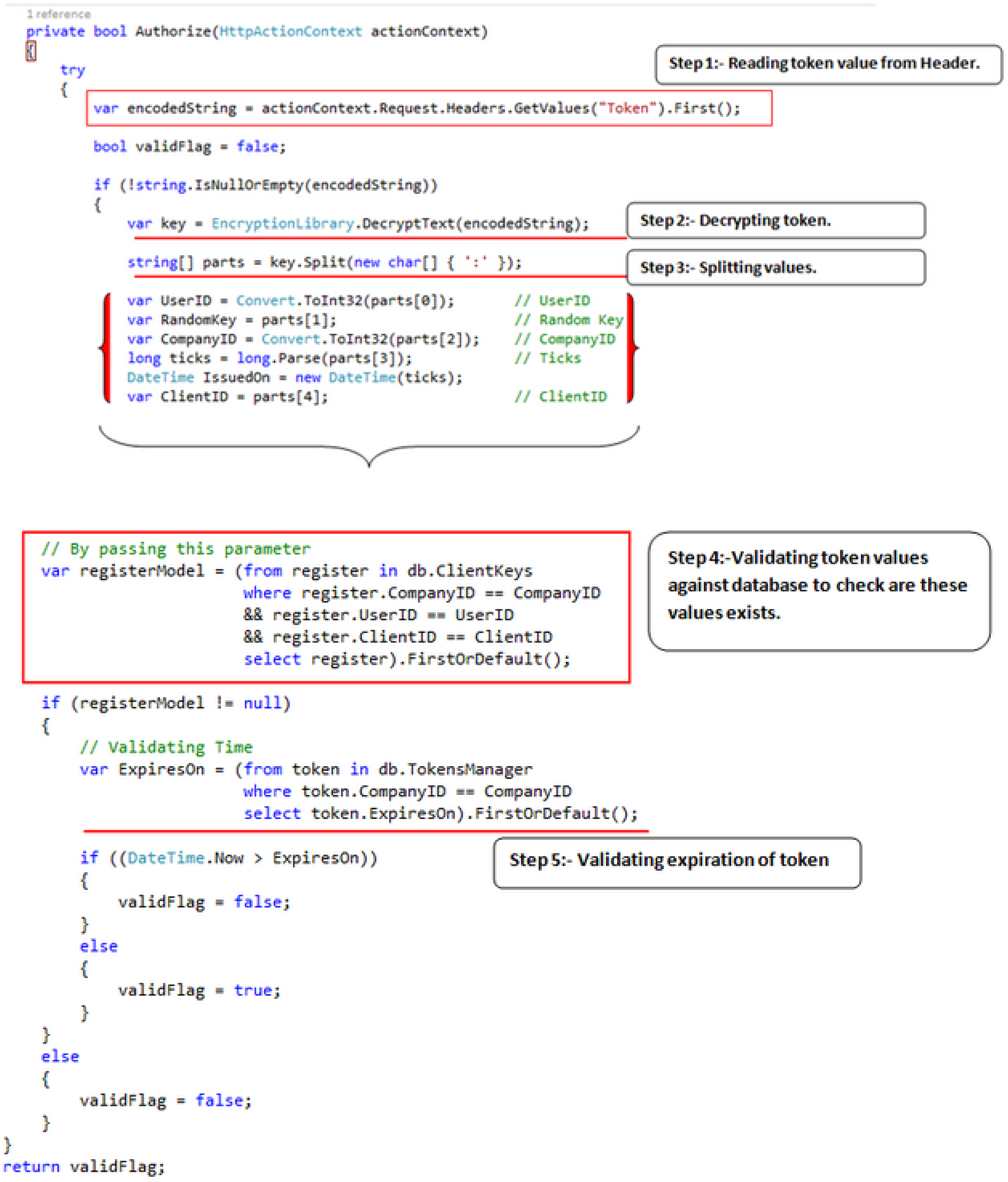
In this part, we are going to create an AuthorizeAttribute with name **"APIAuthorizeAttribute"** in Filter folder.

For adding this AuthorizeAttribute first we are going to a class with name "**APIAuthorizeAttribute**" and this class is going to inherit a class "**AuthorizeAttribute**" and implement all method inside it.

In this "**APIAuthorizeAttribute**" we are going to validate Token which is sent from the client.

1. The first step we are going to receive Token from Client Header.
2. After that we are going to check is this token Null or not
3. Next, we are going to decrypt this Token
4. After decrypting this Token we get string values as output
5. Next, we are going to split (':') string values which we have received.
6. After splitting we get (UserID ,Random Key,CompanyID,Ticks, ClientID) values
7. Next we are going to pass (UserID, CompanyID, ClientID) to database to check are this parameter valid which we have received.
8. After that, we are going to check Token Expiration.
9. If something is throwing error in this step we are going to return a false value.
10. And if we have valid values and token is not expired then it will return true.

Authorize Attribute Snapshot with explanation


Step 1:- Reading token value from Header.
Step 2:- Decrypting token.
Step 3:- Splitting values.
Step 4:-Validating token values against database to check are these values exists.
Step 5:- Validating expiration of token

```
1 reference
private bool Authorize(HttpActionContext actionContext)
{
    try
    {
        var encodedString = actionContext.Request.Headers.GetValues("Token").First();

        bool validFlag = false;

        if (!string.IsNullOrEmpty(encodedString))
        {
            var key = EncryptionLibrary.DecryptText(encodedString);

            string[] parts = key.Split(new char[] { ':' });

            var UserID = Convert.ToInt32(parts[0]);        // UserID
            var RandomKey = parts[1];                      // Random Key
            var CompanyID = Convert.ToInt32(parts[2]);     // CompanyID
            long ticks = long.Parse(parts[3]);             // Ticks
            DateTime IssuedOn = new DateTime(ticks);
            var ClientID = parts[4];                       // ClientID

    // By passing this parameter
    var registerModel = (from register in db.ClientKeys
                         where register.CompanyID == CompanyID
                         && register.UserID == UserID
                         && register.ClientID == ClientID
                         select register).FirstOrDefault();

    if (registerModel != null)
    {
        // Validating Time
        var ExpiresOn = (from token in db.TokensManager
                         where token.CompanyID == CompanyID
                         select token.ExpiresOn).FirstOrDefault();

        if ((DateTime.Now > ExpiresOn))
        {
            validFlag = false;
        }
        else
        {
            validFlag = true;
        }
    }
    else
    {
        validFlag = false;
    }
}
return validFlag;
```

After having a view on Snapshot you will get detail idea how it works next we are going to see complete code snippet of `APIAuthorizeAttribute`.

APIAuthorizeAttribute Code Snippet

Hide Shrink ▲ Copy Code

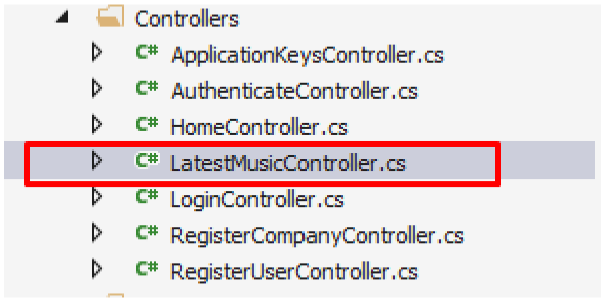
```
using MusicAPIStore.AES256Encryption;
using MusicAPIStore.Context;
using System;
using System.Linq;
using System.Web.Http;
using System.Web.Http.Controllers;

namespace MusicAPIStore.Filters
{
    public class APIAuthorizeAttribute : AuthorizeAttribute
    {
        private DatabaseContext db = new DatabaseContext();
        public override void OnAuthorization(HttpActionContext filterContext)
        {
            if (Authorize(filterContext))
            {
                return;
            }
            HandleUnauthorizedRequest(filterContext);
        }
        protected override void HandleUnauthorizedRequest(HttpActionContext filterContext)
        {
            base.HandleUnauthorizedRequest(filterContext);
        }

        private bool Authorize(HttpActionContext actionContext)
        {
            try
            {
                var encodedString = actionContext.Request.Headers.GetValues("Token").First();

                bool validFlag = false;

                if (!string.IsNullOrEmpty(encodedString))
                {
                    var key = EncryptionLibrary.DecryptText(encodedString);

                    string[] parts = key.Split(new char[] { ':' });

                    var UserID = Convert.ToInt32(parts[0]);       // UserID
                    var RandomKey = parts[1];                     // Random Key
                    var CompanyID = Convert.ToInt32(parts[2]);    // CompanyID
                    long ticks = long.Parse(parts[3]);            // Ticks
                    DateTime IssuedOn = new DateTime(ticks);
                    var ClientID = parts[4];                      // ClientID


                    // By passing this parameter
                    var registerModel = (from register in db.ClientKeys
                                         where register.CompanyID == CompanyID
                                         && register.UserID == UserID
                                         && register.ClientID == ClientID
                                         select register).FirstOrDefault();

                    if (registerModel != null)
                    {
                        // Validating Time
                        var ExpiresOn = (from token in db.TokensManager
                                         where token.CompanyID == CompanyID
                                         select token.ExpiresOn).FirstOrDefault();

                        if ((DateTime.Now > ExpiresOn))
                        {
                            validFlag = false;
                        }
                        else
                        {
                            validFlag = true;
                        }
                    }
                    else
                    {
                        validFlag = false;
                    }
                }
                return validFlag;
            }
            catch (Exception)
            {
                return false;
            }
        }

    }
}
```

After completing with adding `APIAuthorizeAttribute` Next we need to apply this attribute to the controller.

But we have not added the controller (ApiController) on which we need to apply this attribute let's add ApiController with name "**LatestMusic".**

Adding ApiController LatestMusic

In this part, we are going to add ApiController with name "**LatestMusic**" and it will have a single Action Method in it with name **GetMusicStore** which will return a list of latest music from the database.

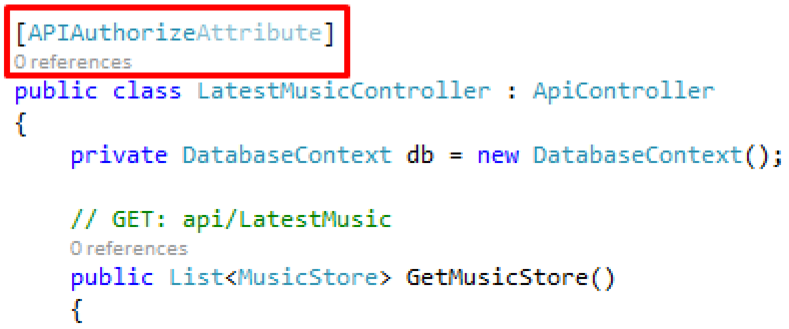

Controllers
ApplicationKeysController.cs
AuthenticateController.cs
HomeController.cs
LatestMusicController.cs
LoginController.cs
RegisterCompanyController.cs
RegisterUserController.cs

LatestMusic Controller Code Snippet

Hide Shrink Copy Code

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using MusicAPIStore.Context;
using MusicAPIStore.Models;
using MusicAPIStore.Filters;

namespace MusicAPIStore.Controllers
{
    [APIAuthorizeAttribute]
    public class LatestMusicController : ApiController
    {
        private DatabaseContext db = new DatabaseContext();
        // GET: api/LatestMusic
        public List<MusicStore> GetMusicStore()
        {
            try
            {
                var listofSongs = db.MusicStore.ToList();
                return listofSongs;
            }
            catch (System.Exception)
            {
                throw;
            }
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }

    }
}
```

In above code snippet if you have a close look then you can see that we have applied Attribute at the controller level.



```
[APIAuthorizeAttribute]
0 references
public class LatestMusicController : ApiController
{
    private DatabaseContext db = new DatabaseContext();

    // GET: api/LatestMusic
    0 references
    public List<MusicStore> GetMusicStore()
    {
```

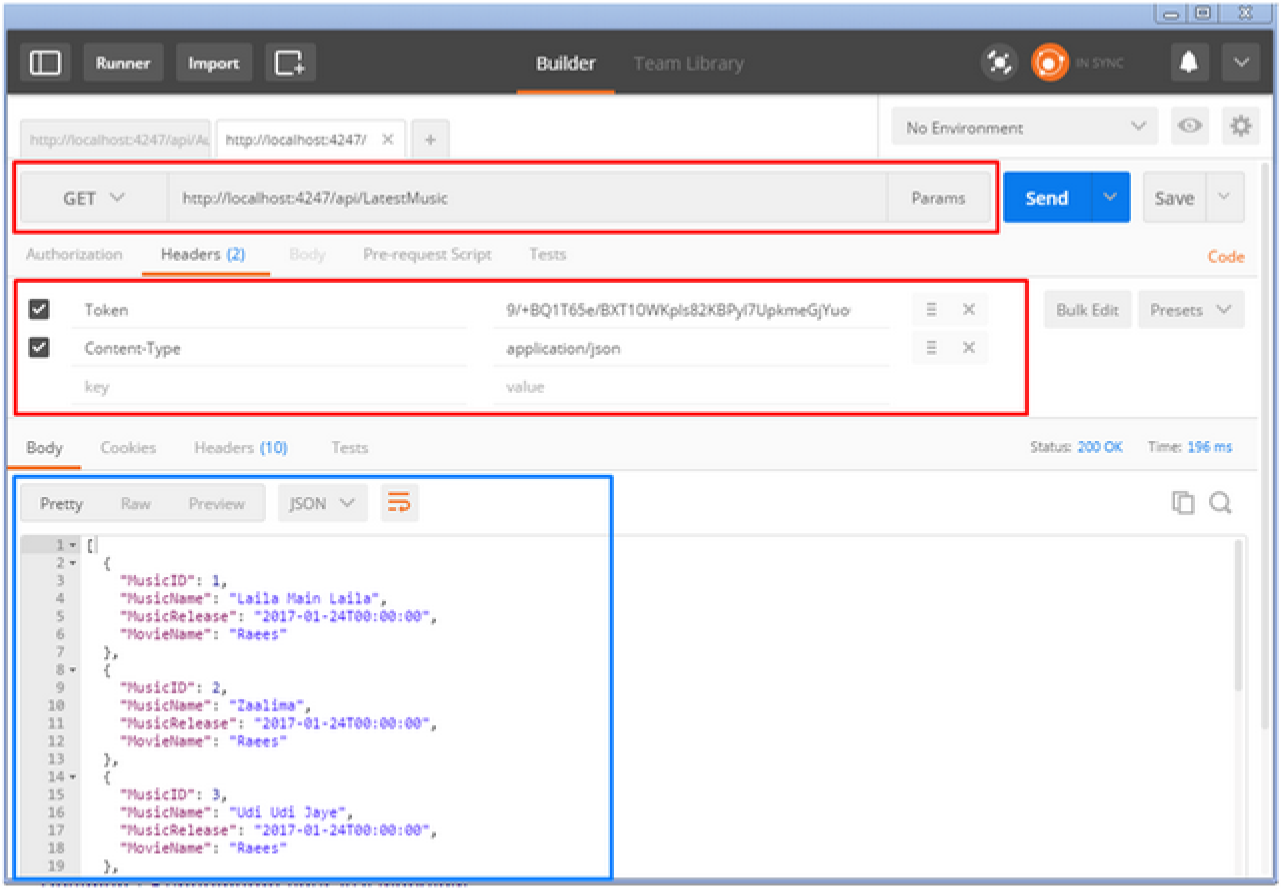
Now, whenever anyone going to access this controller needs to have a valid token after that only he can access **LatestMusic API**.

Accessing LatestMusic API Controller

For accessing **LatestMusic API** we need to send a valid token in the header.

Let's start from authenticating process first after authenticating we are going to receive a valid token in response and this token again we are going to send to access **LatestMusic API**.

1. Authenticating process to get token



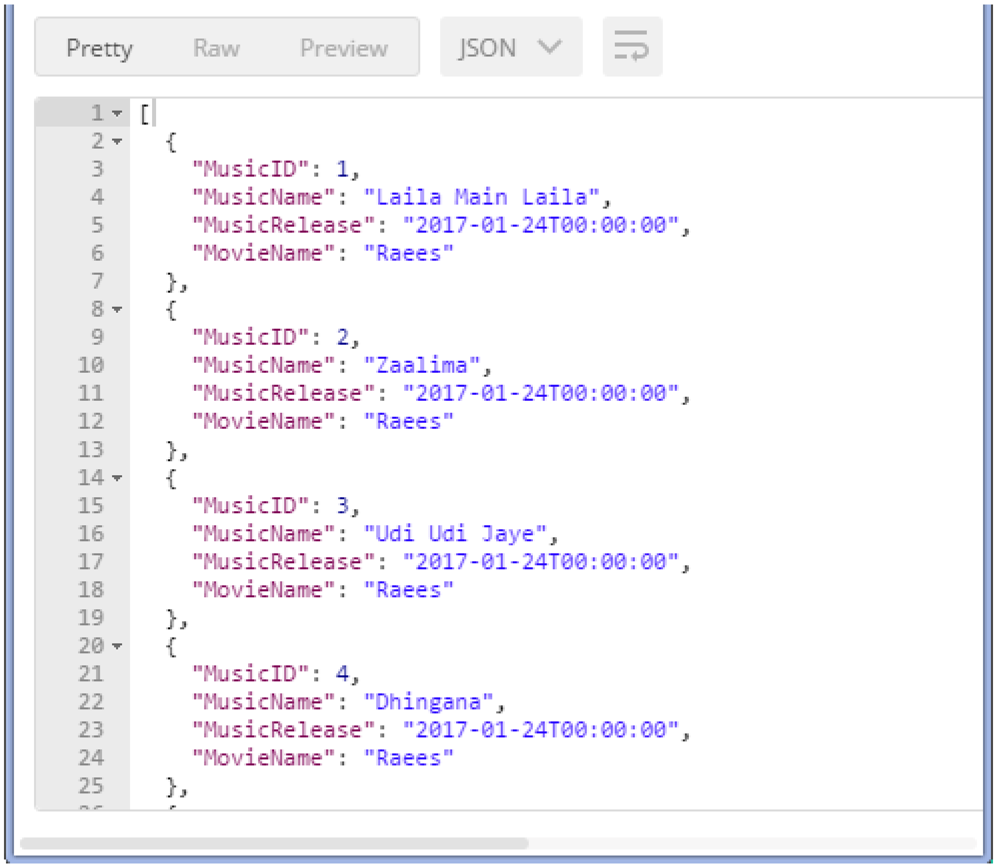
GET http://localhost:4247/api/LatestMusic

Token	9/+BQ1T65e/BXT10WKpls82KBPyl7UpkmeGjYuo
Content-Type	application/json
key	value

Status: 200 OK Time: 196 ms

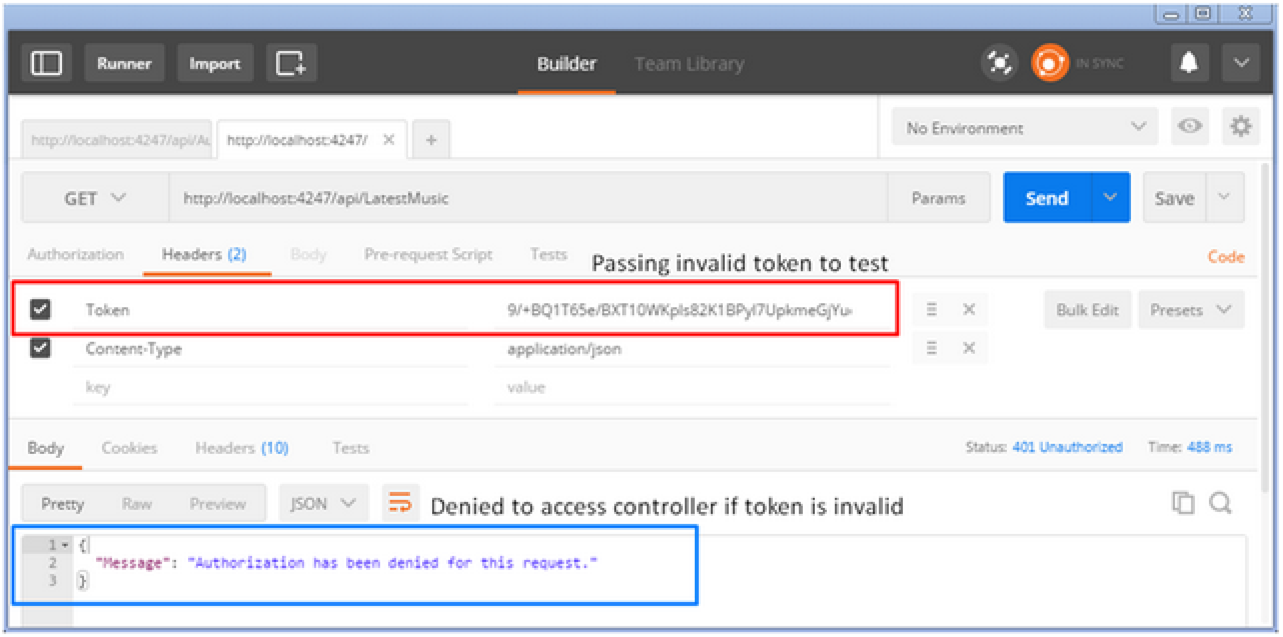
```
[
  {
    "MusicID": 1,
    "MusicName": "Laila Main Laila",
    "MusicRelease": "2017-01-24T00:00:00",
    "MovieName": "Raees"
  },
  {
    "MusicID": 2,
    "MusicName": "Zaalima",
    "MusicRelease": "2017-01-24T00:00:00",
    "MovieName": "Raees"
  },
  {
    "MusicID": 3,
    "MusicName": "Udi Udi Jaye",
    "MusicRelease": "2017-01-24T00:00:00",
    "MovieName": "Raees"
  },
```

2. Sending token to access LatestMusic **API** and getting Top 10 Music hit list in response.



```
[
  {
    "MusicID": 1,
    "MusicName": "Laila Main Laila",
    "MusicRelease": "2017-01-24T00:00:00",
    "MovieName": "Raees"
  },
  {
    "MusicID": 2,
    "MusicName": "Zaalima",
    "MusicRelease": "2017-01-24T00:00:00",
    "MovieName": "Raees"
  },
  {
    "MusicID": 3,
    "MusicName": "Udi Udi Jaye",
    "MusicRelease": "2017-01-24T00:00:00",
    "MovieName": "Raees"
  },
  {
    "MusicID": 4,
    "MusicName": "Dhingana",
    "MusicRelease": "2017-01-24T00:00:00",
    "MovieName": "Raees"
  },
```

Response in details



Passing invalid token to test

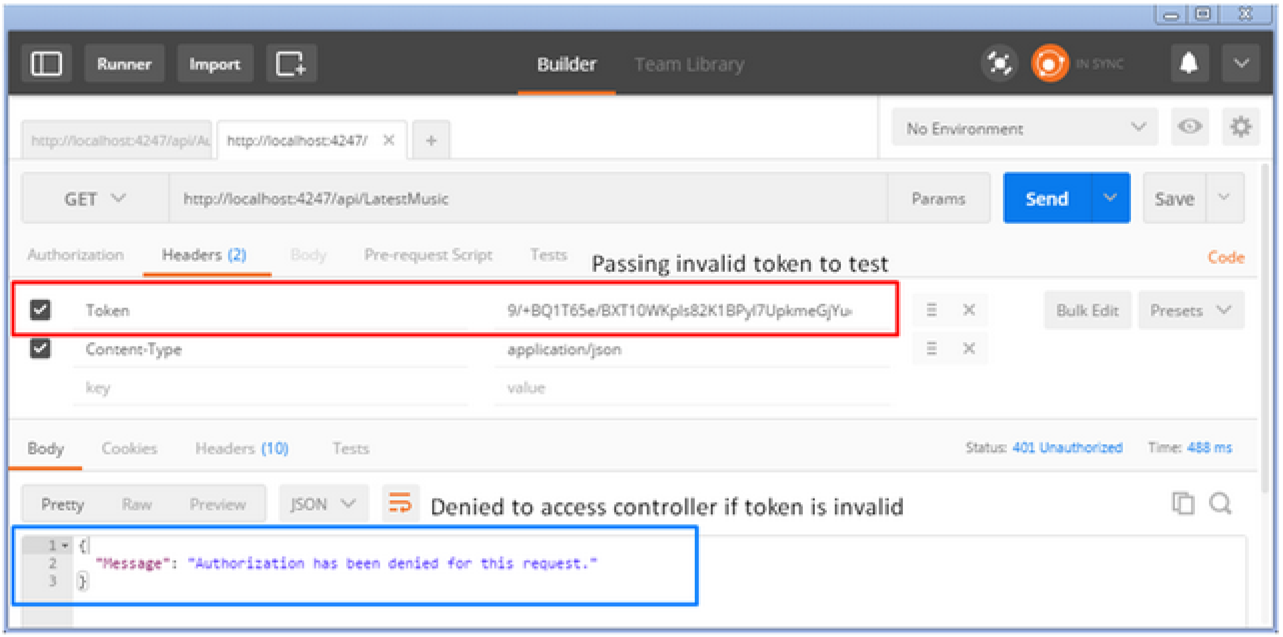
✓	Token	9/+BQ1T65e/BXT10WKpls82K1BPyl7UpkmeGjYu
✓	Content-Type	application/json
	key	value

Status: 401 Unauthorized Time: 488 ms

Denied to access controller if token is invalid

```
{
  "Message": "Authorization has been denied for this request."
}
```

Passing Invalid Token to test Response



Passing invalid token to test

✓	Token	9/+BQ1T65e/BXT10WKpls82K1BPyl7UpkmeGjYu
✓	Content-Type	application/json
	key	value

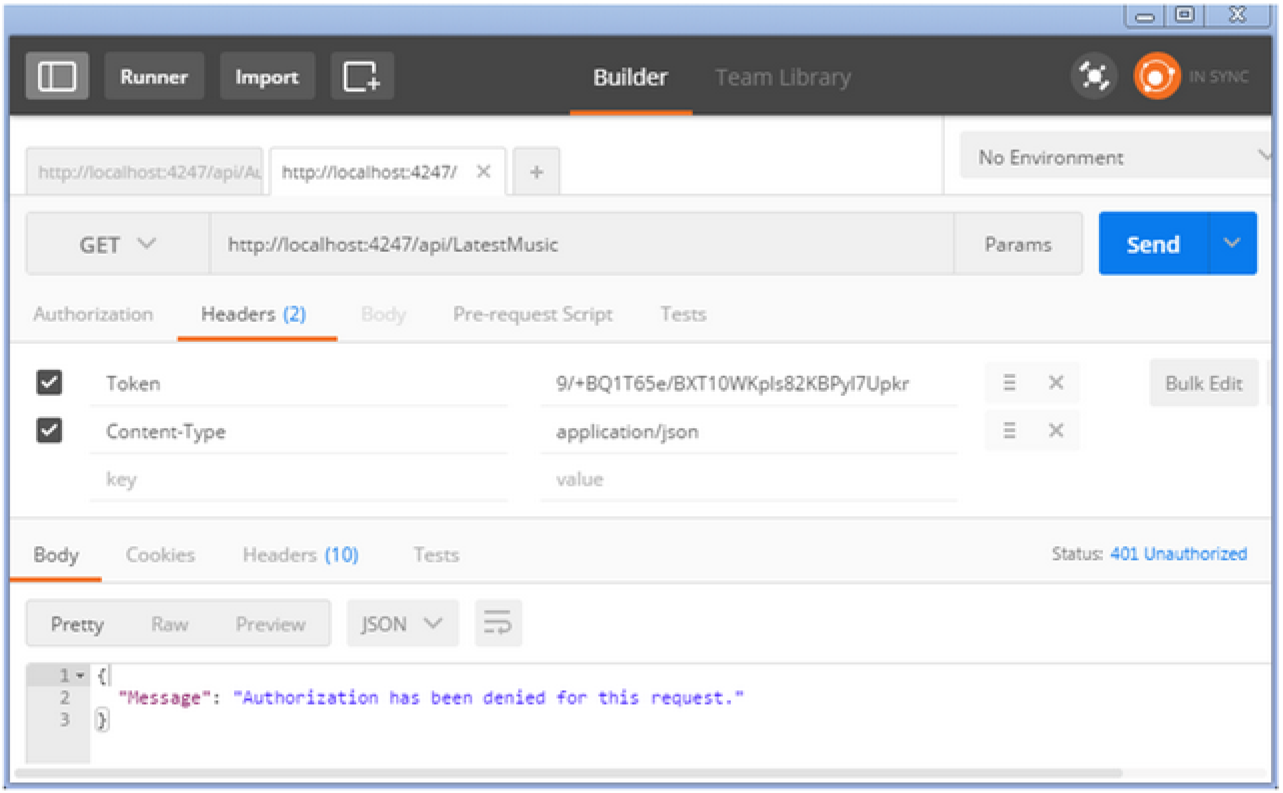
Status: 401 Unauthorized Time: 488 ms

Denied to access controller if token is invalid

```
{
  "Message": "Authorization has been denied for this request."
}
```

Validating Token after Session Expiration

After sending request when the token is expired it will show an error message as shown below.



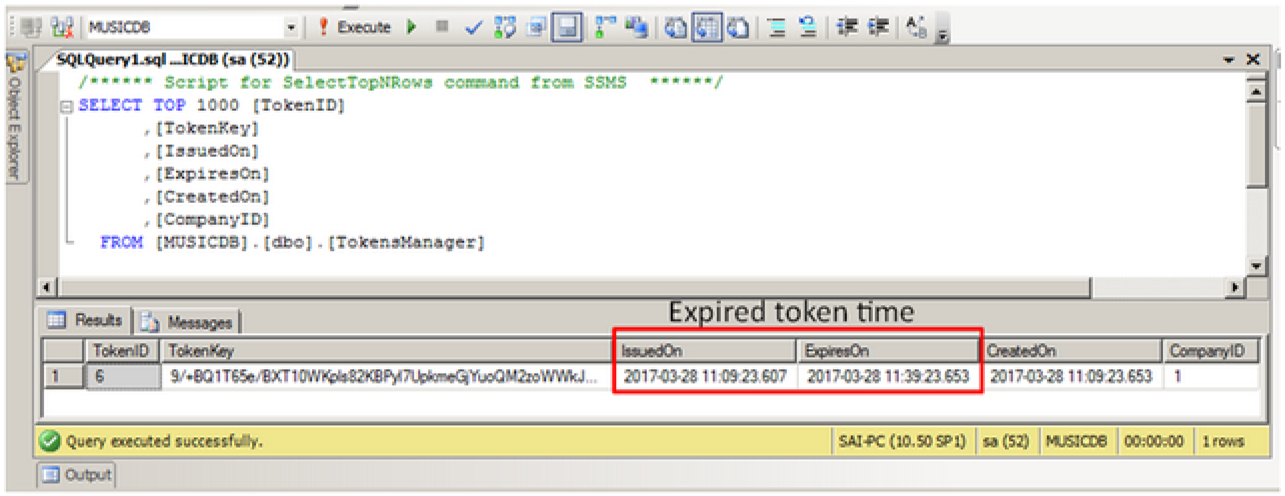
GET http://localhost:4247/api/LatestMusic

Token	9/+BQ1T65e/BXT10WKpls82KBPyl7Upkr
Content-Type	application/json
key	value

Status: 401 Unauthorized

```
{
  "Message": "Authorization has been denied for this request."
}
```

Table where token expiration date and time is stored



```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [TokenID]
      ,[TokenKey]
      ,[IssuedOn]
      ,[ExpiresOn]
      ,[CreatedOn]
      ,[CompanyID]
  FROM [MUSICDB].[dbo].[TokensManager]
```

	TokenID	TokenKey	IssuedOn	ExpiresOn	CreatedOn	CompanyID
1	6	9/+BQ1T65e/BXT10WKpls82KBPyl7UpkmeGjYuoQM2zoWWkJ...	2017-03-28 11:09:23.607	2017-03-28 11:39:23.653	2017-03-28 11:09:23.653	1

Query executed successfully. SAI-PC (10.50 SP1) sa (52) MUSICDB 00:00:00 1 rows

Conclusion

In this article, we have learned how to secure WEB API using token based authentication in step by step way and in detail manner such that junior developer can also understand it very easily, now you can secure your most client based application using this process, and also server based application.

Thank you, I hope you liked my article.

License

Share

About the Author

Saineshwar Bageri

Software Developer (Senior)
India

Follow this Member

Microsoft Most Valuable Professional
Code Project Most Valuable Author
C# Corner Most Valuable Professional

I am Software Developer Working on.Net Web Technology
ASP.NET MVC,.Net Core,ASP.NET CORE, C#, SQL server, MYSQL, MongoDB, Windows

I Also do Freelancing on.Net Technology

You may also be interested in...

Public, Private, and Hybrid Cloud: What's the difference?

Simple Transaction: Microservices Sample Architecture for .NET Core Application

RESTful Day #5: Security in Web APIs-Basic Authentication and Token based custom Authorization in Web APIs using Action Filters

jOOPL: Object-Oriented Programming for JavaScript

Claims And Token Based Authentication (ASP.NET Web API)

Token Based Authentication for Web API where Legacy User Database is used

Comments and Discussions

Add a Comment or Question

Email Alerts Search Comments

First Prev Next

nice article. thank you
diptishinde1983 29-Jan-19 13:34

Hide Copy Code

```
using MusicAPIStore.AES256Encryption;
```

what does this encryption class contain?code for that is not included here. so not able to implement it.
Actually i am trying to do token based user authentication without entity framework.
can you give any inputs. As i am new to mvc and angularjs. thank you

Reply · Email · View Thread

superb excellent article
naren 8198614 6-Dec-18 1:33

Great sir,I am very gratefully to you . what a simple and easy explanation.Thanks a lot.

Reply · Email · View Thread

Amazing article thanks! One issue...
Member 14065726 24-Nov-18 21:45

Great article and it is much appreciated. The one thing I did see in here that could be problematic is that in your APIAuthorizeAttribute after splitting and checking the values to ensure the token is valid, the call to check the expiration does not include the token to match to the one in the DB. Therefore, if someone passes an old, expired token it will still validate as long as that company has an unexpired token. I simply added

Hide Copy Code

```
&& token.TokenKey == encodedString
```

to the call to ensure the token being sent is not itself an old expired one.

Thanks again!

Reply · Email · View Thread

Strange response

apiegoku 11-Sep-18 14:26

Hi,

I've used your code to use it on an Azure web app. It works great if you pass a correct token. But if the token is invalid or the token is expired, it will return a complete html page.(See example).
Any idea whats wrong?

```
<!DOCTYPE html>
<html dir="ltr" class="" lang="en">
<head>
<title>Sign in to your account</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=2.0, user-scalable=yes">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Expires" content="-1">
<meta name="PageID" content="ConvergedSignIn" />
```

Reply · View Thread

My vote of 5

Member 4109344 29-Aug-18 12:44

Thanks a lot

Reply · Email · View Thread

Good code n Explained.

Rahul Kadam Pune 2-Aug-18 9:33

Thank u sirji.....for providing best article.That always. ...

Reply · Email · View Thread

Regarding issue

Member 13300567 14-Jun-18 18:48

I am not able to complete this demo i am getting error when i am going to interact with the database like add() or ValidateUsername()

please find error below,

One or more validation errors were detected during model generation:

```
MusicAPIstore.Context.ClientKeys: : EntityType 'ClientKeys' has no key defined. Define the key for this EntityType.
MusicAPIstore.Context.MusicStore: : EntityType 'MusicStore' has no key defined. Define the key for this EntityType.
```

MusicAPIstore.Context.RegisterCompany: : EntityType 'RegisterCompany' has no key defined. Define the key for this EntityType.
MusicAPIstore.Context.TokensManager: : EntityType 'TokensManager' has no key defined. Define the key for this EntityType.
ClientKeys: EntityType: EntitySet 'ClientKeys' is based on type 'ClientKeys' that has no keys defined.
MusicStore: EntityType: EntitySet 'MusicStore' is based on type 'MusicStore' that has no keys defined.
RegisterCompany: EntityType: EntitySet 'RegisterCompany' is based on type 'RegisterCompany' that has no keys defined.
TokensManager: EntityType: EntitySet 'TokensManager' is based on type 'TokensManager' that has no keys defined.

Reply · Email · View Thread 5.00/5 (1 vote)

Re: Regarding issue
Member 12917715 19-Jun-18 15:16

Make sure you give [Key] attribute while defining the model(s)

eg: [Table("ClientKey")]
public class ClientKey
{
[Key]
public int ClientKeyID { get; set; }
.
.

Reply · Email · View Thread

You rocked
Prafulla Sahu 6-Apr-18 12:42

nice explanation and detailed.

Reply · Email · View Thread

Login form issue
Member 11255091 24-Mar-18 14:18

Hide Copy Code

```
But on submit button there is no any api call ? how to validate user after login ?
```

Reply · Email · View Thread

Very nice article.
Member 13614638 9-Jan-18 19:46

Question is can we just have username and password and generate a token based on that? Do we need the client keys??

Reply · Email · View Thread

Re: Very nice article.
Saineshwar Bageri 10-Jan-18 4:50

ya we can generate token on bases of that with some random parameters such that it always generate unique string.

Reply · Email · View Thread

Single sign on
Meh :D B@y@t 18-Nov-17 8:45

grate article tanks for that. could you please write a article like this i mean token base authentication web API in SSO?

Reply · Email · View Thread 5.00/5 (1 vote)

Looking for Role based example like token example you have given.
Member 10458787 16-Nov-17 18:04

This is such a very nice example i ever found for custom token based Authentication. Please provide a Role based Authorization example without using any framework like OWIN or IdentityFramwork.
Looking for your response.

Neeraj

Reply · Email · View Thread

The relevance of clientkeys table
Member 12752412 15-Nov-17 2:00

What is the need of clientkeystable? Can't we just send user's information and generate token with it?

Reply · Email · View Thread 5.00/5 (1 vote)

How to get a new token when the current token is expired or about to expire
Mohammed Abdul Mateen 10-Nov-17 7:50

what is the recommended approach for a C# client app (bot app) to get a new token when the current token is expired or about to expire, could you please provide suggestions.

We thought of having a retry helper class (retry pattern), where if the request fails to be unauthorized or token expired we will get a new token from web API and retry 1 time.

We're thinking it may be better to fetch a new token when it is expired or about to expire instead of doing it after an API call returns unauthorized/token expired.

Please suggest a recommended approach as soon as possible.

Reply · Email · View Thread

Re: How to get a new token when the current token is expired or about to expire
dan ross 3-Aug-18 3:57

hey Mohammed,

Did you come up with a way to do a refresh?

Reply · Email · View Thread

Thanks
huu loi huynh 9-Nov-17 1:37

It's very helpful and save my time

Reply · Email · View Thread

Re: Thanks
Saineshwar Bageri 9-Nov-17 17:26

Thanks friend

Reply · Email · View Thread

Unable to download code
mvs narayana 26-Oct-17 17:21

Nice article. But unable to download the code from the link. Please update.

Reply · Email · View Thread 5.00/5 (1 vote)

Re: Unable to download code
Saineshwar Bageri 26-Oct-17 19:49

Sir you can download from above link [download project source code] it is work it will take you to dropbox site

Reply · Email · View Thread

Re: Unable to download code
mvs narayana 27-Oct-17 9:20

Sorry. Its my mistake. Couldn't access it from my office network. Thanks for the reply.

Reply · Email · View Thread

Re: Unable to download code
Saineshwar Bageri 27-Oct-17 9:41

ohh

Reply · Email · View Thread

Great
MaartenKumpen 25-Oct-17 8:58

Great article, thanks.

Reply · Email · View Thread

Re: Great
Saineshwar Bageri 25-Oct-17 19:08

Thanks Sir

Reply · Email · View Thread

Refresh 1 2 Next »

General News Suggestion Question Bug Answer Joke Praise Rant Admin