Create a Notes Class in the Models Folder and add the following Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespace OAUTHDEMO.Models
{
    public class Notes
    {
        public int Id { get; set; }
        [StringLength(50)]
        public string NotesTitle { get; set; }
        [StringLength(1000)]

        public string NotesDescription { get; set; }
        public string UserId { get; set; }
    }
}
```

Right After that just add a NotesController and add the following code in this NotesController.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Description;
using Microsoft.AspNet.Identity;
using OAUTHDEMO.Models;

namespace OAUTHDEMO.Controllers
{
    [Authorize]
    public class NotesController : ApiController
    {
        private OAUTHDEMOContext db = new OAUTHDEMOContext();

        // GET: api/Notes
        public IQueryable<Notes> GetNotes()
        {
            string userId = User.Identity.GetUserId();
```

```
    return db.Notes.Where(n => n.UserId == userId);
}

// GET: api/Notes/5
[ResponseType(typeof(Notes))]
public async Task<IHttpActionResult> GetNotes(int id)
{
    Notes notes = await db.Notes.FindAsync(id);
    if (notes == null)
    {
        return NotFound();
    }

    return Ok(notes);
}

// PUT: api/Notes/5
[ResponseType(typeof(void))]
public async Task<IHttpActionResult> PutNotes(int id, Notes notes)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    if (id != notes.Id)
    {
        return BadRequest();
    }
    string userId = User.Identity.GetUserId();
    if (userId != notes.UserId)
    {
        return StatusCode(HttpStatusCode.Conflict);
    }

    db.Entry(notes).State = EntityState.Modified;

    try
    {
        await db.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!NotesExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return StatusCode(HttpStatusCode.NoContent);
}

// POST: api/Notes
```

```
        [ResponseType(typeof(Notes))]
        public async Task<IHttpActionResult> PostNotes(Notes notes)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
            string userId = User.Identity.GetUserId();
            notes.UserId = userId;
            db.Notes.Add(notes);
            await db.SaveChangesAsync();

            return CreatedAtRoute("DefaultApi", new { id = notes.Id }, notes);
        }

        // DELETE: api/Notes/5
        [ResponseType(typeof(Notes))]
        public async Task<IHttpActionResult> DeleteNotes(int id)
        {
            Notes notes = await db.Notes.FindAsync(id);
            if (notes == null)
            {
                return NotFound();
            }
            string userId = User.Identity.GetUserId();
            if (userId != notes.UserId)
            {
                return StatusCode(HttpStatusCode.Conflict);
            }

            db.Notes.Remove(notes);
            await db.SaveChangesAsync();

            return Ok(notes);
        }

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

        private bool NotesExists(int id)
        {
            return db.Notes.Count(e => e.Id == id) > 0;
        }
    }
}
```