**C# Project code summary**

In this project we were working to create a management portal for a construction company that also worked as an employee portal/time card manager. It was a good experience to work more with the ASP.NET MVC and Entity. The first thing I had to do was create the user roles and create an admin user.

|  |
| --- |
| ApplicationDbContext context = new ApplicationDbContext(); var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(context)); var UserManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(context)); |

|  |
| --- |
| if (!roleManager.RoleExists("Admin"))  {   // first we create Admin role   var role = new IdentityRole();  role.Name = "Admin";  roleManager.Create(role);   //Here we create a Admin super user who will maintain the website    var user = new ApplicationUser();  user.UserName = "Admin1";  user.Email = "admin@gmail.com";   string userPWD = "1234@!AZ";   var chkUser = UserManager.Create(user, userPWD);   //Add default User to Role Admin   if (chkUser.Succeeded)  {  var result1 = UserManager.AddToRole(user.Id, "Admin");   }  }   // creating Creating Manager role   if (!roleManager.RoleExists("Manager"))  {  var role = new IdentityRole();  role.Name = "Manager";  roleManager.Create(role);   }   // creating Creating Employee role   if (!roleManager.RoleExists("Employee"))  {  var role = new IdentityRole();  role.Name = "Employee";  roleManager.Create(role);   } |

After that I created the views for the admin and restricted access to things like user creation to the admin. I also created a check to see if the user was an admin

|  |
| --- |
| public bool isAdminUser()  {  if (User.Identity.IsAuthenticated)  {  var user = User.Identity;  ApplicationDbContext context = new ApplicationDbContext();  var UserManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(context));  var s = UserManager.GetRoles(user.GetUserId());  if (s[0].ToString() == "Admin")  {  return true;  }  else  {  return false;  }  }  return false;  } |

I also was tasked with creating the jobsite and the controllers, views, and models. Thankfully leveraging Entity creating all of those things was a breeze and the only thing I had to set up to create all of the things was the model.

namespace ManagementPortal.Models

|  |
| --- |
| {  public class Jobsite  {  [DisplayName("Job Site ID")]  public int JobSiteID { get; set; }  [DisplayName("Address")]  public string Address { get; set; }  [DisplayName("Town")]  public string Town { get; set; }  [DisplayName("State")]  public string State { get; set; }  [DisplayName("Zip")]  public string Zip { get; set; }  } } |

I also created a new homepage that would that would redirect all AnonymousUsers.

The last thing I did was make something that would show a map of all of the jobsites for all of the employees to see on the jobsite pages that were assigned to them. I used Google maps API and wrote it in JavaScript

|  |
| --- |
| //This code tells the browser to execute the "geocode" method only when the complete document model has been loaded.  $(document).ready(function () {  geocode();  });   //This gets the address from the model and feeds it to the Google Geocoding API to get the latitude and longitude to use with the mapping API  function geocode() {  var location = "@Model.Address @Model.Town, @Model.State @Model.Zip";  console.log(location)  //Axios is a lightweight library for making Ajax requests  axios.get('https://maps.googleapis.com/maps/api/geocode/json', {  params: {  address: location,  key: '@System.Web.Configuration.WebConfigurationManager.AppSettings["mapKey"]'  }  })  .then(function (response) {  console.log(response);  var lat = response.data.results[0].geometry.location.lat;  var lng = response.data.results[0].geometry.location.lng;  console.log(lat + ', ' + lng)  mapping(lat, lng);  })   .catch(function (error) {  console.log(error);  });  }   //this take in the latitude and longitude of geocode() and uses the Google Place API to create a map and insert it into the html  function mapping(lat, lng) {   // Google has tweaked their interface somewhat - this tells the api to use that new UI  google.maps.visualRefresh = true;  var JobSite = new google.maps.LatLng(lat, lng);   // These are options that set initial zoom level, where the map is centered globally to start, and the type of map to show  var mapOptions = {  zoom: 16,  center: JobSite,  mapTypeId: google.maps.MapTypeId.G\_NORMAL\_MAP  };   // This makes the div with id "map\_canvas" a google map  var map = new google.maps.Map(document.getElementById("map\_canvas"), mapOptions);   // This shows adding a simple pin "marker"  var myLatlng = new google.maps.LatLng(lat, lng);   var marker = new google.maps.Marker({  position: myLatlng,  map: map,  title: 'Job Site'  });  } |

Overall this was a great project to work on and I found javascript and the ASP.NET MVC very enjoyable to work with. Also working with Entity was great because its a real god send when creating new views and controllers in a code first database.