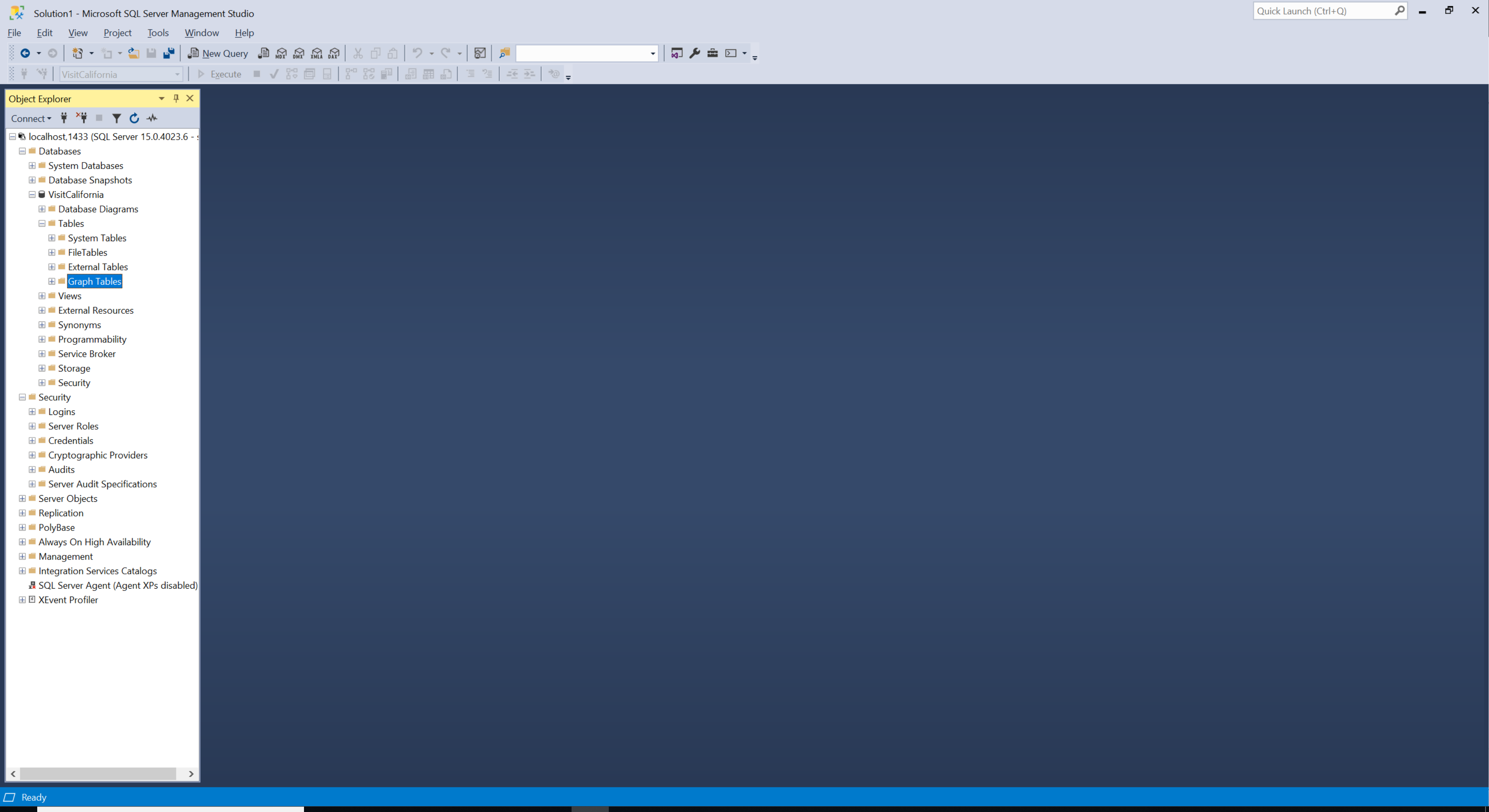
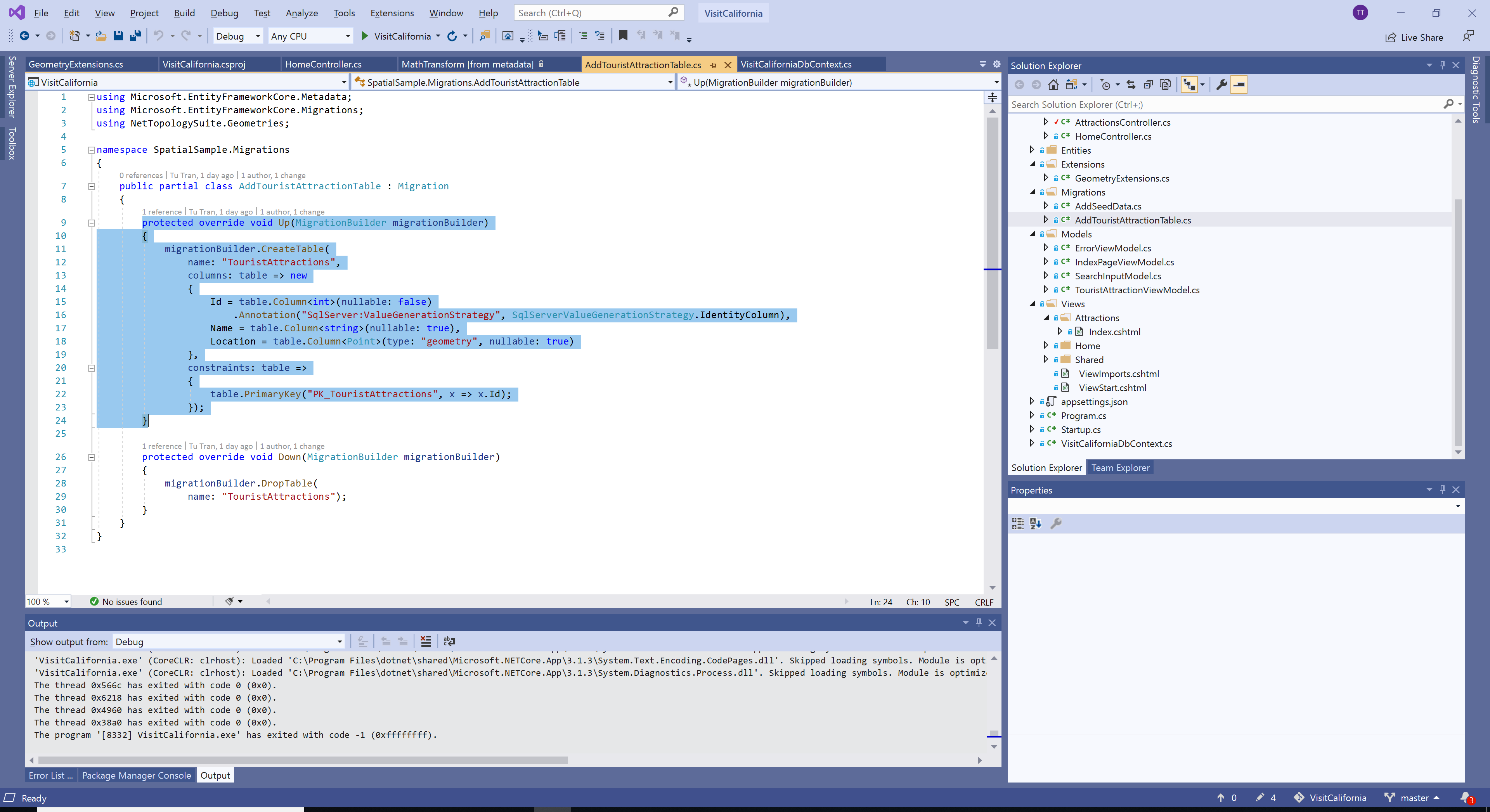
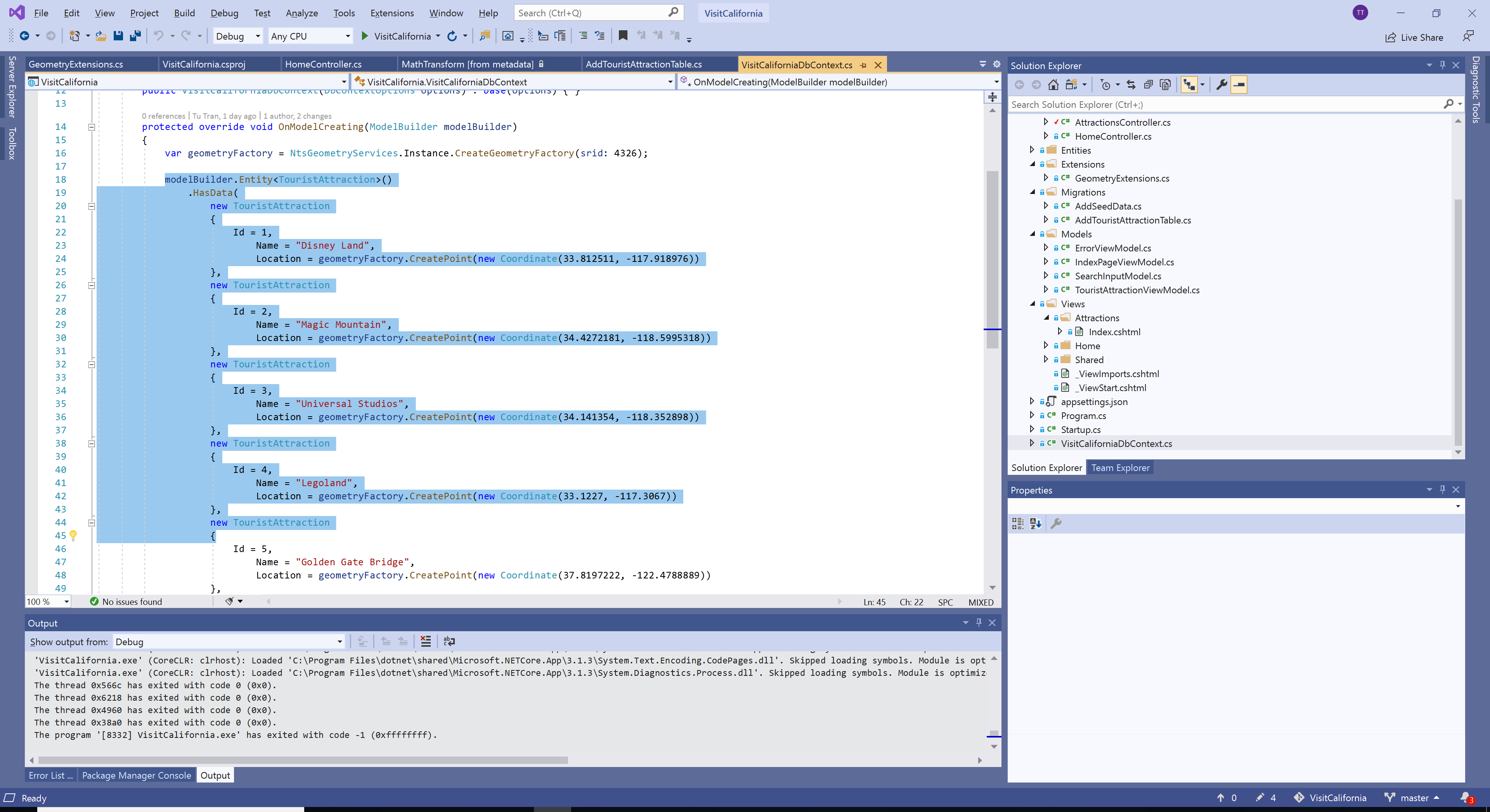
1. Verify the database is empty

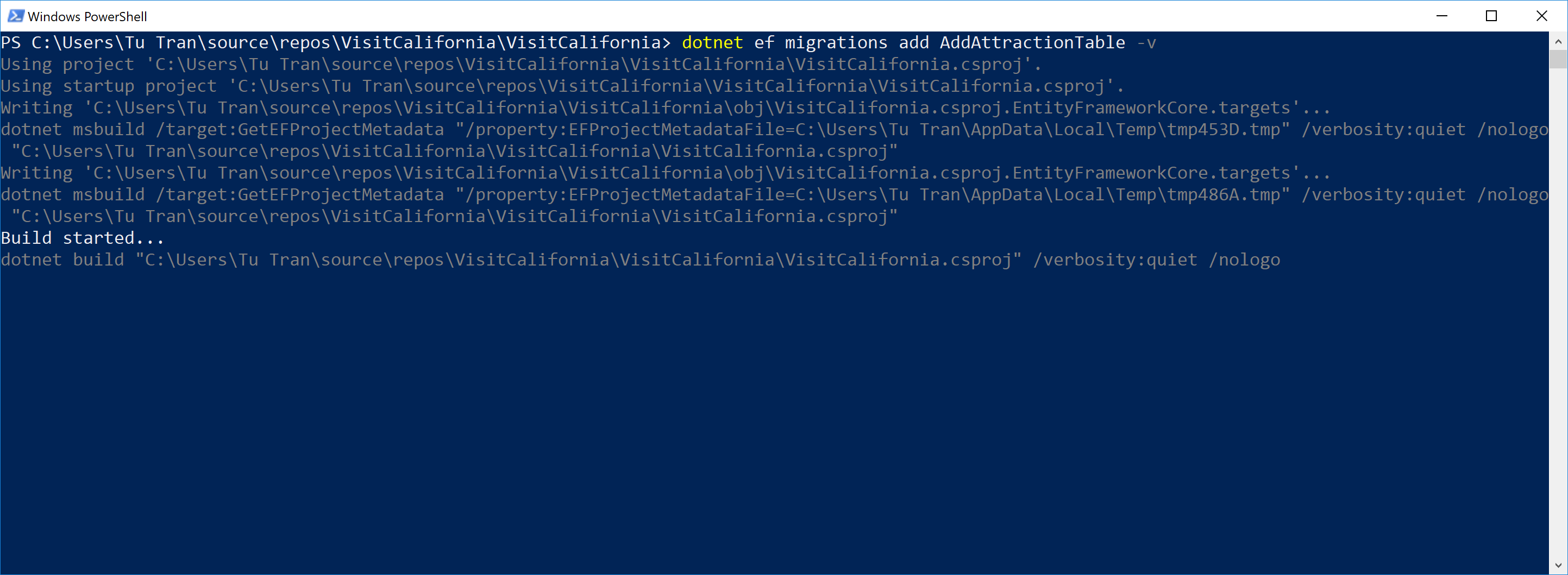


1. Class that extends Migration is used in ‘Code First’ approach to reconstruction tables from the existing models as well as creating seed data. There are no SQL statements beforehand but the framework will know how to generate the table structures and data based on C# code. Hence, it’s aptly named ‘Code First’ approach.

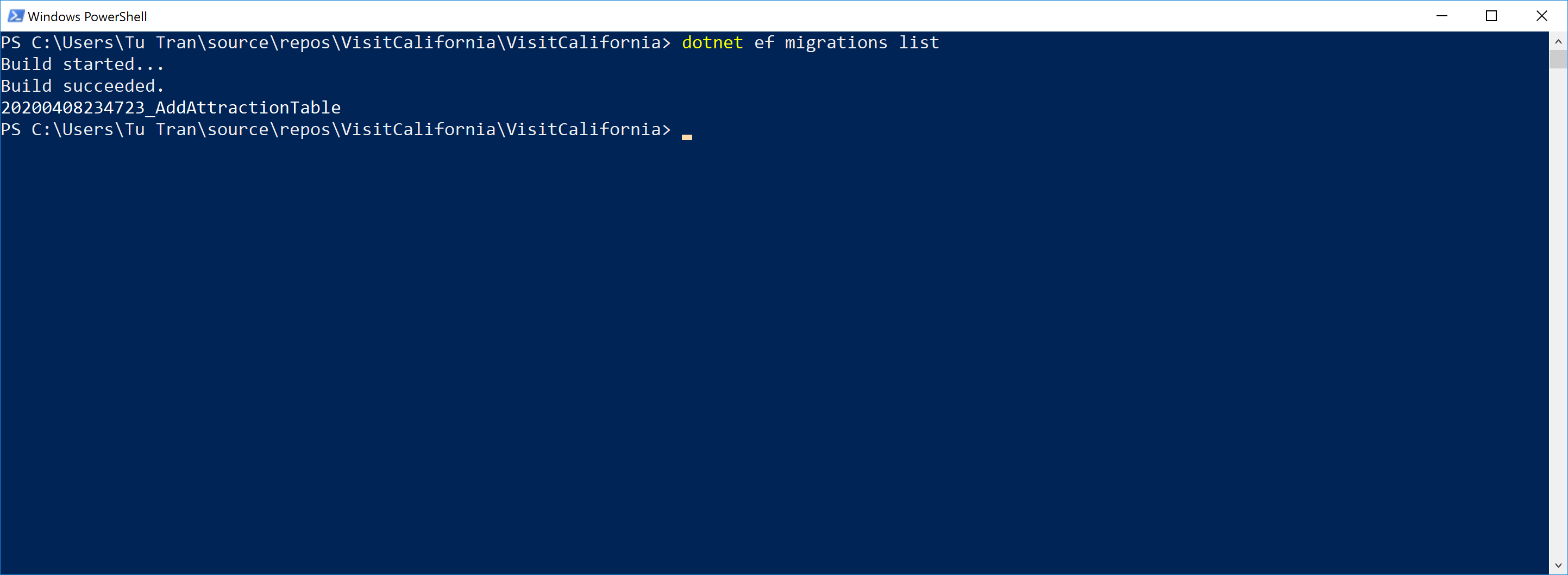




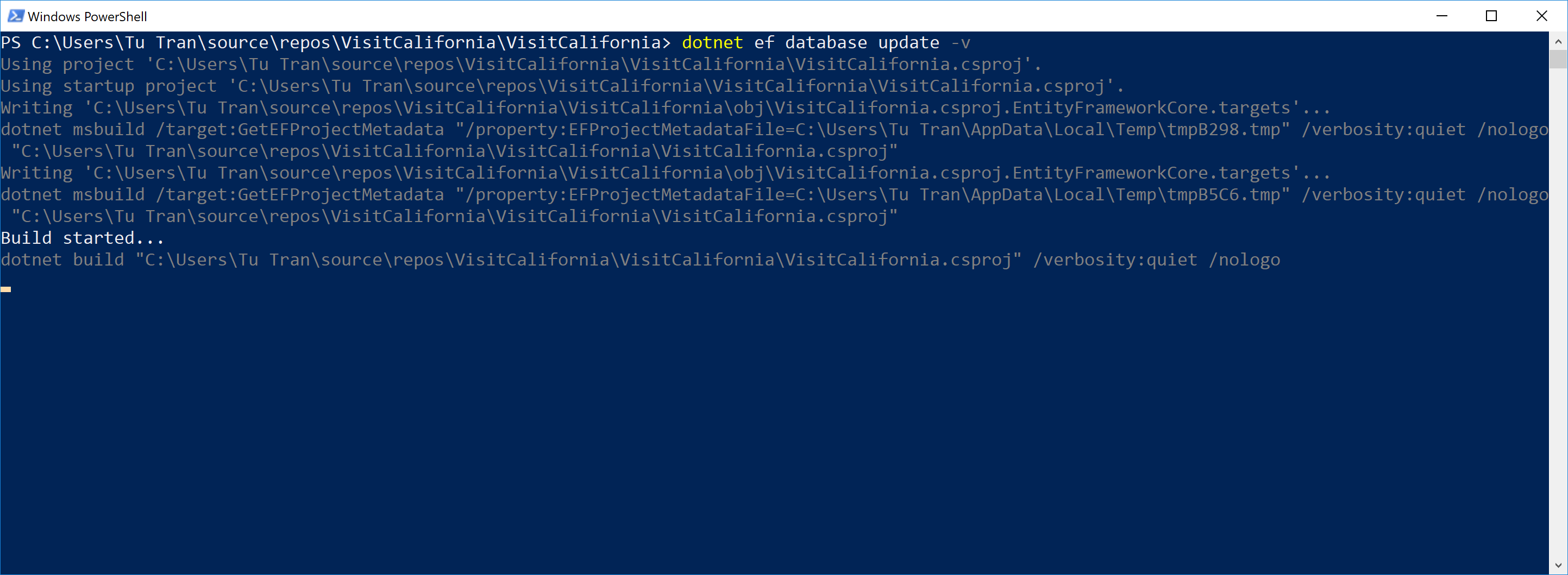
1. Add the migration class.



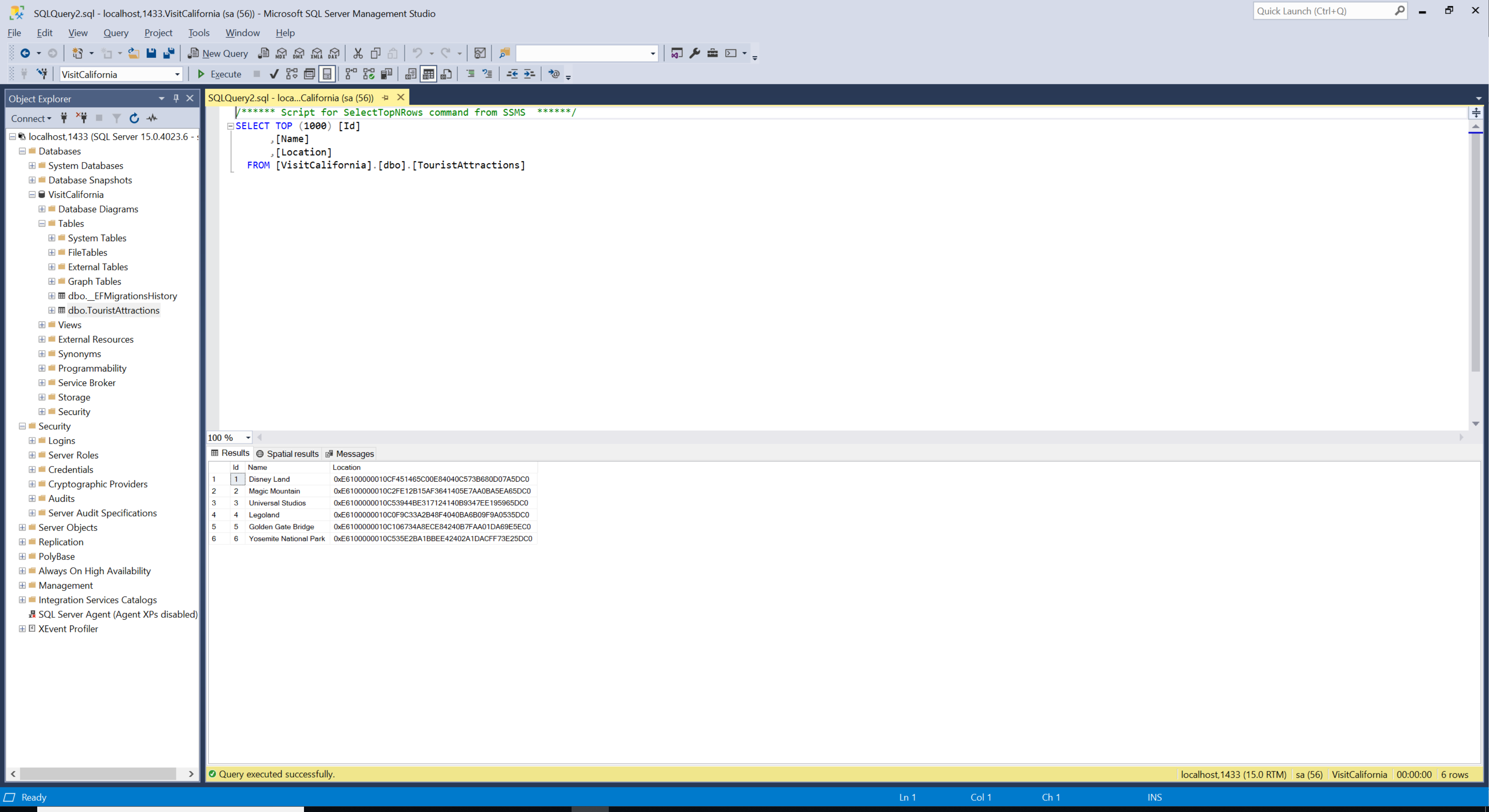
1. List the pending migration class



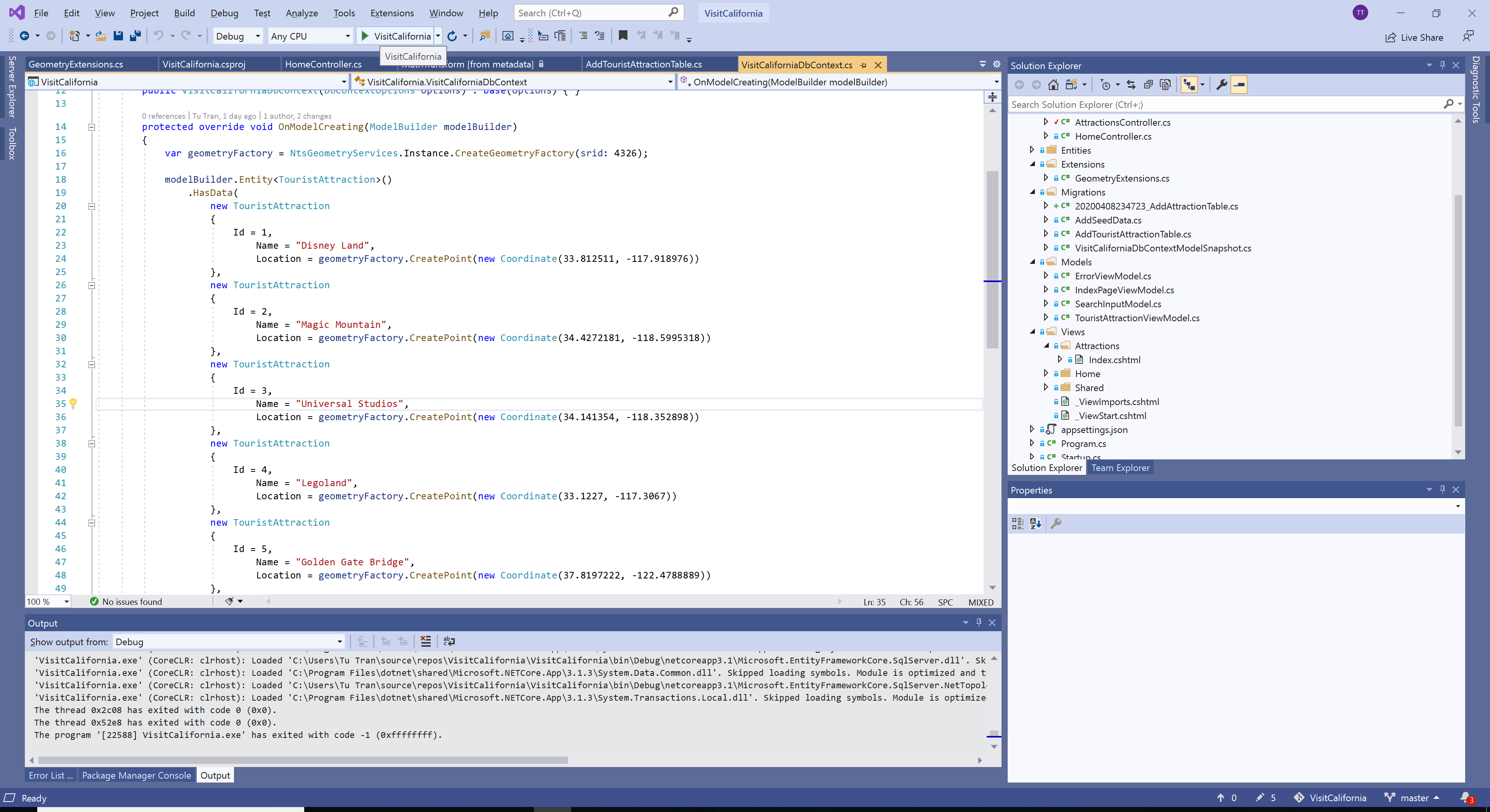
1. Apply the changes to the database



1. Verify the changes took effect. Here, we have TouristAttractions table being create along with the seed data. Note that there is no distance field because we will calculate it on-the-fly using NetTopologySuite.

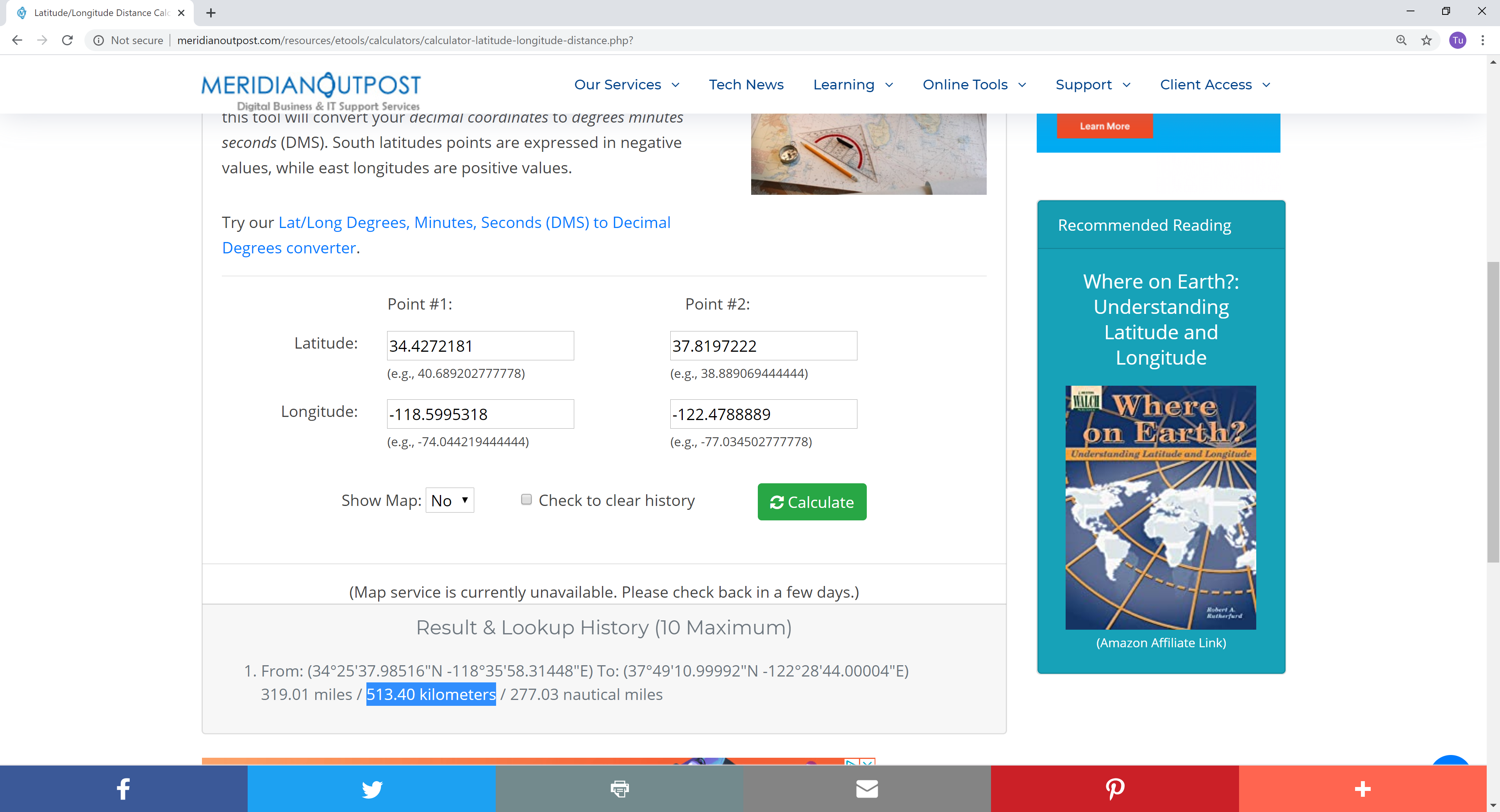


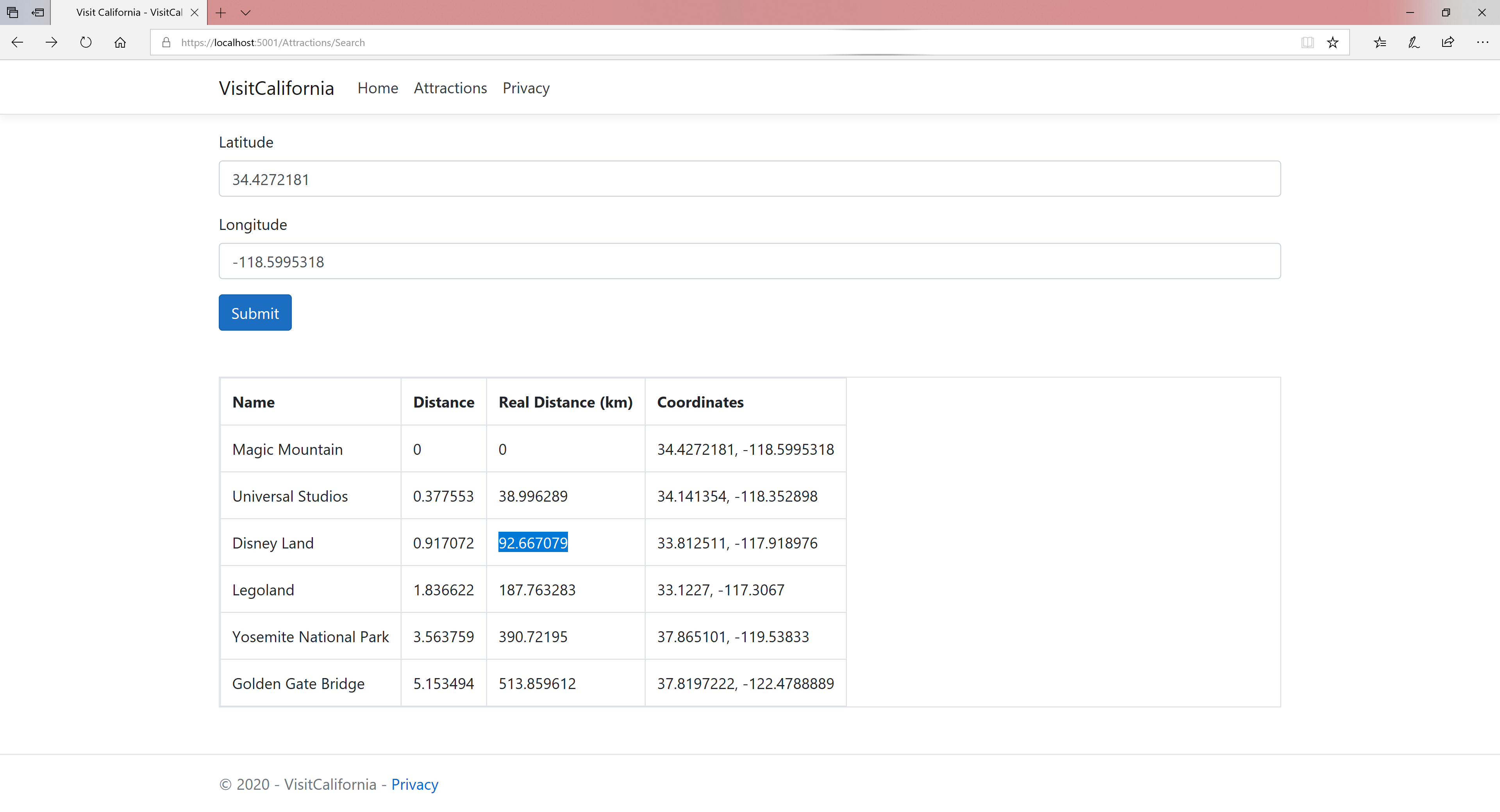
1. Start the application

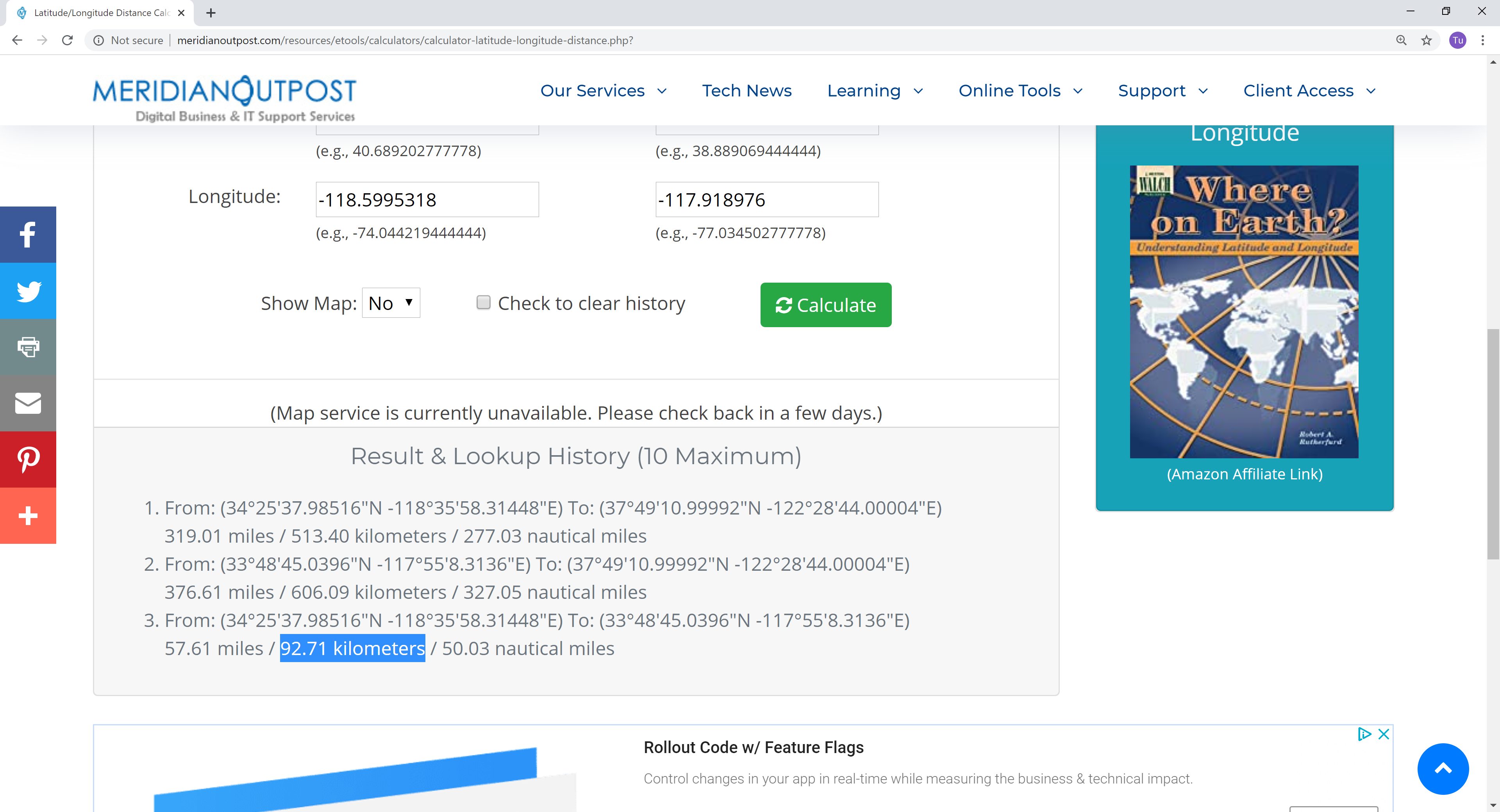


1. Verify that Magic Mountain is around 514 kilometers away from the Golden Gate Bridge and 93 kilometers from Disneyland, which coincides with the data from MeridianOutpost.com









1. Code for Controller: we get the lat/long from the input and create the original point. We use query through dbContext as a way to access data using Entity Framework. Then we do mapping: for each of the resulting row, we calculate distance of the attraction’s from the original point. Distance calculation does not yield any meaningful result of the unit of measurement of a particular coordinate system is in degree. The software can just do a Cartesian coordinate calculation of degree distance. Since we want to show something meaningful, we need to transform the points into another coordinate system with unit in meter. Here, we project the point from WGS84 with SRID 4326 into US National Atlas Projected coordinates with SRID 2163

|  |
| --- |
| using Microsoft.AspNetCore.Mvc;  using NetTopologySuite.Geometries;  using System;  using System.Linq;  using VisitCalifornia.Extensions;  using VisitCalifornia.Models;  namespace VisitCalifornia.Controllers  {  public class AttractionsController : Controller  {  VisitCaliforniaDbContext \_dbContext;  public AttractionsController(VisitCaliforniaDbContext dbContext)  {  \_dbContext = dbContext;  }  public IActionResult Index()  {  return View();  }  [HttpPost]  public IActionResult Search([FromForm] AttractionViewModel indexModel)  {  var indexViewModel = new AttractionViewModel  {  SearchInput = indexModel.SearchInput  };  // Convert the input latitude and longitude to a Point  var location = new Point(indexModel.SearchInput.Latitude, indexModel.SearchInput.Longitude) { SRID = 4326 };  var RealLocation = new Point(location.Y, location.X) { SRID = 4326 };  // Fetch the tourist attractions and their  // distances from the input location  // using spatial queries.  var touristAttractions = \_dbContext  .TouristAttractions  .Select(t => new { Place = t, Distance = t.Location.Distance(location), RealDistance = new Point(t.Location.Y, t.Location.X) { SRID = 4326 }.ProjectTo(2163).Distance(RealLocation.ProjectTo(2163)) })  .ToList();  // Ordering the result in the ascending order of distance  indexViewModel.TouristAttractions = touristAttractions  .OrderBy(x => x.Distance)  .Select(t => new TouristAttractionViewModel  {  Distance = Math.Round(t.Distance, 6),  Latitude = t.Place.Location.X,  Longitude = t.Place.Location.Y,  Name = t.Place.Name,  RealDistance = Math.Round(t.RealDistance / 1000, 6),  }).ToList();  return View("Index", indexViewModel);  }  }  } |

1. Code for GeometryExtension: Note that we can import new coordinate systems by copying WKT (Well-Known-Text) from epsg.io and the transformation will take place automatcally

|  |
| --- |
| using NetTopologySuite;  using NetTopologySuite.Geometries;  using ProjNet;  using ProjNet.CoordinateSystems;  using ProjNet.CoordinateSystems.Transformations;  using System.Collections.Generic;  namespace VisitCalifornia.Extensions  {  static class GeometryExtensions  {  static readonly NtsGeometryServices \_geometryServices = NtsGeometryServices.Instance;  static readonly CoordinateSystemServices \_coordinateSystemServices  = new CoordinateSystemServices(  new CoordinateSystemFactory(),  new CoordinateTransformationFactory(),  new Dictionary<int, string>  {  // Coordinate systems:  // (3857 and 4326 included automatically)  // This coordinate system covers the area of our data.  // Different data requires a different coordinate system.  [4326] = @"GEOGCS[""WGS 84"",  DATUM[""WGS\_1984"",  SPHEROID[""WGS 84"",6378137,298.257223563,  AUTHORITY[""EPSG"",""7030""]],  AUTHORITY[""EPSG"",""6326""]],  PRIMEM[""Greenwich"",0,  AUTHORITY[""EPSG"",""8901""]],  UNIT[""degree"",0.0174532925199433,  AUTHORITY[""EPSG"",""9122""]],  AUTHORITY[""EPSG"",""4326""]]  ",  [2163] = @"PROJCS[""NAD83 / UTM zone 11N"",  GEOGCS[""NAD83"",  DATUM[""North\_American\_Datum\_1983"",  SPHEROID[""GRS 1980"",6378137,298.257222101,  AUTHORITY[""EPSG"",""7019""]],  TOWGS84[0,0,0,0,0,0,0],  AUTHORITY[""EPSG"",""6269""]],  PRIMEM[""Greenwich"",0,  AUTHORITY[""EPSG"",""8901""]],  UNIT[""degree"",0.0174532925199433,  AUTHORITY[""EPSG"",""9122""]],  AUTHORITY[""EPSG"",""4269""]],  PROJECTION[""Transverse\_Mercator""],  PARAMETER[""latitude\_of\_origin"",0],  PARAMETER[""central\_meridian"",-117],  PARAMETER[""scale\_factor"",0.9996],  PARAMETER[""false\_easting"",500000],  PARAMETER[""false\_northing"",0],  UNIT[""metre"",1,  AUTHORITY[""EPSG"",""9001""]],  AXIS[""Easting"",EAST],  AXIS[""Northing"",NORTH],  AUTHORITY[""EPSG"",""26911""]]"  });  public static Geometry ProjectTo(this Geometry geometry, int srid)  {  var geometryFactory = \_geometryServices.CreateGeometryFactory(srid);  var transformation = \_coordinateSystemServices.CreateTransformation(geometry.SRID, srid);  return Transform(geometry, transformation.MathTransform);  }  public static Geometry Transform(this Geometry geometry, MathTransform mathTransform)  {  geometry = geometry.Copy();  geometry.Apply(new MathTransformFilter(mathTransform));  return geometry;  }  private sealed class MathTransformFilter : ICoordinateSequenceFilter  {  private readonly MathTransform \_mathTransform;  public MathTransformFilter(MathTransform mathTransform)  => \_mathTransform = mathTransform;  public bool Done => false;  public bool GeometryChanged => true;  public void Filter(CoordinateSequence seq, int i)  {  var (x, y, z) = \_mathTransform.Transform(seq.GetX(i), seq.GetY(i), seq.GetZ(i));  seq.SetX(i, x);  seq.SetY(i, y);  seq.SetZ(i, z);  }  }  }  } |

References:

<https://epsg.io>

<https://docs.microsoft.com/en-us/ef/core/modeling/spatial#nettopologysuite-nts>

<https://medium.com/trimble-maps-engineering-blog/brief-intro-to-nettopology-in-net-core-51a944ec566b>