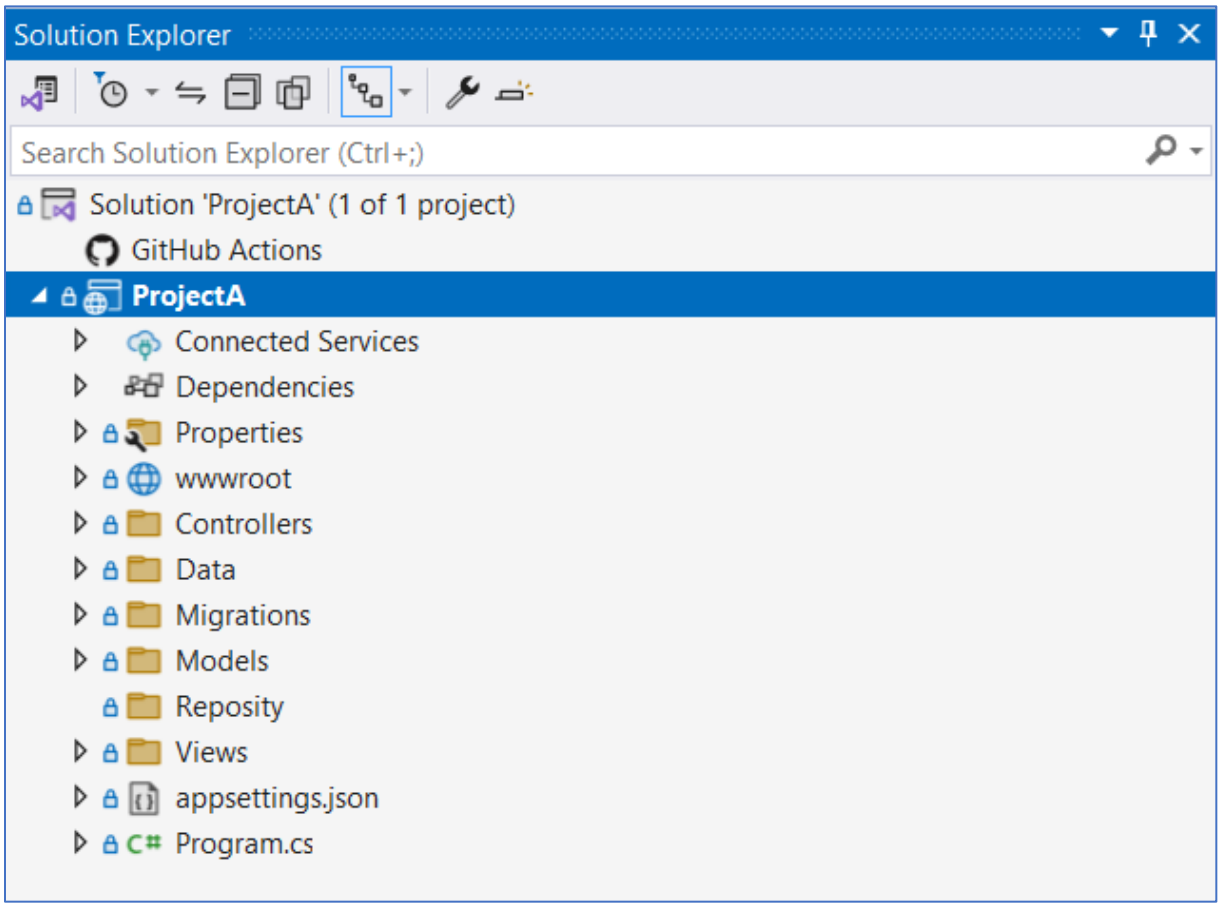
ASP.NET Core MVC Project Structure



The **Connected Services** node contains the list of external services, APIs, and other data sources. It helps in the integration with various service providers, such as Azure, AWS, Google Cloud, and third-party services like authentication providers or databases.

The **Dependencies** node contains the list of all the dependencies that your project relies on, including NuGet packages, project references, and framework dependencies. It contains two nodes, Analyzers and Frameworks.

- *Analyzers* are extensions for static code analysis. They help to enforce coding standards, identify code quality issues, and detect potential problems in code. Analyzers can be custom rules or third-party analyzers provided by NuGet packages.
- *Frameworks* node contains the target framework that the project is designed to run on.

The **Properties** node includes a launchSettings.json file which includes Visual Studio profiles of debug settings. launchSettings.json helps developers to configure the debugging and launch

profiles of their ASP.NET (also known as ASP.NET Core) applications for different environments such as development, staging, production, etc. The following is a default launchSettings.json file.

The **wwwroot** folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root all the CSS, JavaScript, and external library files should be stored here which are being reference in the HTML file.

The **Controllers, Models, and Views** folders include controller classes, model classes and cshtml or vbhtml files respectively for MVC application.

The **appsettings.json** file is a configuration file commonly used in .NET applications, including ASP.NET Core and ASP.NET 5/6, to store application-specific configuration settings and parameters. It allows developers to use JSON format for the configurations instead of code, which makes it easier to add or update settings without modifying the application's source code.
The last file **program.cs** is an entry point of an application. ASP.NET Core web application is a console application that builds and launches a web application.

CREATE CLASSES

For this project, I have total 3 controller: UserController, DeviceController, HomeController and CategoryController.

Home Controller: has some method for the home page of website.

UserController: Control the CRUD of user.

DeviceController: Control the CRUD of device

CategoryController: Control the CRUD of category.

Implement CRUD

- Create

```
// GET: Categories/Create
0 references
public IActionResult Create()
{
    return View();
}

// POST: Categories/Create
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create([Bind("Id,Name,Description")] CategoryModel category)
{
    if (ModelState.IsValid)
    {
        _context.Add(category);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(category);
}
```

- Read

```
// GET: Categories
3 references
public async Task<IActionResult> Index()
{
    return View(await _context.CategoryModel.ToListAsync());
}

// GET: Categories/Details/5
0 references
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var category = await _context.CategoryModel
        .FirstOrDefaultAsync(m => m.Id == id);
    if (category == null)
    {
        return NotFound();
    }

    return View(category);
}
```

- Update

```
// POST: Categories/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Description")] CategoryModel category)
{
    if (id != category.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(category);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!CategoryExists(category.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(category);
}
```

- Delete

```
        // GET: Categories/Delete/5
        0 references
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var category = await _context.CategoryModel
                .FirstOrDefaultAsync(m => m.Id == id);
            if (category == null)
            {
                return NotFound();
            }

            return View(category);
        }

        // POST: Categories/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var category = await _context.CategoryModel.FindAsync(id);
            if (category != null)
            {
                _context.CategoryModel.Remove(category);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
```