User Secrets

As tmpst has to use some API keys to communicate with external data sources, these keys need to be kept secure from potential malicious users as a compromised key may cause the data source to lock out our application, or in the case of the Google Maps API, cost us money by making excess calls.

Due to this, tmpst has used ASP.NET Core as the framework for its application. ASP.NET Core offers a user secret function which allows tmpst to store its sensitive API information in a file called secrets.json.

In this tutorial:

Pre-requisites:

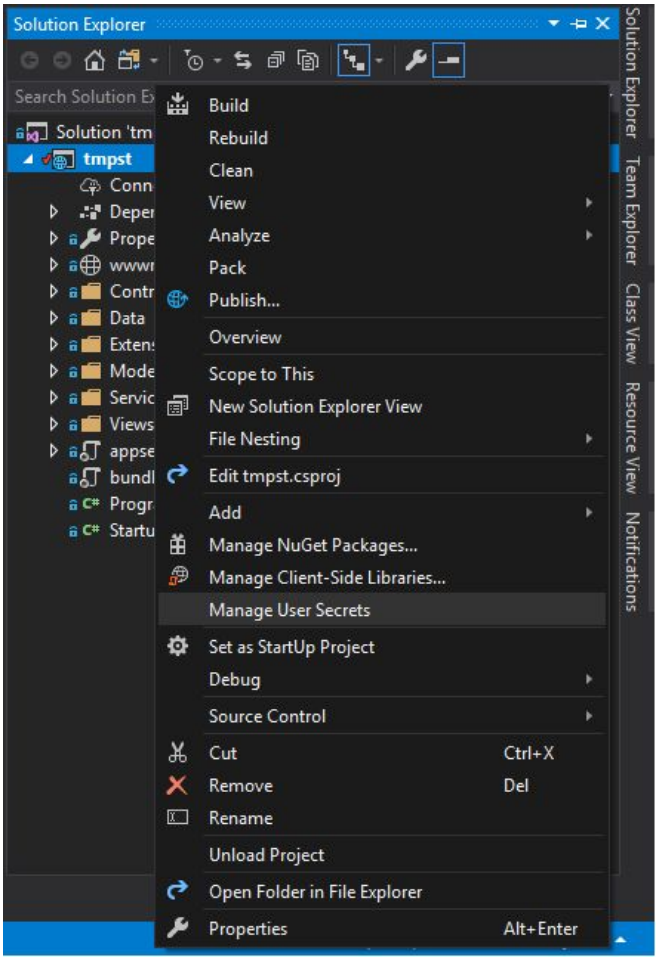
- Visual Studio 2017 (Community, Professional or Enterprise)
- .NET Core v2.2
- Secrest to store (API Keys etc.)
- ASP.NET Core Web Application

Aims:

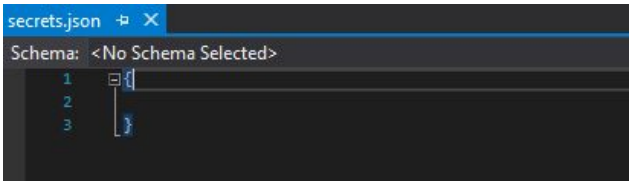
- Learn how to add user secrets to the secrets.json
- Learn how to access the secret data securely in C# and store them in a model
- Learn how to pass the secret data into a cshtml view

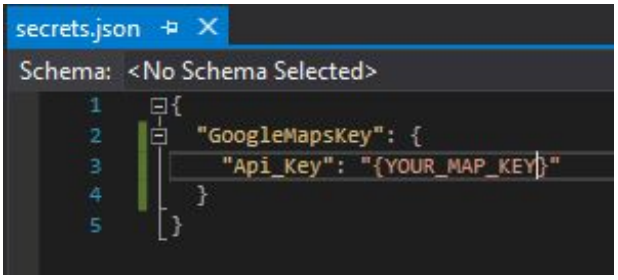
Step 1: Access secrets.JSON

Assuming your web application is already setup, right-click on the web application icon in the Visual Studio solution explorer as shown :



Step 2: Add applicable secrets to file
The image on the left shows the secrets.json as an empty file. The image on the right shows the secret file with a sample Google map secret declaration





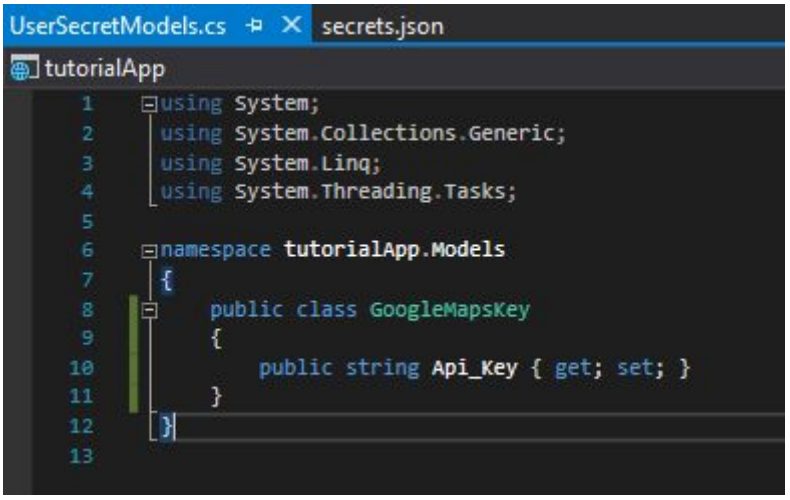
In the example above, we have Google maps as an example. You can add any secret that you want. Below is a breakdown of the secrets configuration:

- “GoogleMapsKey” - Name of the secret declaration. This can be called anything that you want it to be.
- “Api_Key” - Name of the attribute you want to store. This can also be called anything you want it to be.
- “{YOUR_MAP_KEY}” - Value of the attribute that you want to be kept secure. This value does matter. In your example, replace {YOUR_MAP_KEY} with the value you wish to store. For security, we have not included our actual Google maps key

Step 3: Create Models to access secrets
Now that you have added your secrets to the file, you will have to setup models to access the values. These models should have the same names as the ones you have used in your secrets file so that you know what secrets map to what models. Below is an example model setup using the existing Google map example:

Steps:

1. Create a new class file called UserSecretModels in the Models folder of the application
2. Remove the class UserSecretModels from the file (Not necessary, but removes unneeded model)
3. Create a new class within the file with the same name as the secret declaration (in our example GoogleMapsKey)
4. Add the attribute(s) you added to the secrets to the new class (in our example Api_Key)



The image to the left shows a GoogleMapsKey model which contains an attribute for our Google maps key.

Step 4: Add middleware to access secrets
Now that the secrets are in the json file and there are models to store the values, we need to add code to the startup class to ensure we can access these secrets at runtime:

Steps:

1. Navigate to the ConfigureServices() method of the Startup.cs. The Startup class should be located near the bottom of the solution explorer in our case
2. Add the appropriate line of code to access the secrets (shown below)

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    // Add application services.
    services.AddTransient<IEmailSender, EmailSender>();

    services.AddMvc();

    // Secrets
    services.Configure<GoogleMapsKey>(Configuration.GetSection("GoogleMapsKey"));
}
```

Above is the line we need to add to the method to ensure we can access our secrets. Below is a breakdown of the line's content:

- GoogleMapsKey - The name of the model we have created to store a section of our secrets at runtime.
- "GoogleMapsKey" - The name of the section in secrets.json that we want to store in the model.

At runtime the compiler will attempt to map the contents of the specified section of secrets.json to the stated model. In our case this will not be a problem as we have given the model and the secret section the same names.

Step 5 (Final Step): Use dependency injection to access secrets in controller
Before continuing with this step, ensure that you have the following setup:

- Section of secrets - Make sure you have added a syntactically correct section to the secrets.json with an appropriate attribute(s) (See Step 2)
- Class to contain secrets - Make sure that you have a class setup to map to your section of secrets (See Step 3)
- Middleware to process secrets at runtime - Make sure that you have written the line of code that makes the compiler process your secrets (See Step 4)

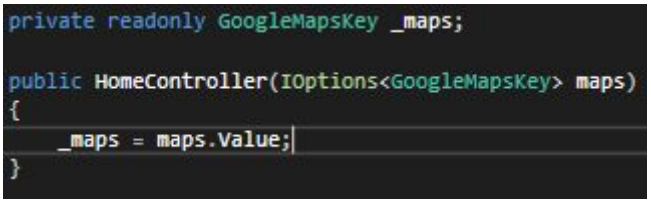
If you have completed all of the above items, then you can proceed to actually access the value of your secrets by using dependency injection

Note: For this section of the tutorial, we will be using the HomeController as our example controller. The following code will work on any class that inherits from ControllerBase, however, as tmpst is a single page application (and that page is accessed from the HomeController) we will use this as our example.

Steps:

1. Create a private readonly instance of the class which stores your secrets
2. Create a constructor of the Controller you are using
3. Add an IOptions<> instance of the model to the constructor of the controller (Name the parameter differently from the private instance created earlier)
4. Assign the value of the parameter to the private instance within the constructor

Below is an image of what the final constructor should look like. Once again for this example we are using our Google maps secret:



The private instance of the user secret model can be called anything. However, once again it would be a good idea to give this a good name - especially if your dealing with multiple secret models

Conclusion

That's it. Your user secrets can now be accessed from the private instance of the secret model and you can do what you want with them. The model can be passed to methods and views alike. By making this instance read only, we avoid any risk of mutating the contents of the class. You can access the attributes of the model b using the normal notation (_maps.). In our case the attribute we can access is Api_Key.

Common Issues:

Normally, user secrets can be setup without any mistakes However, some of the most common related to this tutorial are:

- Invalid secret syntax - This error occurs when a user has entered a secret in an incorrect format. The secrets.json file accepts json, therefore, using a JSON editor such as https://jsoneditoronline.org/ will help you format the data correctly
- Don't remove initial braces - In the secrets.json file there is a pair of { } brackets which should contain all secret data. It is common that people will remove these when adding in their secret data. If these have been removed then the secrets will not work as the file is no longer correct JSON. Make sure you have an opening { and a closing } in your secrets.json

- .Value is required - When using dependency injection to populate your private instance, you must ensure you store the .Value of the parameter in the private instance. If you do not the secrets can't be accessed and you will receive an error
- Forgetting steps - When setting up secrets it is common to forget a step in the process which will cause an error. If the tutorial does not work properly for you, then re-read the tutorial in case that is helpful

References

Although we hope this tutorial has been all the help you need, here are some useful links that may be of use:

Useful Resources:

- Microsoft User Secret Docs - https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-2.2&tabs=windows
- Why user secrets should be used - https://dotnetcore.gaprogman.com/2017/09/07/user-secrets-what-are-they-and-why-do-i-need-them/

Tools:

- JSON Editor - https://jsoneditoronline.org/