read or update values into appsetting-json

Sometimes it is necessary to update values into appsetting-json in the application(By Admin),here is implementation(Read and Write)

I used `Microsoft.Extensions.Options For Read Config` and Used `IOptionsSnapshot and System.Text.Json` For update Values

Example appsetting-json

```
"AppSettings": {
    "AppVersion": "1.2",
    "TypeLog": "**",
    "IP": "***",
    "AppId": 0,
    "AppName": ""
  },
```

Example of Settings

```
public class AppSettings
    {
       public string AppVersion { get; set; }
        public string TypeLog { get; set; }
        public string IP { get; set; }
        public long AppId { get; set; }
        public string AppName { get; set; }
    }
```

Read

```
1.Add configuration to Startup file or Program file:

services.AddOptions();
services.Configure<AppSettings>(Configuration.GetSection("AppSettings"));

2.You could also use below code to gain access to the settings values

private readonly AppSettings _settings;
public HomeController(
            IOptionsSnapshot<AppSettings> settings = null)
{
            if (settings != null) _settings = settings.Value;
}

public IActionResult Index()
{
   var _resul = _settings;
   return View();
}
```

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi

update values into appsetting-json

1.Add configuration to Startup file or Program file (For reload Appsetting After Change)

```
public Startup( Microsoft.Extensions.Hosting.IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: false,
reloadOnChange: true)
            .AddEnvironmentVariables();
            Configuration = builder.Build();
          //Configuration = configuration;
        }
```

2.Add Base Class(you need to add the basic code to the project)

3.Update AppSetting

```
 var settingsUpdater = new AppSettingsUpdater();

settingsUpdater.UpdateAppSetting(Key, value);
```

Example:

```
   public IActionResult Index()
        {

            var settingsUpdater = new AppSettingsUpdater();
            settingsUpdater.UpdateAppSetting("AppSettings:AppVersion", "1.7.5.2");

            settingsUpdater.UpdateAppSetting("AppSettings:IP", "1.07.05.2");
            settingsUpdater.UpdateAppSetting("AppSettings:AppName", "Test");
            settingsUpdater.UpdateAppSetting("AppSettings:TypeLog", "sql");
            settingsUpdater.UpdateAppSetting("AppSettings:AppId", 1014);

            return View();
        }
```

Add Base Classes:

I used of this is post(https://stackoverflow.com/a/67917167/10193401) ,customized based on my own code

```
  public class AppSettingsUpdater
    {

        Microsoft.Extensions.Hosting.IHostingEnvironment env;
        public AppSettingsUpdater(Microsoft.Extensions.Hosting.IHostingEnvironment _env)
        {
            env = _env;

        }
        public AppSettingsUpdater()
        {
```

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi

```
        }


        private const string EmptyJson = "{}";
        public void UpdateAppSetting(string key, object value)
        {
            // Empty keys "" are allowed in json by the way
            if (key == null)
            {
                throw new ArgumentException("Json property key cannot be null", nameof(key));
            }

             string settinsgFileName = $"appsettings.{env.EnvironmentName}.json";
            // We will create a new file if appsettings.json doesn't exist or was deleted
            if (!File.Exists(settinsgFileName))
            {
                File.WriteAllText(settinsgFileName, EmptyJson);
            }
            var config = File.ReadAllText(settinsgFileName);

            var updatedConfigDict = UpdateJson(key, value, config);
            // After receiving the dictionary with updated key value pair, we serialize it
back into json.
            var updatedJson = JsonSerializer.Serialize(updatedConfigDict, new
JsonSerializerOptions { WriteIndented = true });

            File.WriteAllText(settinsgFileName, updatedJson);
        }

        // This method will recursively read json segments separated by semicolon
(firstObject:nestedObject:someProperty)
        // until it reaches the desired property that needs to be updated,
        // it will update the property and return json document represented by dictonary of
dictionaries of dictionaries and so on.
        // This dictionary structure can be easily serialized back into json
        private Dictionary<string, object> UpdateJson(string key, object value, string
jsonSegment)
        {
            const char keySeparator = ':';

            var config = JsonSerializer.Deserialize<Dictionary<string, object>>(jsonSegment);
            var keyParts = key.Split(keySeparator);
            var isKeyNested = keyParts.Length > 1;
            if (isKeyNested)
            {
                var firstKeyPart = keyParts[0];
                var remainingKey = string.Join(keySeparator, keyParts.Skip(1));
                // If the key does not exist already, we will create a new key and append it
to the json
                var newJsonSegment = config.ContainsKey(firstKeyPart) && config[firstKeyPart]
!= null
                    ? config[firstKeyPart].ToString()
                    : EmptyJson;
                config[firstKeyPart] = UpdateJson(remainingKey, value, newJsonSegment);
            }
            else
            {
                config[key] = value;
            }
            return config;
        }
    }
```

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi