

## Name Sorter Application

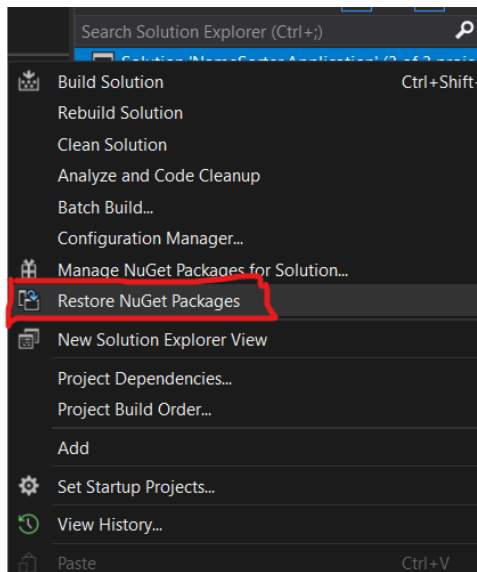
### Setting up the project

Clone the repository using the following command:

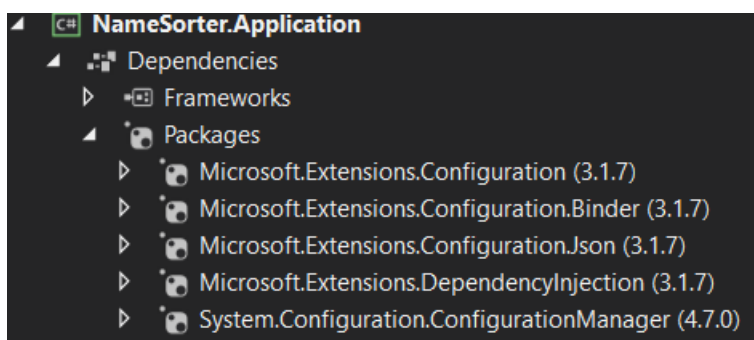
```
git clone https://github.com/adheeb-adb/NameSorter.git
```

Once the project is cloned, open it with visual studio and ensure it builds.

if the project does not build, restore NuGet packages by right clicking on the solution and selecting the 'Restore NuGet Packages' as shown in the image below.



Once NuGet packages are resolved, the following packages should be available in the 'NameSorter.Application' project.



If not automatically restored, these can be installed via the **NuGet package manager console**.

## Project Configurations

The configurations for the Name Sorter application can be found in the '**appSettings.json**' file in the '**NameSorter.Application**' project.

Under the '**NameSorterConfigurations**' section, the paths to the '**unsorted-names-list.txt**', '**sorted-names-list.txt**' and the **max allowed number of names per name** can be configured.

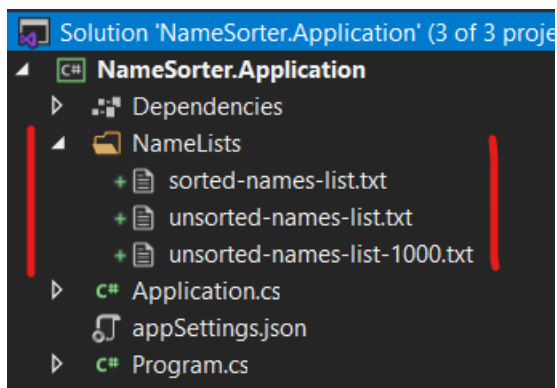
```
"NameSorterConfigurations": {  
  "UnsortedNamesFilePath": "../../../NameLists/unsorted-names-list.txt",  
  "SortedNamesFilePath": "../../../NameLists/sorted-names-list.txt",  
  "MaxAllowedNamesPerName": 4  
}
```

The '**unsorted-names-list.txt**' and '**sorted-names-list.txt**' files are located in the '**NameLists**' folder in the '**NameSorter.Application**' project.

**MaxAllowedNamesPerName** is set to 4 by default (first name, last name, 2 given names).

A name having more than the configured number of names will be considered as invalid.

In addition to the '**unsorted-names-list.txt**', '**sorted-names-list.txt**' files, the folder also contains a '**unsorted-names-list-1000.txt**' file with more than 1000 random names which can be used to test the application.



## Configurations – ConsoleTexts

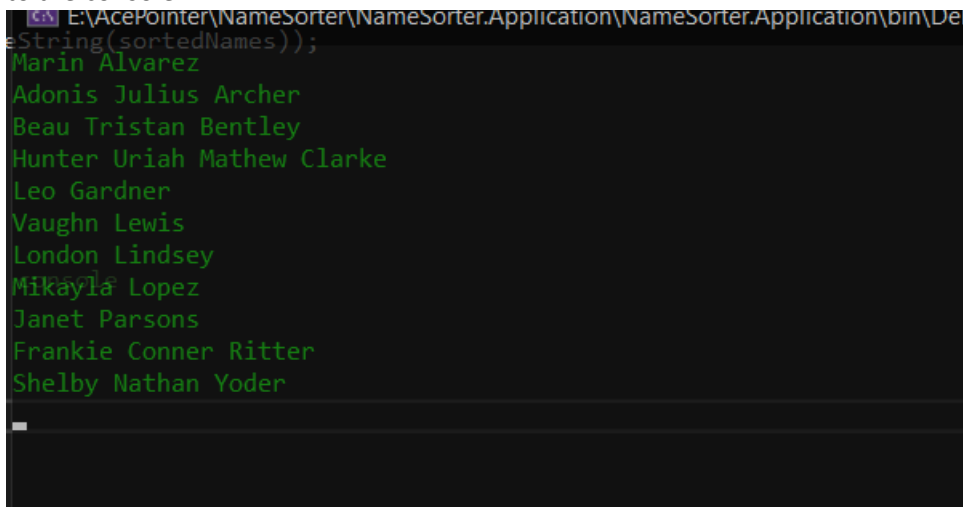
The '**appSettings.json**' also contains a section called '**ConsoleTexts**'.

All texts displayed in the console can be configured using this section.

```
"ConsoleTexts": {  
  "InputFileNotInDefaultPath": "The file does not exist at the configured path!",  
  "EnterFilePath": "Please enter the full path to the file and press 'Enter'",  
  "ExitProgram": "The program will exit"  
}
```

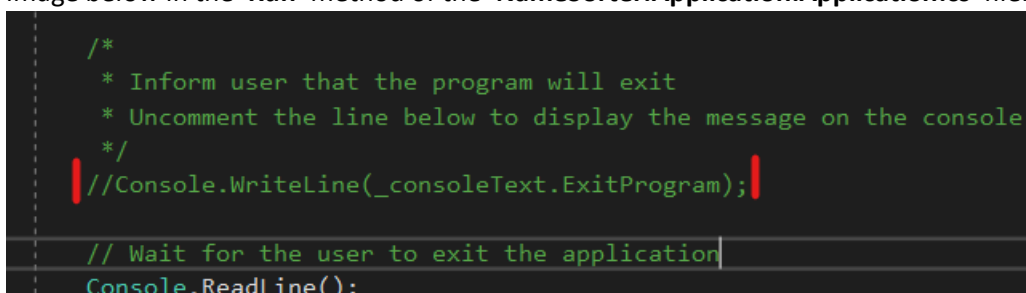
## Running the application

- If the file paths are configured correctly, the application will run and print the sorted name to the console.



```
E:\AcePointer\NameSorter\NameSorter.Application\NameSorter.Application\bin\De
String(sortedNames));
Marin Alvarez
Adonis Julius Archer
Beau Tristan Bentley
Hunter Uriah Mathew Clarke
Leo Gardner
Vaughn Lewis
London Lindsey
Mikayla Lopez
Janet Parsons
Frankie Conner Ritter
Shelby Nathan Yoder
```

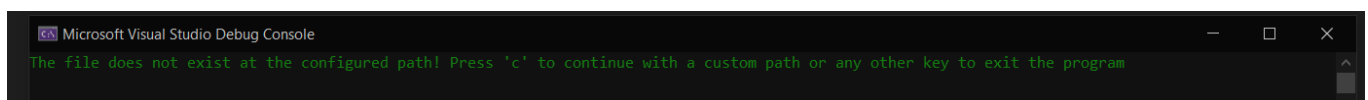
- The program will exit when the user presses the any key.
- If there are no automated validation performed on the console output, a message notifying the user to press a key can be displayed by un commenting the line of code shown in the image below in the 'Run' method of the 'NameSorter.Application.Application.cs' file.



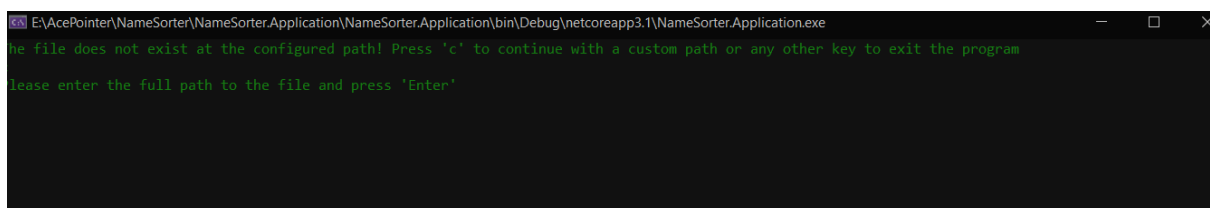
```
/*
 * Inform user that the program will exit
 * Uncomment the line below to display the message on the console
 */
//Console.WriteLine(_consoleText.ExitProgram);

// Wait for the user to exit the application
Console.ReadLine();
```

- If the program is unable to read the file and there is an exception, a message will be displayed on the console, prompting the user to press 'c' to continue and enter a valid file path, or any other key to exit.
- If the user wished to continue, the 'c' key should be pressed. Then the user will be prompted to enter a valid file path, and the unsorted names will be read from the entered path.



```
Microsoft Visual Studio Debug Console
The file does not exist at the configured path! Press 'c' to continue with a custom path or any other key to exit the program
```



```
E:\AcePointer\NameSorter\NameSorter.Application\NameSorter.Application\bin\Debug\netcoreapp3.1\NameSorter.Application.exe
The file does not exist at the configured path! Press 'c' to continue with a custom path or any other key to exit the program
Please enter the full path to the file and press 'Enter'
```

## Solution Architecture

The solution contains 3 projects:

1. NameSorter.Application
2. NameSorter.Domain
3. NameSorter.Services

### NameSorter.Application

- Program.cs:
  - Contains '**Main**' method of the console application
  - Reads configurations and registers services and configuration objects that can be injected to service classes
- Application.cs
  - Contains the code for the application flow

### NameSorter.Domain

- Contracts
  - All interfaces of the service classes are defined in this folder. Further this has sub folders based on the type of service
- Models
  - Contains the model classes

### NameSorter.Services

- Contains the services which contain the core logic and algorithms of the sorting application and helper and utility services that provide file operation and text manipulation services.

## Extending the Application with a Web API project

Though this is a simple console application, the solution can be easily extended to add a Web API project or Azure functions project (with an http triggered function for a serverless API) to expose name sorting services via an API.

Simply add a new Web API or Azure function project, copy the contents in the '**Startup**' method of the '**NameSorter.Application.Program.cs**' to the '**Startup**' method of the new project.

Next copy the contents of the 'appSettings.json' to the 'local.settings.json' file of the new project.

Once the above steps are done, the service classes can be injected and called via the controller classes (if a Web API project is added), or in the azure function.

## Extending the functionality of the Application

### Adding new sort functionality

If a new name sort functionality needs to be added, it can be done by adding a new method to the 'NameSorter.Service.Sort.NameSorterService.cs' file. The method needs to be defined in its interface at 'NameSorter.Domain.Contracts.Sort.INameSorterService.cs'.

### Adding a new service

If new service is required;

- First define its interfaces in the '**NameSorter.Domain.Contracts**' project in a suitable folder (create a new folder if required).
- Then add a new service class in the " project in a suitable folder as above.
- Once the service interface and class are defined, it needs to be registered at the project startup in the required manner (singleton, scoped etc.)

```
// Singleton services and objects
services.AddSingleton<IFileService, FileService>();
services.AddSingleton<ITextLineUtilityService, TextLineUtilitySer
services.AddSingleton(nameSorterConfigurations);
services.AddSingleton(consoleTexts);

// Transient services
services.AddTransient<IApplication, Application>();

// Scoped services
services.AddScoped<INameSorterService, NameSorterService>();
```