Content

[Characteristics of the automation object 2](#_Toc161343540)

[Brief information about the automation object 2](#_Toc161343541)

[Description of the processes to be automated 3](#_Toc161343542)

[General provisions 3](#_Toc161343543)

[General description of the development stage and its components 4](#_Toc161343544)

[System requirements 5](#_Toc161343545)

[Requirements for the system in general 5](#_Toc161343546)

[Requirements for the structure and operation of systems 5](#_Toc161343547)

[Functionality requirements 5](#_Toc161343548)

[Naming convention 6](#_Toc161343549)

[Programming style recommendations 8](#_Toc161343550)

# Characteristics of the automation object

## Brief information about the automation object

The object of automation is the user of the operating system, the function of which is to save his data. Data backup is a critical procedure to ensure the security and availability of important user information. Here are a few reasons why data backup is important for the average operating system user.

#### Protection against accidental data loss

People can accidentally delete or damage files without warning. Backup allows you to restore important data in case of loss or damage.

#### Protection against hardware malfunctions

Hard disks, flash drives, and other hardware can fail due to accidental breakdowns or failures. Backup allows you to restore data that was saved on damaged media.

#### Protection against device loss

If your device is stolen or lost, you could lose all your important data. Backing up allows you to save copies of your data elsewhere so that it's available even if your primary device is lost or stolen.

## Description of the processes to be automated

### General provisions

The program for backing up data and virtual safes automates a number of processes for convenient and secure user management. Here is a description of the processes that this program automates:

#### Automatic data backup

The user can set up automatic backup intervals, for example, every time the operating system starts. The program automatically backs up data according to the specified parameters.

#### Keyboard shortcut for quick data transferring

Users can set keyboard shortcuts to quickly transfer data to a virtual safe directly from the program interface.

#### Simplified backups organization

The program allows the user to create virtual safes to store backup copies of data, which simplifies the work with backup and the organization of this backup.

#### Logging of data backup operations

The program automatically keeps a log of data backup operations, allowing the user to track and analyze the actions performed.

### General description of the development stage and its components

The development is divided into the following stages:

#### Project development planning

During this stage, approaches will be evaluated and technologies will be selected for project development. Documentation for the future project will be drawn. Drawing up a theoretical approximate project layout. For example, to figure out the naming convention for this project, etc.

#### GUI development

The main goal of this stage is to create a GUI for the application that will not yet perform any actions related to the file backup functionality, except for the transition between Views and some basic implementations. The result should be а defined, as far as possible, GUI concept that will only need to implement the functionality.

#### Developing functionality part

Once you have the GUI, you start coding a set of methods that will implement the functioning of the GUI. The result is a set of methods for implementing the project's functionality.

#### Combining GUI and functionality part

At this stage, you need to link the GUI and functional parts by coding classes and services in the GUI that will use implementations. As a result, you should have a ready-made program.

#### Software testing

At the end of development, the program is tested and bugs are fixed. The result is a tested program that should ideally be bug-free.

# System requirements

## Requirements for the system in general

The program should provide the user with a convenient interface for managing files and the ability to select specific files and reserve them in one of the specific virtual safes. The interface of this application should be coded using the .Net MAUI Blazor Hybrid technology, because this technology allows using the Blazor Class Library to transfer Views to the website as needed, and also the ability to scale to other platforms. If necessary during development, it is worth dividing the functionality of the program into a separate C# Class Library.

## Requirements for the structure and operation of systems

### Functionality requirements

Below are the requirements for functionality

#### Virtual safe

The program should be able to create virtual safes where data can be stored and backed up. The safe can be a separate local disk, a place on a local disk, or a flash drive.

It should be possible to set up automatic data backup. For example, a file should be backed up when the operating system starts. To simplify backup, it should be possible to create a shortcut to quickly back up files.

Each action on the safes and files in them should be stored in the history, which should be available for viewing.

#### Smartphones and optical disks

The program should provide an interface that will allow you to download photos and videos from a smartphone and transfer them to one of the safes, as well as write the contents of the safes to an optical disc.

### Naming convention

Requirements for the naming convention are shown in Table 1.

Table 1. List of naming conventions.

|  |  |  |
| --- | --- | --- |
| Object | Description | Example |
| Variable | <Camel Case> | int myField; |
| Enum | <Pascal Case> + “Enum” | public enum MyEnum {} |
| Flags enum | <Pascal Case> + “Flags” | [Flags]  public enum MyFlags {} |
| Class | <Pascal Case> | public class MyClass {} |
| Struct | <Pascal Case> + “Struct” | public struct MyStruct {} |
| Class field | <Camel Case> | private int myField; |
| Class method | <Pascal Case> | public void MyMethod() {} |
| Class property | <Pascal Case> | public int MyProperty{ get; set; } |
| Property field | “\_” + <Camel Case> | private int myProperty; |
| Interface | “I” + <Pascal Case> | public interface IMyInterface{} |
| Asynchronous method | <Pascal Case> + “Async” | public async Task MyMethodAsync() |
| Event processing methods | “On” + <Pascal Case> + <Action Name> | public async Task OnMyMethodClickAsync() |
| Model | <Pascal Case> + “Model” | public class MyClassModel {} |
| Entity | <Pascal Case> + “Entity” | public class MyClassEntity {} |
| Data transferring object | <Pascal Case> + “Dto” | public class MyClassDto {} |
| Request | <Pascal Case> + “Request” | public class MyClassRequest {} |
| Repository | <Pascal Case> + “Repository” | public class MyClassRepository : IMyClassRepository {} |
| Service | <Pascal Case> + “Service” | public class MyClassService : IMyClassService {} |
| Mapper | <Pascal Case> + “Mapper” | public static class MyClassMapper {} |
| Extension | <Pascal Case> + “Extension” | public static class MyClassExtension {} |
| Command | <Pascal Case> + “Command”,  Add “Execute” part at start if it is execute method or “CanExecute” part at start if it is “canexecute” method | public class MyClassCommand : CommandBase {}  // or  ICommand OnShowHomeViewClick = new RelayCommand (ExecuteShowHomeViewCommand);  void ExecuteShowHomeViewCommand() {} |
| Controller | <Pascal Case> + “Controller” | public class MyClassController : Controller {} |
| ViewModel | <Pascal Case> + “ViewModel” | public class MyClassViewModel : ViewModelBase {} |
| View | <Pascal Case> + “View” | public partial class MyClassView : ContentView {} |
| Razor Page | <Pascal Case> + “Page” | public partial class MyClassPage : ComponentBase {} |
| Content Page | <Pascal Case> + “Page” | public partial class MyClassPage : ContentPage {} |

### Programming style recommendations

Below are some programming style recommendations. In the listings, it is advisable to replace the code with a green background with a code with a red background.

#### Number of namespaces per file

There should be only one namespace per file.

#### Number of classes per file

There should be only one class per file.

#### Using Types

Use predefined types, not framework types.

#### var VS target-typed new

If faced with a choice between using var or new()/new[] in a project, it is better to use new()/new[] as shown in Listing 1. There is no performance difference.

Listing 1. Code style recommendation.

|  |
| --- |
| var person = new PersonModel(); |
| PersonModel person = new(); |

#### “” VS string.Empty

It is recommended to use string.Empty instead of “”, as shown in Listing 2.

Listing 2. Code style recommendation.

|  |
| --- |
| var str1 = ""; |
| var str2 = string.Empty; |

#### Namespace

Since only one namespace can be used per file, we use file syntax instead of block syntax, as shown in Listing 3.

Listing 3. Code style recommendation.

|  |
| --- |
| namespace some  {  public class PersonModel  {  private string \_name = string.Empty;  public string Name  {  get => \_name;  set  {  if (!Regex.IsMatch(value, "^[a-zA-Z]+$"))  throw new ArgumentException("Invalid characters in the name");  \_name = value;  }  }  }  } |
| namespace some;  public class PersonModel  {  private string \_name = string.Empty;  public string Name  {  get => \_name;  set  {  if (!Regex.IsMatch(value, "^[a-zA-Z]+$"))  throw new ArgumentException("Invalid characters in the name");  \_name = value;  }  }  } |

#### Method group conversion

Avoid such expressions in LINQ, but generally use as shown in Listings 4 and 5.

Listing 4. Code style recommendation.

|  |
| --- |
| Action<object> writeObject2 = obj => Console.Write(obj); |
| Action<object> writeObject1 = Console.Write; |

Listing 5. Code style recommendation.

|  |
| --- |
| list.Select(Math.Abs); |
| list.Select(el => Math.Abs(el)); |

#### Switch expression

It is preferable to use the switch statement as shown in Listing 6.

Listing 6. Code style recommendation.

|  |
| --- |
| int number = 1;  string result;  switch (number)  {  case 1:  result = "One";  break;  case 2:  result = "Two";  break;  case 3:  result = "Three";  break;  default:  result = "Other";  break;  } |
| int number = 1;  string result = number switch  {  1 => "One",  2 => "Two",  3 => "Three",  \_ => "Other"  }; |

#### Index operator

It is preferable to use the index operator as shown in Listing 7.

Listing 7. Code style recommendation.

|  |
| --- |
| value[value.Length - 1] |
| value[^1] |

#### Range operator

It is preferable to use the range operator as shown in Listing 8.

Listing 8. Code style recommendation.

|  |
| --- |
| value.Substring(1, value.Length - 2) |
| value[1..^1] |

#### Tuple swap

It is preferable to exchange values without an extra variable, as shown in Listing 9.

Listing 9. Code style recommendation.

|  |
| --- |
| int a = 5;  int b = 10;  int temp = a;  a = b;  b = temp; |
| int a = 5;  int b = 10;  (a, b) = (b, a); |

#### Null check

It is preferable to check for null as shown in Listing 10.

Listing 10. Code style recommendation.

|  |
| --- |
| // 1  if (s == null)  {  throw new ArgumentNullException(nameof(s));  }  // 2  if (func != null) { func(args); }  // 3  var v = x == null ? y : x; // or  var v = x != null ? x : y;  // 4  var v == null ? null : o.ToString(); // or  var v != null ? o.ToString() : null; |
| // 1  this.s = s ?? throw new ArgumentNullException(nameof(s));  // 2  func?.Invoke(args);  // 3  var v = x ?? y;  // 4  var v = o?.ToString(); |

#### Pattern matching

It is recommended to use pattern matching to increase code readability, as shown in Listing 11.

Listing 11. Code style recommendation.

|  |
| --- |
| // 1  return num == 1 || num == 2;  // 2  if (o is int)  {  var i = (int)o;  }  // 3  var s = o as string;  if (s != null) {}  // 4  if (!(o is string s)) |
| // 1  return num is 1 or 2;  // 2  if (o is int i) {}  // 3  if (o is string s) {}  // 4  if (o is not string s) |