Hypothetically we have a code review check list in our team

|  |  |  |  |
| --- | --- | --- | --- |
|  | Metrics | Descriptions | principles |
|  | Test coverage | Is there a need to test more cases? |  |
| 1 | Abstraction |  | OOP basic principles |
| 2 | Encapsulation |  | OOP basic principles |
|  | Polymorphism |  | OOP basic principles |
|  | Inheritance |  | OOP basic principles |
|  | Association, Aggregation and Composition |  | OOP |
|  | Composition over inheritance |  | OOP |
| 1 | Single responsibility |  | Solid Principal |
|  | Open–closed |  | Solid Principal |
|  | Liskov substitution |  | Solid Principal |
|  | Interface segregation |  | Solid Principal |
|  | Dependency inversion |  | Solid Principal |
|  | Creator |  | GRASP Principles |
|  | Information Expert |  | GRASP Principles |
|  | Low Coupling |  | GRASP Principles |
|  | High Cohesion |  | GRASP Principles |
|  | Controller |  | GRASP Principles |
|  | Pure Fabrication |  | GRASP Principles |
|  | Polymorphism |  | GRASP Principles |
|  | Readability | Are there any redundant code and comments |  |
|  | **Security** | Does the code expose the system to a cyber attack |  |
|  | **Architecture** | Does the code use **encapsulation** and **modularization** to achieve separation of concerns |  |
|  | **Reusability** | Does the code use reusable components, functions, and services |  |
|  | Long Method |  | Code Smell |
|  | Large Class |  | Code Smell |
|  | Long Parameter List |  | Code Smell |
|  | Hidden dependency |  | Code Smell |
|  | Gang of four Design pattern |  |  |
|  | **Team Convention** | Code Convention, Code Preferences |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Regarding to avoid **Hidden Dependency** I suggest Constructor dependency injection for all of your class instead of Resolving Dependency by IOC container in the class constructor
2. Regarding **Single Responsibility** you can delegate the responsibility of AssignValuesFromArguments() to AppSettings object because it has already read setting from ConfigurationManager in ReadAllSettings()
3. Exception Handling Issue

**First Problem**: Environment.Exit(exitCode) in HandleError in ConsoleErrorHandler suppresses and exit the app when get called, then if you have a finally{} in upper-level or below of catch that call HandleError(), this finally{} has been never called. Because in catch you exit from app.

**Second Problem**: redundant try-catch throughout the project that they don’t do any specific business.

I would handle the general exception in Main() of Programs.cs.

1. Backup.cs class can implement IDisposable and in Despose() method check the business of File deletion. Consequentially we can delete Try Catch Finally in Run() from this file. Also we should call backup.run() in using (){} in Program.cs. that is garanteed Dispose() get called for releasing the resource and memory
2. It doesn’t need to keep AppSettings properties static, if your purpose was that keeping them alive during the app life-cycle, you have already register AppSettings as a singleton instance. From my perspective, I stay away from static classes or methods as much as possible unless in a rare situation.
3. Get(string key) in AppSettings need to check if ValuePairs.TryGetValue(key, out value); cannot find the key. Then we can have a guard that if there isn’t a key throw an exception.
4. In Download() method in SalesForceWebDownloader the baseAddress is built by some format and setting. And it isn’t only here, there are a lot of places we create some URLs and address for reaching to our third-party provider. According to single responsibility I prefer having another essence and nature which can consider it as a service (Pure Fabrication) to provide us with addresses and call it AddressProvider.
5. In LogIn() method in SalesForceWebDownloader if the login get fail we should have a guard and handle it with appropriate exception

The other point here is that, why sfClient.Url and sfClient.SessionHeaderValue are initialized, while only sessionId is returned. The scope of sfClient instance is just during the login method and it doesn’t have any effect.

1. DownloadExportFile() method in SalesForceWebDownloader doesn’t need to be static method
2. Why DownloadWebpage() method in SalesForceWebDownloader isn’t an async Method? Instead of take the Result of client.SendAsync() we can await it, Await is an asynchronous wait but result is a blocking wait.

Note: As you know we have a console app which seems for each time it gets run and take a backup then get closed at the end automatically. current code shows we don’t have business requirement for parallel processing. Then asynchronous wait maybe doesn’t have any place for discussion. But I prefer have a comprehensive approach and design through entire the app. For example instead of having DownloadExportFile() method as async and DownloadWebpage() method as sync, I prefer having both of them as Async (one signature), then finally await backup.run() in Main method in Progaram.cs.

1. Download() method in SalesForceWebDownloader instead of returning files.ToArray() can return files; consequentially in the method return signature instead of string[] would be better return Task<List<string>>.
2. I extract two addition functionality which can separate them in another methods in DownloadListOfExportFiles() method in SalesForceWebDownloader

var page = await DownloadWebpage(\_appSettings.Get(AppSettingKeys.DataExportPage), sessionId);

var matches = GetMatchingItems(page);

return GetExportFiles(matches);

1. In IUploader interface I would name the input of Upload as a filePath instead of file. Because the nature of it is path
2. In S3Uploader we have a try catch and for having the better performance I would use when in front of catch instead of check the if in Catch

1. It doesn’t need to keep ExitCode enum inside Enums class. You can take ExitCode enum out of this class and delete the Enums class. Also edit the file name from Enums.cs to Exitcode.cs

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

1. Up to now we reviewed the code as it is, for avoiding off topic discussion I assumed this code is based on our requirement and just reviewed the code. Maybe our servers are only able to execute .net framework not .net 6,7,8. But from now on as additional discussion I want to take the step forward and discuss beyond the present topic.
2. code should have a test coverage. We consider test as first client which tests the business. We consider tests as online document.
3. If commented explanation above each method or class is part of our team convention, I don’t have a problem with it, but if not, I prefer don’t having them, instead I use self-explanatory naming for each element also using unit test coverage. Then my code review start point is reviewing test scenario to understand the functionality.
4. Now is the September of 2024, if we assume that we don’t have any issue to use .net core or .net 7,8, then I prefer having a .net 8 one, it is a small project and it doesn’t take too much effort to migrate it to the .net 8 one.

Note: My-dotNet8-Refactored-SalesForceBackup project is accessible in [my GitHub by this address](https://github.com/salimian65/MySaleForceBackup) . it is WIP (work in progress) and I am working on it now.

1. We assume knowing about the advantages of .net 8.
2. We can Use the advantages of IHostBuilder also can have a BackgroundService which named it as a worker. The worker can listen to the channel or TPL Dataflow.
3. Then we can design entire of the flow as an async process. User from the UI send a request for making a new backup, then as soon as request comes to the Controller, we dispatch BackupCommand to download the files form SalesForceWebPage and put them on the channel, then respond to customer that “the backup is processing”. Also, we can use SignalR as a push notification to show a progress bar to customer. Then Worker is listening to channel and take the downloaded file one by one from the channel and try to process them and uploads them as an Uploader to Azure or AWS. When the process is finished then push a notification to the customer that it gets finished. I will elaborate it in Third Task.
4. Using HttpClientFactory instead of HttpClient for taking the advantage of httpClientPool and having better management on http request

Some Code Review Metric

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Inspection rate** | The speed at which your team reviews a specific amount of code, calculated by dividing lines of code (LoC) by number of inspection hours | **code review metrics** |
|  | Defect rate | The frequency with which you identify a defect, calculated by dividing the defect count by hours spent on inspection | **code review metrics** |
|  | **Defect density** | The number of defects you identify in a specific amount of code, calculated by dividing the defect count by thousands of lines of code (kLOC). | **code review metrics** |