Documentation

Introduction

This file includes documentation on a coding assessment in C#. The assessment covers a file reading application, with related other requirements like security & authorization.

The application has been coded with VS Code (and not Visual Studio) for the following reasons:

- requirements are not very complex
- VS Code is lightweight, easy to use, quick startup, …
- only a few packages had to be installed manually

Requirements

User Perspective

The user would like to read files of different types, where these files could possible be encrypted. Next to that, a role based authorization functionality should be included.

The user will provide the file type, the file path, whether the file is encrypted or not and his role (admin, user, …). The user would like to read multiple files without the program restarting.

The user expects that for each file, the program returns the file contents of that file.

Required functionalities

Reading files of different types

The application should be able to read txt-, JSON- and XML-files.

Reading encrypted files

The application should be able to read encrypted files. The encryption/decryption algorithm itself does not matter at this moment in time, but it is required that the algorithm can easily be replaced.

Reading files in a role based context: authorization required

The application should provide some type of role based authorization functionality.The authorization method is currently not relevant. An example would be a user being allowed to read txt-files only and an admin being allowed to read txt-, JSON-and XML-files. Again, it is required that this Authorization functionality can be replaced easily.

Practical user interface/GUI

The application should contain some type of interface, whether it be an CLI-program or a GUI.

Analysis

General

Based on the provided requirements and user stories, it is clear that more functionality might be necessary to add in the future. Some examples would be:

- reading .xlsx files
- adding more user roles/changing the authorization method
- adding different decryption algorithms
- implementing encryption functionality (instead of decryption functionality only)
- …

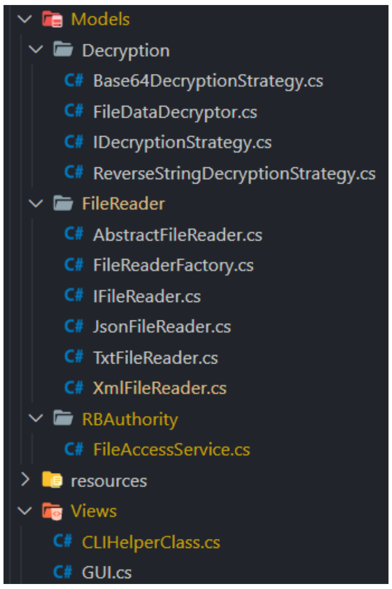
Based on the requirements and examples, and with the goal of easy maintenance and scalability of the code, the application will be based on the following principles/design patters/architecture.

Architectural Design

The application is designed based on the MVC design pattern. This pattern provides a clear structure, by separating the View (User interface), the Controller (acts as a middle man between the user interface and the business logic) and the Model (the business logic).

This MVC design pattern facilitates the maintenance of the code and ensures a more practical extension of the current functionality.

Below, the architecture is presented:



```
Models
  Decryption
    Base64DecryptionStrategy.cs
    FileDataDecryptor.cs
    IDecryptionStrategy.cs
    ReverseStringDecryptionStrategy.cs
  FileReader
    AbstractFileReader.cs
    FileReaderFactory.cs
    IFileReader.cs
    JsonFileReader.cs
    TxtFileReader.cs
    XmlFileReader.cs
  RBAuthority
    FileAccessService.cs
resources
Views
  CLIHelperClass.cs
  GUI.cs
```

Design Patterns (focused on the business logic)

File Reading Functionality: Factory Pattern

Since a user will read an unknown number of files, which requires different file readers to be created with different functionality, a factory pattern is chosen. This simplifies the creation and logic of reading different types of files. Next to that, it ensures an easy way to extend the application with new file reading functionality, like reading a .xlsx-file, by simply adding a new file reader implementation, and extending the FileReaderFactory- and FileController-class.

Decryption Functionality: Strategy Pattern

For each file, user input will be required on whether the file is encrypted or not. If it is required, the application should first decrypt the file contents and then read the file. If decryption is not required, the application should simply read the file contents.

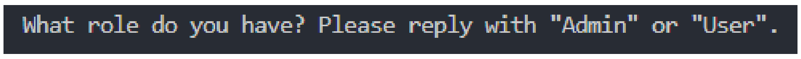
This leads to the fact that it should be easy to change this behaviour while the application is running. For that reason, a Strategy Pattern is chosen.

For the default decryption behaviour, a base64-decryption algorithm has been assigned.

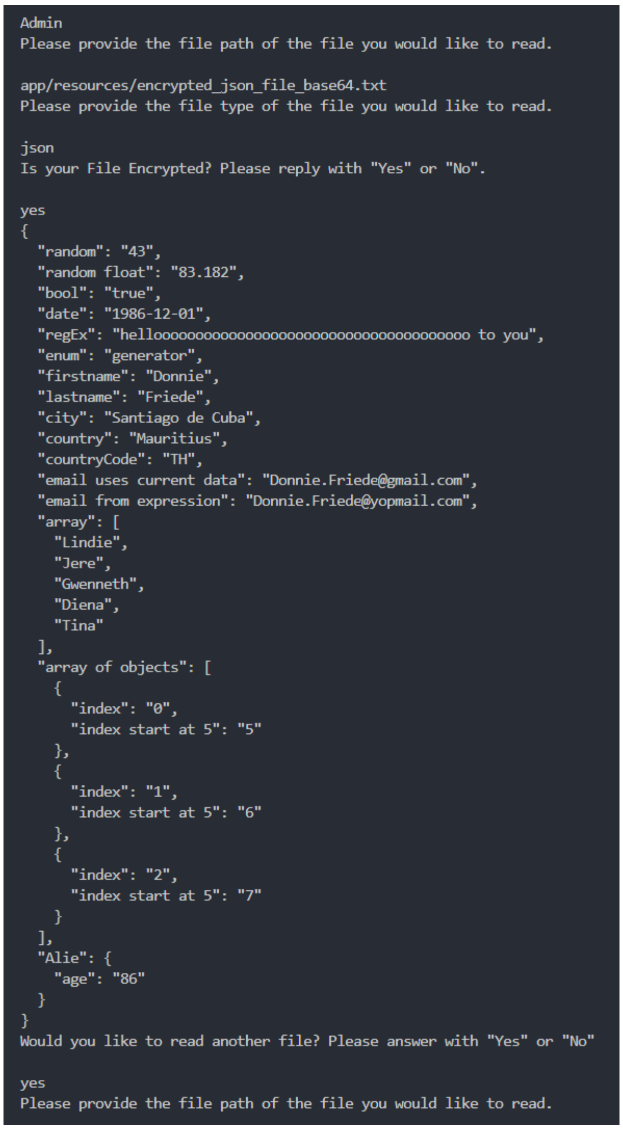
Future work/remarks:

- Improve interaction with user:
  - As-Is: user provides role, file path, file type, if file is encrypted or not
  - To-Be: user provides role, file path, if file is encrypted or not
    - if the file is encrypted, user also provides file type. This is required since encryption algorithms change the file type that the
    - if the file is not encrypted, file type is easily defined based on file path.
- Clean up output of user interaction: better formatting and spacing if questions/file content output/…
- Since decryption is so closely related to reading a file, the actual decryption is executed in the File Reader class. This means that the FileReader-objects require a FileDataDecryptor-object as parameter, with the assigned decryption algorithm. It might be a good idea to separate the decryption functionality, with a decorator pattern for example. When decryption is required, the filereader-object would be encapsulated with another FileReader object that overwrites the decryption logic, including decryption.

Example of user interaction:



```
What role do you have? Please reply with "Admin" or "User".
```



```
Admin
Please provide the file path of the file you would like to read.

app/resources/encrypted_json_file_base64.txt
Please provide the file type of the file you would like to read.

json
Is your File Encrypted? Please reply with "Yes" or "No".

yes
{
  "random": "43",
  "random float": "83.182",
  "bool": "true",
  "date": "1986-12-01",
  "regEx": "hellooooooooooooooooooooooooooooooooooooooo to you",
  "enum": "generator",
  "firstname": "Donnie",
  "lastname": "Friede",
  "city": "Santiago de Cuba",
  "country": "Mauritius",
  "countryCode": "TH",
  "email uses current data": "Donnie.Friede@gmail.com",
  "email from expression": "Donnie.Friede@yopmail.com",
  "array": [
    "Lindie",
    "Jere",
    "Gwenneth",
    "Diena",
    "Tina"
  ],
  "array of objects": [
    {
      "index": "0",
      "index start at 5": "5"
    },
    {
      "index": "1",
      "index start at 5": "6"
    },
    {
      "index": "2",
      "index start at 5": "7"
    }
  ],
  "Alie": {
    "age": "86"
  }
}
Would you like to read another file? Please answer with "Yes" or "No"

yes
Please provide the file path of the file you would like to read.
```