Allan Gray

# May 31, 2022

### Business Requirements

1. Requirements for the Allan Gray assessment are as follows:
   1. Read the included tweets.txt file.
   2. Read the included user.txt file.
   3. Output the results in the form of a twitter feed.

### Target Audience

Allan Gray hiring manager

### Technical Specs

1. Language: C#
2. Framework: .net Core 5.0
3. Unit test: xUnit
4. IDE: Visual Studio 2019

Solution

### Project 1. GA.Cli



Fig. 1 - CE.Api / CE.Cli

This project consists of the Main.cs file and theTweetManager.

The TweetManager class contains the Run method which is invoked by the Main.cs class. The Run method contains the high-level logic to read and build a user and tweet list. The display of the results is executed by the Run method, but the logic is separated into different services and classes. This should make testing easier.

### Project 2. GA.Data

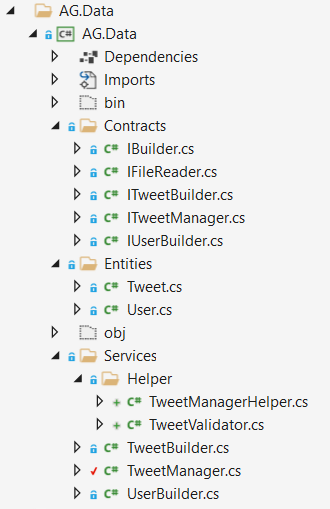


Fig 2. AG.Data

The AG.Data project contains five contracts. These interfaces define the functionality of the services declared in the Services folder. The IBuilder defines the Build function which is inherited by the ITweetBuilder and IUserBuilder interfaces. This conforms to Interface Seggregation Principle. Additional specialized functions can be defined in ITweetBuilder and IUserBuilder.

1. IUserBuilder is implemented as a service to perform the reading of the user.txt file. The actual reading is delegated to the IFileReader which is injected into the UserBuilder service during runtime. This separates the file reading logic into its own class and can easily be swopped out for another implementation. The rest of the UserBuilder service creates a list of users from the text file.
2. ITweetBuilder is implemented as a service to read the tweet.txt file. The IFileReader is injected into the services during runtime. This will read the file and tweets are generated and stored in a list.
3. IFileReader is implemented is implemented as a service used to reading a provided text file. The concrete implementation is stored in the AG.Infrastruture as this can be seen as cross-cutting concerns.

These interfaces are injected through dependency injection and makes it easier to perform testing. Implementation can be easily swopped out just by defining another concrete implementation in the DI container. The concrete classes are stored in the AG.Services. Further separation of logic is stored in the TweetManagerHelper and TweetValidator static classes declared in the Helper folder. This keeps the high-level classes clean from low level details. This conforms to the Single Responsibility Principle.

### Project 3 - AG.IoC

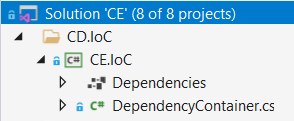


Fig 3. AG.IoC

The Di project registers all the interfaces for dependency injection. This function is normally done inside the Main.cs class but externalizing this into a separate project, it keeps the Main.cs cleaner.

### Project 4 - AG.Test

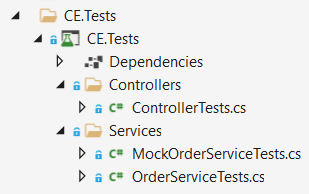


Fig 4. AG.Test

The last project is an Xunit test project for running unit tests. The application is developed using a TDD approach.