Software Design Principles

We could have all these services inherit (is a) from online service, but what if we have 20 different services? Are we going to have a class for each permutation, no we must use a good design pattern: builder design pattern for example.

Builder Design Pattern

- The builder allows for us to create objects of different use. This differs from the decorator pattern, because this pattern creates new objects. Allowing for more than one to be edited, rather they are created.

Loose Coupling

- Components interacting with each other should be independent of each other relying on the knowledge of the other components as little as possible.
- Seperation of concerns, allows for more general code and more reusability

Type Generalization

- We want to define logical abstractions which can define certain functionality which is used by concrete types
- Sort of the idea behind interfaces

```
List<string> list;
HashSet<int> HashSet;
Stack<float> stack;

// these are all able to be passed into due to type generalization
internal static void PrintAll<T>(IEnumerable<T> input){
        foreach(T element in input)
        {
                Console.WriteLine(element?.ToString());
        }
}
```