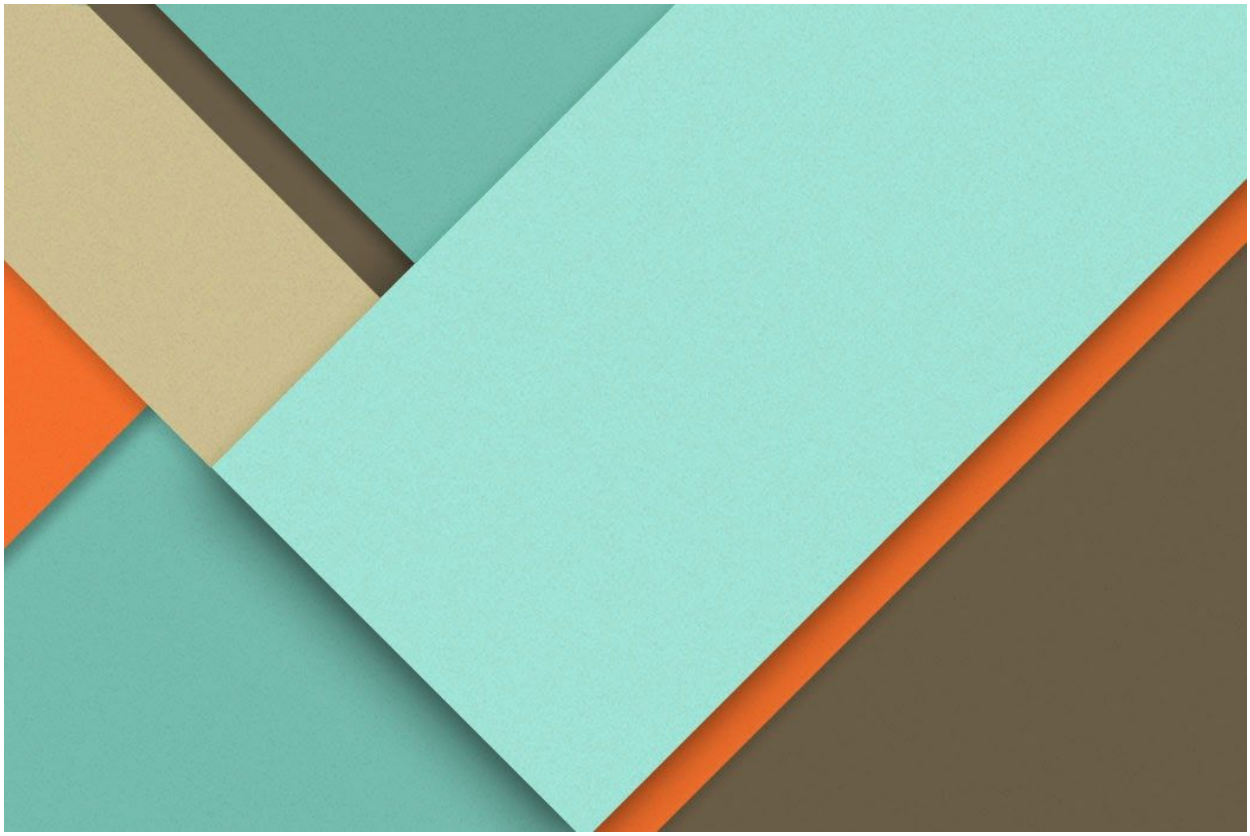


# Cook Delivery Option Finder (API rewrite and architecture)

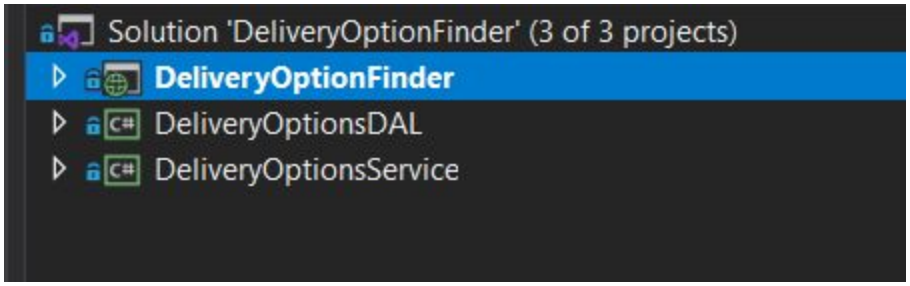
09.09.2020

Munir Syed

# Overview

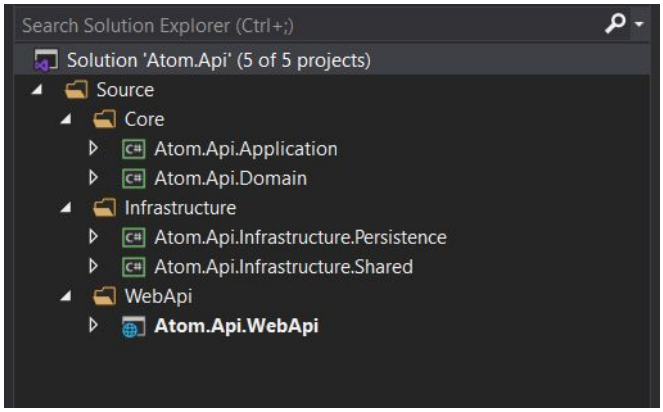
The solution proposed for the coding challenge for the Cook lead role involves a simple three layered architecture. The UI itself is an MVC .net Web UI, and we have a service layer called DeliveryOptionsService and a data access layer called DeliveryOptionsDAL.

See screenshot below.




However, this is not the approach I usually recommend for a clean architecture solution and recommend using .Net Core. This solution given here, is only to showcase very quickly what are the techniques that can be used to add flexibility and adhoc inclusions of changes, as the Business grows.

For clean architecture, I recommend the following approach that I have implemented in another solution where time wasn't of the essence. See screenshot below:




With the above structure, we are still using a layered approach to resolve the requirements, however, further split is added whereby domain and application level functionality is moved to the CORE layer, persistence and shared code moved to Infrastructure and

Then there is the client facing aspect which the client application.

Yet the hierarchy structure in the solution given here is the original layered architecture and it works fine, and many times used in production.

## The Layers

The three layers in this solution, provide decoupling and encapsulation, and each layer provides part of the logic that is needed to get the data from the JSON file (in our case) to the Web UI.

The **Data Access Layer**: Called DeliveryOptionsDAL, provides all the logic to fetch, and return a list of matching delivery options, for a given postcode passed in.

This layer is uniquely working to handle data. That is its sole purpose.

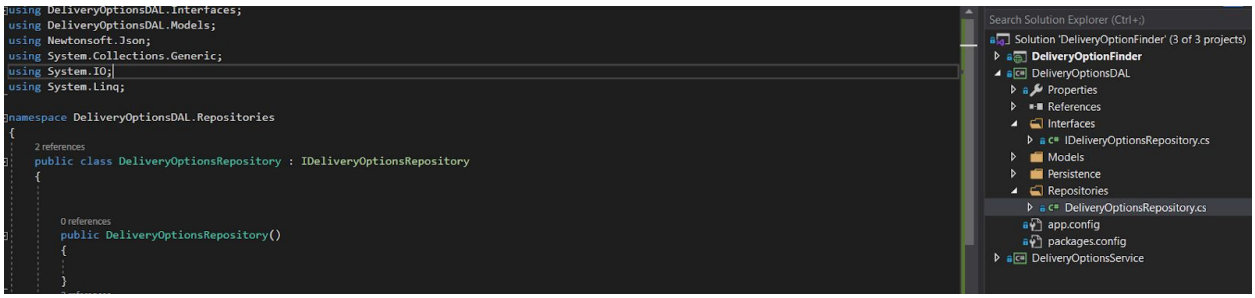
We have then the **Service** layer, which sits between any consumer of the data, and the DAL.

Of course, then we have the **UI Layer**.

## Decoupling and dependency injection

Both the service layer and data access layer implement interface based approach to build functionality, thus guaranteeing decoupling of functionality and re-usability at the same time.

The following screenshot shows in the DAL, how the interface is being used:

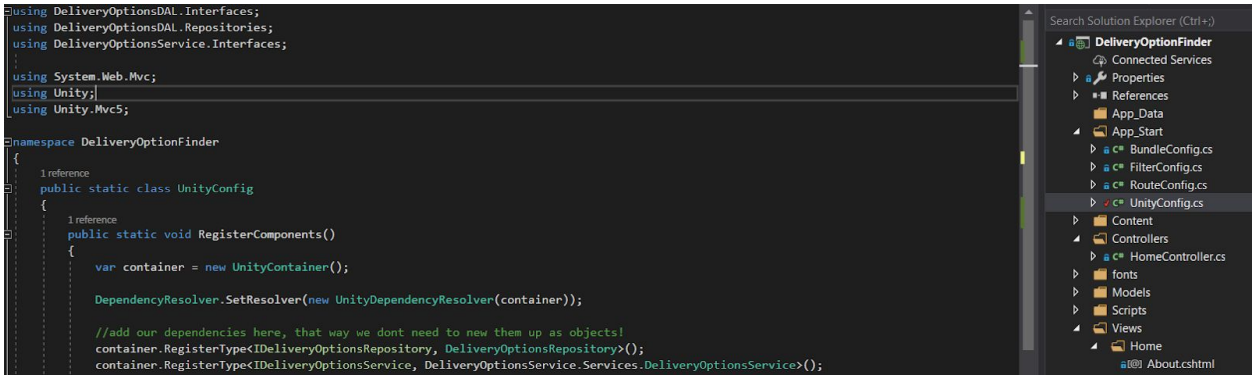



This is replicated in the service layer, and thus both layers make use of interfaces to provide decoupling

This approach enables the application layer to change the functionality without changing much of the application layer code, since the application layer now is able to use **Dependency Injection** to inject the functionality into the relevant parts of the application, mainly controllers.

## Dependency Inject Containers

The application uses a dependency injection container which is needed to inject service layer logic so that it can use the service. See screenshot below, showing the Unity Container and code to register types.



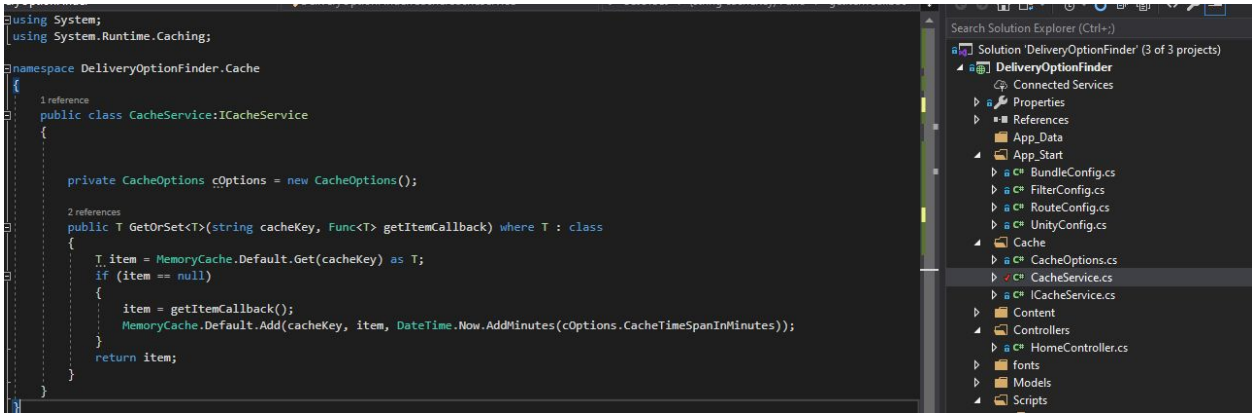

This is a key concept as the solution requested must be able to provide the ability to change the look up postcode functionality when the Business needs it. With this approach, we can register another type against the interface ***IDeliveryOptionsService*** and should be able to cope with the change with minimal impact.

## Caching

It is very important in the context of search and lookup scenarios, that frequently requested data is cached and therefore uses a simple Cache object.

The caching mechanism uses dependency injection and makes use of the Unity container to register the tye, thus the actual implementation of the caching mechanism can be altered with minimal impact.

Screenshot below shows the ICacheService implementation :




## Other recommendations

Not implemented but security and validation measure are important aspects which have been left out in this prototype. Also, going to production, suggestion would be to use a distributed caching mechanism, like Redis.

## A note on the UI

Please note the UI was implemented adhoc using the existing project template and adding Javascripts to call the page method.

No guidelines were followed in the UI, other than the DI techniques and the caching mechanism.

The project should run directly from the DeliveryOptionsFinder web project set as Startup project.

## Code

The final code can be downloaded or clone from the following git locations:

**https://github.com/Muniro/DeliveryOptionFinder.git**