ExpressVPNCoding Challenge

Project Overview

Implemented in C# / WPF
Implementation follows typical MVVM pattern making use of a well known third party library MVVMLight as scaffolding
IDE: Visual Studio 2019
Testing Framework: NUnit

Assemblies

All assemblies target .net Framework 4.8

`ExpressVPNClient`	Windows Application	Container App
		
		Project References:
		
		`ExpressVPNClientView`
		`ExpressVPNClientViewModel`

`ExpressVPNClientView`	Class Library	The View Project References: `ExpressVPNClientViewModel`
`ExpressVPNClientViewModel`	Class Library	The ViewModel Project References: `ExpressVPNClientModel`
`ExpressVPNClientModel`	Class Library	The Model Project References: `PingService`

`ExpressVPNTestLib`	Class Library	NUnit Test Library
		
		Project References:
		`ExpressVPNClientModel`
		`PingService`

`PingService`	Class Library	Self-contained reusable class lib for obtaining ping statistics using an async background task

-

Building The Application

1. Restore Packages
2. Build All

Running The Application

The executable `ExpressVPNClient.exe` is a standalone Windows 10 exe.

Configuration is possible in two ways:

Default Configuration using app.config

```
<appSettings>
    <add key="ServerLocatorURI" value="https://private-16d939-codingchallenge2020.apiary-mock.com/locations"/>
  </appSettings>
```

Edit the value of `ServerLocatorURI in app.config to point to an alternative API`

Command Line Configuration (optional)

Supply a URI to the command line to override that in `app.config`

The URI can be an HTTP API reference *or* a local file reference.

HTTP URI will be identified starting with `http:// or https://`

File references will be identified starting with `file://` and must be followed by a qualified local path.
e.g. file://d:\dev\expressvpn\testdata\mockreponse1.xml

NB: The document referenced by either web or file URI is expected to confirm the XML format laid out in the specification.

Dependency Injection

A simple DI container (`class DIContainer`) is leveraged from MVVMLight for 2 purposes:

1. To link the View and ViewModels

   For example, see the XAML declarative setting of DataContext in the VPNServersView user control

   ```
   DataContext="{Binding VPNServersVM, Source={StaticResource Locator}}">
   ```

2. To configure the concrete class used to process data. Depending on whether file or web data is being consumed, the appropriate class will be registered to implement `IRequestProcessor`

   See the constructor of `MainViewModel`

A static global instance of `DIContainer` is declared in the ResouceDictionary see App.xaml, and thus is accessible throughout the application, from XAML and in code.

Testing The Application

In Visual Studio : Test > Run All Tests

The unit tests can be found in

ExpressVPNTestLib\WebApiTests

Specifically these unit tests aim to test the behaviour of `XMLWebRequestProcessor` independently of other classes \ components.

Note: the `Process` method in `IRequestProcessor` takes an optional argument of `WebRequestFactory`.

This enables a lightweight system of web API mocking whereby the HTTP request and response can be simulated rather than making a genuine HTTP request.

ExpressVPNTestLib\ModelUnitTests.cs

These tests are at a higher level and test the composite behaviour of the `ServerModel` instance end to end.

ExpressVPNTestLib\PingServiceTests.cs

Tests the ping service independently

Threading

The ping service operates on a background thread. This does not block the UI but call-backs from this thread do update the UI. This feature was not in the project spec but useful from a UI perspective and for debugging.

The ping service is shutdown cleanly before application termination using a cancellation token.

The refresh operation and the ping service do update elements of a shared data structure. Appropriate locking with monitor is in place around these sections.