A high level description of the underlaying model class design can can be found in Section. **??**. This section will focus on the code required to get the model running and linked to the gui.

0.1 Libraries and Frameworks

- **Windows Forms** - the graphical library, this library is very accessible and can be picked up at a very rapid pace so its almost ideal for a first time project using a GUI in C#.
- **AngleSharp** - A DOM parsing library, this allows us to access the DOM using a logical class structure greatly simplifying accessing specific DOM elements when necessary.
- **Microsoft Visual Studio Unit Testing Framework** - This framework has been used to write the unit tests, it has built in integration to the IDE used for developing this software: Visual Studio 2019

0.2 Interacting with the model

Assuming you have representations of the GUI elements described in Section. **??**, first insantiate the **History** & **Favourites** classes so you can assign their event handlers. See Fig. 1

```
1          public partial class ExampleBrowser
2          {
3              private Favourites fav;
4              private Histroy hist;
5              public ExampleBrowser()
6              {
7                  fav = Favourites.Instance;
8                  hist = History.Instance;
9                  fav.EntryChanged += Favourites_Changed;
10                 hist.EntryChanged += History_Changed;
11                 // Deserialize any Favourites or History elements stored locally
12                 fav.DeserializeCollection();
13                 hist.DeserializeCollection();
14             }
15             private void Favourites_Changed(object sender, EntryRecordChanged e)
16             { /* Update Favourites gui elements */}
17             private void History_Changed(object sender, EntryRecordChanged e)
18             { /* Update History gui elements */}
19         }
```

Figure 1: Get the history and favourites instances and assign event handlers

Now that we have our History and Favourites instances we can instantiate a **PageContent** object. See Fig. 2

```
public partial class ExampleBrowser
{
    private PageContent content;
    // This method is called by a gui element event that is triggered
    // upon first loading the gui
    private async void ExampleBrowser_Load(object sender, EventArgs e)
    {
        content = await PageContent.AsyncCreate(fav.HomeUrl, hist, fav);
        content.ContextChanged += content_OnContextChanged;
        // at this point we can use the properties of content to assign
        // values to gui elements.
    }
    private void content_OnContextChanged(
        object sender,
        ContextChangedEventArgs e)
    { /* Update all gui elements realated to the current web page here */ }
}
```

Figure 2: Async instantiate PageContent with the Home URL

At this point you just need to use the public methods in the **PageContent**, **History**, and **Favourites** classes to update the view, and to send control commands to the model. See Fig.**??** for a description of the **PageContent** class important public methods and their signatures. Likewise see Fig. **??** for a non-exhaustive summary of useful public **History**/**Favourites** methods.

0.3 GUI elements

This solution uses 4 Windows Forms classes, **BrowserWindow**, **EntryCollectionEditor**, **EntryEditor**, and **FavouritesDialogue**.

BrowserWindow is the class representing the actual browser itself, all the GUI components for this class but the dropdown menu are visible in Fig. **??**. Once the basic shape of the Windows forms controls has been defined its just a matter of linking up all the events with appropriate methods called within the **PageContent**, **Favourites** and **History** classes.

EntryCollectionEditor holds a reference to the **EntryRecord** that contains the List being painted onto the ListView that way any change made to the underlaying source List means that the

source List can be used to repaint the ListViewContents. The rest is mostly event handlers that trigger behaviour based on button presses, selection changes ect.

EntryEditor is a very simple component, it is comprised of 2 text input boxes corresponging to the title and url of the entry. If the user makes a change and presses ok, the underlaying element gets mutated an event gets triggered and the window closes.

FavouritesDialogue has very similar appearence to **EntryEditor** but it doesnt use any event handlers. it simply allows the user to instantiate a new EntryElement for the Favourites object.