Course code and name:	**F20SC Industrial Programming**
Type of assessment:	**Individual**
Coursework Title:	**CW1 Browser**
Student Name:	**Kate Levi**
Student ID Number:	H00347035

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature: Kate Levi

Date: 22/10/2022

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

F20SC CW 1

H00347035

Introducion

The goal of the project is to create a simple web browser. A simple web browser needs the following objective:

- Browse pages
- Track History
- Page navagation (Refresh, back page, forward page)
- Error handling (Do not want your browser to crash when a page does not have a title!)
- A user interface to interact with It was decided to use gtk# (Mono) + Glade# to achieve the goals, this allows us to use the c# core in any platform, current c# version with mono is .Net 6 The application is designed to work on any computer system (Unix,Unix-Like, and windows). Development is completed on Windows and Macos.
  This report will cover Design Considrations, Developer Guide, User Guide, Testing and conclusions.
  Design Considrations will cover justifications of the project structure (Taking into account performance, Extendability, and general approuch).
  Developer Guide will explain in detail how the program works, with a guide of how to approuch the system.
  User Guide will be a overview of what is the program for and how to use it as a general user (What do the menus and buttons do!).
  Testing will cover how the system was tested to ensure reliability.
  Conclusions will include reflections and critical anlysis of the project.

Requirments Checklist

Requirment	Done	Notes
Sending HTTP request messages for URLs typed by the user	Yes	Can be done through the URL bar and either press enter or the "go" button, calls `urlBar_icon_press` or `urlBar_activate`
Receiving HTTP response messages	Yes	Completed via the HTTP class and display function in class TabWindow
Http status code handling	Yes	Expection handling in `loadUrl(string url)` TabWindow class
Display HTTP response code	Yes	`Display` function in class TabWindow
Display HTTP response status code, title of web page.	Yes	Done via getTitle in class TabWindow
Reload current page, display contents, title and response status code.	Yes	Done via refresh button UI element, calls `loadUrl`, `display` in TabWindow
Home Page	Yes	Loads home page on start with `onDraw` in TabWindow class, options are loaded on start in function `Main` in class Program
Home Page edit	Yes	Done via UI options button and the following classes: `Options` (`map.update`) and OptionsWindow (`onSave`).
Favourites set alias	Yes	Done via UI add button and 2 text entry elements, calls `Faviortes` (`map.add`) and `FavoritesWindow` (`addNodeUrl`).
Favourites modification	Yes	Done via a UI button and 2 text entry elements, calls `Faviortes` (`map.update`) and `FavoritesWindow` (`editNodeFav`).
Favourites Load on start	Yes	Loads from DB to cache (`map.reload()`) within `Main` in class program.
Favourites visit	Yes	Double click to visit (`visitUrl`)
History	Yes	Similar methdology to Favourites, with a difference of History containing a time stamp instead of alias (and less UI and code features).
Bulk	Part	`loadUrl` provides body (as a string), url, and status code. UI is missing for Bulk.

Design Considrations

Enviorment

It was chosen to use Mono for the compiler, as it provides a cross platform compiler (versus microsoft's c# compiler which only runs on windows). Mono implmeants c# version 6.

It was decided to use glade# for the UI, as it provides a quick and performant way to develop UIs.

Glade uses a MVC (Model View Controller) model. MVC is a popular structure for UIs, it follows the following "Model" as models codes as UI elements, "View" All the view functions that interfact with the user, and "Controller" which is a middle man between the model and view.

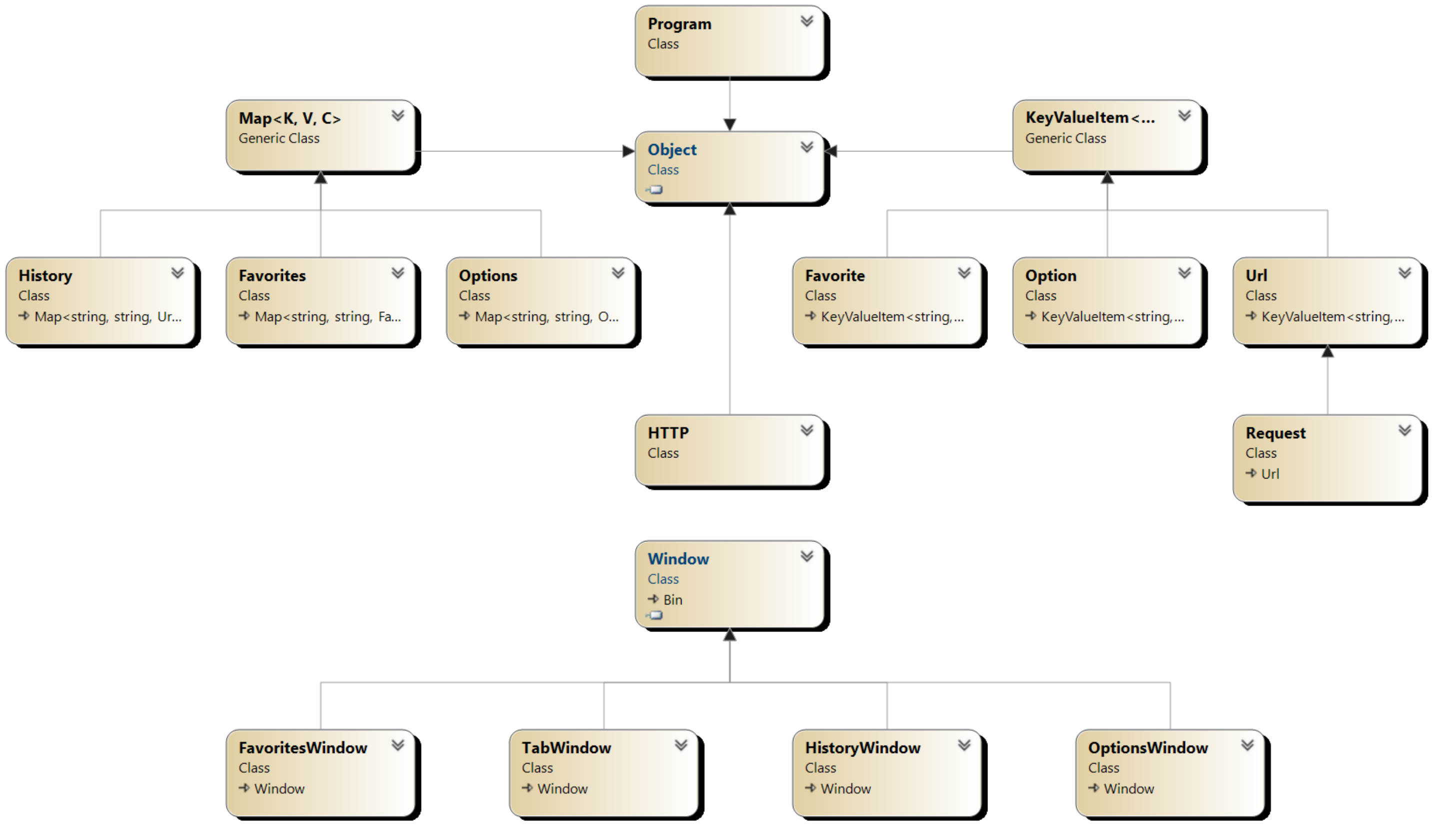
For interacting with sqlite, it was chosen to use the entity framework core (EF core), EF core is a .NET Standard 2.0library, and is part of the .Net core.

EF core provides a datbase access based on "Sets" and each set has it's own unique data type. EF core, can create a database model and gives you direct access to the sets; meaning you can manipulate data quickly, without clutter (when compared to using sqlite directly, on its own). This is because it's all model based, drastically making code more readable and maintainable.

Linq was chosen to query to sqlite and EF core, Linq allows simple syntex yet very powerful. Thus omitting the need to manually write SQL quries. Simplyfing maintainability and readability of code.

Class Diagram

The current diagram, split between UI and non UI.



HTTP class

Initiates a http client. Its request function takes a url(string) and returns a Request object.

Request class

Contains http client, a HTTP response. its get functions sets its class's response.

Class KeyValueItem

The class is a generic class providing a geneic key and a generic value. The key is also the primary key for DB purposes. It contains a copy constructor, an empty constructor and a setter constructor. The empty constructor is there so generic classes can set it's key and value.

Class Favorite

Inherted from KeyValueItem of type <string,string> where the key is the Url and value is Alias

Class Option

Inherted from KeyValueItem of type <string,string> where the key is the Name and value is Value

Class Url

Inherted from KeyValueItem of type <string,string> where the key is unix time and value is Url

Class Map

Generic class which provides cache and DB interface for a KeyValueItem class. Provides Add, Update, and Delete opreations for both the datbase and the internal map (cache).
dictionary was chosen as it's very fast for getting the values with a search value of O(1) a and write value of O(1). Making it ideal for browser usage, speed will stay the same regardless of the amount of items you have stored.
All DB operations use the EF core to get, update, delete, or add item(s) to/from tables.

Class History

Inherted from map, with a type of Url.

Class Favorites

Inherted from map, with a type of Favorite.

Class Options

Inherted from map, with a type of Option.
Adds a defaultOptions function, which loads default options to cache and DB through `map.add()`

Class Program

Contains the main functions, inits classes and DB. (Loads data to or from DB on startup)
Also inits the GUI.

Window classes

Window classes hold UI elements and functions to drive those UI elements. When a UI element is triggered, a singal is emitted and calls predefined function for the signal.
This allows the program to setup the UI before it's showed to the user, creating a very seemless experince.

TabWindow Class

Holds element htmlBox to present data, such as raw html.
Also holds a cache for backwards/forwards: holding the request with it's response, meaning when a user goes back and forfth there is 0 delay and saves us from resending requests.
Refreshing and new pages work as expected with cache (refreshing will refresh the page and new pages will overwrite previous forwards).

This cache is not saved because of EF not allowing custom class values to be inserted to DB. Additionally the cache would grew quite big on scale so it does not make sense to store in permemant memory.

Class HistoryWindow

Holds a UI table of a url and a timestamp of the url, interacted via double click, selection, and buttons.

Class OptionsWindow

Holds a labal a text entry of home page, and a save button, text entry is loaded from cache on draw. Save buttons updates the cache, DB, and closes the options window.

Class Favorites

Similar to history, holds 2 additional buttons (add and save) and 2 text entries (url and alias).

Class Structure Conclusion

The classes are structed around abstraction, with a heavy focus on generics and loosy coupling.
This done to ease reuseability of classes and make maintance easier.

Where the GUI is tightly coupled, this is as all windows rely on the TabWindow in order to load urls. This is needed as otherwise it's very hard to access other windows' information.
For example: Visting a page from history requires access to TabWindow.
This heavily reduces maintainbility, reusability, and extendability.

Data Structures

Using a dictionary (an implmeantion of a hastable) for storing map instances has a drawback: It has to be loaded after startup. Which includes sorting which will take O(n log(n)) long. To solve this we can use SortedDictionary. However it will slow down searching, inserting, and deleting to O(logN). Meaning our applications will not scale well if we were to use SortedDictionary.

Therefore it was chosen to stick with a dictionary based map class as the application will always feel fast after startup. This increases start time of the application, which its affects are hidden by hiding the app until data is loaded.

User Guide

Running The Application

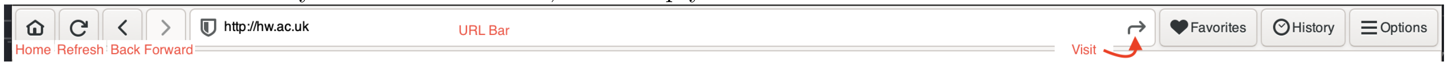
Only windows binaries are provided, so users of other systems would need to compile from source.

Windows

Requires gtk3, version 3.24.24 downloadable from here:
https://github.com/GtkSharp/Dependencies/raw/master/gtk-3.24.24.zip
Extract to `%LOCALAPPDATA%\Gtk\3.24.24`. Ensure the folders Gtk and subfolder 3.24.24 exist. Then extract your "win-x64" to your chosen folder. Now you can run the app by entring the folder and double click "F20SC_CW1.exe".

Tab Window

This is the main window you will interact with, in the top you will see a tool bar:



To the left most you have your Home Page, which goes to your home page, on default it loads "http://hw.ac.uk". The home page is loaded on startup, you can modify the website in the options menu to the right most.

In the middle of the tool bar you will use a url bar, which will show the current page you are on.
To visit a different website click on the URL and delete the conents. Replace the content with your chosen website, either click enter or press the go button to the right most of the URL Bar.

Important note: URLs must either start with "http://" or "https://" no other protocol is supported!
Additionally options will let you set any URL, therefore make sure it's valid before changing to it.

To the left of the URL bar you will see 3 Buttons: Refresh, Backwards, And Forwards.
Refresh reloads the current website. Backwards goes back to a previous visited page in the session, which is only enabled if you visited a new page. Forwards goes to a newly visited page in the session, only enabled is you went backwards.

To the right of the URL bar you will see 3 Buttons: Favorites, History, and Options.
Favorites will show a window to add aliases to your favorite website, allowing you to keep track of them. History shows a window with all of your previously visited websites. For the options window refer to either the start of this section or to the Options Window subsection.

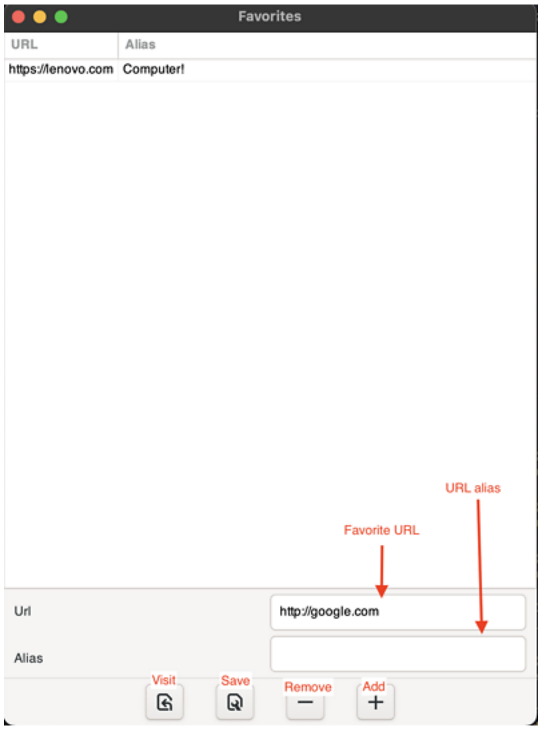
HtmlBox

Below the URL bar there is a box, any errors or website content will be displayed there.

Title

The titel of the window bar displays the html status code and if there is a title it will so it too.

Windows Overview



Above is the Favorites window, which has more features than History window (but very similar. You can visit a website by double clicking on its row in the table. When selecting a website by clicking on its row once you can interact with the text entry to edit the website's alias and interact with the buttons below the text entries.

Options contains the home page text entry, which when modified and saved will change the home page. After saving the options page will exit.

Developer Guide

Explains how navigate the code base and how to use the the core classes, which will let you, the deverloper the ability to extend the code base.

Exploring the code base

A design rational and overview is covered in "Class Diagram", to understand how the code base ties together please give that subsection a read.
Information about dependencies can be found in the following:
Mono,gtk#,Glade, EF Core

Comments

The code base is commented where there is more complexity.
For example a simple code like:

```
public List<Option> loadDefaults()
{

    List<Option> defaults = new List<Option>() {
            new Option("homePage","http://hw.ac.uk")
            };
    return this.add(defaults);
}
```

Does not need to be explained, as the naming is enough to explain what the code does.
For example here it's clear that the functions loads default options and returns those added options.

Where in more complex codes, commenting is very needed. For example:

```
public void checkNavButtons()
{
    if (this.index == 0) // newest page
    {
        goForward.Sensitive = false;
    }
    ...
    if (UrlWindowHistory.Count < 1) // If zero pages are loaded
    {
    ...
    }
    else if (Math.Abs(this.index) + 1 == UrlWindowHistory.Count) // Check if we are on the oldest page
    else // We are in between newest and oldest
    ...
}
```

As shown here, more complex statements where developers can get confused are explained. Additionally the vast majoirty of functions use XML comments.
Example:

```
/// <summary>
/// method  <c>checkNavButtons</c> Tracks index location in cache; disables and enables UI elements in rel
/// </summary>
```

Explaining exactly what the function does.

When extending or modifying the code, use xml comments in order to keep the code base understandable.

Http requests and responses

The file "Http.cs" contains two class "HTTP" and "Request".

HTTP class

Initiates a http client.

Request function

Takes a string and returns a Request object.
Sends a request through the http client, saves the response, ensures the request was successful (if not throws error) and returns a Request object.

Request class

Contains http client, a HTTP response

Get function

sets the class's HTTP response through the http client.

Things to keep in mind

DB save operation

EF Core is not built with multiple saves at the same time, and as a result any database save operation can go wrong, with two ways of dealing with such issue. You can either override the cache or the DB. Meaning either the DB "wins" or the DB client. This project defaults to the DB client "winning" as it's important the newest information gets saved, to avoid data loss. More information Here.

Exceptions

Custom exceptions are used heavily within this project, when calling for a function ensure handling of the correct exceptions. Custom exceptions are in the "Exceptions.cs" file.

Only One HttpClient

HttpClient is meant to have one instance per application. Therefore in the project it exists within the HTTP class, with it's instance being created in `Main` within the Program class. It is then passed when needed. More information: Here

Testing

Test	Argument	Expected results	Actual Results	Notes
loadUrl	http://hw.ac.uk	Html to display, and title change	Html to display, and title change	As expected
loadUrl	no://test.com	to display Error in htmlBox	displayed Error in html box	As expected
loadUrl	http://httpstat.us/404	title to show 404 and htmlbox to say not found	title 404 and htmlbox not found	As expected
loadUrl	http://httpstat.us/400	title 400 and htmlbox bad request	title 400 and htmlbox bad request	As expected
loadUrl	http://httpstat.us/403	title 403 and htmlbox forbidden	title 403 and htmlbox forbidden	As expected
loadUrl	https://was.sdsd.ds	Dns error	Dns error	As expected
history	remove when empty	nothing	console error log (no crash)	Not as expected
history	visit when empty	nothing	nothing	As expected

Many more tests were done in development, hence you will see proper exception handling in most (if not all) functions.

Class map, function add: covers if saving from DB fails with 2 corner cases (concurrency and deletion of DB after startup).

```
catch (ArgumentNullException e)
    Console.WriteLine(e.Message); // Null argument
catch (DbUpdateConcurrencyException e)
{
    // Update original values from the database
    var entry = e.Entries.Single();
    entry.OriginalValues.SetValues(entry.GetDatabaseValues());
}
catch (DbUpdateException e)
    Console.WriteLine(e.Message); // Unable to save to DB
```

Similar tests are done throughout the applications.

Refelection

A few things could have been implmeanted better when looking back on the project, although as a whole I'd mark the project as a success. However, can we all reflect to be better, so here are the main issues with the project:

Tight UI Coupling

The biggest issue with the current project is the structure of window classes, every single window class relies on TabWindow. Which means if it's changed then other classes needs to be changed. Thus leading to lower maintainability and readability of code.

Comments

XML comments are used but are quite lacking in terms of detail, they should be to a higher standard.

Database model

EF code is model based, meaning along with Linq fetching, updating, deleting, and adding items to database is a breeze. As EF core supports Linq using it's syntax was simple.

However using EF Core limits you to primitive, string, and datetime types.
Meaning you can not easily store complex or custom classes with EF Core.

Advanced language features

Using collections allows us to use object of a the same class together and allows us to iterate through all elements of the collection. Therefore collections (dictionary, list) were used in Map(map) and TabWindow(Cache)

Using generics, different types can be used with the same class, extending the usability of the class. Therefore generics were used in Map<K,V,C> and KeyValueItem<K,V>

Using indexes it's easier for developers to use your classes, as there are no functions to be called in order to get or set an item. Used in map to quickly get item either buy key or index.

Using custom exceptions allows us to deal with issues in advance. Thus it's key when implmeaning new classes. Therefore created in "Exceptions.cs" and used mainly in map.

Languages

Using c# is ideal for a browser, as it's compiled language. Meaning it does not have to go through a "middleware"(interpreter) as that would slow down any operation done by the browser. As browsers are expected to be performant using a scripting language (a language based on an interpreter) would be less ideal.

Conclusion

Non UI code is what I am most proud of in the project, as with the usages of generics, EF core, indexes, and other features of c# I have used in this project. Which made the project a lot more readable and pleasant to build. This is my first big project with c#. Cross platform gui development is a lot less documented compared to JavaFX (library my last GUI Java project used). However my experince with that project had helped me a lot, as JavaFX also uses a MCV model.

If I had time I would have implmeanted concurrency to make the applications multi-threaded, so tasks such as saving to DB can be done in the background which would improve the feel of the application. Additionally I would want to add unit testing to the application.

Refrences

https://github.com/mono/gtk-sharp/blob/main/sample/GtkDemo/DemoListStore.cs
https://www.bigocheatsheet.com
https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-6.0
https://learn.microsoft.com/en-us/dotnet/core/rid-catalog
https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-build
https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/generics
https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/indexers/using-indexers
https://zetcode.com/gui/gtksharp/
https://learn.microsoft.com/en-us/ef/ef6/saving/concurrency
https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/
https://www.mono-project.com/docs/about-mono/languages/csharp/
https://www.mono-project.com/docs/gui/gtksharp/beginners-guide/
https://help.gnome.org/users/glade/stable/
https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?source=recommendations&tabs=netcore-cli