**Industrial Programming**

**F20SC**

Coursework 1 – Web Browser

By Abu Saed Sayem

ID - H00144207

**Table of Content**

1. Introduction.………………………………………………….3

2. Assumptions..………………………………………….……..3

3. Requirements checklist…………………………………...….3

4. Design Consideration………………………………………...4

5. User Guide………………………………………………….10

6. Developer’s guide…………………………………………..16

7. Testing………………………………………………………22

8. Conclusion………………………………………………….34

9. References………………………………………………….34

**Introduction**

The purpose of this coursework is to implement a simple web browser using the C# programming language. The web browser developed can send HTTP request messages, receive HTTP response messages and displaying it in a RichTextBox, display HTTP error messages received (if any), home page button, favourites option, URL history and use multi-threading for browser-server communications. The purpose of this report is to guide both the user and developers. Users will find guidelines on how to use the browser. The developers will find guidelines and code snippets in order to understand the structure of the implementation.

**Assumptions**

Assumption 1: [www.google.com](http://www.google.com) is the default homepage and this site will be loaded by the web browser after initial startup.

Assumption 2: when the user clicks on the ‘homepage’ button, the current URL is going to be the new homepage. In the next web browser start up, this new URL will be loaded as a homepage.

Assumption 3: When a user clicks on a link in History or Favorites, the URL will be loaded onto the current tab.

Assumption 4: Correct URL is typed by the user and incorrect URL is not handled by the code.

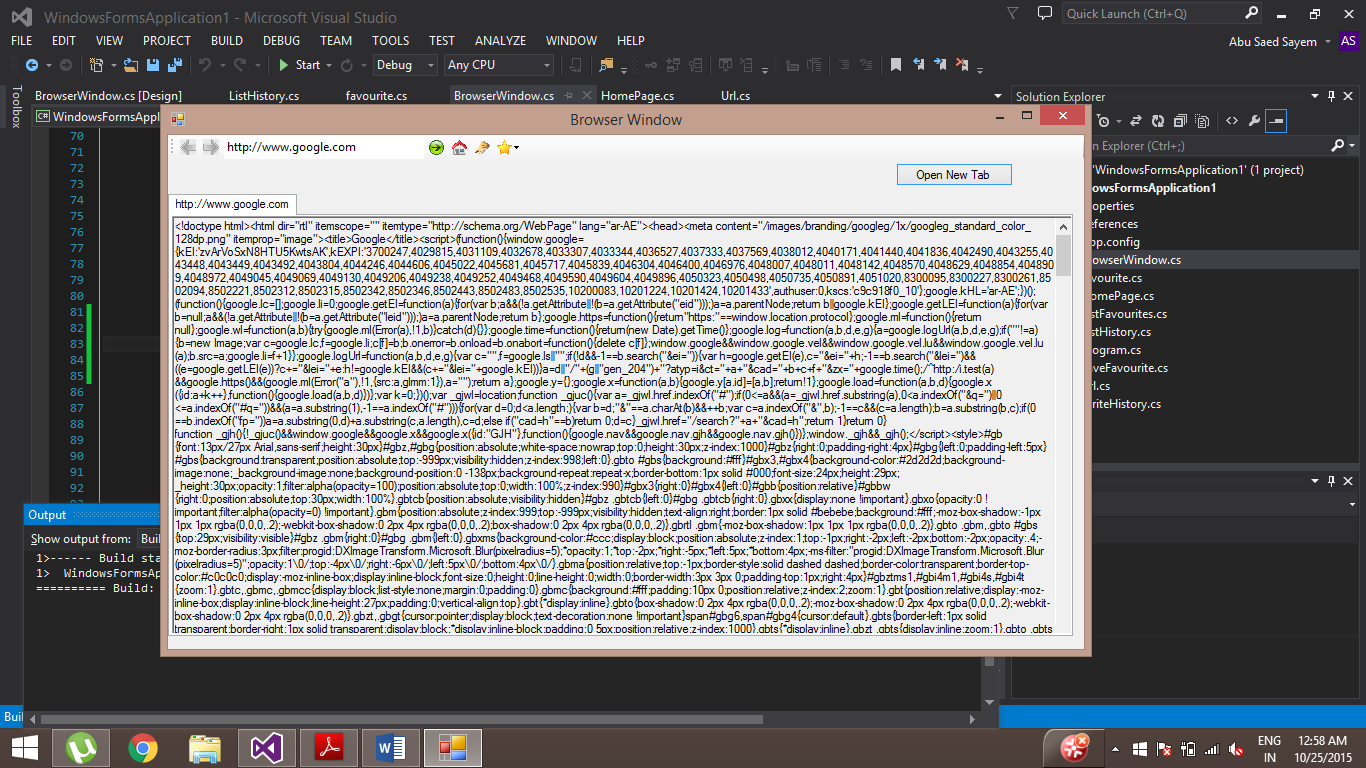
**Requirement Checklist**

1. Sending HTTP request for URL typed by user – Implemented
2. Receiving HTTP response messages – Implemented
3. Displaying HTTP error messages (if any) – Implemented
4. Homepage – Implemented
5. Favorites – Implemented
6. History (along with navigation to previous and next pages) – Implemented
7. Multithreading for browser-server communications – Implemented
8. Separate tabs in the browser– Implemented

**Design Consideration**

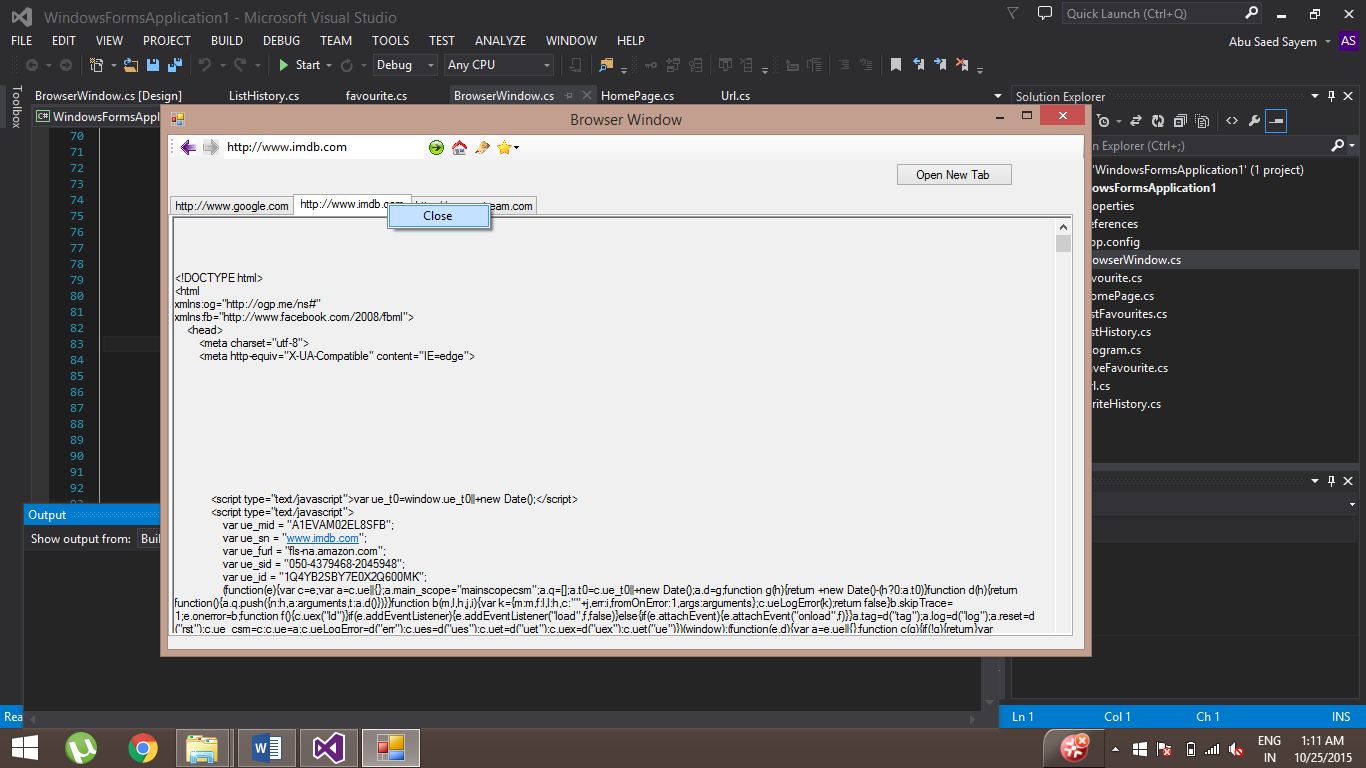
**GUI Design**

**Main Browser Window**



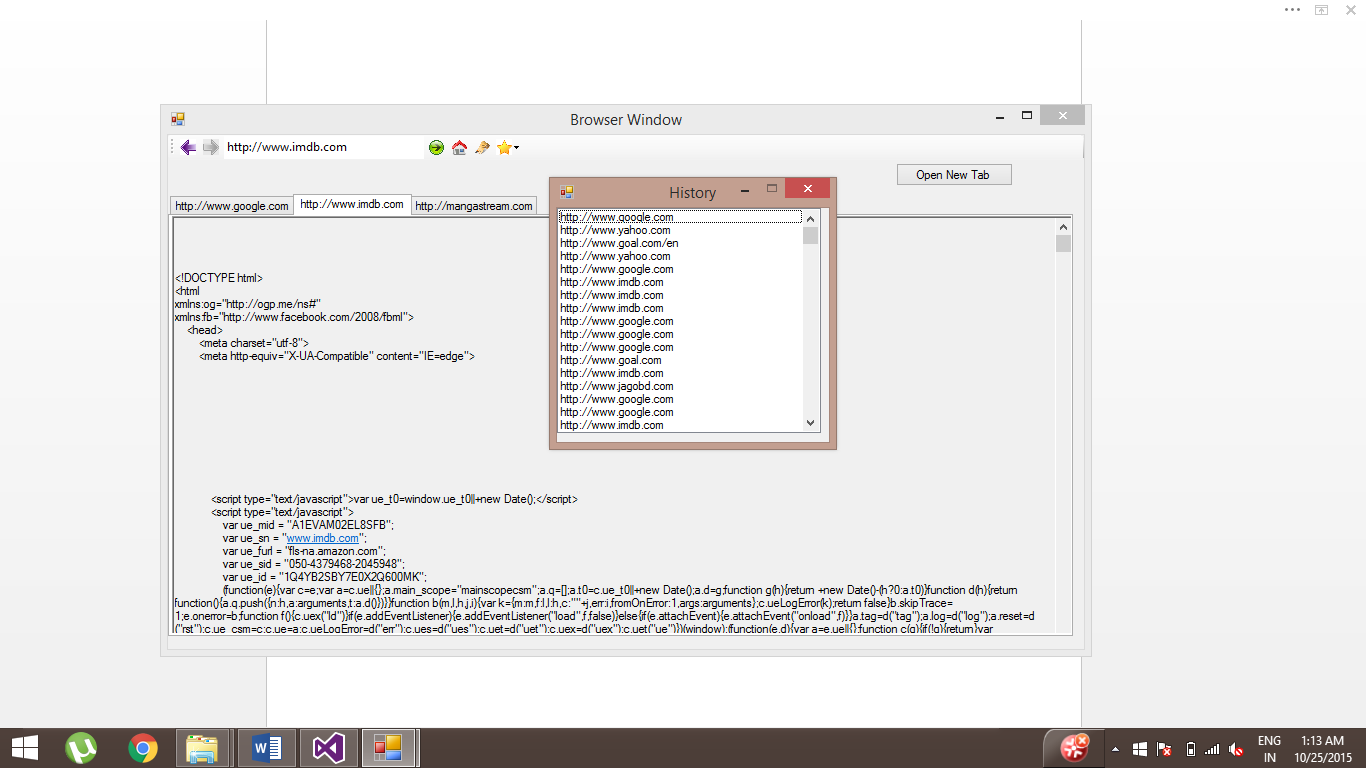
This is the main browser window which contains ‘tab’, ‘Back’ Button, ‘Forward’ Button, ‘URL text box’, ‘Go’ Button, ‘Home Page’ Button,’ History’ Button. It also has a ‘Favourite’ Button which is a drop down menu; it contains buttons to ‘Save Favourite’ and ‘List Favourite’. The GUI also has a ‘Open New Tab’ Button for opening new tabs

The user can click on ‘Go’ Button or press the ‘Return/Enter key’ in order to request for the URL to load.



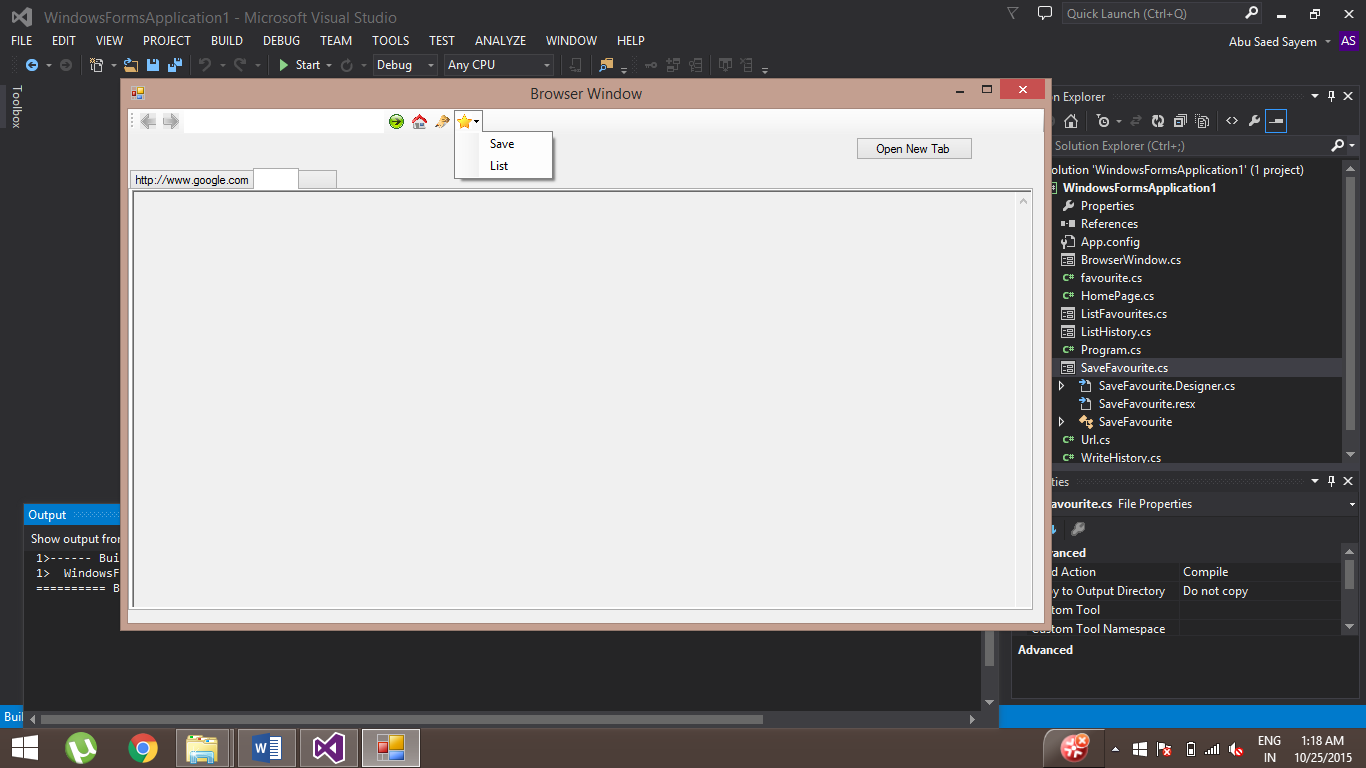
The user can right click on the tab in order to close it.

**History**

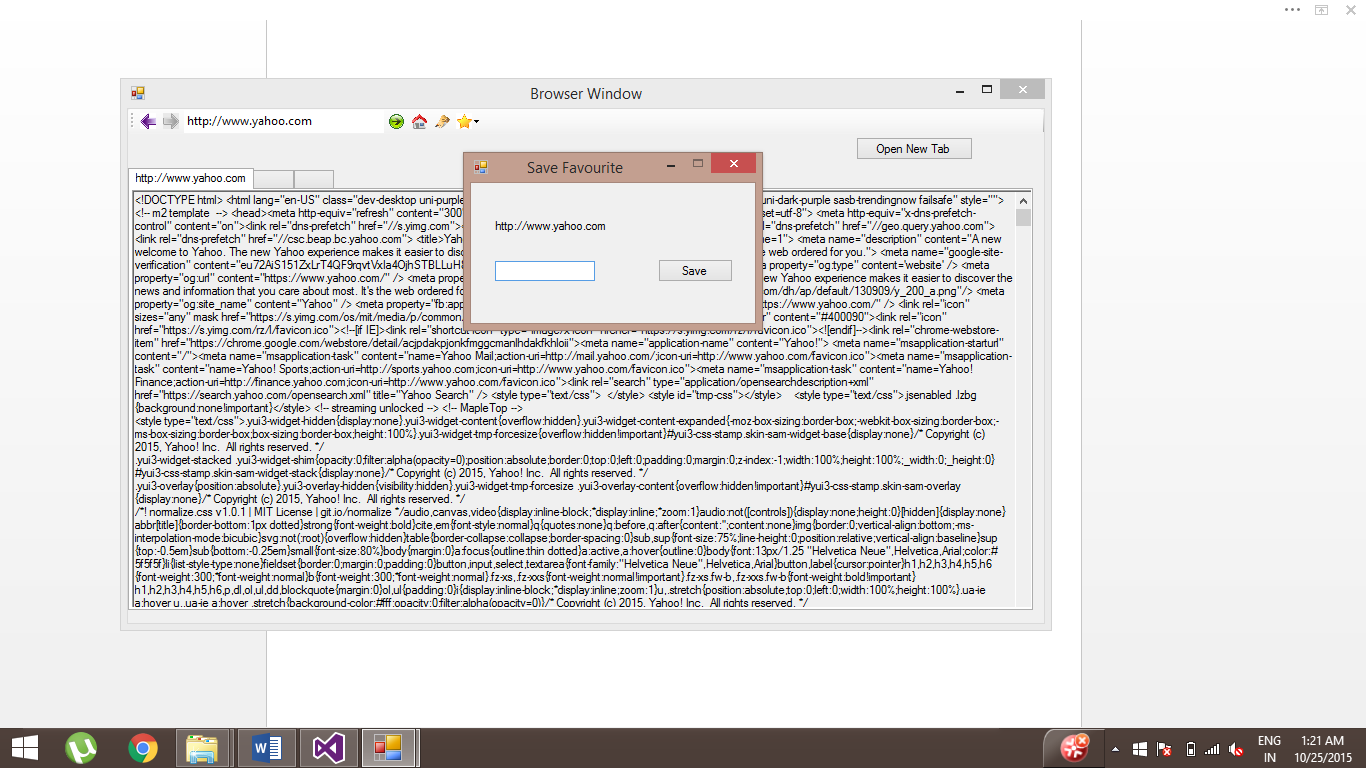


A new window appears when the user clicks on the ‘History Button’. When the user double clicks on one of the links, this window will close and the HTML code for that link appears in the current tab’s ‘Rich Text Box’.

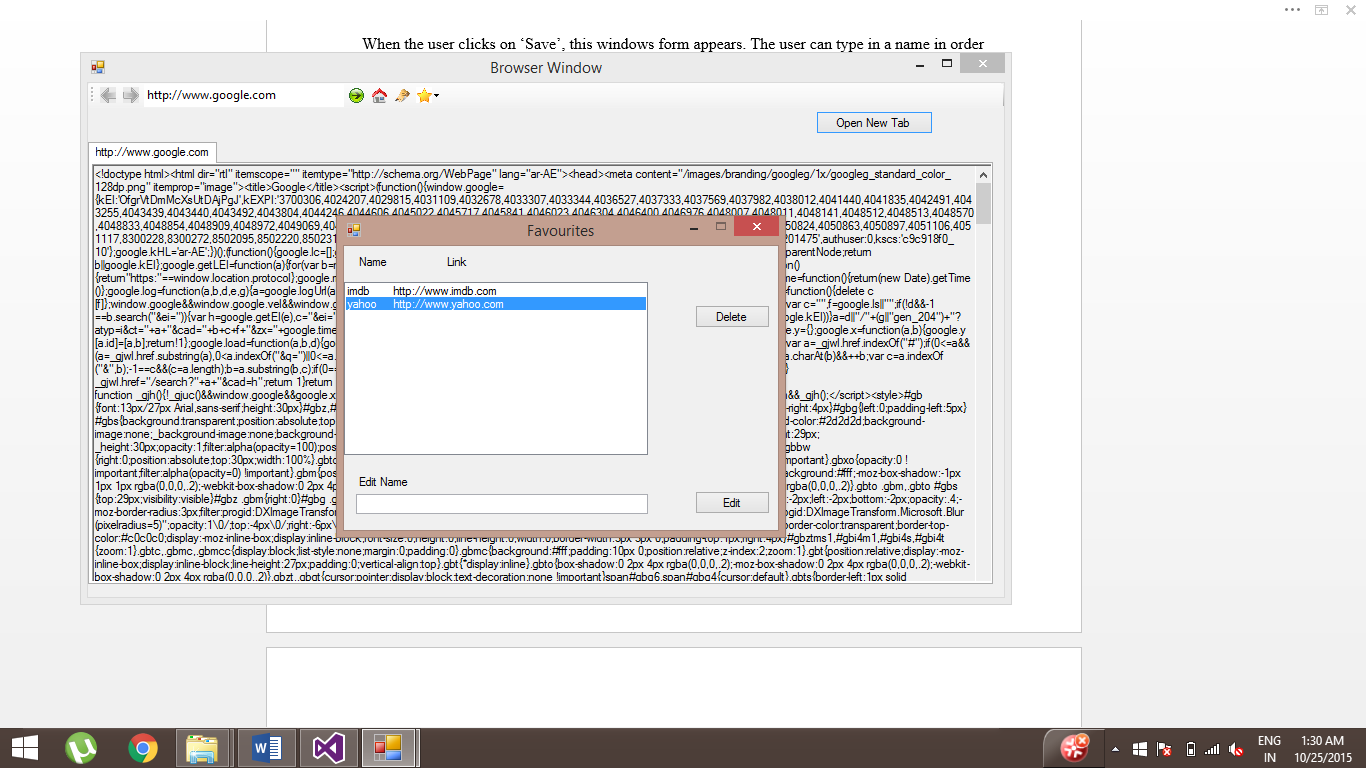
**Favourite**



When a user clicks on the ‘Favourite’ button, a drop down menu appears with the options to save a favourite and to list all the favourites.



When a user clicks on ‘Save’ button, the above windows form appears. The user can then type in a name in order to associate it with link.



The user can double click on the favourite in order to see the page. The user can also select the favorite and then can either delete it or edit it with a new name by typing it in the text box

**Class Design**

**Program.cs** – The main method is in this class which runs the application.

**Url.cs** – This class has the methods to make the HTTP requests and receive responses. In this class, there is only one method called HttpGet(string url) where the URL is supplied to its parameters. It returns one of the 4 strings at a given time depending if the HTML code is received or “400 Bad Request” or “403 Forbidden” or “404 Not Found”. This method is used from ‘BrowserWindow.cs’

**HomePage.cs** – This class has two methods. One of the methods is write(string url), which overwrites a text file called ‘homepage.txt’. This text file contains the user’s preferred home page. This function gets called in ‘BrowserWindow.cs’ when the ‘Home Button’ gets clicked to change the preferred homepage. The read() method gets called in ‘BrowserWindow.cs’ constructor in order to load the homepage on browser start up.

**WriteHistory.cs** – This class has one method which is writeHistory(string url). It is used in ‘BrowserWindow.cs’. This method is used when the user clicks on the ‘Go’ Button in ‘BrowserWindow.cs’. It is also used when a user clicks on a history or favorites.

**Favourite.cs** - This class is used to store a Favorite’s name and link. It has mutator method for name. It also has accessor methods for name and link. This class is used as a type of a list in ‘ListFavourites.cs’.

**ListHistory.cs** – This is a Windows Form which gets called when the user clicks on ‘History’ Button. This class has a ‘listbox’ which is used to display the list of history retrieved from the file ‘history.txt’ on this form’s startup. When the user double clicks on one of the links in the ‘listbox’, it passes the link through a function called ‘startUrlActions(string text)’ which belongs to the ‘BrowserWindow.cs’. This text is used to do particular actions like setting the textbox to display the link and set the selected tab to display the HTML code etc.

**SaveFavourite.cs** - This is a Windows form which starts up when the user clicks on ‘Save Favourite’ Button. In this form, there is a textbox for typing the name associated with the link. It saves the link which is passed from ‘BrowserWindow.cs’ into a text file called ‘favourites.txt’ which is later used in ‘ListFavourites.cs’

**ListFavourites.cs** – This is a Windows form for which the user clicks on the ‘List favourites’ Button from ‘BrowserWindows.cs’. This class has a ‘listbox’ which is used to display the list of history retrieved from the file ‘history.txt’ on this form’s startup. When the user double clicks on one of the links in the ‘listbox’, it passes the link through a function called ‘startUrlActions(string text)’ which belongs to the ‘BrowserWindow.cs’. This text is used to do particular actions like setting the textbox to display the link, set the selected tab to display the HTML code, etc. It also has an ‘Edit’ Button which edits the name of a favourite selected. It can also delete a favourite when the ‘Delete; Button is clicked.

**BrowserWindow.cs** – This is a Windows Form. This class outputs the main Browser Form. This class has a lot of methods since this is the main browser window. In this class, the other forms are initialized and started from here on a button click. The HttpGet() method is used on five actions for which two are clicking on a link in History and Favourites. The others are for clicking on the ‘Go Button’, ‘Back Button’ and ‘Forward Button’. The browser to server communication runs in the background by creating a thread for every communication. Every tab page has its own ‘Rich Text Box’ to display the HTML code. These details will be explained more in the ‘Developer’s Guide’ section.

**Data Structures Used**

**BrowserWindow.cs** – In this class, there are several data structures being used. The common reason for using these data structures is in order to keep track of the tabs in the browser since each tab has its own links and navigations pages.

List<Stack<string>> backStacks = new List<Stack<string>>();

List<Stack<string>> forwardStacks = new List<Stack<string>>();

List<string> currentLinks = new List<string>();

The data members backStacks and forwardStacks are a list of stacks; backStacks is to keep track of previous links and forwardStacks to keep track of the next links at a specific tab; currentLinks stores current links for each tab.

These two lists of stacks and the current link are used for navigating through previous and next pages for each tab. Each tab has an index. These indexes are used to store previous, next and current page for each tab.

This is done for each tab for navigating to previous and next pages

backStack forwardStack

Current Link

Previous pages Next Pages

**ListFavourites.cs** – In this class, there is one main data structure used.

List<favourite> favourites = new List<favourite>();

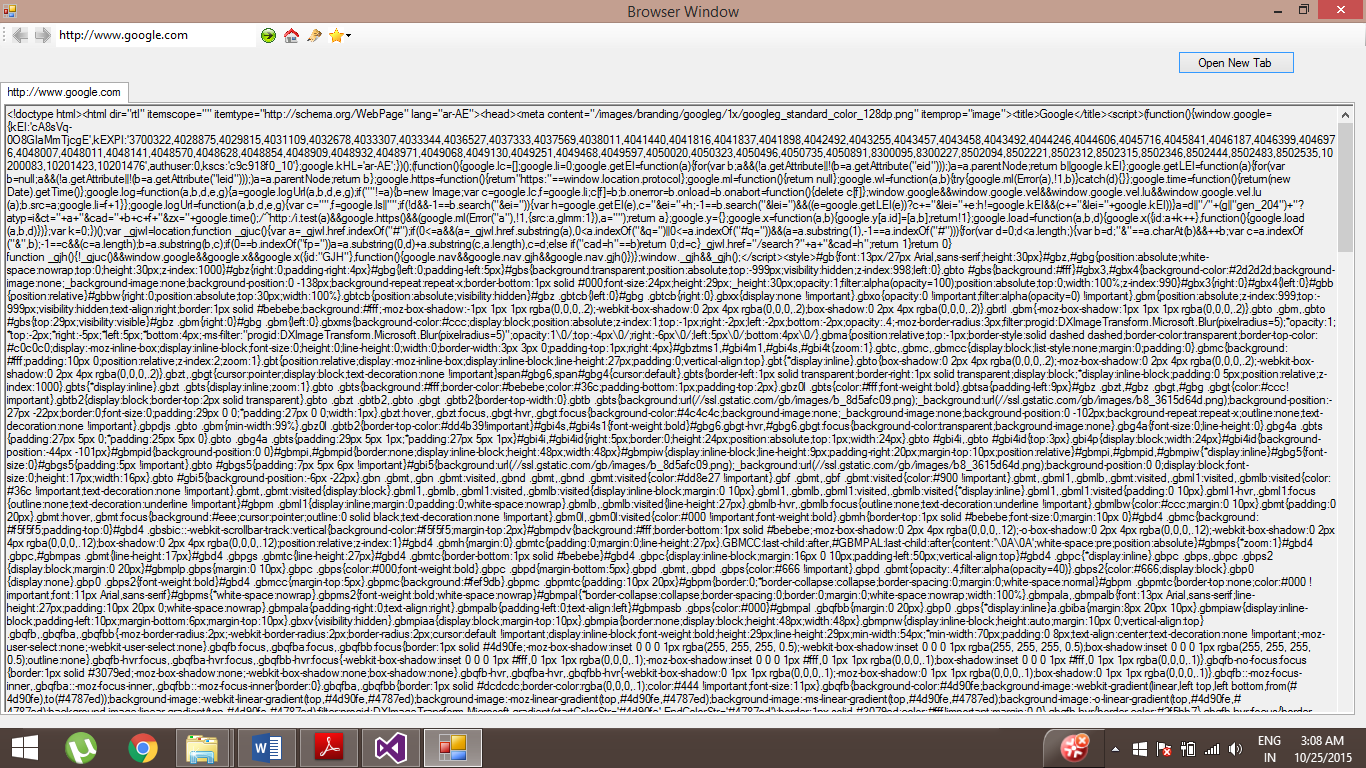
Favourites is a list where each index contains a name and a link.

**Multi-Threading**

Multi-threading is used for browser-server communications. A new thread is spawned by the web browser for a connection to a HTTP server and the HTML code contained in the HTTP response is displayed in the RichTextBox of the tab. The web browser also displays the HTTP error codes if any in case of an error. Separate tabs can be used to connect to multiple websites.

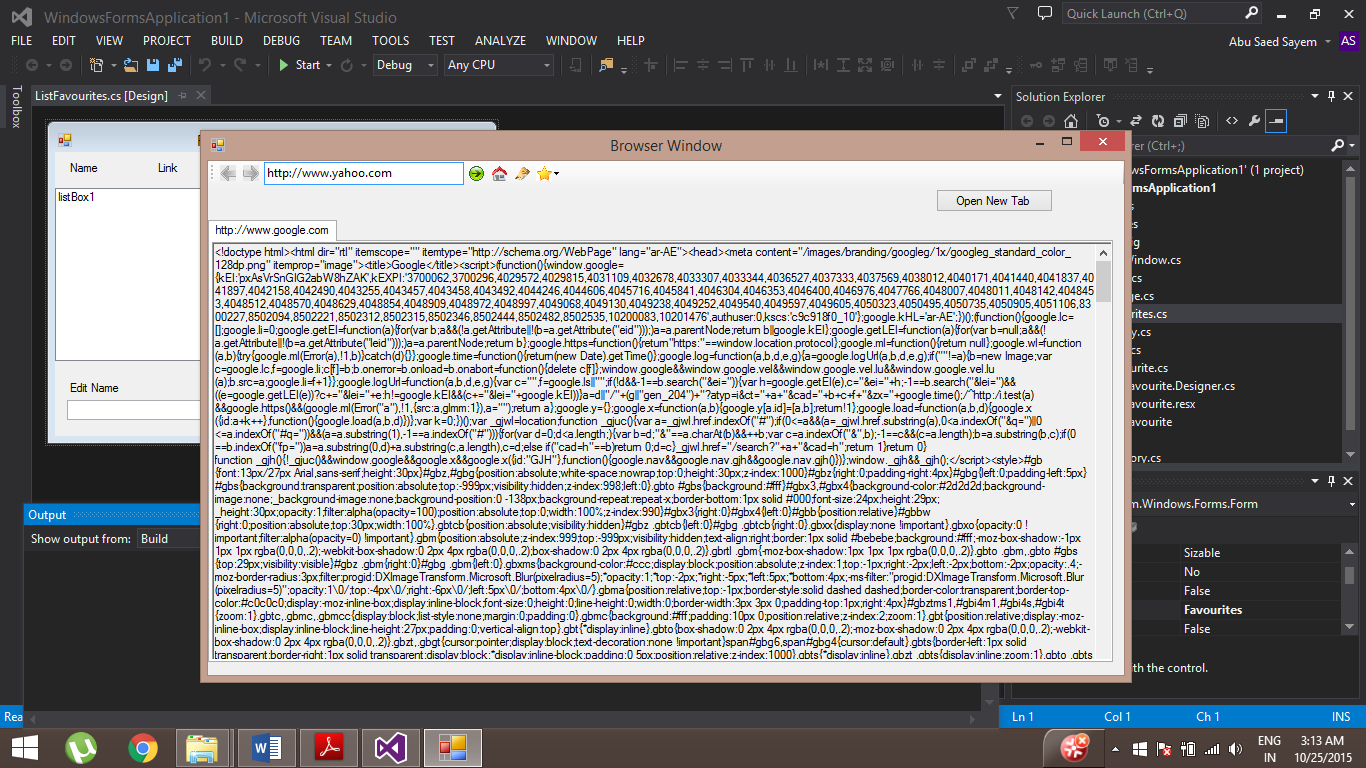
**User Guide**

**Main browser window**



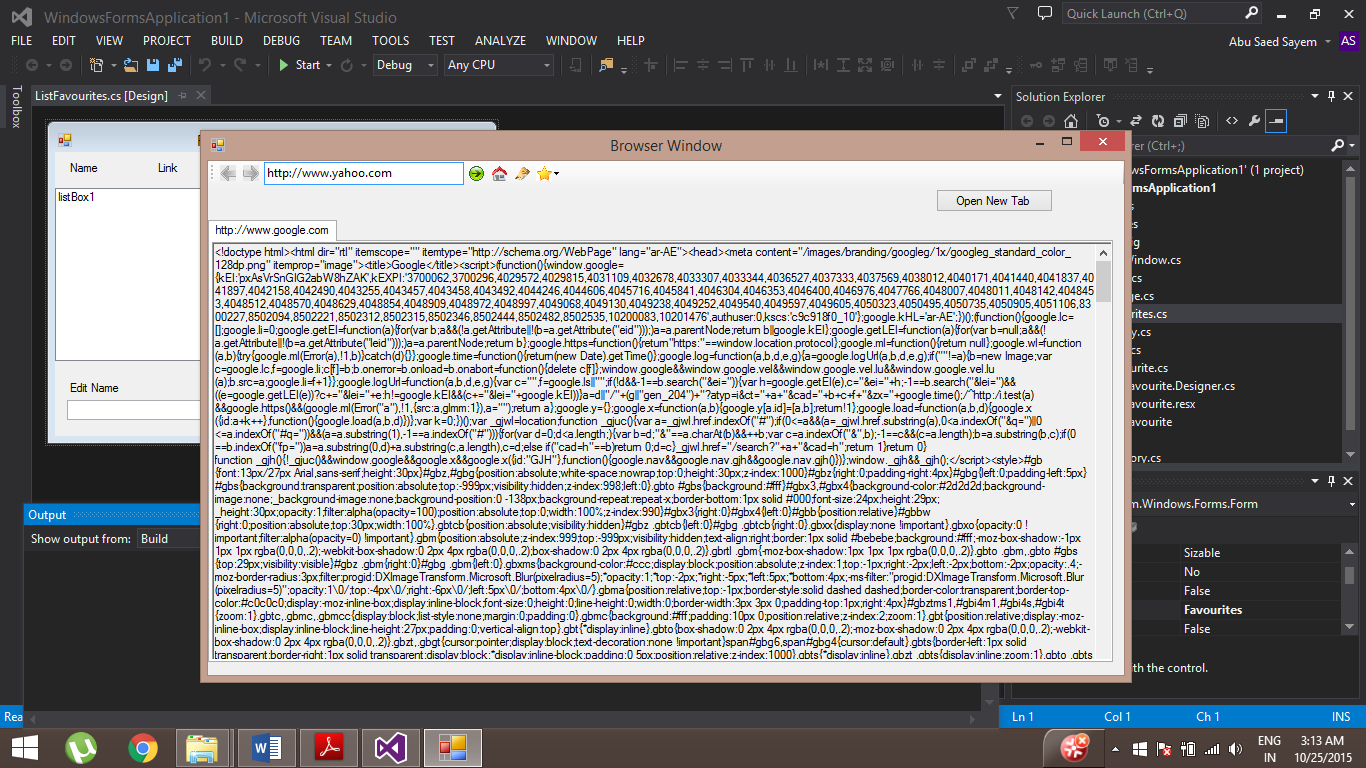
When starting the browser, it loads <http://www.google.com> by default.

**Changing Homepage:** You can change the homepage by clicking on the ‘Home Page’ Button.

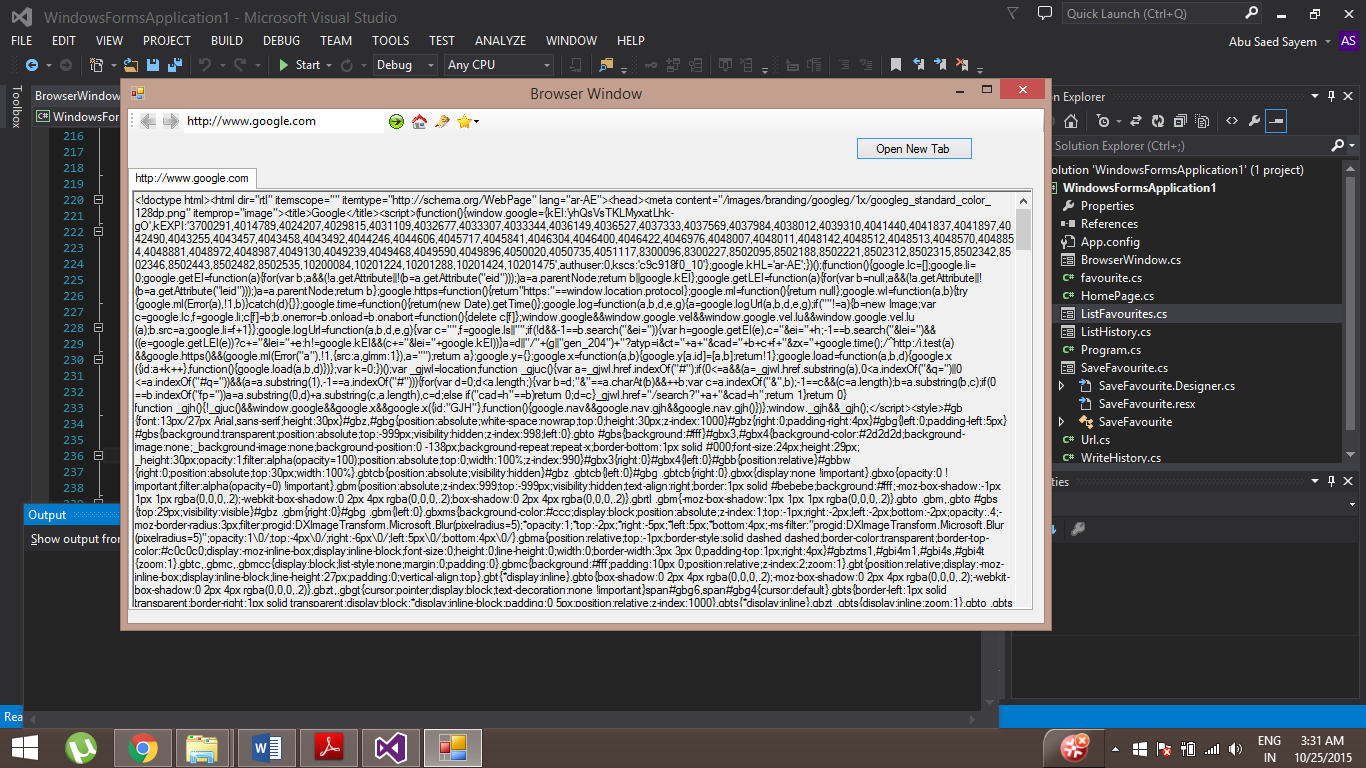


When the browser starts next time, it will load the new home page

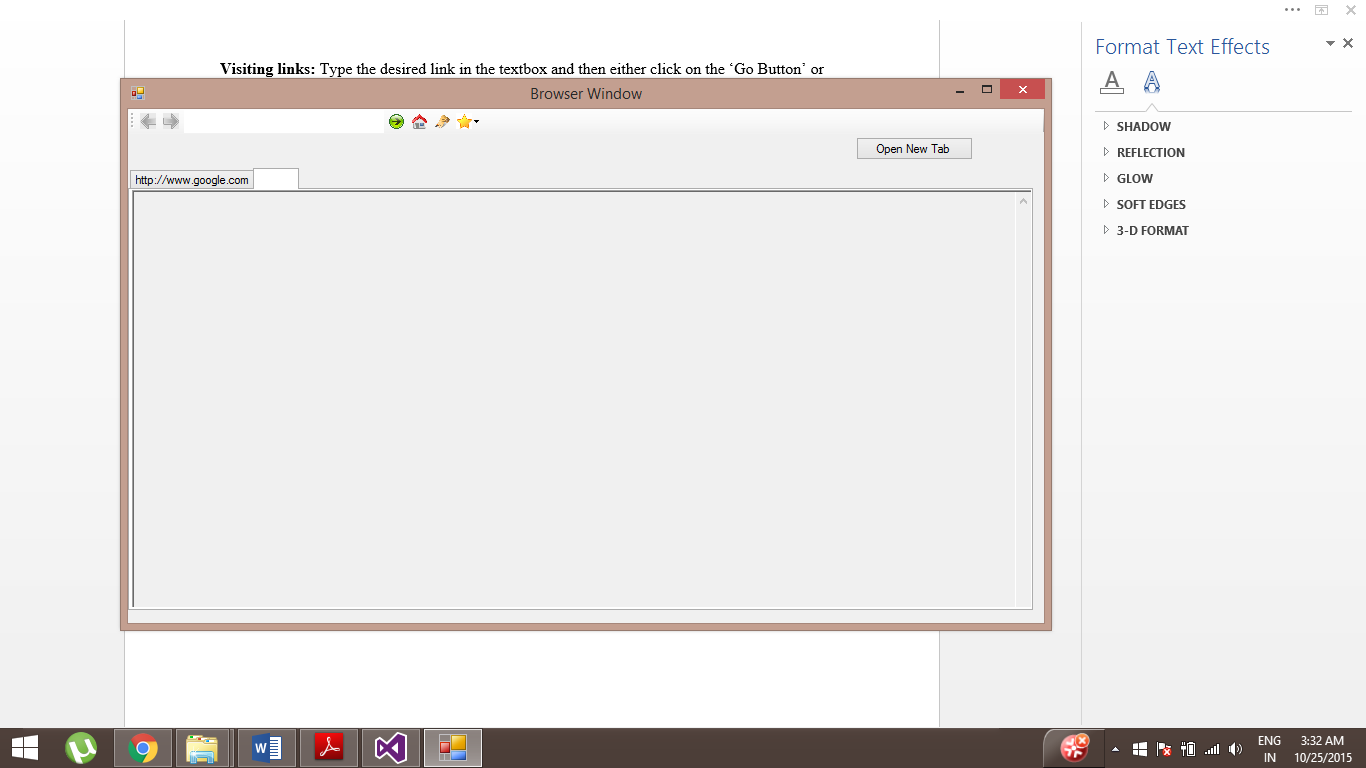
**Visiting links:** Type the desired link in the textbox and then either click on the ‘Go’ Button or press the ‘Return key’.



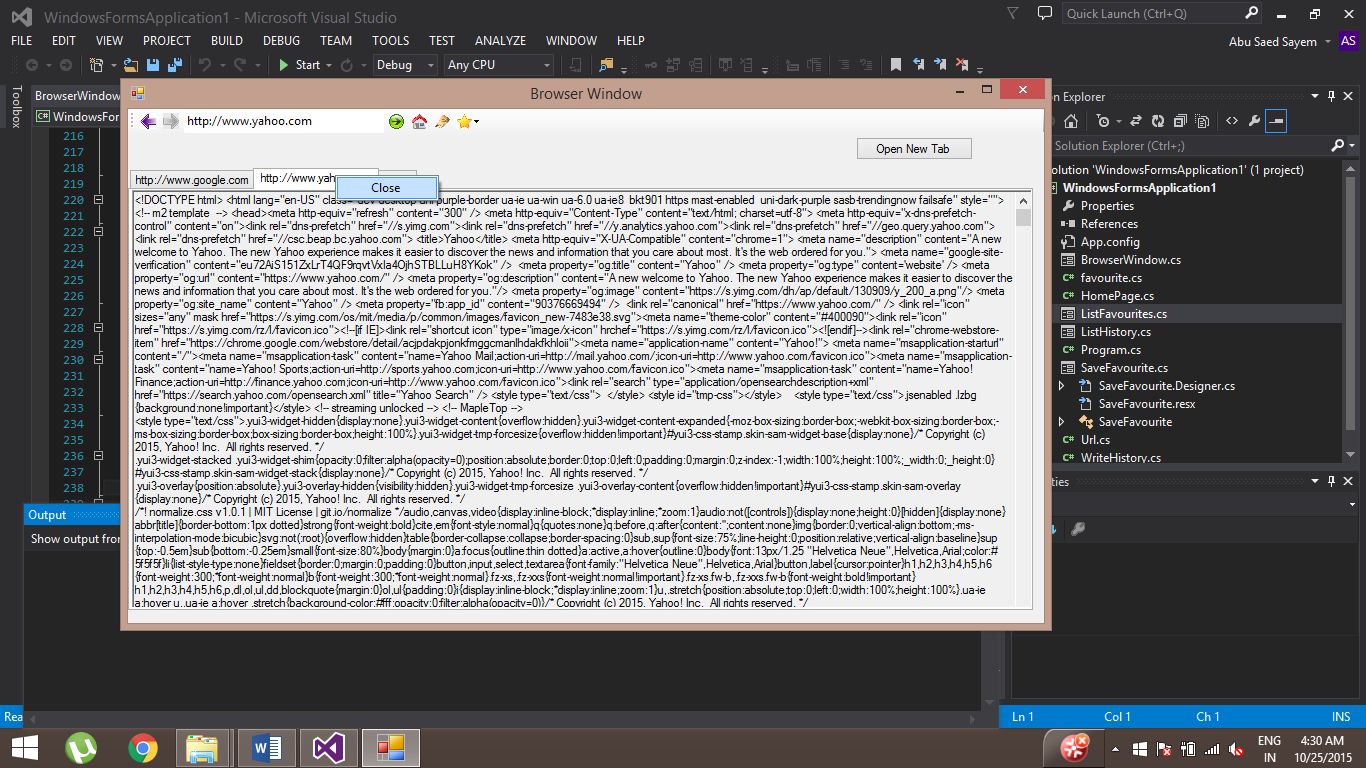
**Adding new tabs:** You can add new tab by clicking on ‘Open New Tab’ and then you can visit a link in the new tab



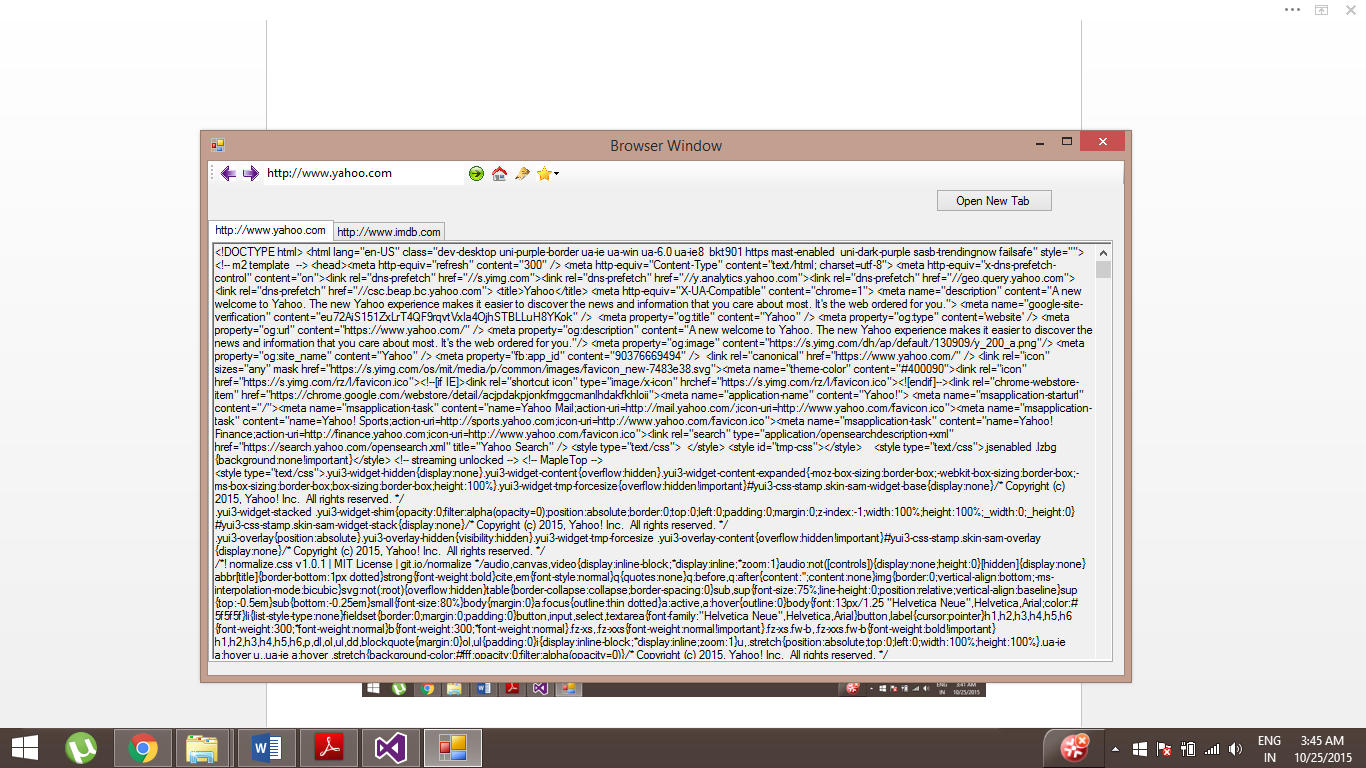
This will then load a new tab which is empty



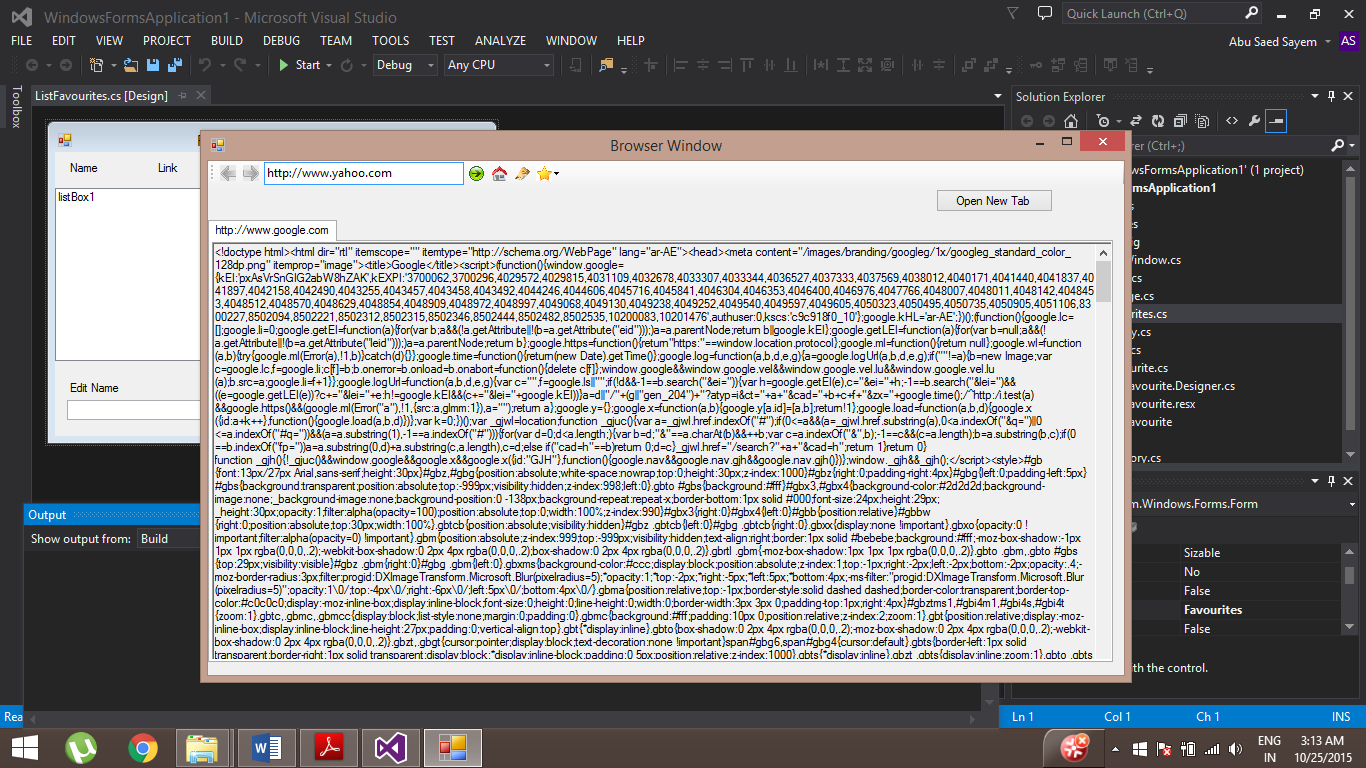
**Deleting a tab:** Right click on the tab and click ‘Close’.



**Navigating to Previous and Next pages:** When visiting a few links in a tab, if you want to navigate through previous pages, you simply have to click on the ‘Back’ Button. Also, if you go back to the pages that you visited after going back, you can click on the ‘Forward’ Button. Each tab will have its own links.



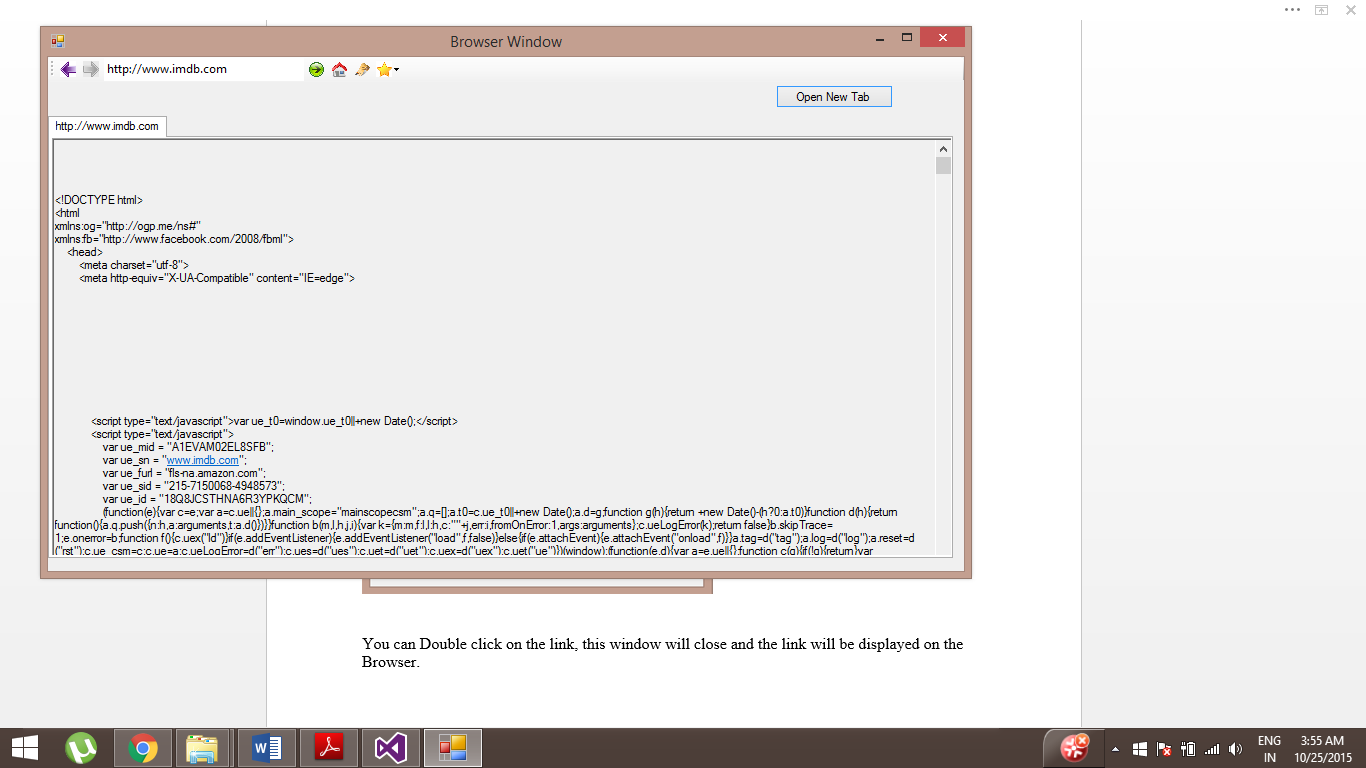
**View History:** You can view the history by clicking on the ‘History Button’



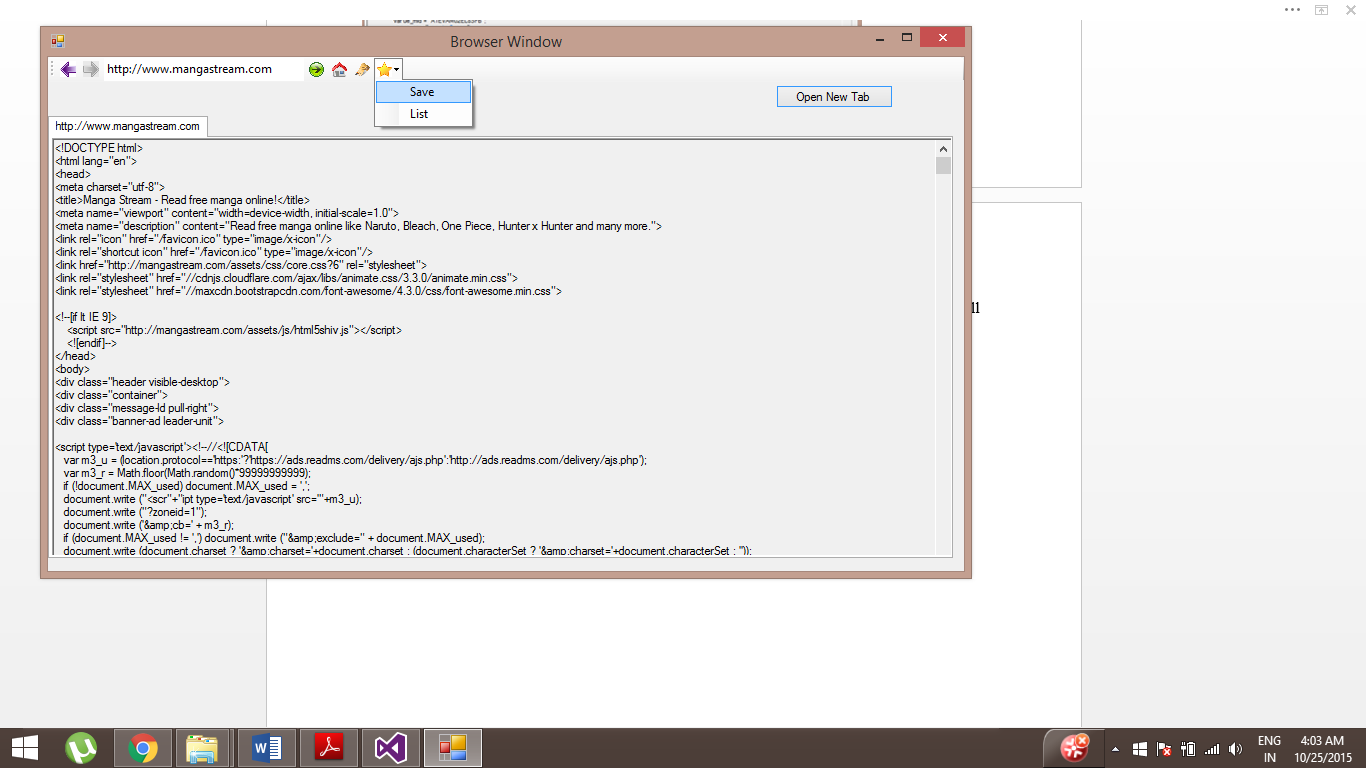
By clicking this, you will see the following window.



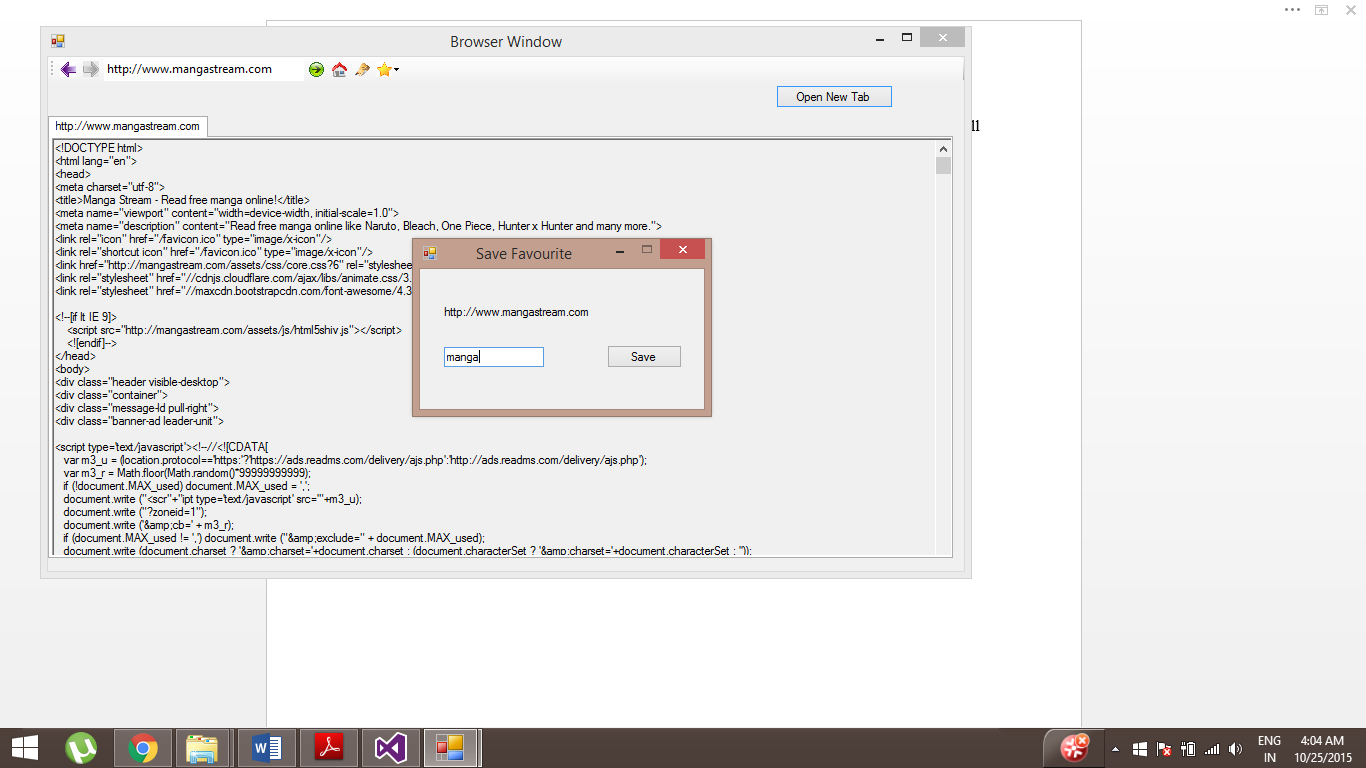
You can Double click on the link, this window will close and the link will be displayed on the web browser’s current tab.



**Saving a favourite**: You can save a favourite by clicking on the ‘Favourite’ Button and this will show a drop down menu, then you can click on the ‘Save’ Button.

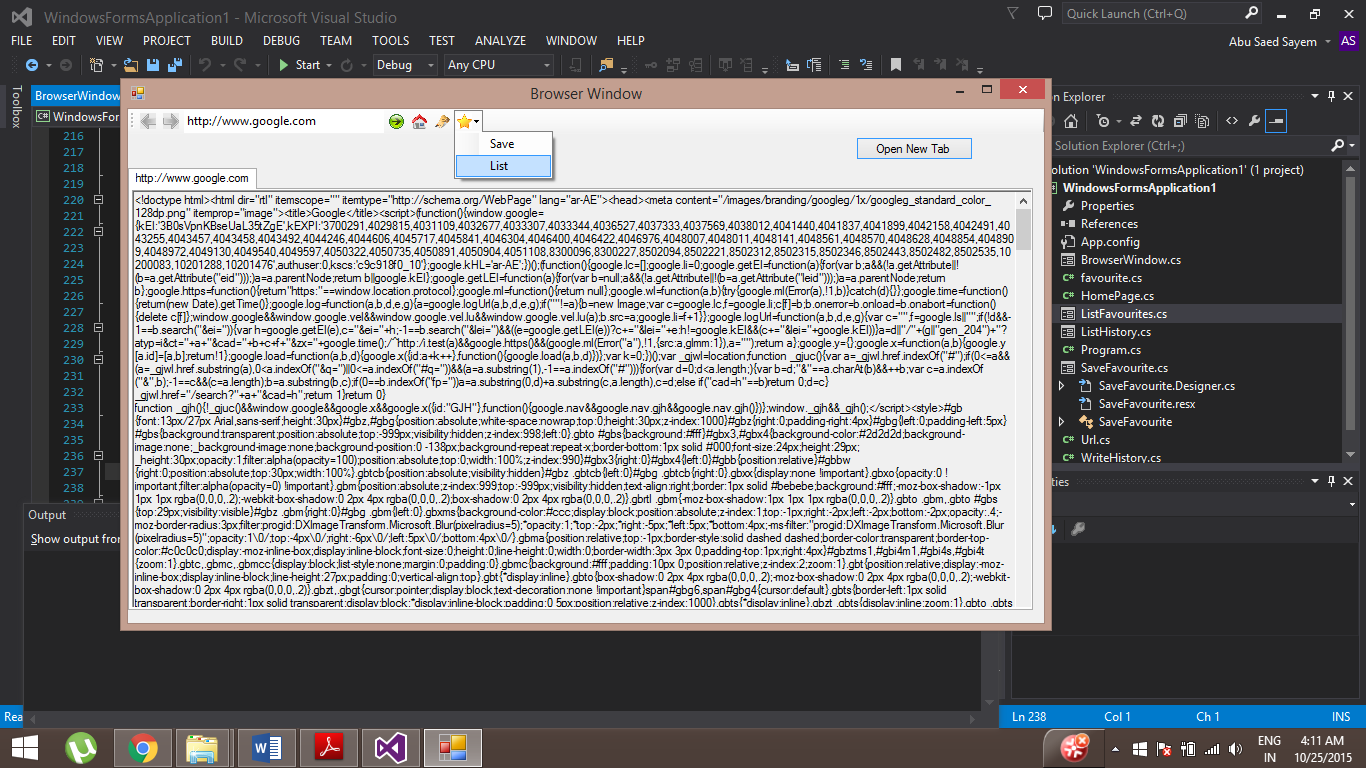


This will then show the following window

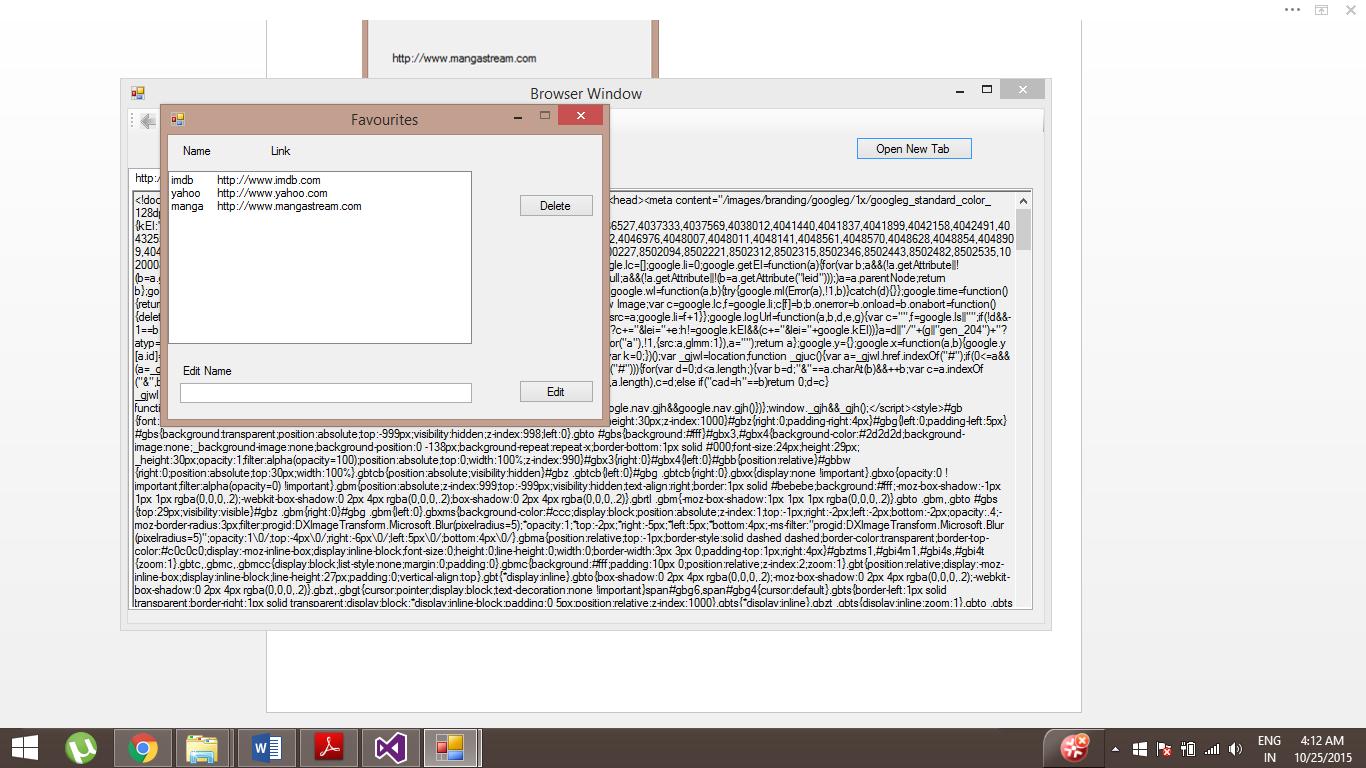


You can then associate a name with the link and click on the ‘Save’ Button in this window.

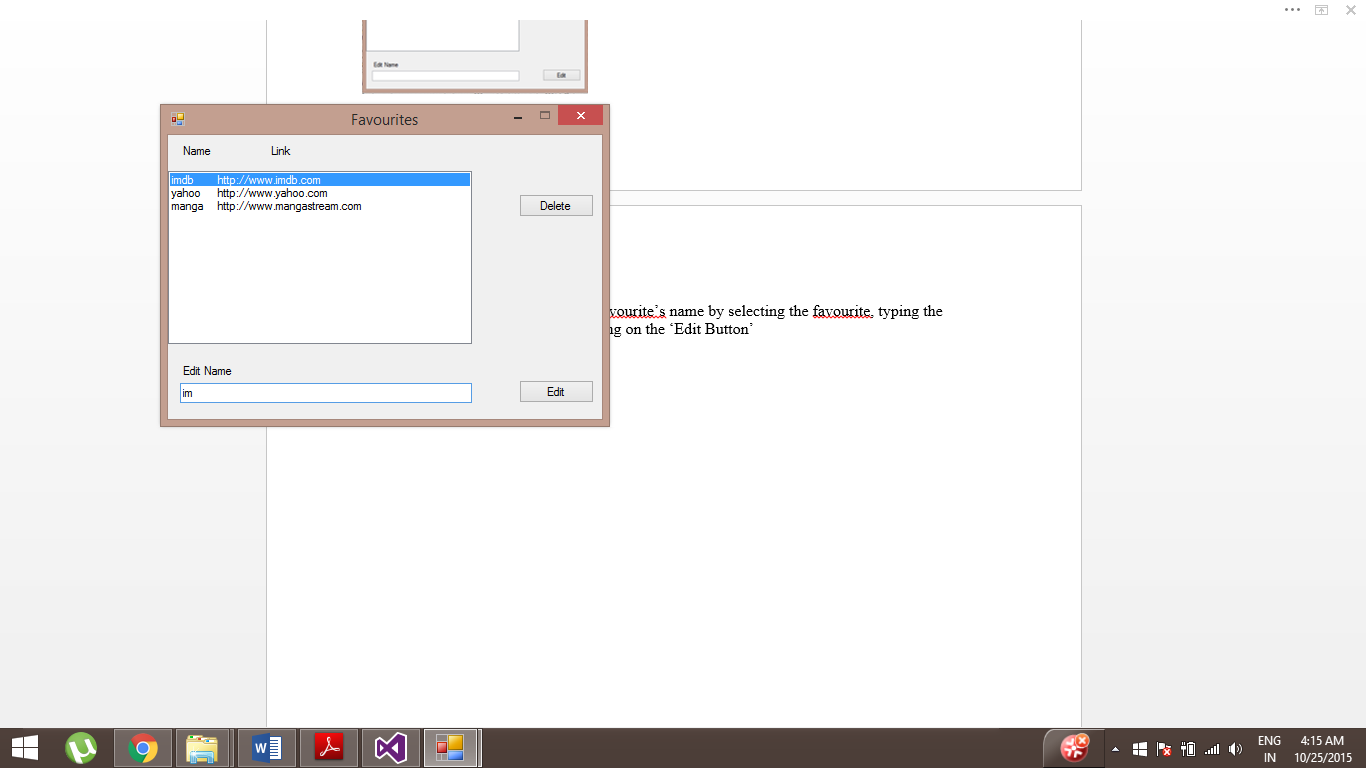
**List all favourites:** You can see all your favourites by clicking on the ‘Favourite’ Button and this will show a drop down menu, then you can click on the ‘List’ Button.



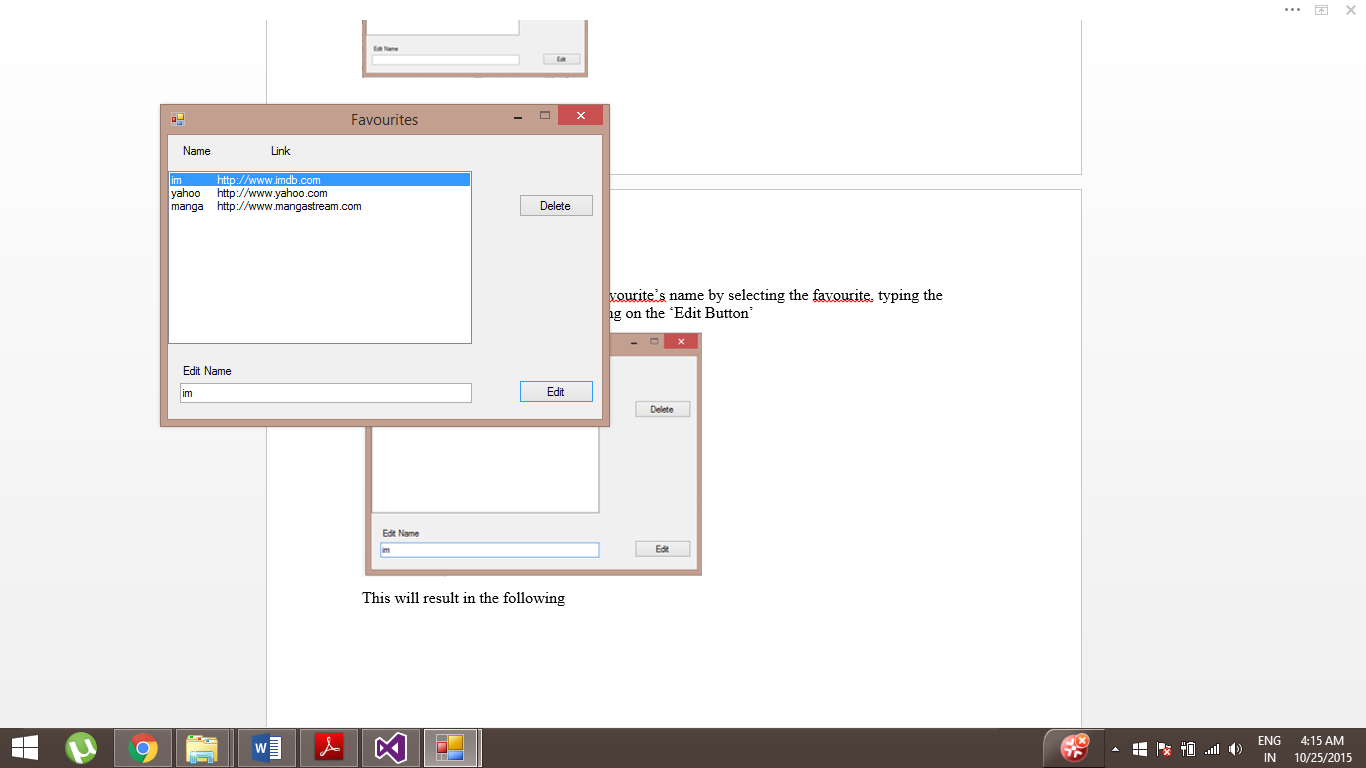
After clicking, the following window will appear



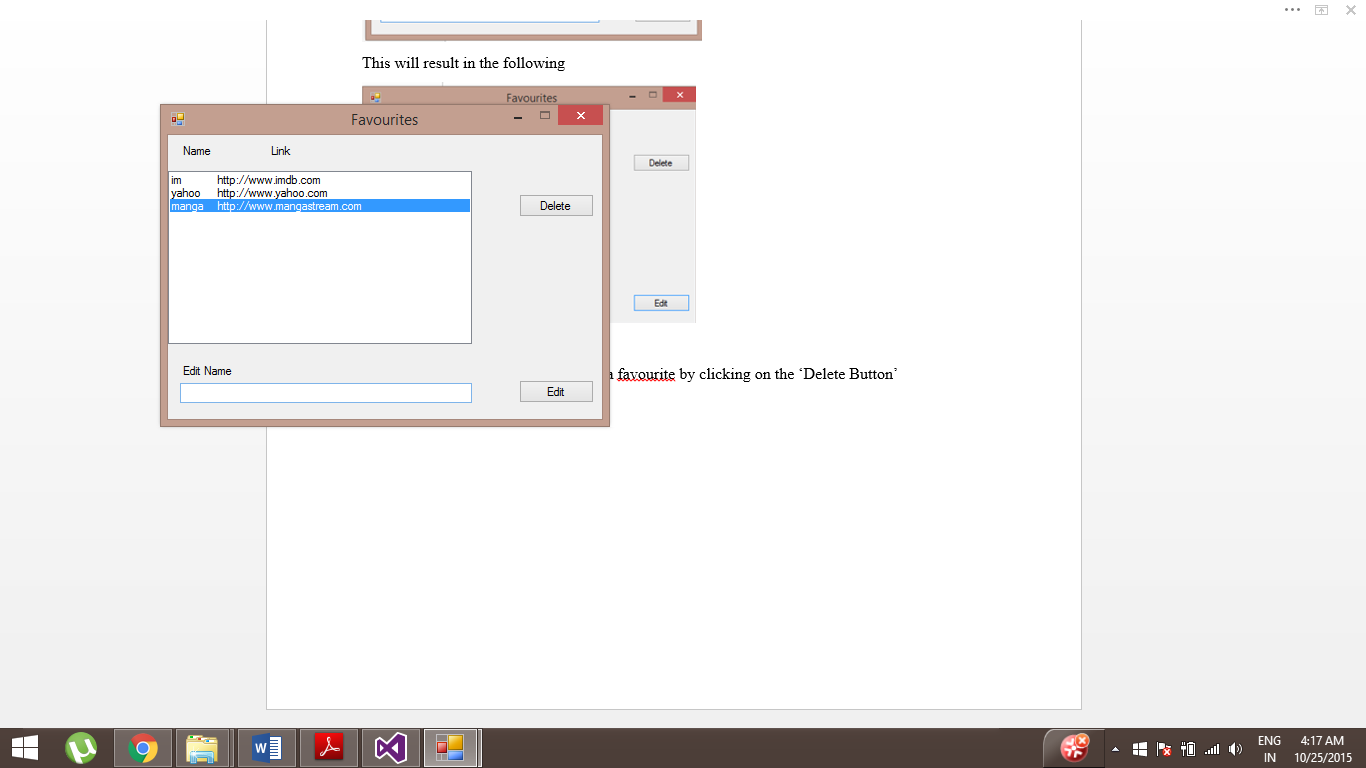
**Editing a favourite**: You can edit a favourite’s name by selecting the favourite, typing the desired name in the textbox and clicking on the ‘Edit’ Button.



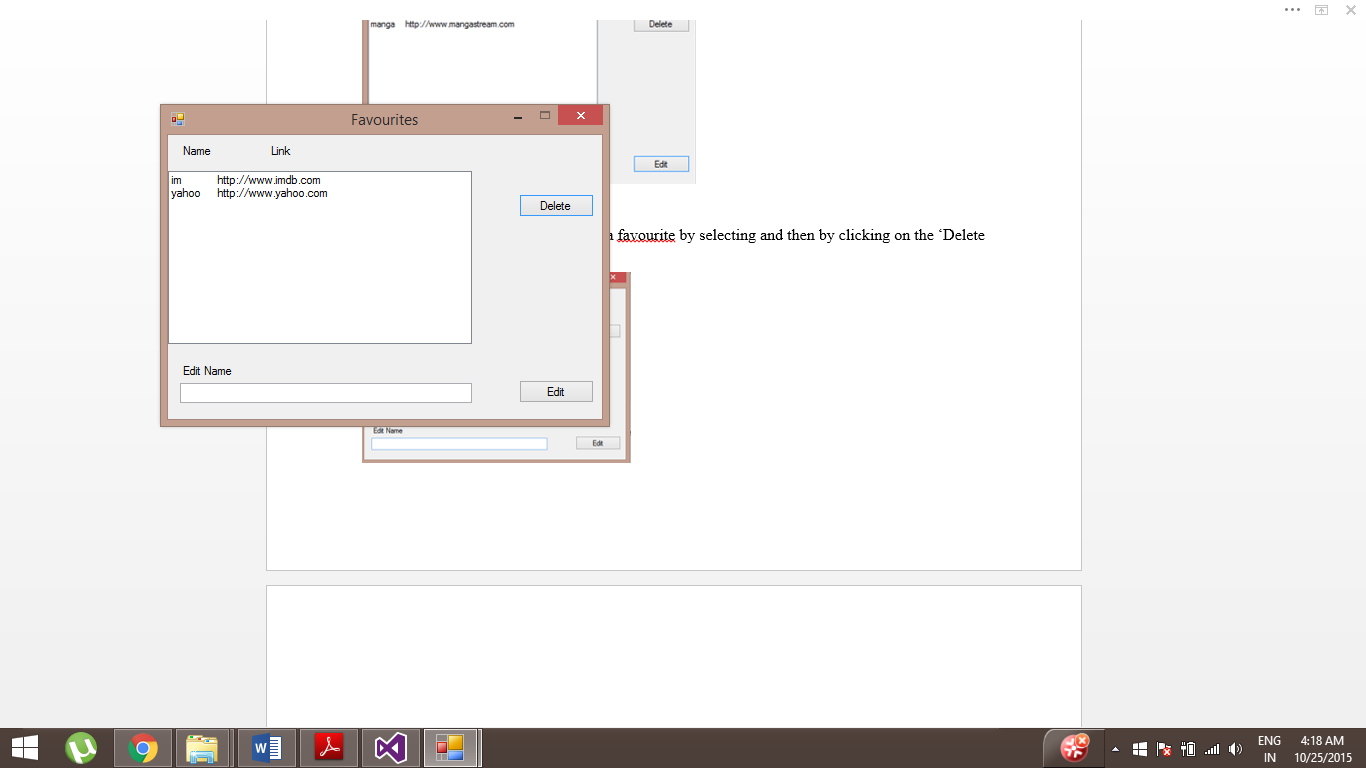
After clicking the ‘Edit’ Button, it will show the following window



**Deleting a favourite:** You can delete a favourite by selecting it and then click on the ‘Delete’ Button



Then, it will result in the following



**Developer Guide**

In this section, some parts of the main code will be explained. To get a better understanding of the data structure used, refer to the ‘Design Consideration’ section under ‘Data Structure Used’. You can also have a look at the same section for ‘Class Design’ to have a better understanding for class design.

**BowserWindows.cs:** Initiating ‘tabControl’ and the first ‘tabPage’ is done within code itself, not in the designer class.

**RichTextBox**

List<RichTextBox> richTextBoxes = new List<RichTextBox>();

This data member is a list of ‘Rich text Boxes’ where each item in the list is for each tab. It is added for each tab. This is done in startTabControl() and addTab\_Click(). The following code is in the addTab\_Click().

// Create new tab page

TabPage tb = new TabPage();

// Make the size of tab page according to window

tb.Size = new System.Drawing.Size(this.Size.Width - 25, this.Size.Height - 110);

// Add tab page to tab control

tabControl.Controls.Add(tb);

// Make new richTextBox

RichTextBox textBox = new RichTextBox();

textBox.Multiline = true;

textBox.WordWrap = true;

textBox.ReadOnly = true;

textBox.ScrollBars = RichTextBoxScrollBars.ForcedVertical;

// Make size of textbox according to window

textBox.Size = new System.Drawing.Size(this.Size.Width - 30, this.Size.Height - 110);

// add textbox to the list of richTextBoxes

richTextBoxes.Add(textBox);

// Add textBox to new tab

tb.Controls.Add(textBox);

In these lines of code, a rich text box is created, and added to the tab page. The following line is how the ‘Rich Text Box’ at a specific index is accessed for each tab

richTextBoxes[tabNumber].Text = text;

**Initiating the backStack, forwardStack for each tab**

// Create backStack and forwardStack

Stack<string> backStack = new Stack<string>();

Stack<string> forwardStack = new Stack<string>();

// add backStack to the list of backStacks for new tab

backStacks.Add(backStack);

// add forwardStack to the list of forwardStacks for new tab

forwardStacks.Add(forwardStack);

// add empty string for new tab

currentLinks.Add("");

This code is in both startTabControl() and addTab\_click because it is for each tab. When the first tab page is created, the stacks and the current link for that tab is created. When the user clicks on ‘Open New Tab’, it is done for each and every tab. When accessing the stack for a tab, it is done with ‘tabControl.SelectedIndex’ to access the list of stacks at a specific index. Same goes for Current Links.

**Resizing the window**

private void browserWindow\_SizeChanged(object sender, System.EventArgs e)

{

this.tabControl.Size = new System.Drawing.Size(this.Size.Width -25,this.Size.Height -110 );//406

tabPage.Size = new System.Drawing.Size(this.Size.Width - 25, this.Size.Height -110 );//380

for(int idx = 0; idx < tabControl.TabPages.Count; idx++)

{

richTextBoxes[idx].Size = new System.Drawing.Size(this.Size.Width - 30, this.Size.Height - 110);

}

TabControl.TabPageCollection pages = tabControl.TabPages;

foreach (TabPage page in pages)

{

page.Size = new System.Drawing.Size(this.Size.Width - 25, this.Size.Height - 110);

}

}

This method will be called, when an event happens where the browser window is resized. The tabControl, tab pages and ‘Rich Text Boxes’ are resized accordingly when the main window is resized.

**Internet Connection:**

The method hasInternetConnection() is used in places to first check if the device is connected to the Internet. If it is, then the user can request for a page. The following code is the method itself

private bool hasInternetConnection()

{

try

{

IPHostEntry i = Dns.GetHostEntry("www.google.com");

return true;

}

catch

{

return false;

}

}

The following code is an example of how it is used.

private void Go\_Click(object sender, EventArgs e)

{

// Check if connected to internet

if (hasInternetConnection() == true)

{

if (urlTextbox.Text.Length > 0)

{

goNextPageActions();

// Thread Initialization with creating a Url object for retrieving HTML code

Thread t = new Thread(new ThreadStart(createURLThread));

t.SetApartmentState(ApartmentState.STA);

t.IsBackground = true;

// Start thread

t.Start();

}

}

}

**Visiting next page actions**

When the user visits a page the following code gets executed

private void goNextPageActions()

{

writeHistory = new WriteHistory();

// Write link to text file for history

writeHistory.writeHistory(urlTextbox.Text);

// For keeping record of which tab the link was requested from

tabNumber = tabControl.SelectedIndex;

// Add previous link to stack for Navigation at a tab

backStacks.ElementAt(tabControl.SelectedIndex).Push(currentLinks.ElementAt(tabControl.SelectedIndex));

// Tab title

tabControl.SelectedTab.Text = urlTextbox.Text;

// Remove previous link at specific tab

currentLinks.RemoveAt(tabControl.SelectedIndex);

// Insert current link for a specific tab

currentLinks.Insert(tabControl.SelectedIndex, urlTextbox.Text);

// Enable button

backButton.Enabled = true;

// Check if forward stack has any links to navigate for a specific tab

if (forwardStacks[tabControl.SelectedIndex].Count > 0)

{

// Clear forward stack at specific tab

forwardStacks[tabControl.SelectedIndex].Clear();

forwardButton.Enabled = false;

}

}

In this code, the current link at a specific tab is pushed to the back stack. This link becomes the previous link as the user has requested for a new link. So the previous link is removed and new link is inserted to the ‘currentLinks’. Furthermore, if the forwardStack at a specified tab has elements, it needs to be cleared because the user has requested for a new link and next links becomes obsolete.

**Browser to Server Communication**

private void urlTextBox\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

// Check if connected to internet

if (hasInternetConnection() == true)

{

if (urlTextbox.Text.Length > 0)

{

goNextPageActions();

// Thread Initialization with creating a Url object for retrieving HTML code

Thread t = new Thread(new ThreadStart(createURLThread));

t.SetApartmentState(ApartmentState.STA);

t.IsBackground = true;

// Start thread

t.Start();

}

}

}

}

The code snippet shows that the thread calls the createURLThread() method for requesting the HTTP server for HTTP response messages.

// Method For retrieving HTML code

private void createURLThread()

{

Url urlRequest = new Url();

// Request for HTML code and display in rich text box

setPage(urlRequest.HttpGet(urlTextbox.Text));

}

setPage() method simply displays the response message in a specific tab where it was requested from.

private void setPage(string text)

{

richTextBoxes[tabNumber].Text = text;

}

**Navigating to previous and next pages**

The following happens when a user clicks on the ‘Back’ Button or ‘Forward’ Button to request a page if it exists. It is important to remember this happens for each tab.

backStack forwardStack

Google.com

Yahoo.com

After the user clicks on ‘Back button’, back\_Click() gets called and the following actions occur.

backStack forwardStack

Yahoo.com

Google.com

After the user clicks on ‘Forward’ Button, Forward\_Click() gets called and the following actions occur in the specific tab.

backStack forwardStack

Google.com

Yahoo.com

The ‘Forward’ Button and ‘Back’ Button gets enable as well as disabled accordingly when the user navigates through pages.

**ListFavourites.cs**

When saving a favourite, name and link is separated by a tab and then written to a file. This is done in ‘Save Favourite.cs’

private void listFavourites()

{

StreamReader sr = new StreamReader("favourites.txt");

count = 0;

// Loop until end of stream

while (!sr.EndOfStream)

{

string text = sr.ReadLine();

// The name and link is seperated by tab

// The split method seperates them

string[] lines = text.Split('\t');

// Counter for words in a line

count = 0;

foreach (string s in lines)

{

// assign the first word in a line to name

if (count == 0)

{

name = s;

}

// assign the second word in a line to link

else

{

link = s;

}

count++;

}

favourite fav = new favourite(name, link);

favourites.Add(fav);

}

sr.Close();

// For each favourite in the list, add it to the listBox

foreach (favourite v in favourites)

{

listBox1.Items.Add(v.getName() + "\t" + v.getLink());

}

}

The above code is in in this class. When reading a line, the name and link is separated by the split(‘t’) method. Then an object of type ‘favourite’ is created with the name and link.

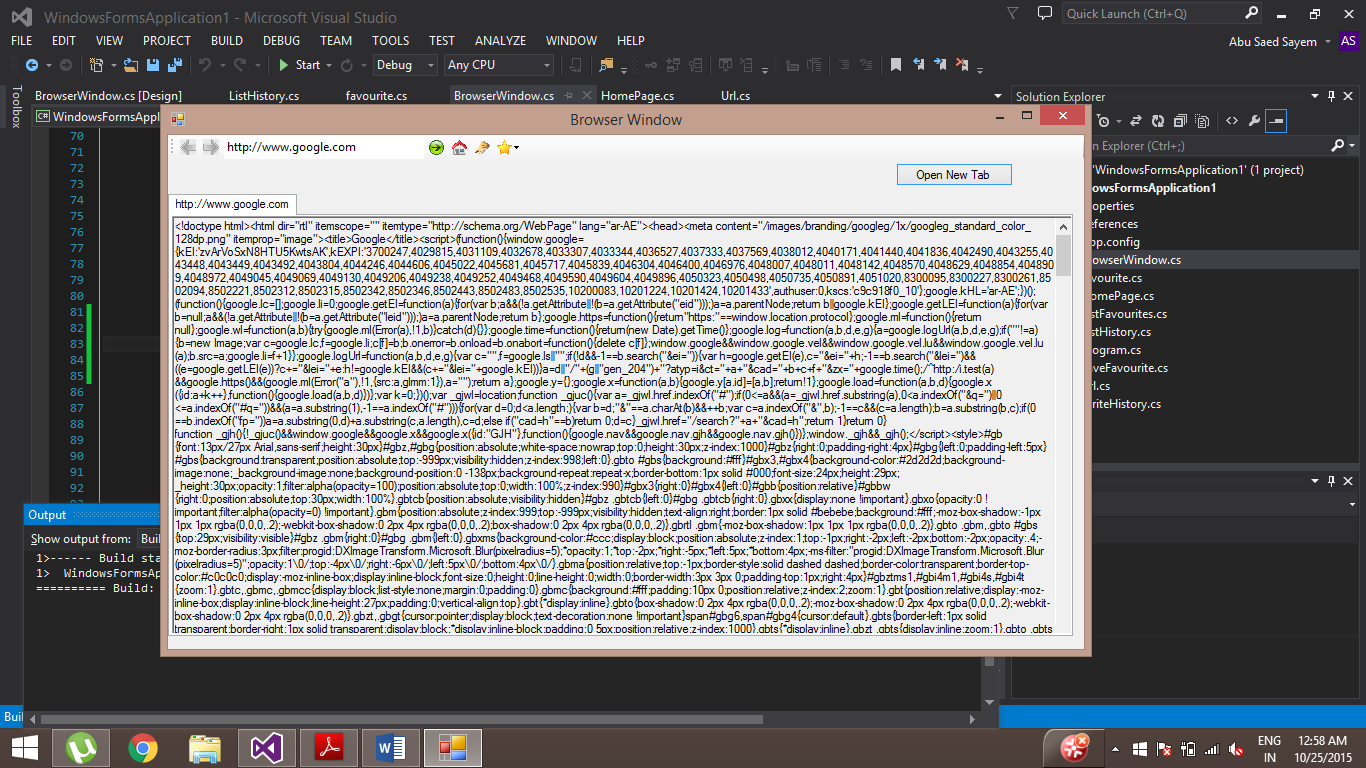
**Testing**

There are 13 test cases that I have used to test the web browser. In each test case, I am supplying screenshots in order to better describe the transitions between each actions and whether it is successful or not.

**Test Cases**

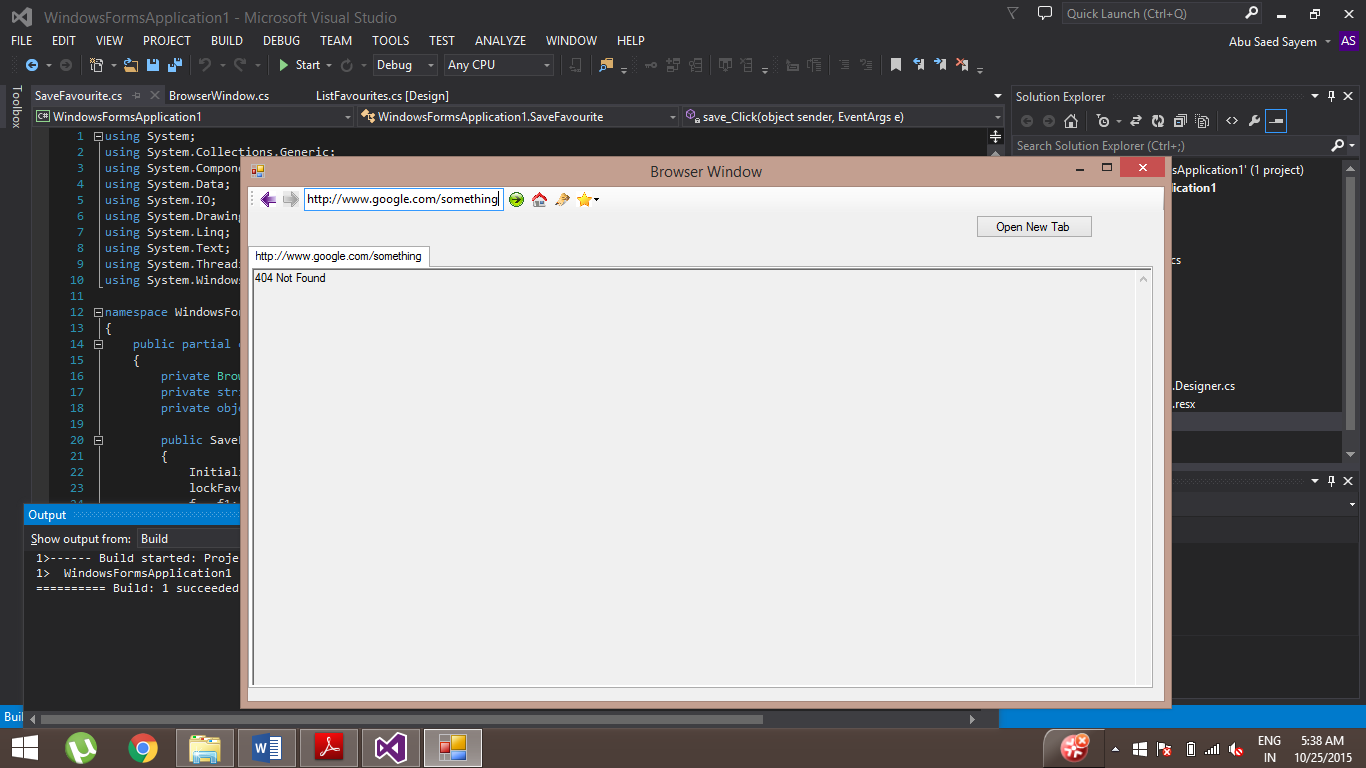
|  |  |
| --- | --- |
| 1. HTML code | HTML code is returned to the textbox at the selected tab |
| 1. 404 Not Found | If the web site is not found , string ‘404 Not found’ should be in the textbox at selected tab |
| 1. 403 Forbidden | If the user does not have access to a file, then string ‘403 Forbidden’ should be in the textbox at selected tab. |
| 1. 400 Bad Request | Returns a string ‘400 Bad Request’ when the browser client sends a request that does not follow the HTTP RFC format for HTTP requests like GET, POST, etc |
| 1. Changing homepage | User changes the homepage and this new homepage loads on the next browser startup |
| 1. Saving a Favourite | User can save a favourite |
| 1. Deleting a favourite | User can delete a favourite after selecting it and clicking on the ‘Delete’ Button |
| 1. Editing a name of a favourite | User can edit a favourite after selecting it, typing the name in the textbox and clicking on the ‘Edit’ Button |
| 1. Double clicking on a favourite | User can double click on a favourite and this link will appear in the current tab of browser |
| 1. Double clicking on a history | User can double click on a history and this link will appear on the main browser window |
| 1. Go to previous page | User can visit the previous page after clicking on the ‘Back’ Button |
| 1. Go to next page | User can visit the next page if it exists after clicking the ‘Forward’ Button |
| 1. Visiting links on different tabs | User is able to visit links in different tabs |

**Test case 1- HTML code**



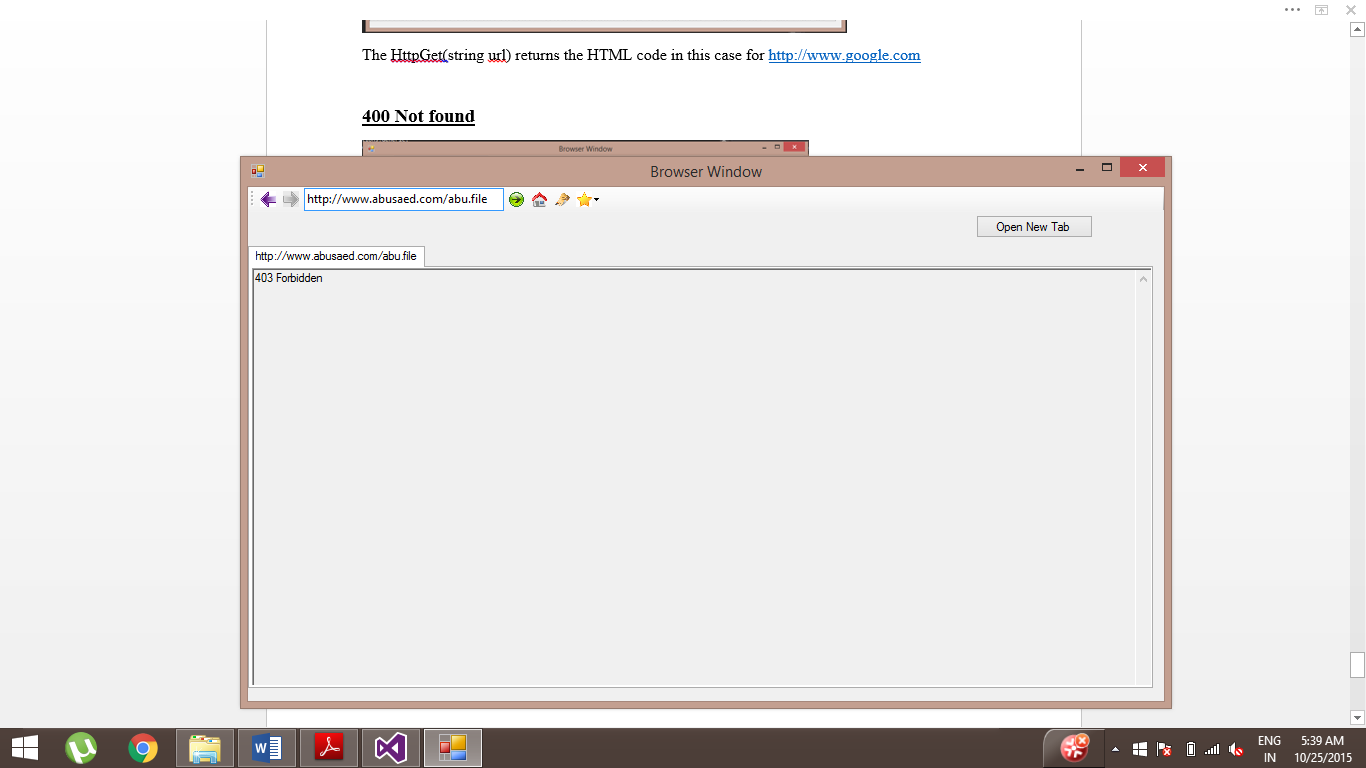
The HttpGet(string url) is implemented in ‘Url.cs’ returns the HTML code in this case for <http://www.google.com>

**Test case 2 - 404 Not found**



Google does not have a directory or a page with <http://www.google.com/something>. . That is why the response.statusCode is returning ‘400’. Then, I convert the response to a string and append ‘Not found’.

**Test case 3 - 403 Forbidden**



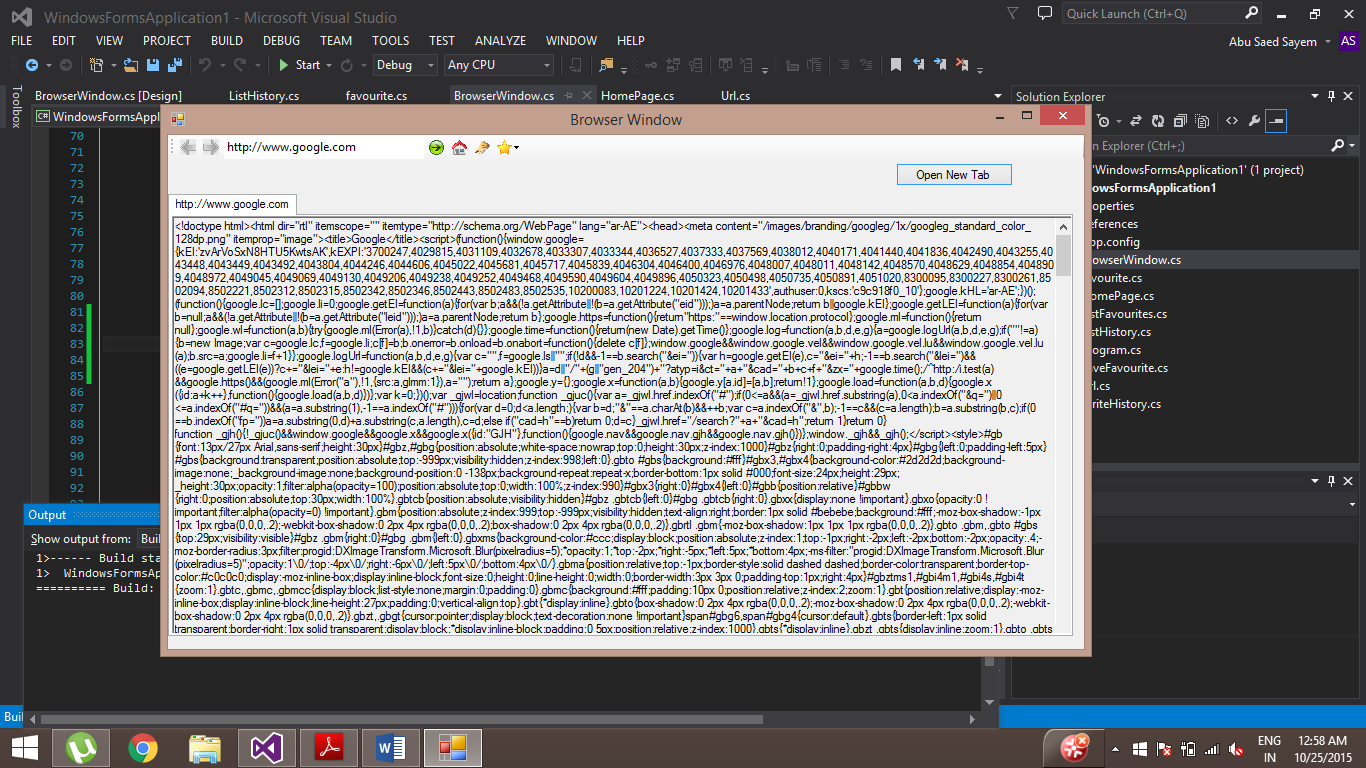
Only root has access to this abu.file on the server and not anyone else. That is why the response.statusCode is returning ‘403’. Then I convert the response to a string and append ‘Forbidden’.

**Test case 4 - 400 Bad Request**

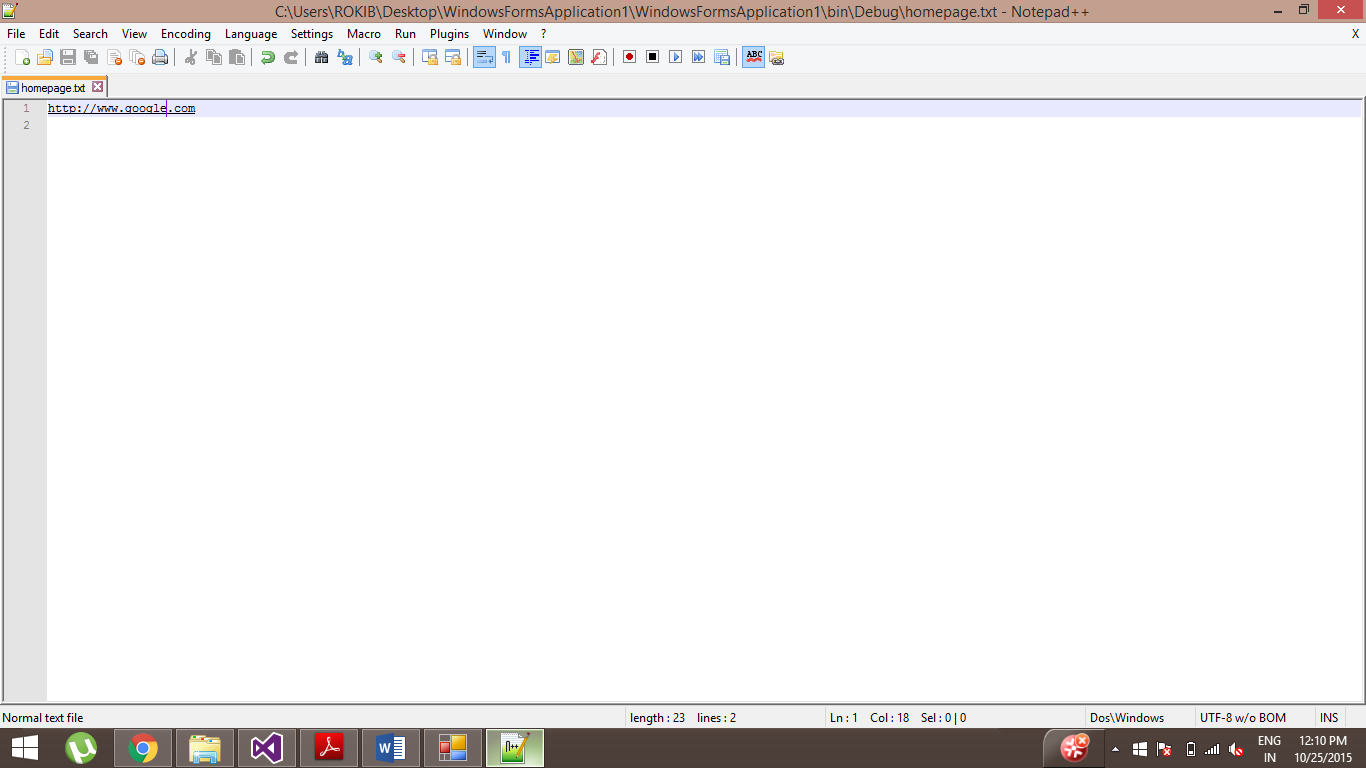
I have implemented to handle this case in ‘Url.cs’. The bad request is reported by the HTTP server when the browser client sends a request that does not follow the HTTP RFC format for HTTP requests like GET, POST etc.

**Test case 5 – Changing Homepage**

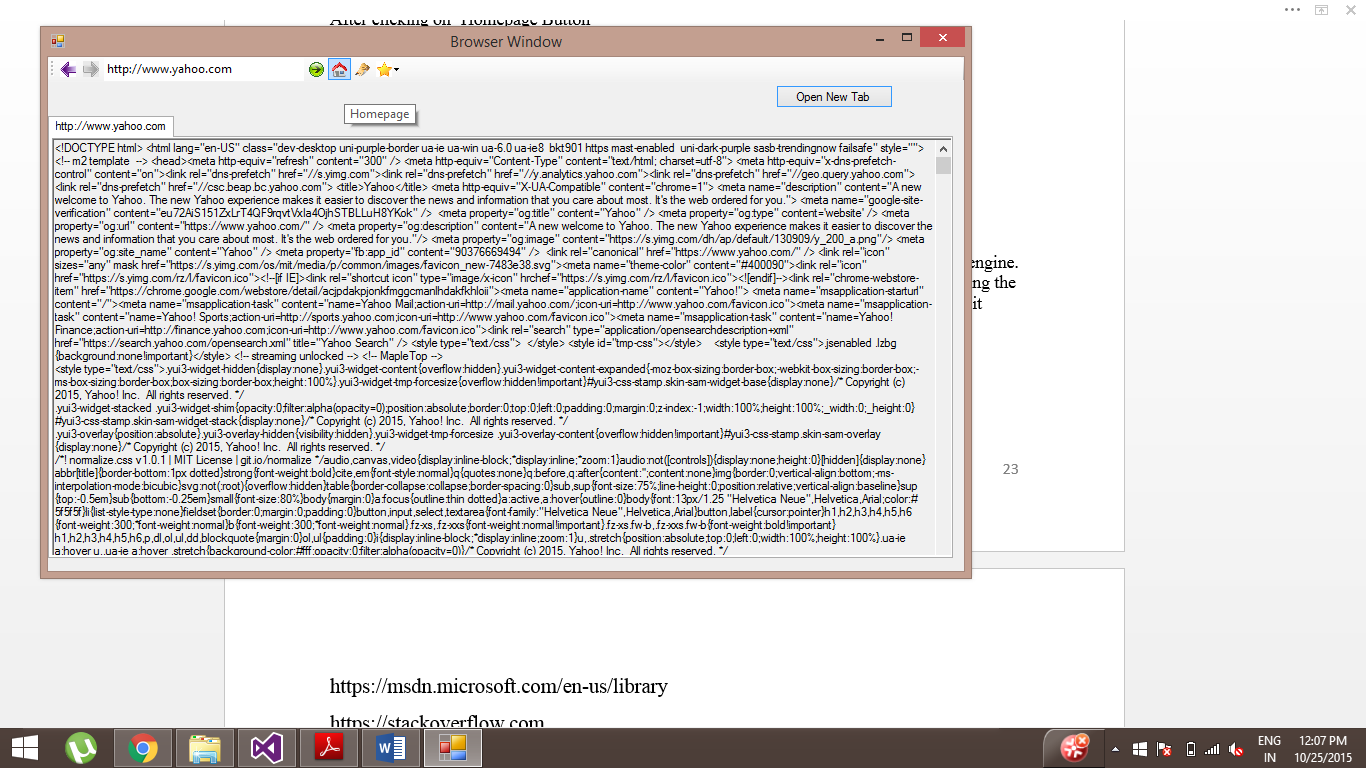
Initial startup



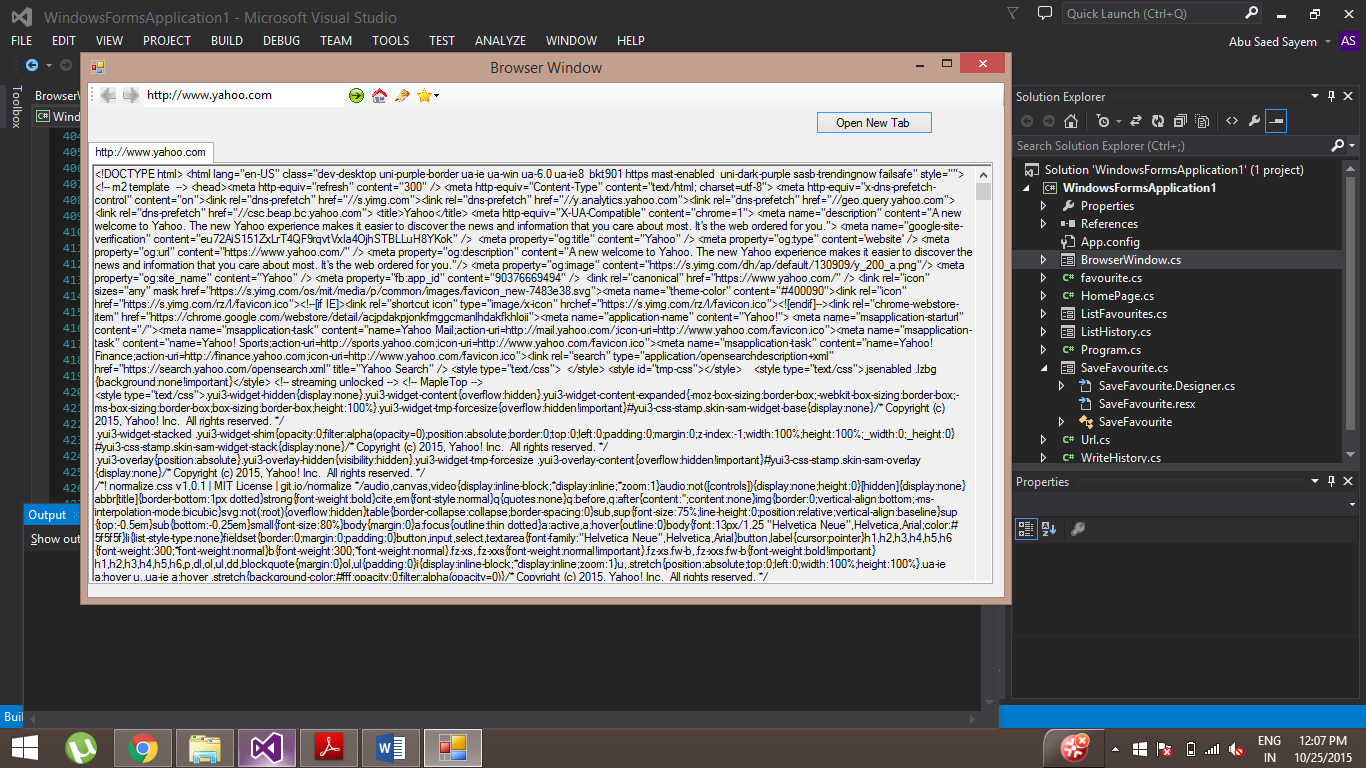
Below is the text file where the homepage link is saved



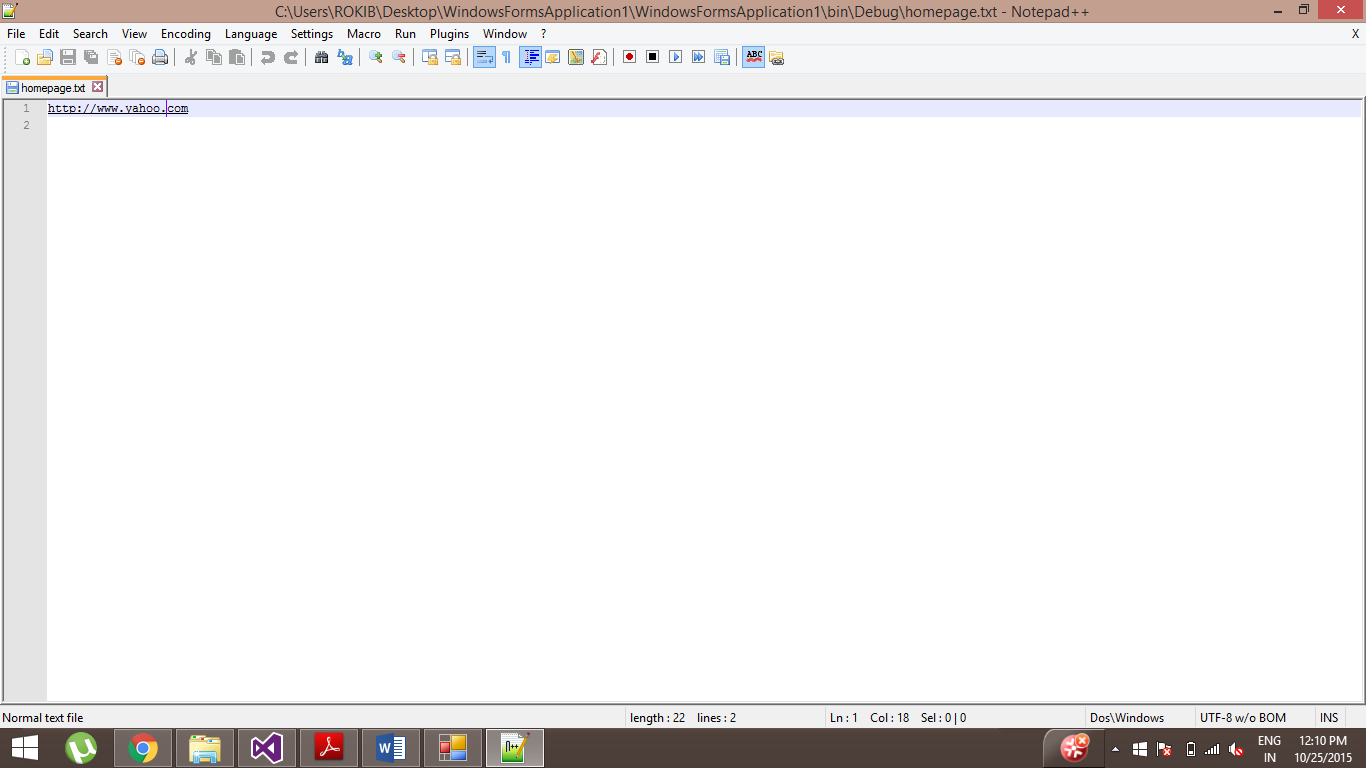
After clicking on ‘Homepage Button’



After closing and then restarting the browser again

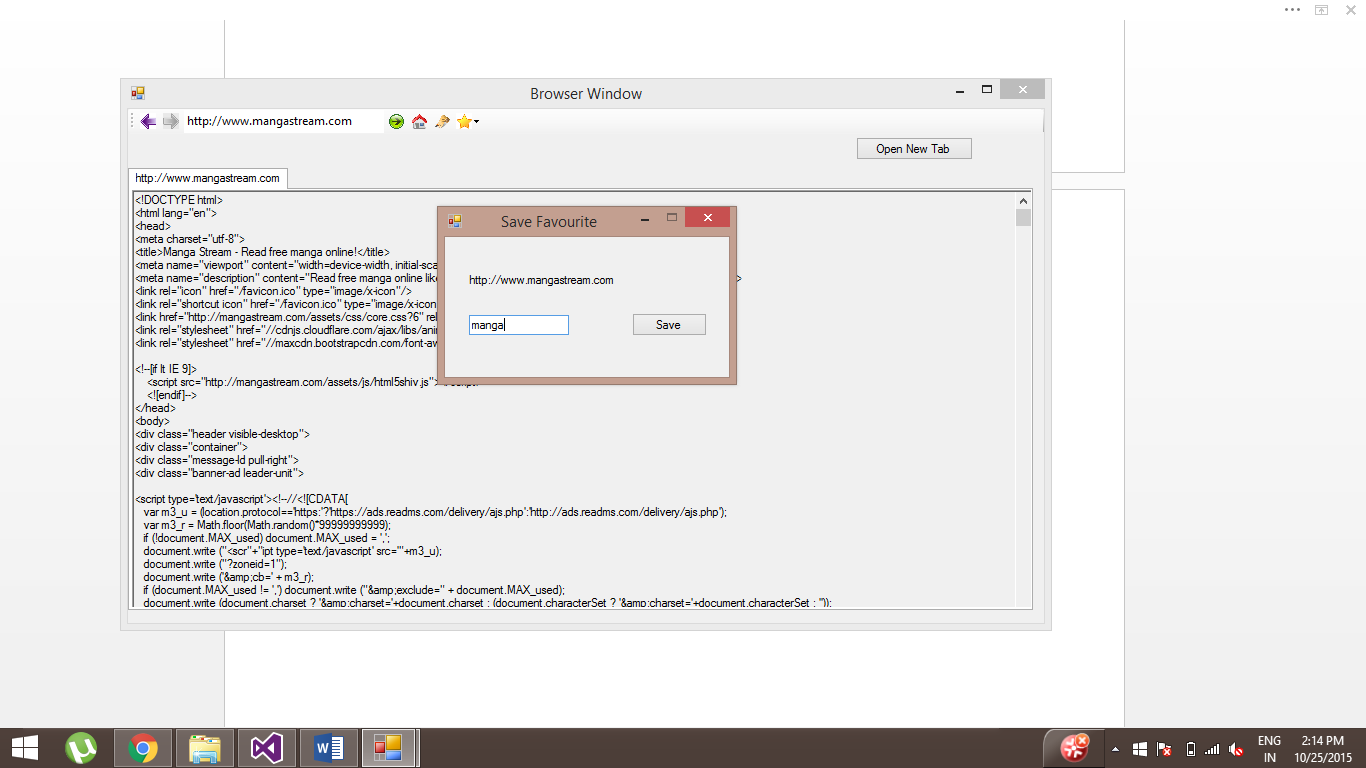


Below is the updated text file where the homepage link is saved.

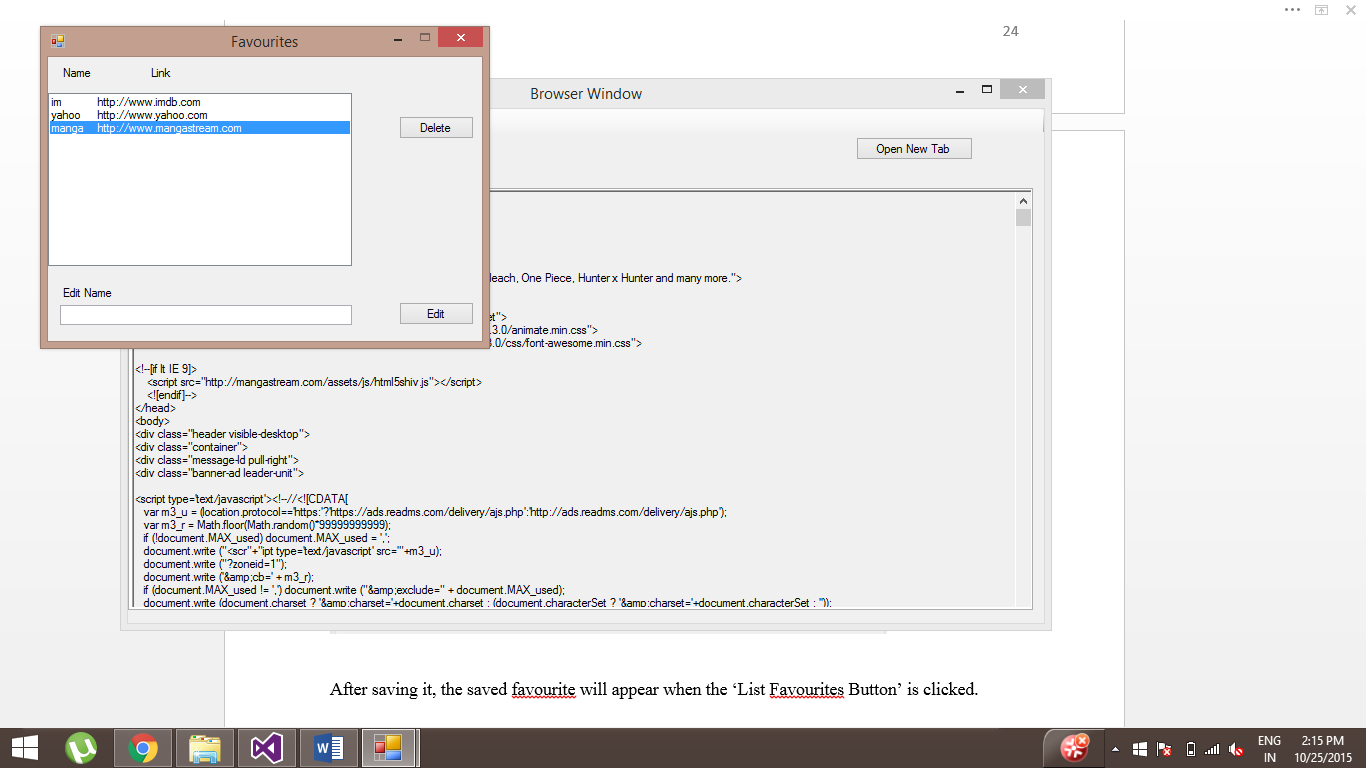


**Test case 6 – Saving a Favourite**

After clicking on the ‘Save Favourite Button’, the following window appears

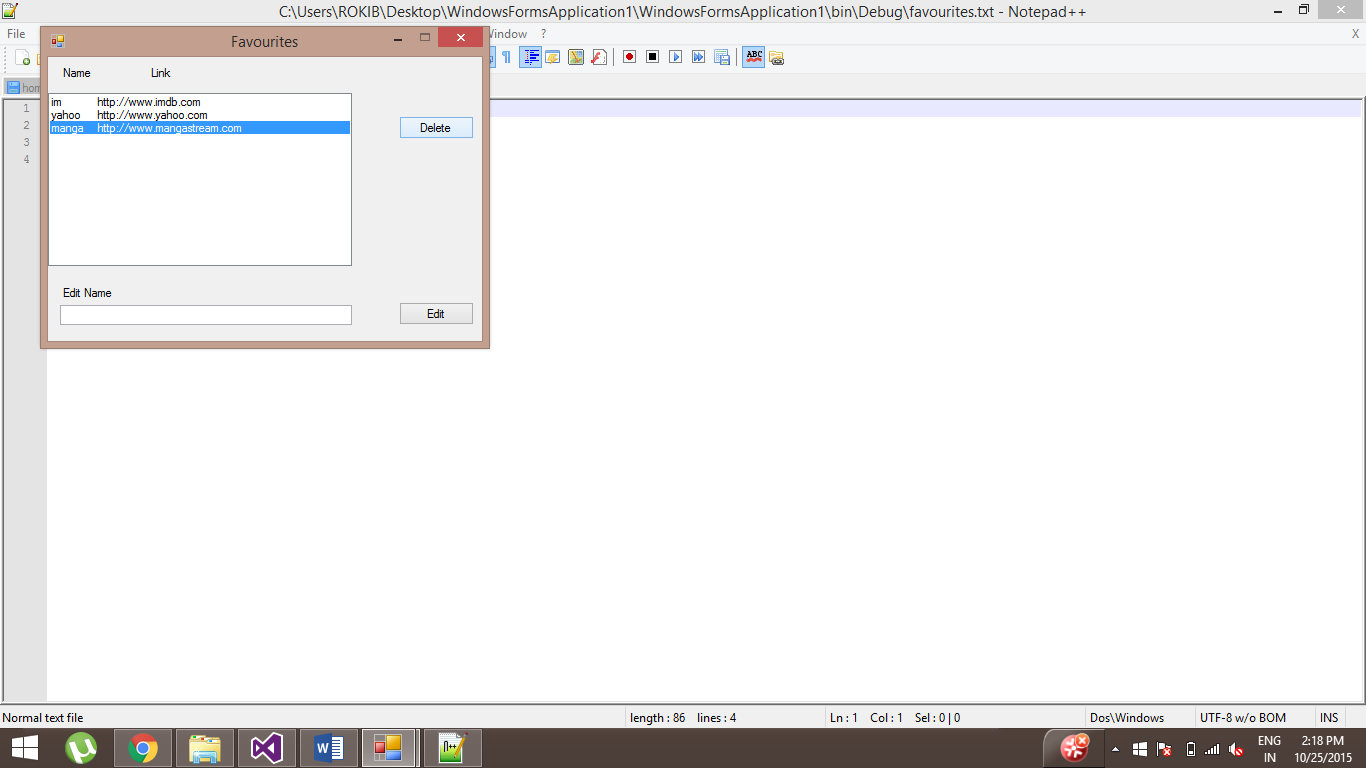


After saving it, the saved favourite appear s when the ‘List Favourites’ Button is clicked.

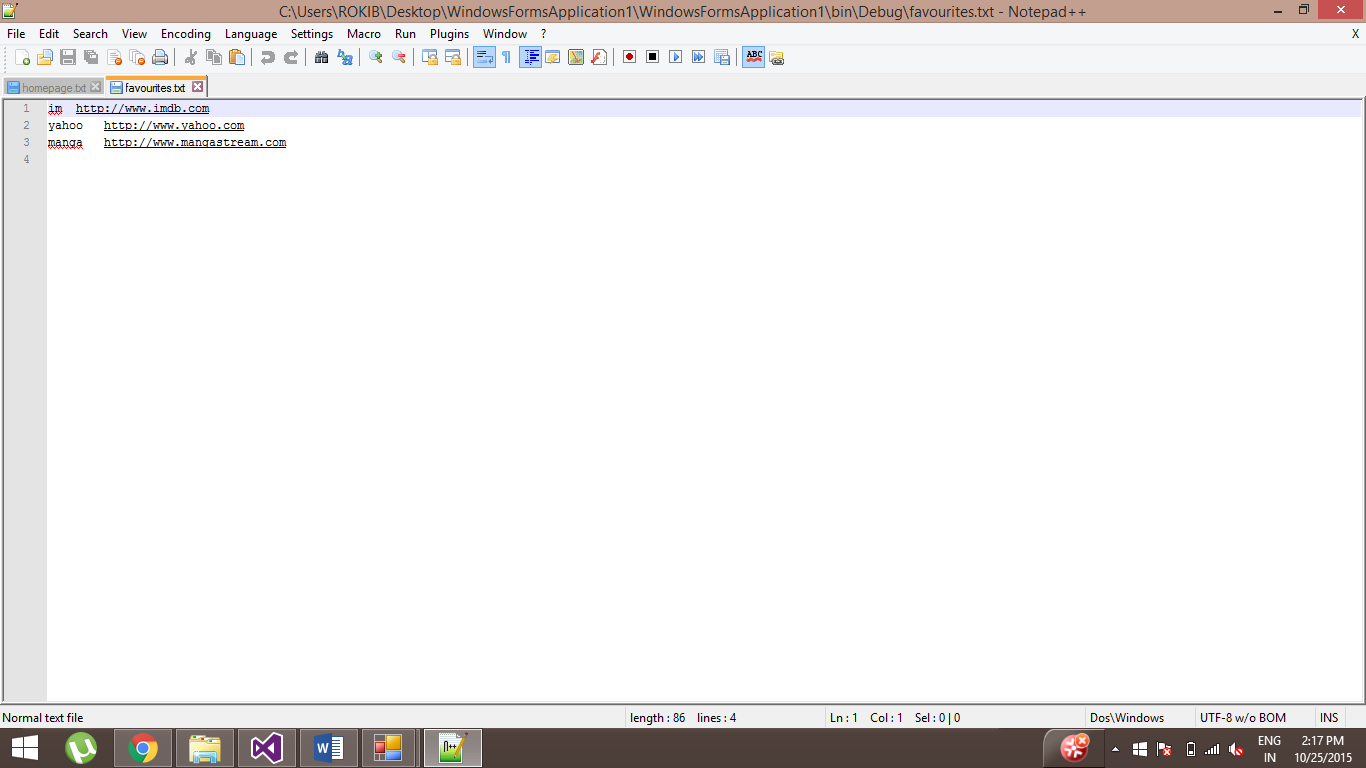


**Test case 7 – Deleting a Favourite**

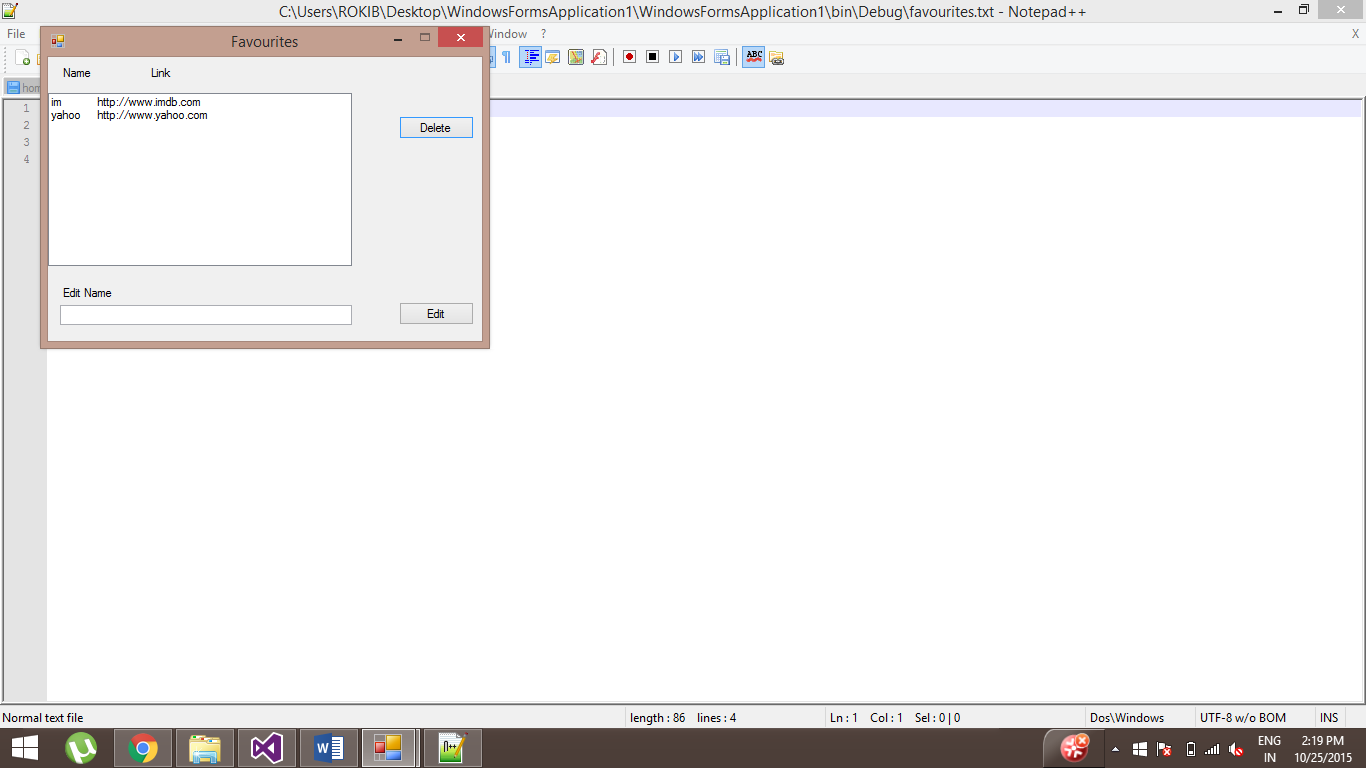
When a favourite is selected and then the user clicks on the ‘Delete’ Button; the favourite will be deleted from the list.

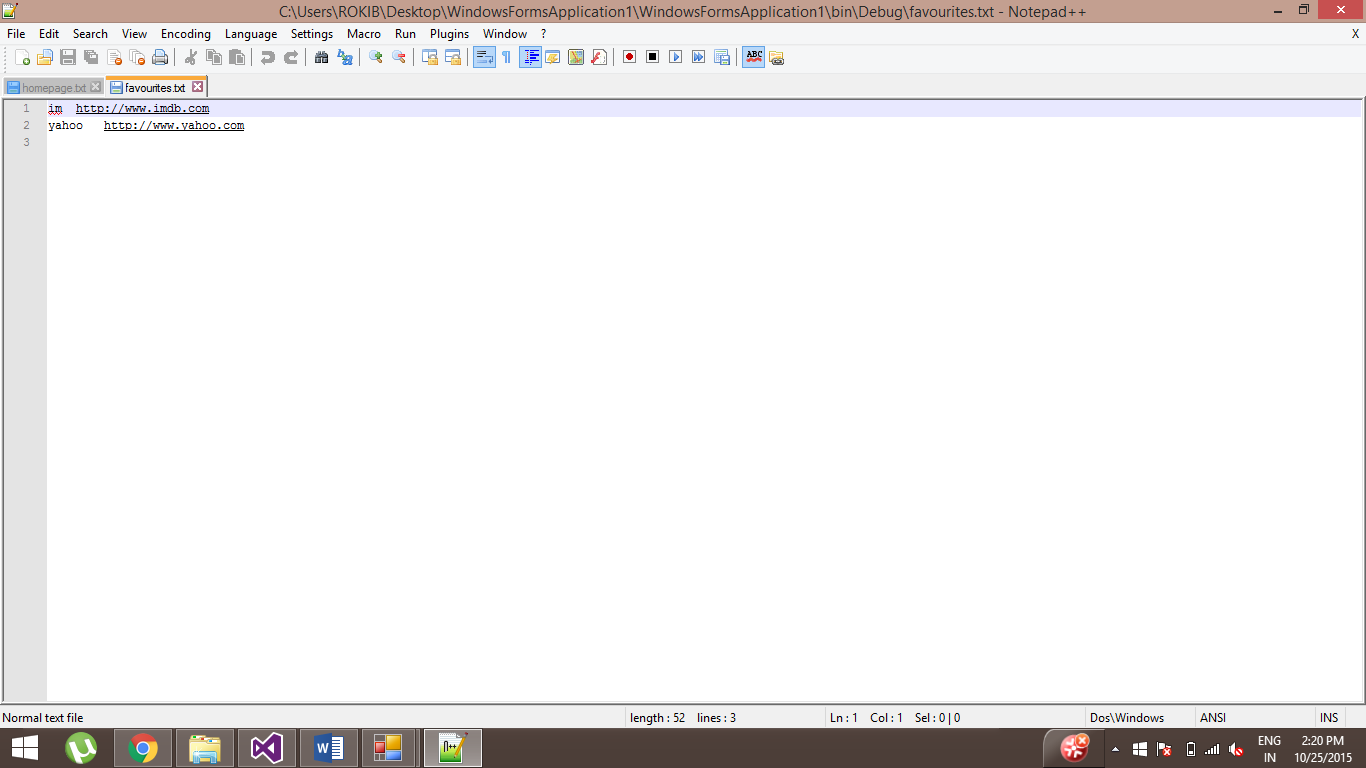


Below is the text file where the favourites are stored.



After clicking the ‘Delete’ Button, the following action takes place

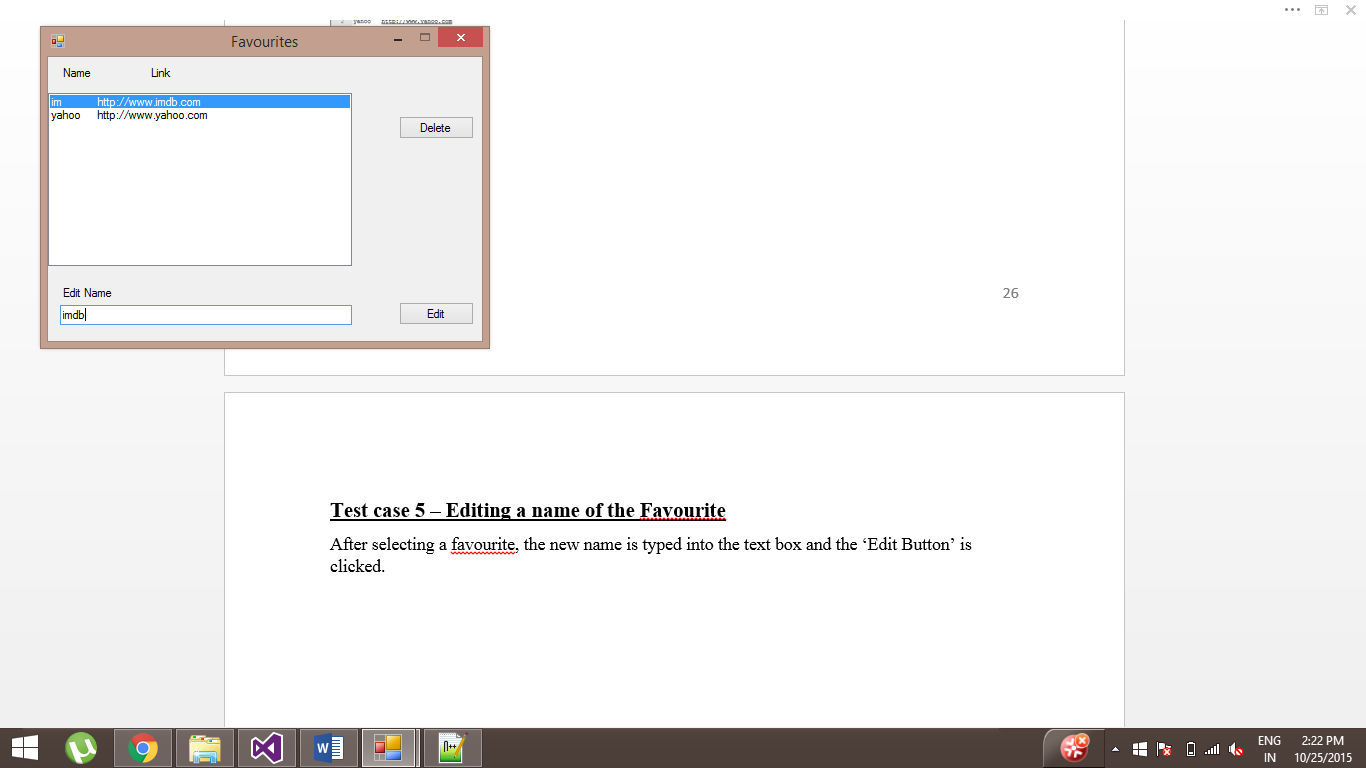




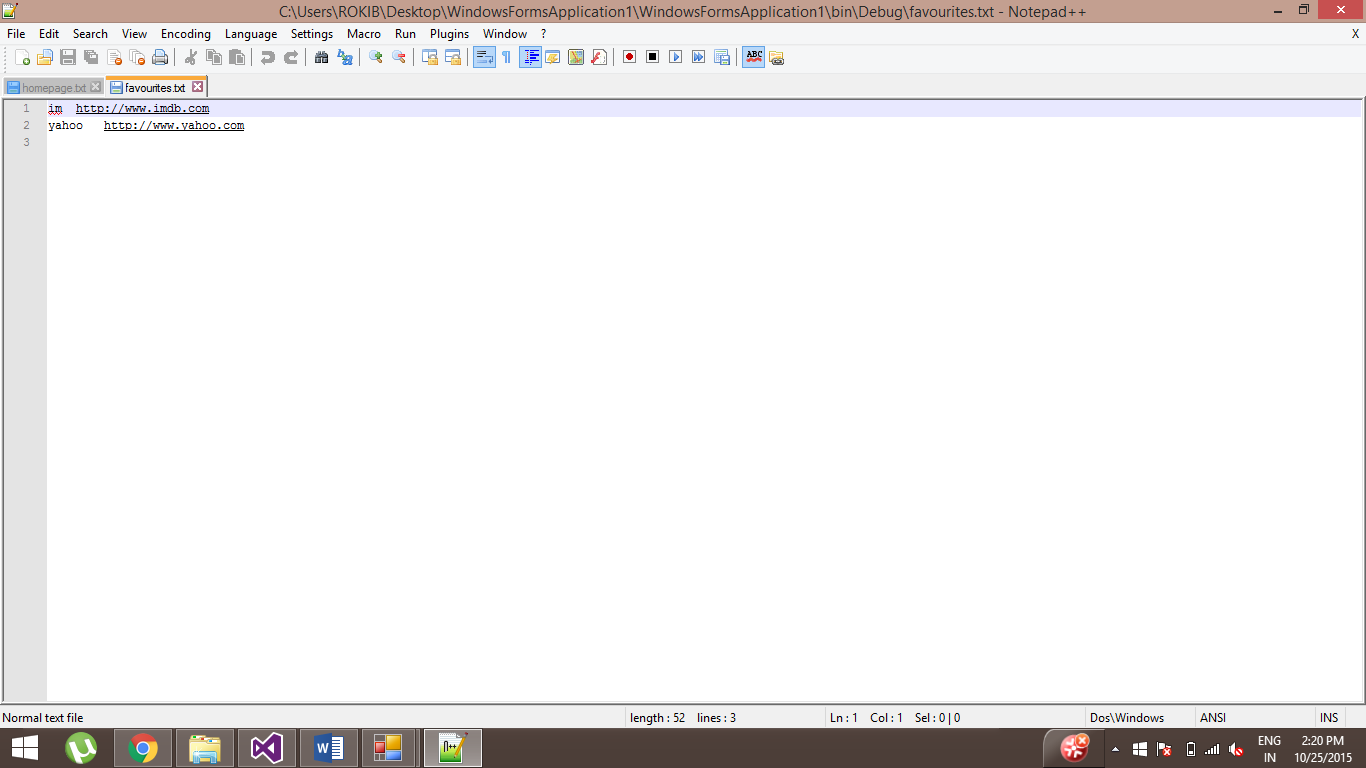
**Test case 8 – Editing a name of the Favourite**

After selecting a favourite, the new name is typed in the text box and the ‘Edit’ Button is clicked.

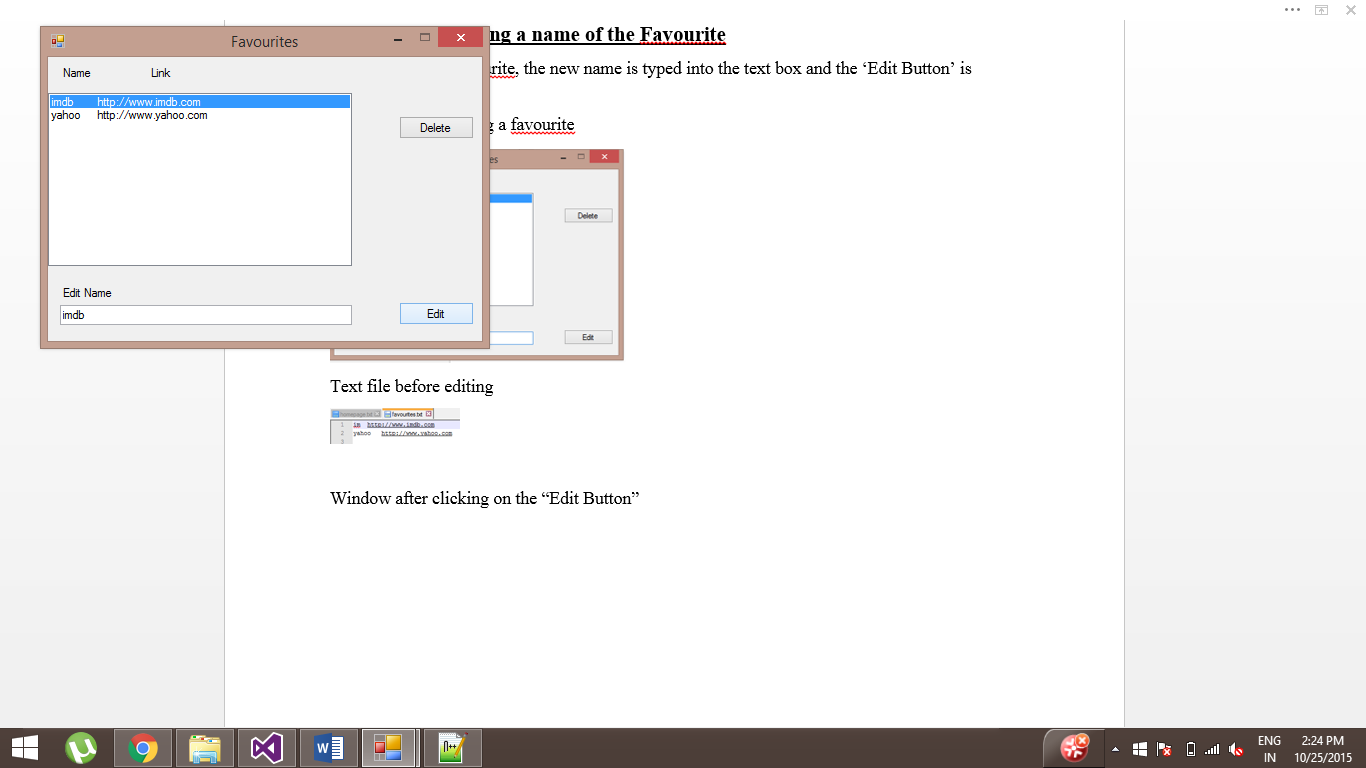
Below is the Window before editing a favourite

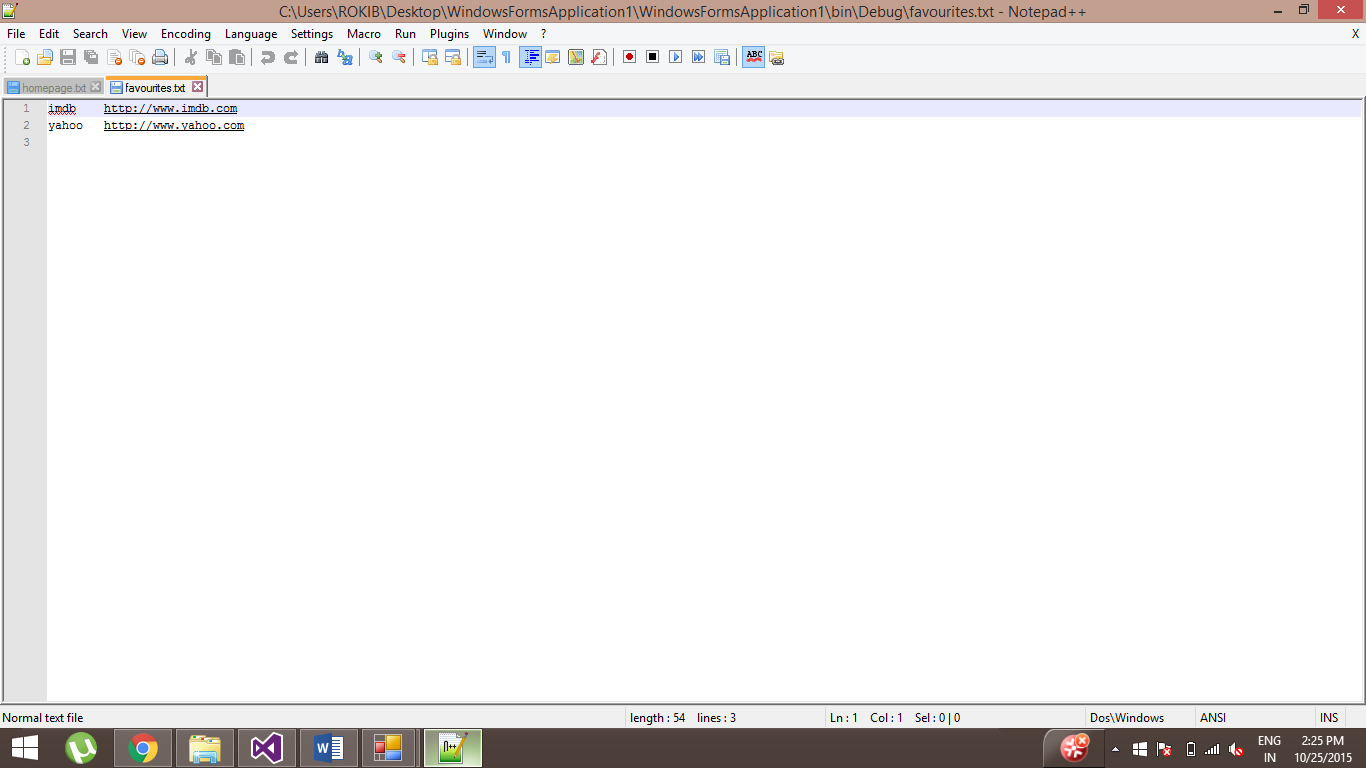


Text file before editing



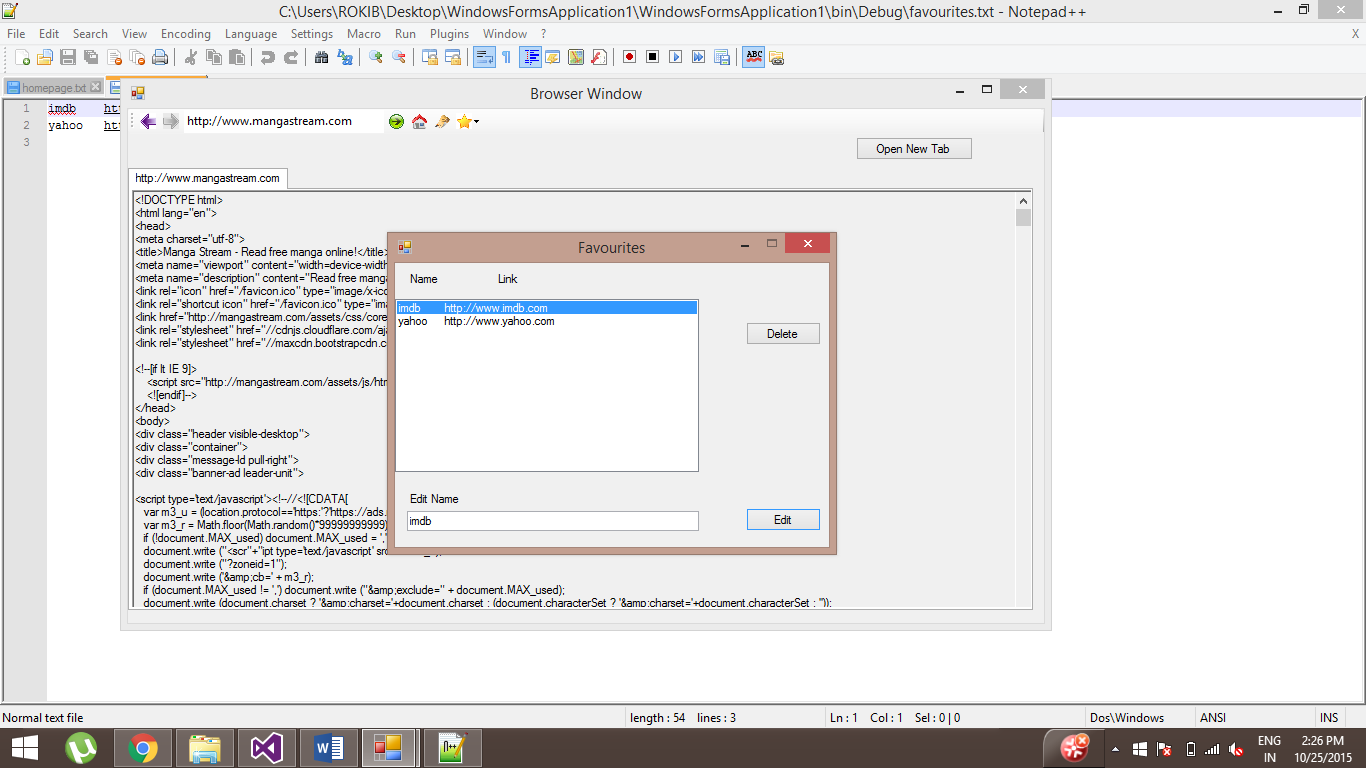
Window and text file after clicking on the ‘Edit’ Button



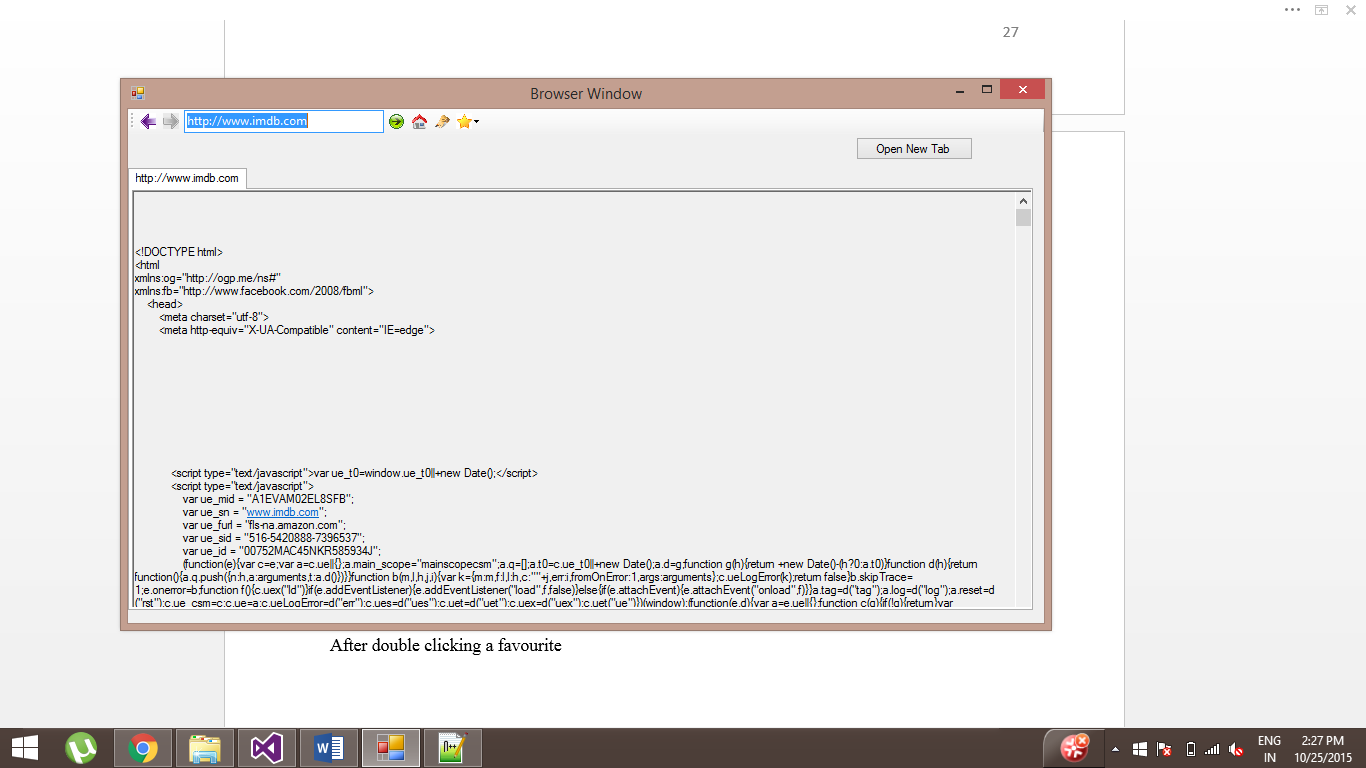


**Test case 9 – Double clicking on a favourite**

Initial window

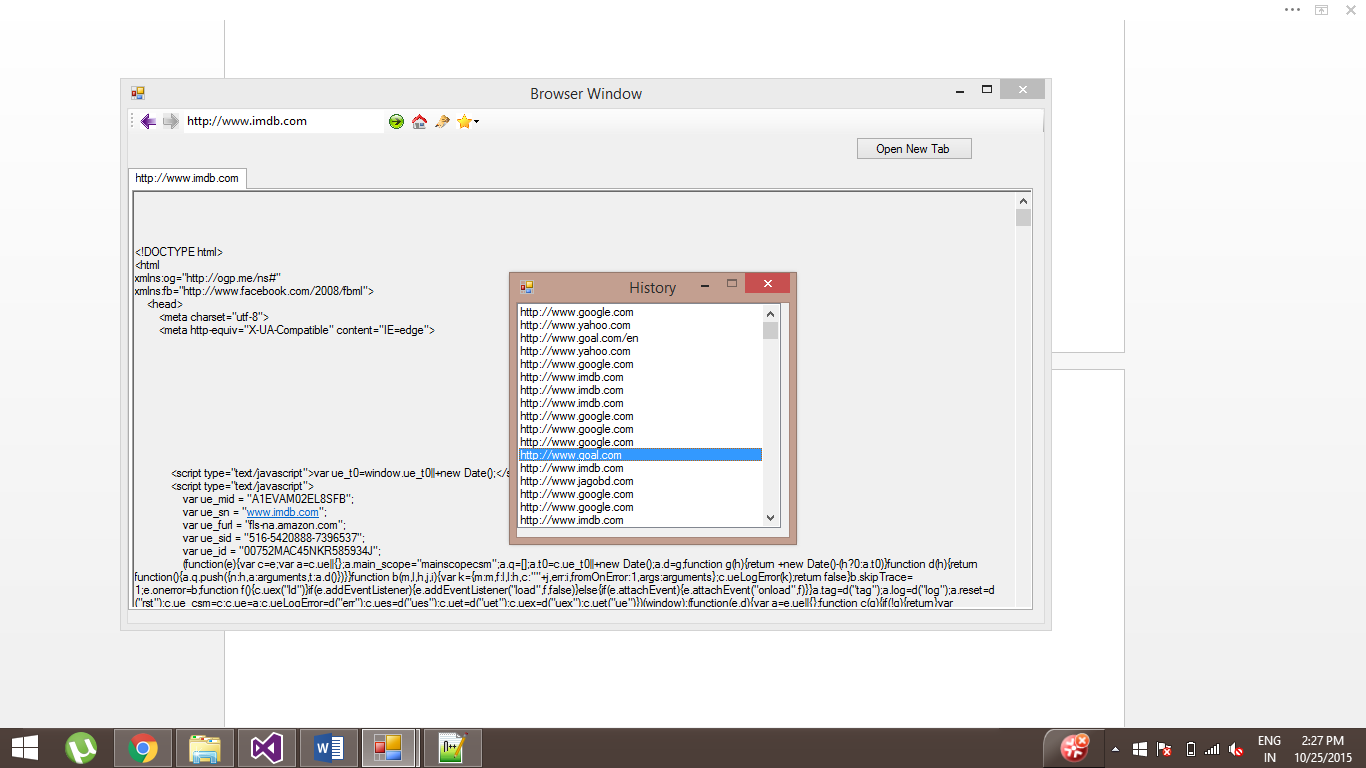


After double clicking on a favourite

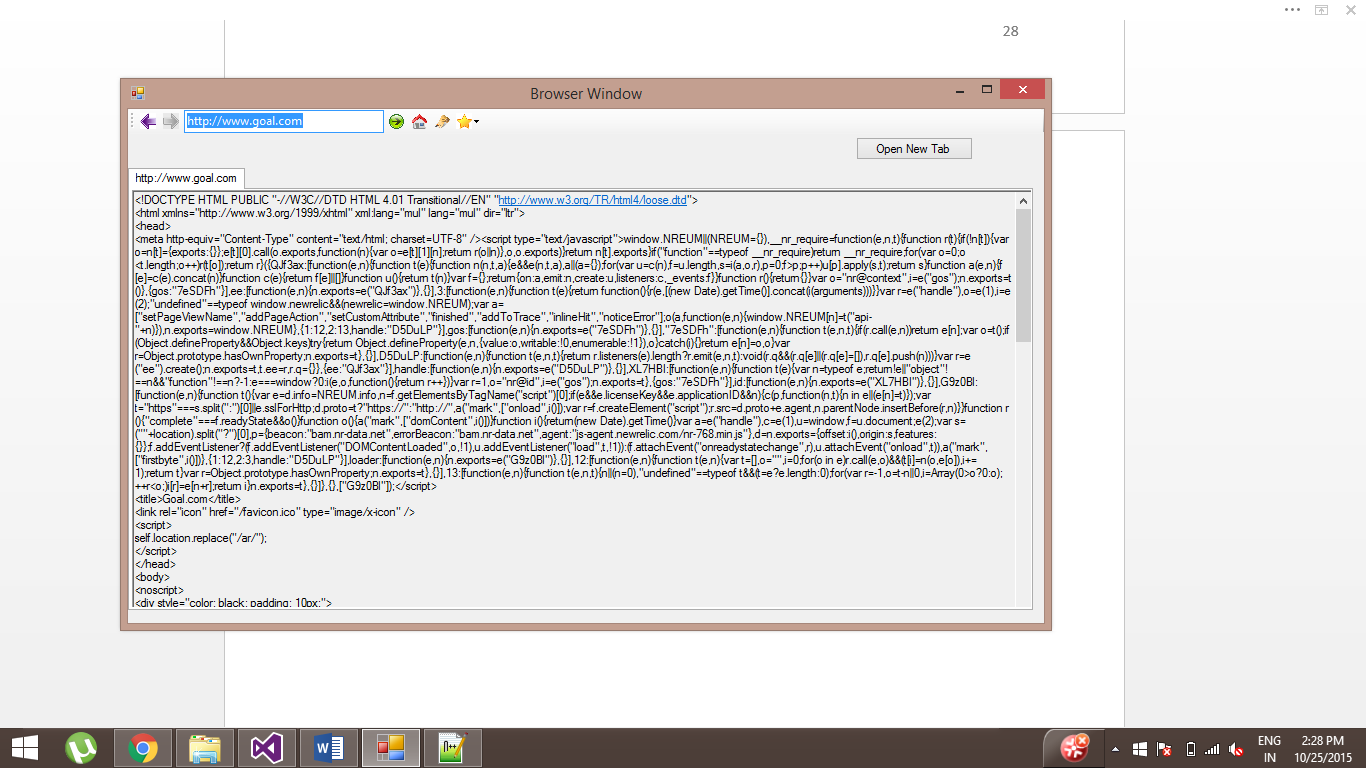


**Test case 10 – Double clicking on a history**

Initial Window

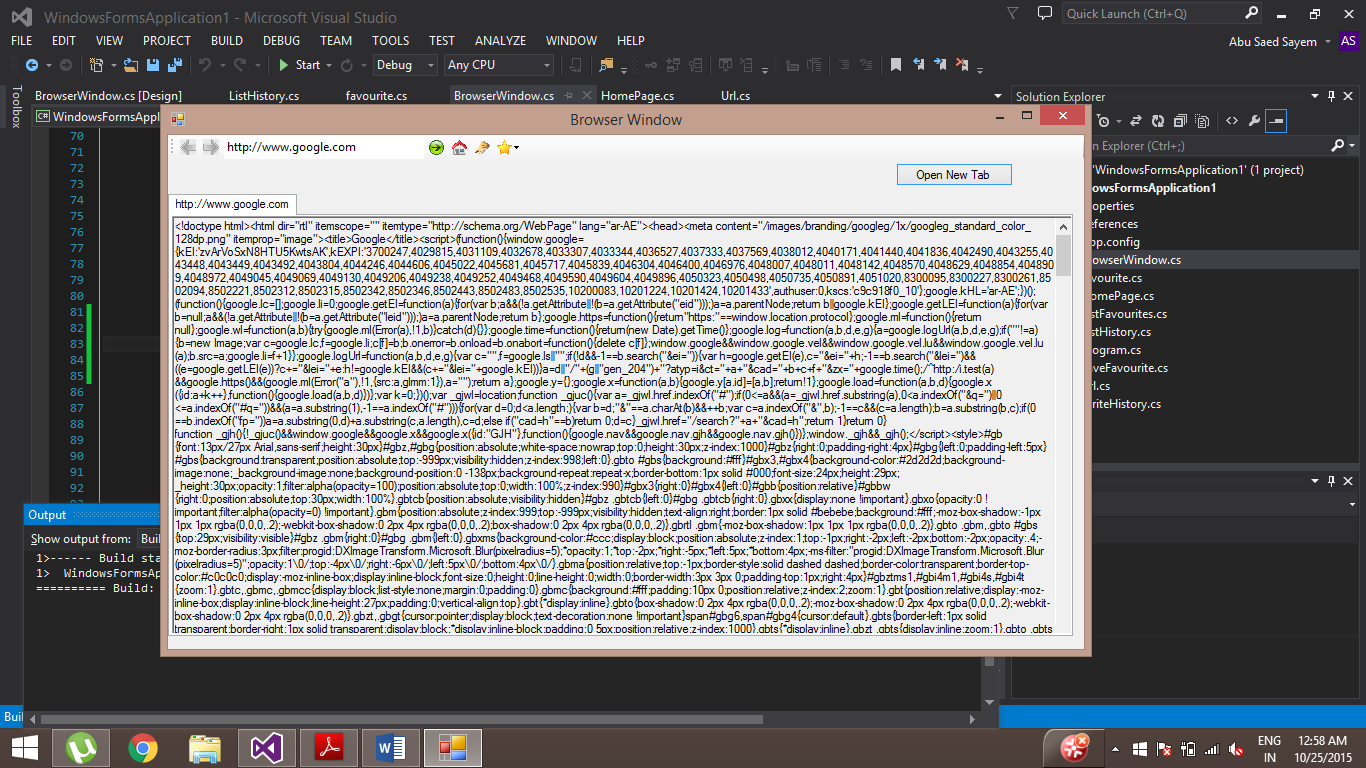


After double clicking



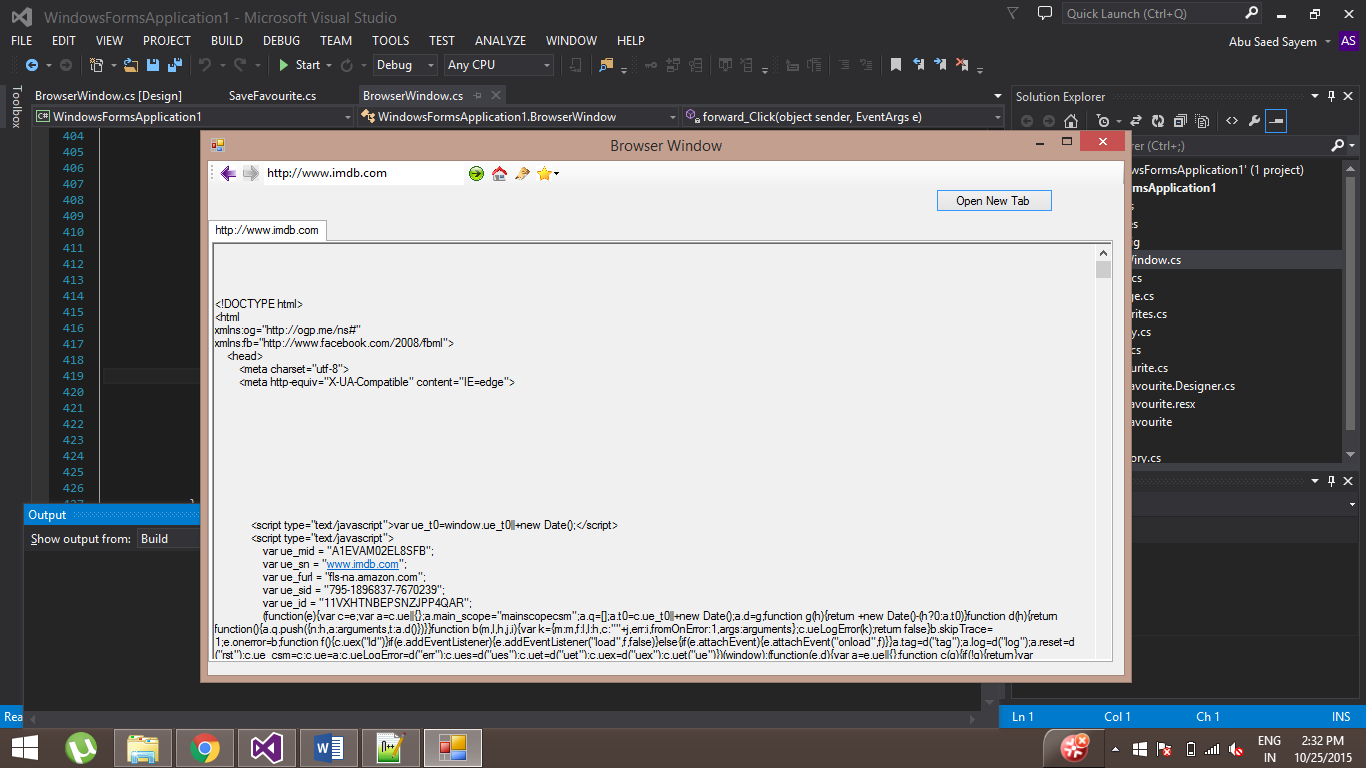
**Test case 11 – Go to previous page**

Initial Window

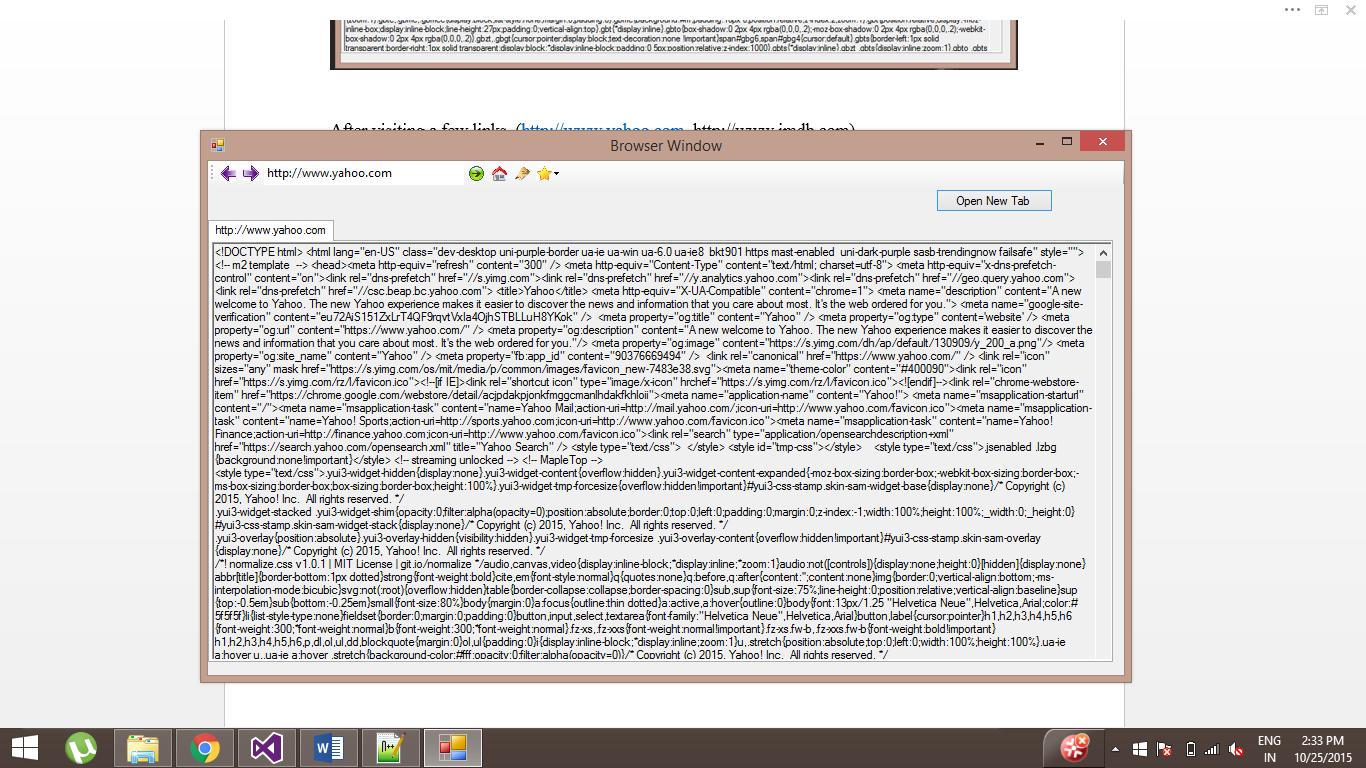


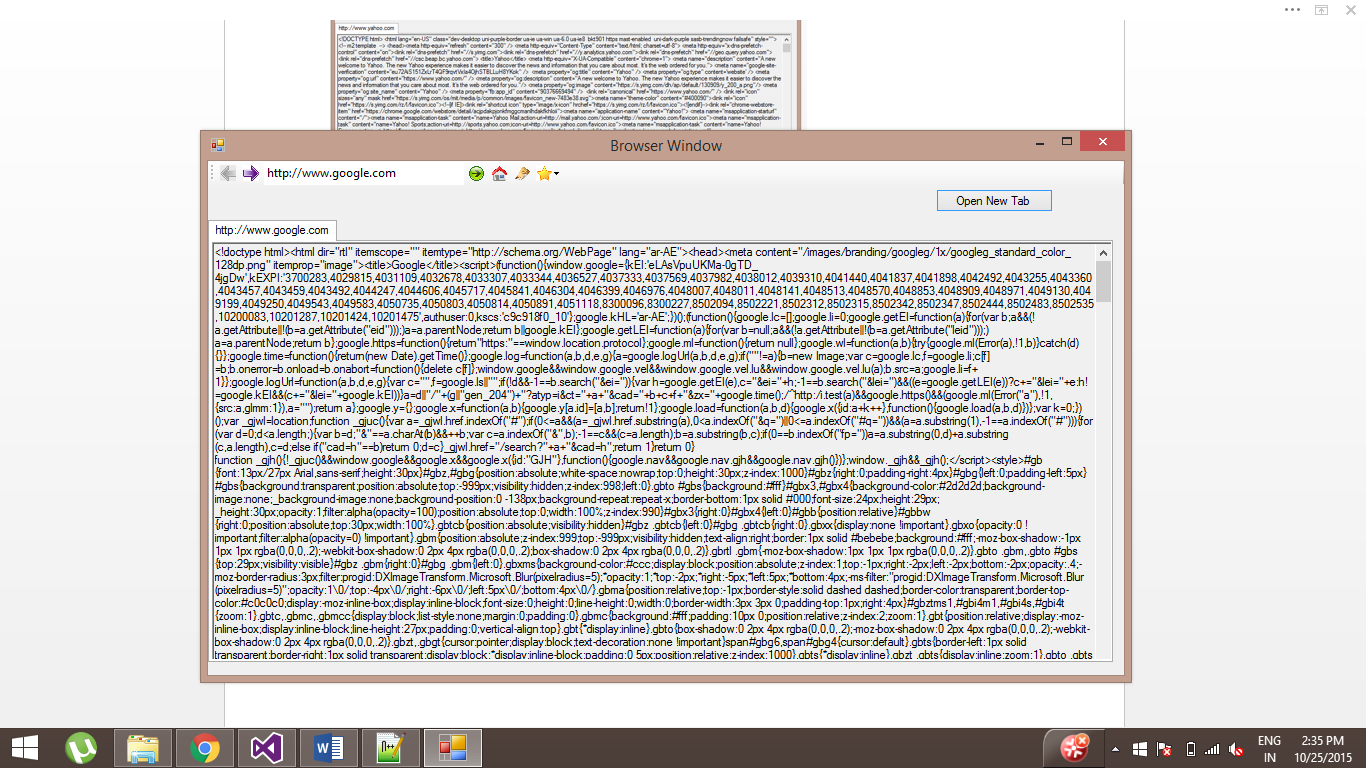
After visiting a few links (<http://www.yahoo.com>, http://www.imdb.com)

The following screenshots shows how the user can navigate through pages visited.



After clicking the ‘Back’ Button, it will display the previous page

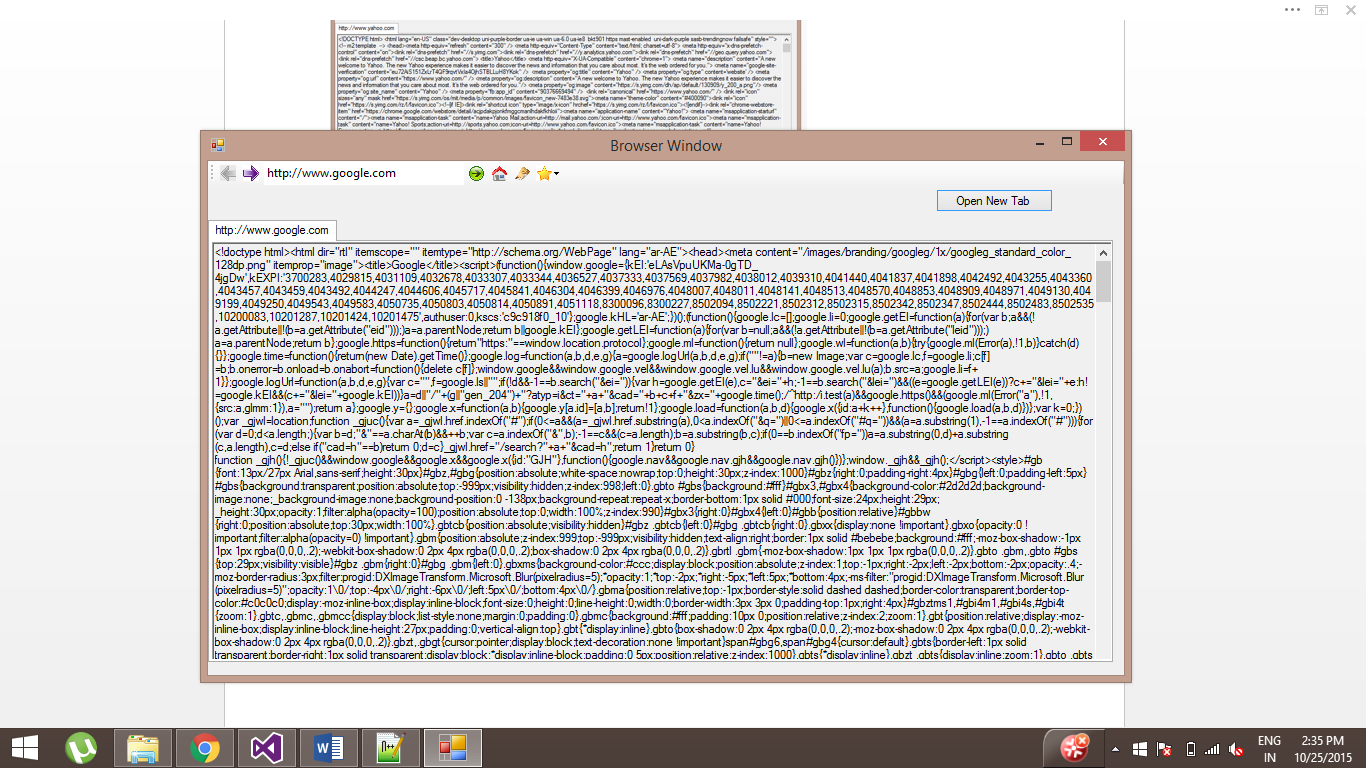




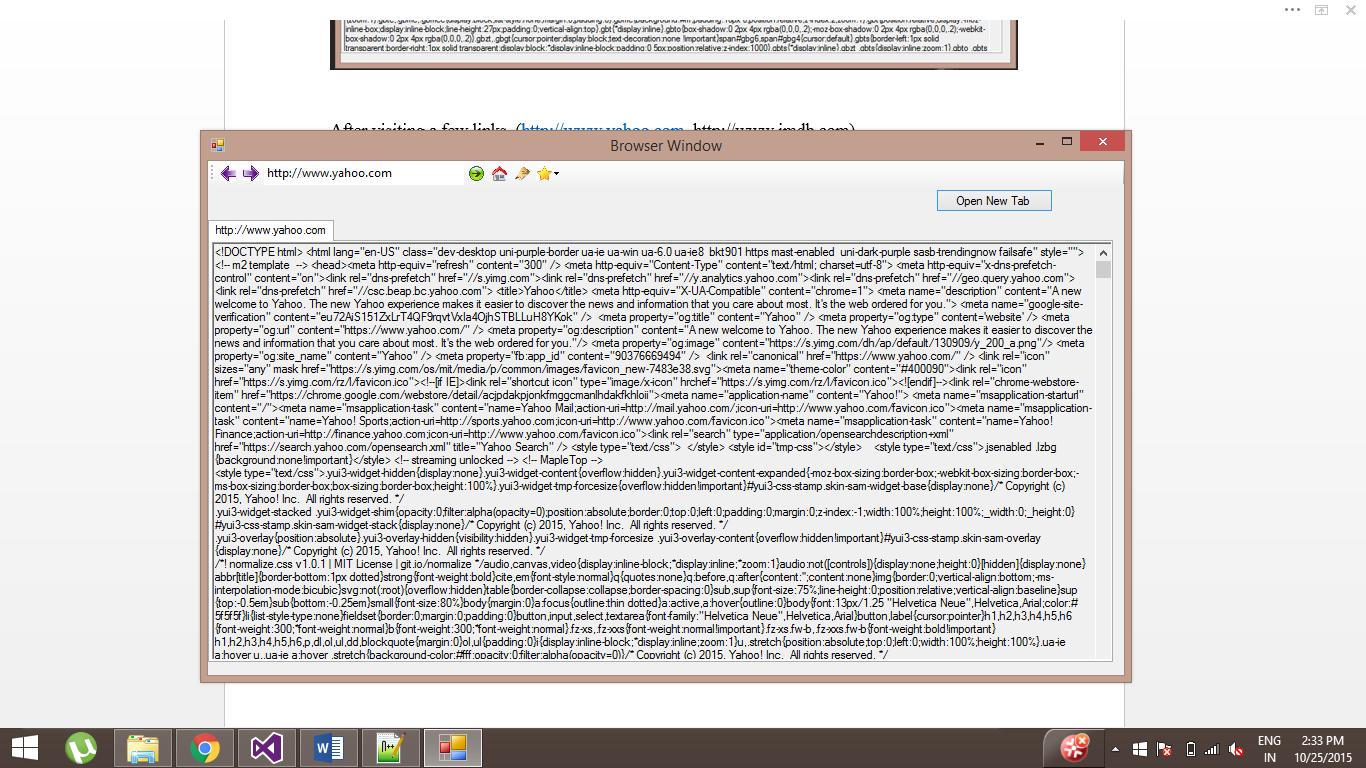
**Test case 12 – Go to next page**

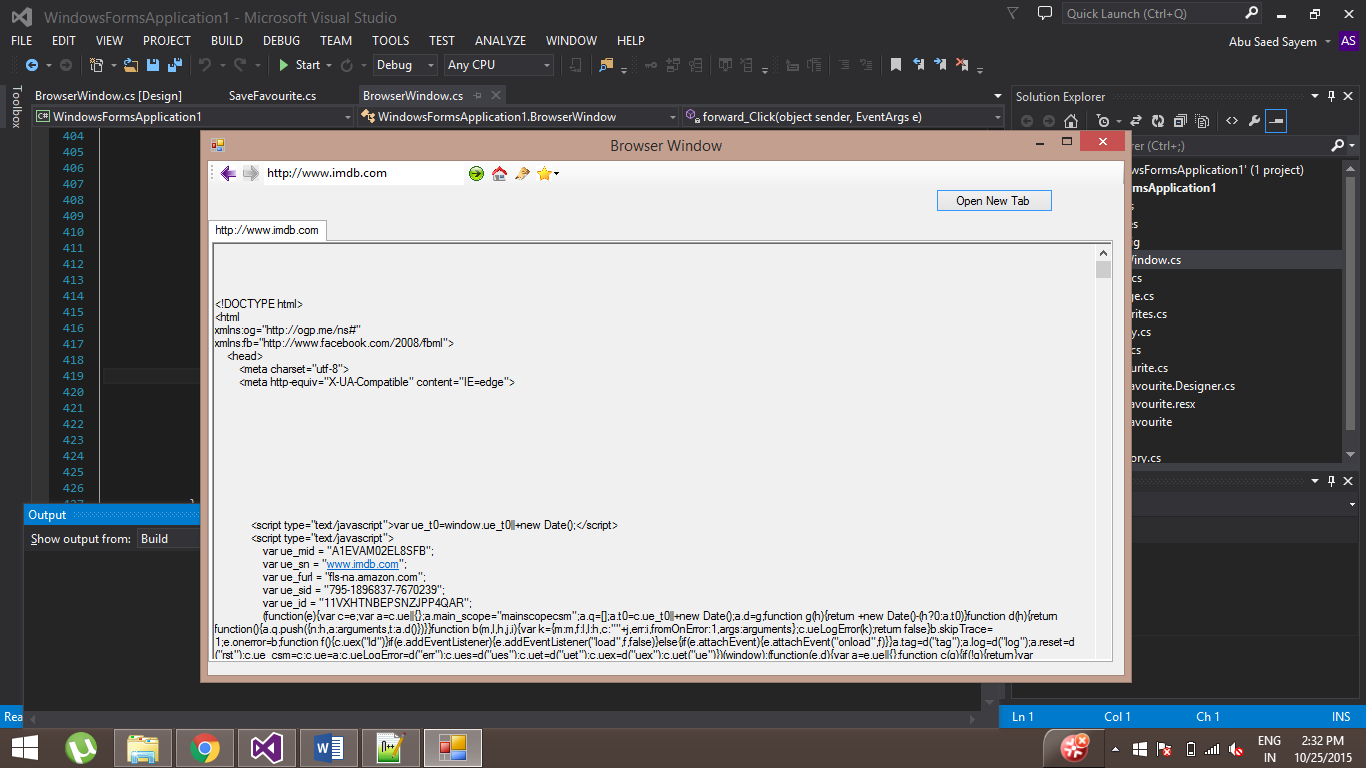
Clicking on the ‘Forward’ Button

The following screen shot takes place by pressing the ‘Back’ Button to the first page

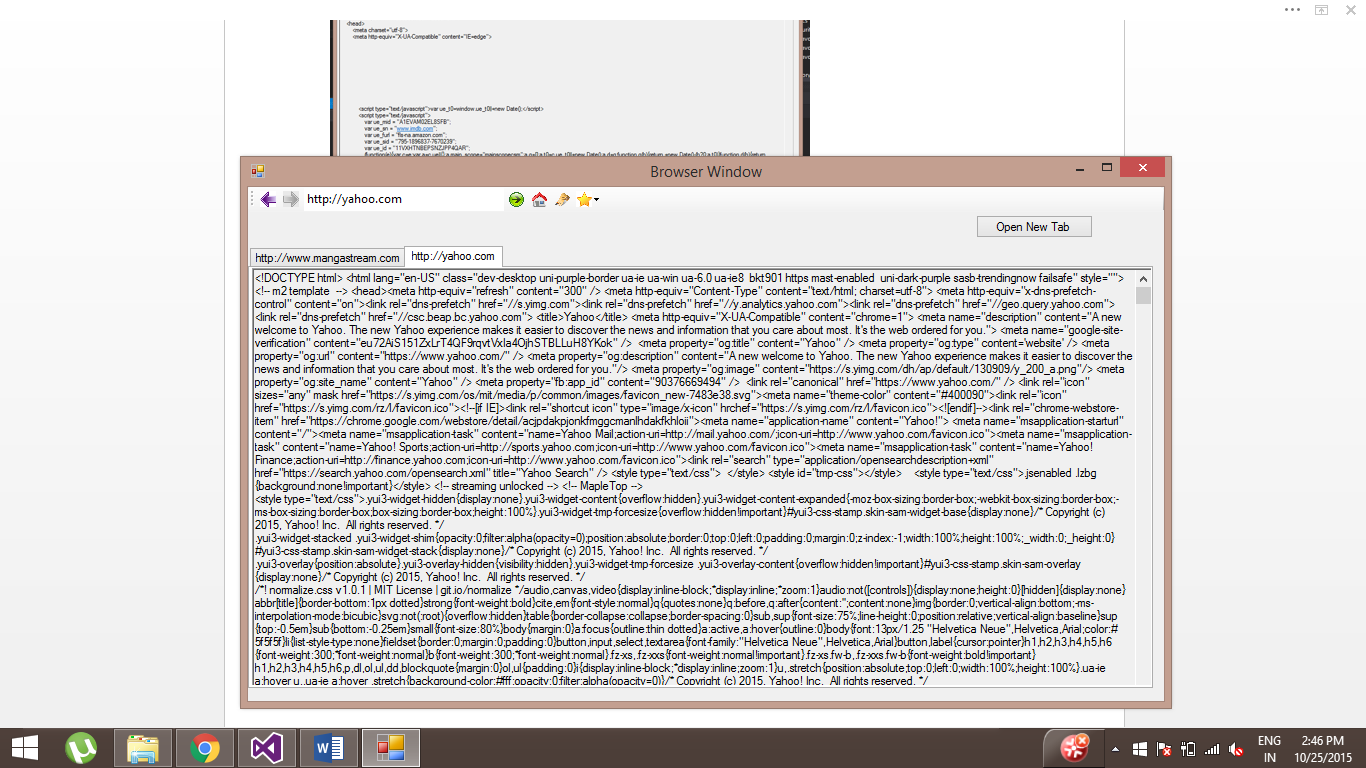


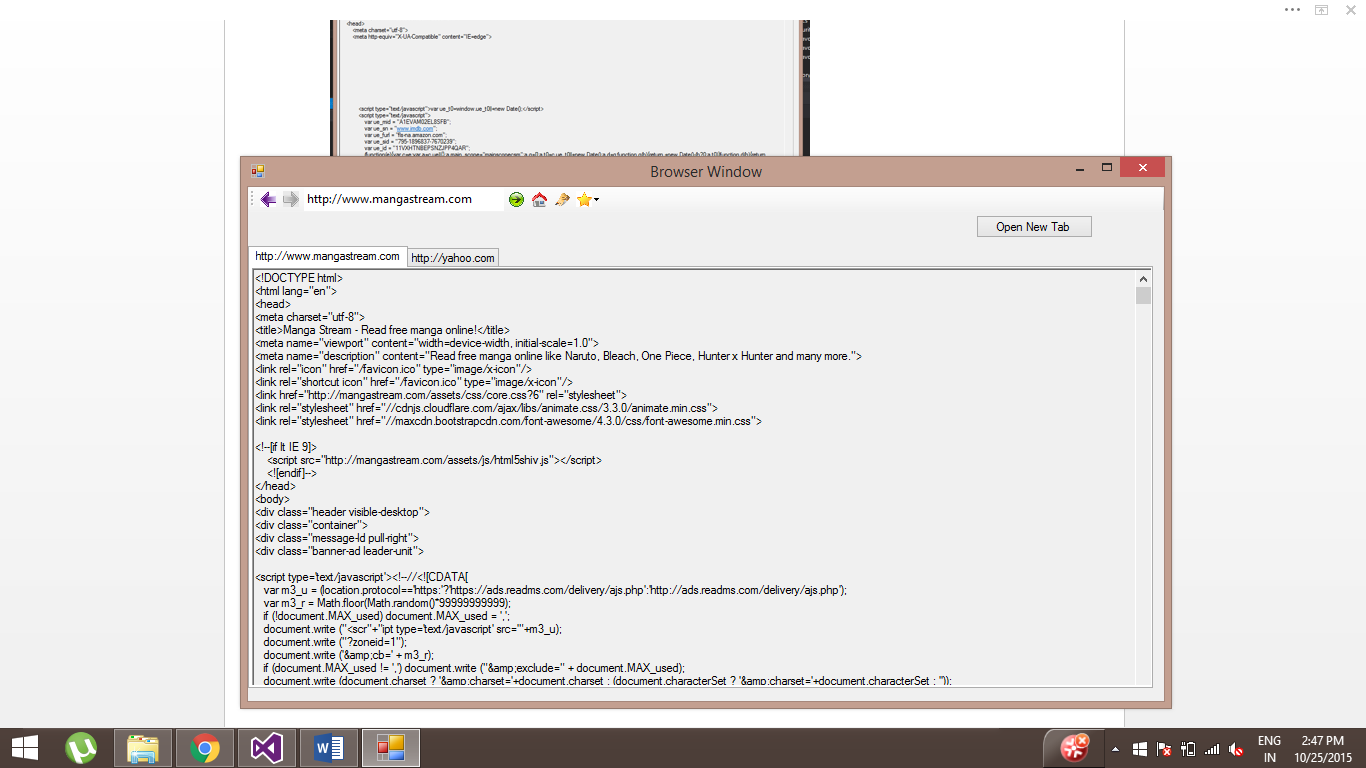
The following screenshots are after clicking on the ‘Forward’ Button





**Test case 13 – Visiting links on different tabs**





**Conclusion**

If time permitted then I would have liked to try to render the html code using a free source rendering engine. One of the things that I like about my application is the implementation of the different tab pages by using the different data structures. It took me a while to implement this feature and that is why I am proud of it. The web browser coursework gave the opportunity to learn the C# programming language through implementation, familiarization with the .net framework and using multi-threading for browser-server communications.

**References**

https://msdn.microsoft.com/en-us/library

https://stackoverflow.com