

## Introduction

The purpose of this reports is to take the professor along in our efforts to make an original web browser on our own. Originally we gave each other different sections of the project to work on, but in the end, we all worked on the project together to finish the web browser we call Obsidian. During the project, we made several assumptions. We were originally going to use Qtwebkit, but after further researching, we instead went with using Qtwebengine to create the web browser. Another assumption we made is the fact that we thought that the project was going to be easier to combine each section with each other, but because it was harder than expected to do this, we decided to all work together on each part.

## Assignment Checklist

The following are the things we have implemented into the browser:

- The sending of HTTP request messages for URLs typed by the user will take the user to a desired website.
- The receiving of HTTP response messages and displaying the contents of said messages to the interface.
- There is a Home page whose URL is able to be edited. The Home page's URL will be loaded up when the browser starts up.
- There is a favorites section where the user can add a URL for a web page to a list of URL's. The user can add and edit a URL and then select a URL from the list and navigate to it just by pressing on it.
- There is a History section in which the browser maintains a list of URLs, corresponding to the web pages requested by the user. The user can also

navigate to previous and next pages and jump to a page by clicking on the links in the list of URLs in the History tab.

- The browser also includes Multi-threading within the tabs as if you are watching a YouTube video in one tab and navigate to a new page, the video will continue to play in the background and you will be able to listen to what is on the video.
- We also make use of menus with appropriate shortcut keys in order to increase accessibility. Such examples are the Browser, Favorites, and History buttons that navigate you to their respective places. There is also the button with the <3 that adds the web page that you are currently on to the list of favorites.

### Design Considerations

The Obsidian browser interface was designed to be simple. At first we were going to make a more standard looking application with a toolbar and top positioned tabs; however, this approach seemed to be unexciting and overcomplicated for a simple browser. We desired our interface to be more akin to a browser like Apple's Safari, with a simple layout and easy to understand features. We decided to use a tab system to organize the application and all of its different features. As for the code, originally we had intended on using the QWebKit library to get the ability to open web pages. This was unfortunately not possible because the QWebKit Object is deprecated within the newest version of Qt. Thus we had to turnback and reinvestigate our options. We ended up using the QWebEngineWidgets library and the QWebEngineView object to load webpages to the browser.

## User Guide

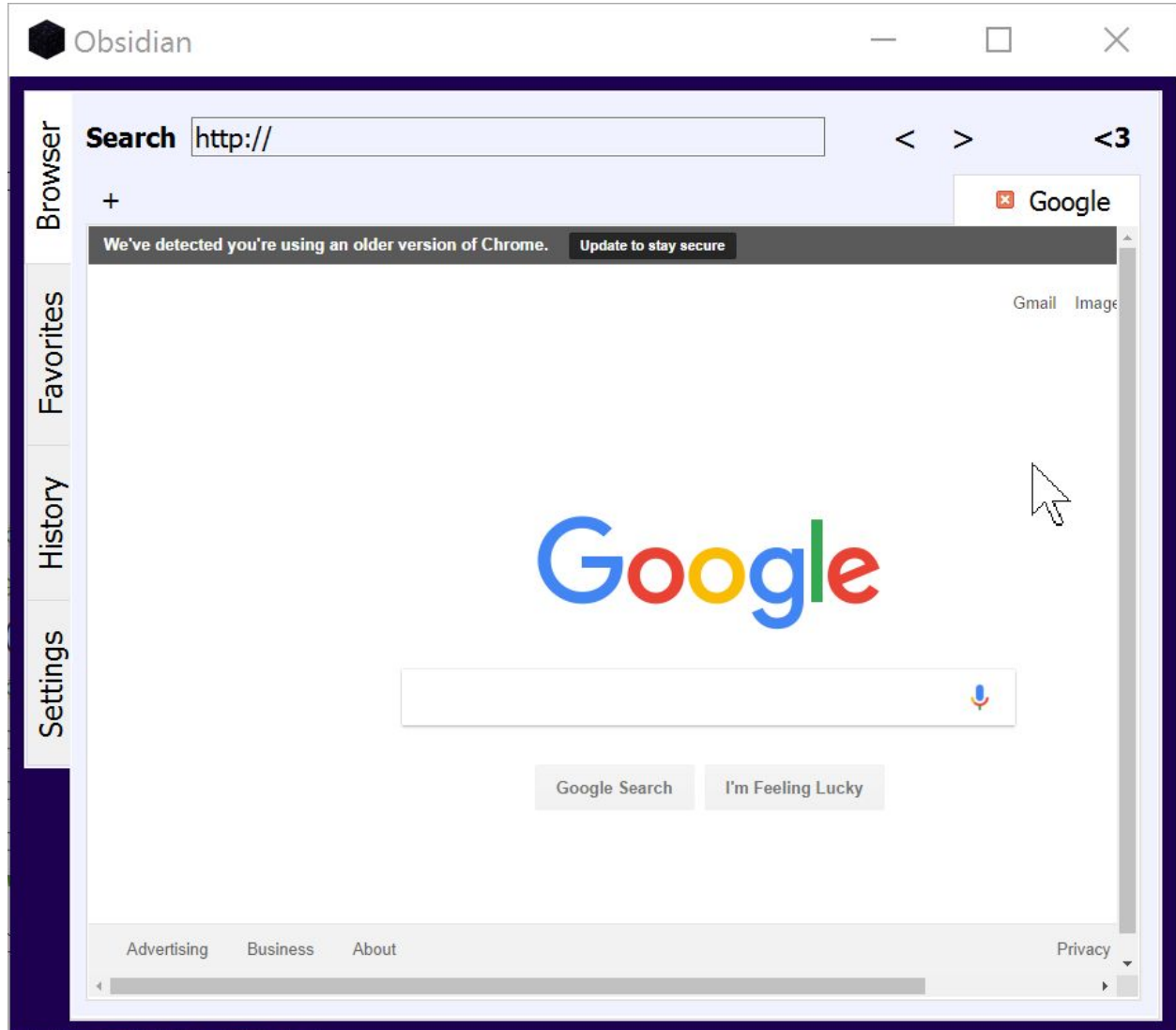
Obsidian is a simple web browser to use. The application has a tab system on the left side that takes the user to each part of the user interface. By default, Obsidian loads the Browser tab.

To run the program, one first must open it in Qt. Then, during the first time of use, they must build the program by pressing this icon . After it is built, one will press the run button



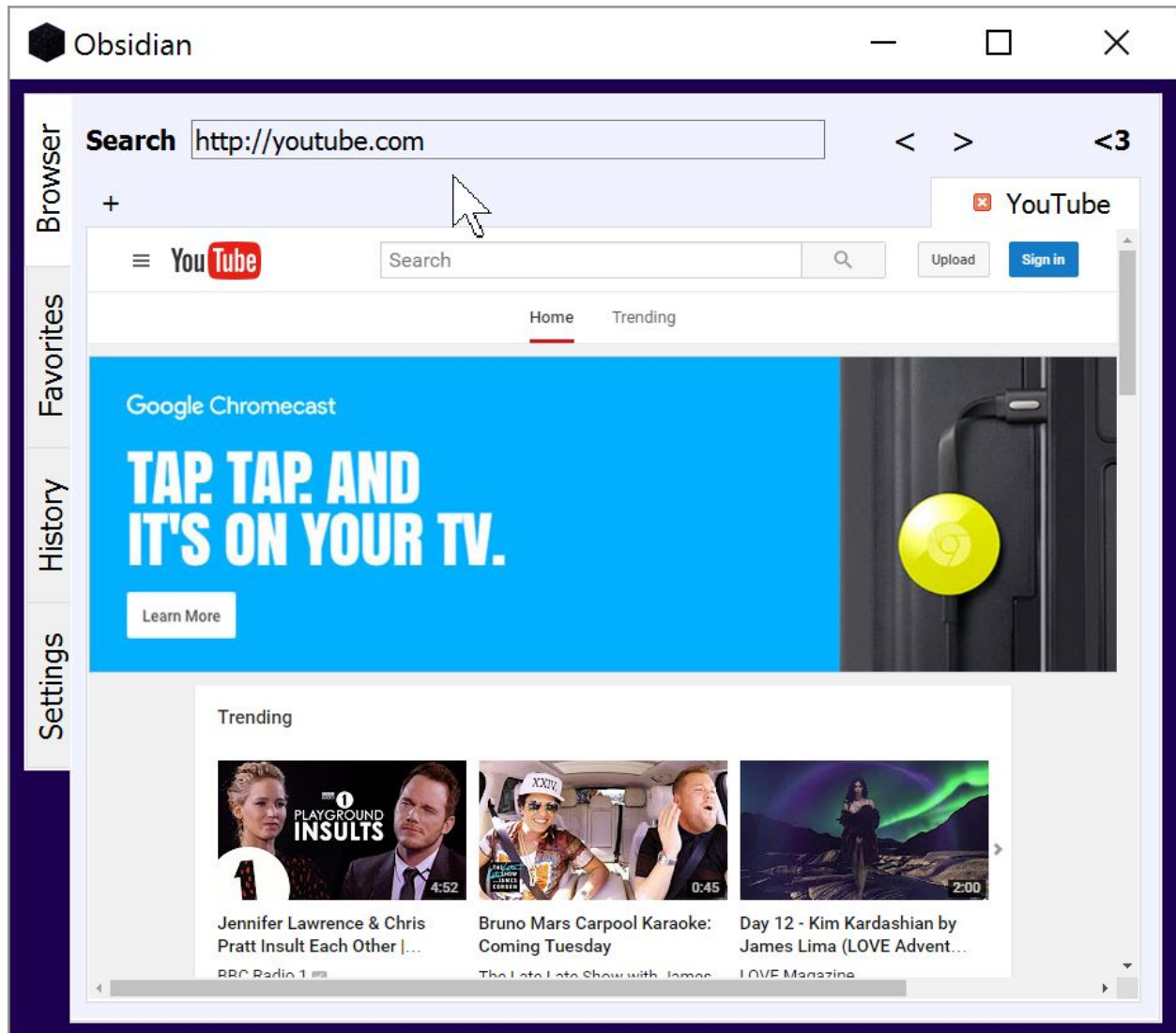
. After they press the run button, the Obsidian browser should be running on the screen


with the home page open.



To navigate to a website, simply type in the address to the search bar at the top of the program

**Search**  , and then press enter.




The web page you desired will open, and you can view the website. To navigate back and forth, simply press the forward or back arrows located on the top of your browser, right next to the search bar . To add a new tab, simply press on the plus symbol in the top left of your screen, right under the search bar, and a new tab with your homepage loaded should open

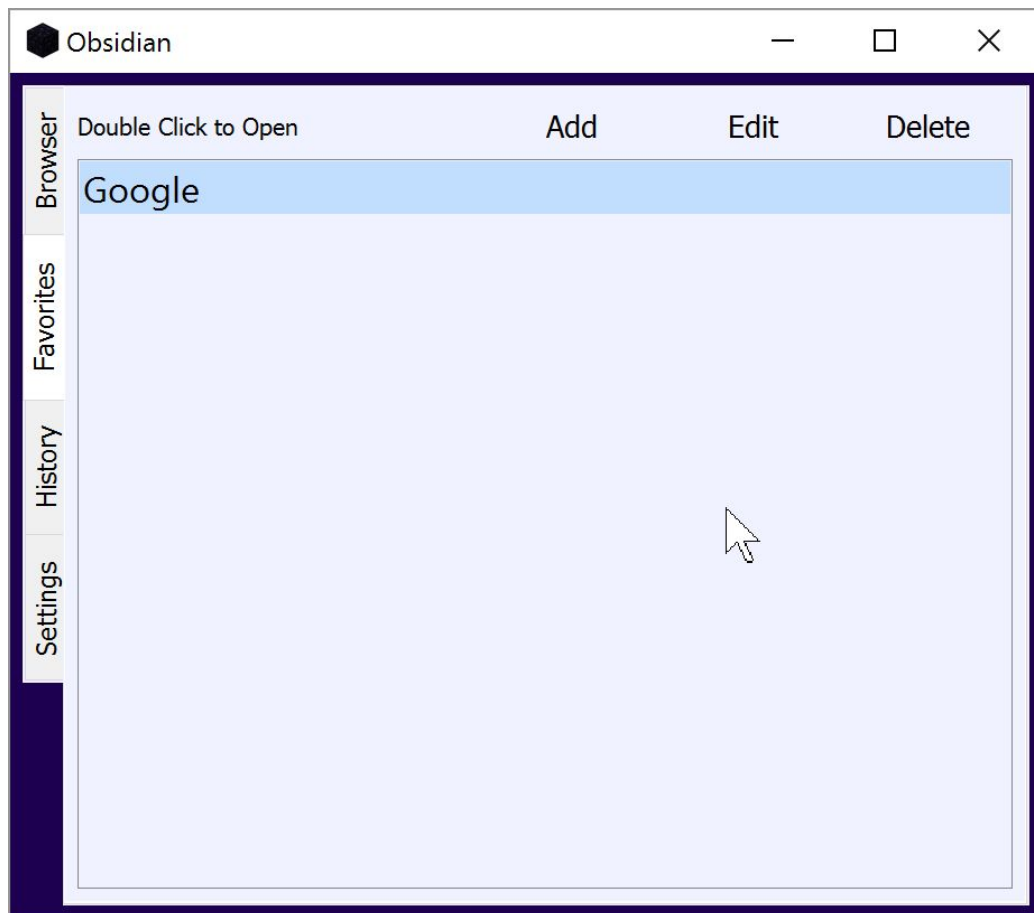


On the left-hand side of the browser, you can locate different tabs, that do different specified

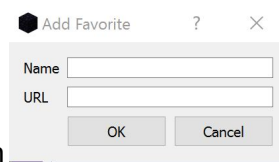
Browser  
Favorites  
History  
Settings

Favorites

thing . To use the Favorites tab  , simply click on it, and it should navigate you to this page.



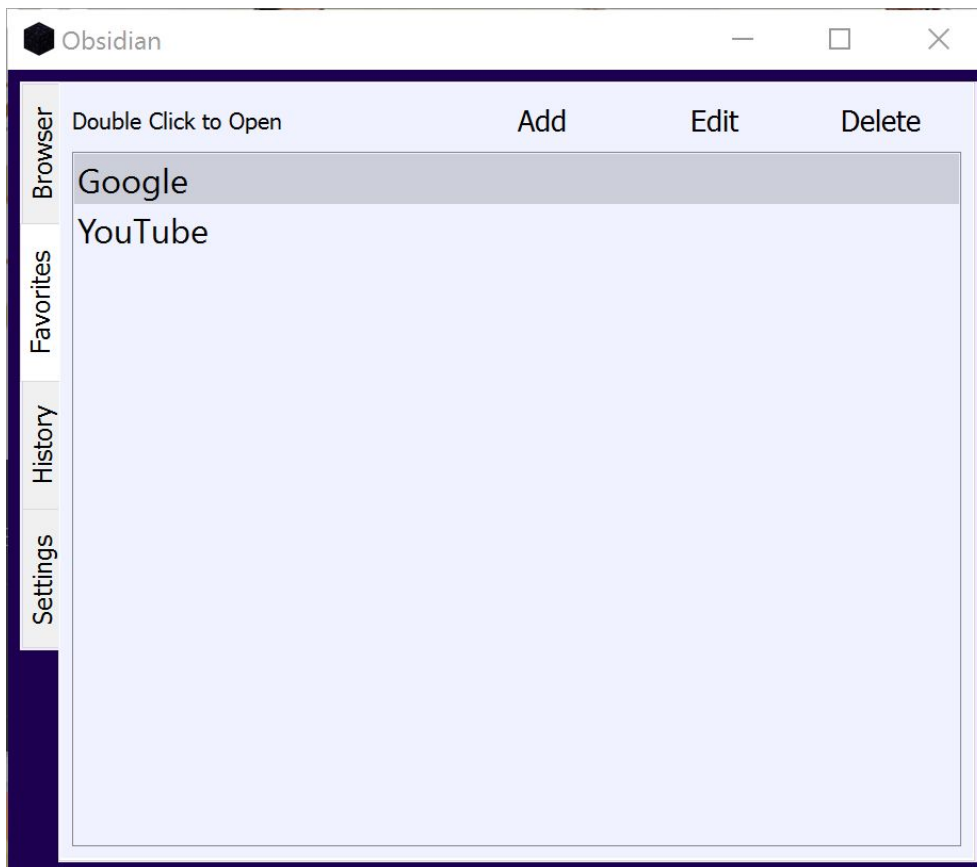
When you open the Favorites tab, if you have not added any websites to Favorites, this should be empty. To add a website to Favorites, you can do one of two things. The first of these, is to press the “Add” button at the top of your browser



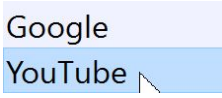
This should pop up on your screen, in which you can type in a desired Name and URL for your newly created Favorite. The other way of adding a favorite would be to navigate back to the browser tab, and press on the “<3” button in the top, right of the screen



. After you have added a Favorite, your Favorites tab will update and now show that new item.

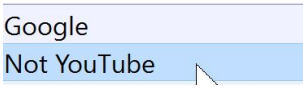



To open one of your Favorites, simply double click on it, and then navigate back to your browser tab, and whatever tab you were currently on in Browser, should now be updated to that specific website. You can also edit or delete a Favorite. To edit a Favorite, simply press on the website

you want to edit , and then press on the “Edit” button on the top of the screen



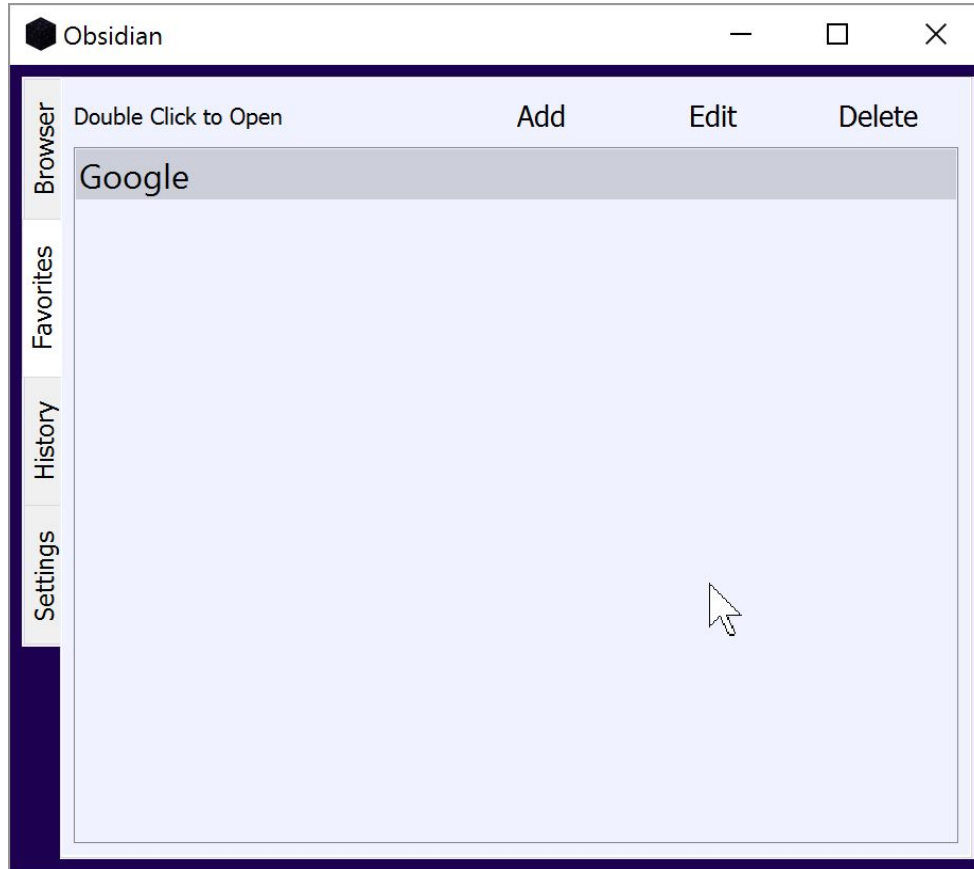
should pop-up on your screen, on which you can edit that webpage, and it will change in the list

of favorites . Finally, to delete a website from Favorites, simply press on

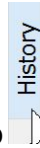
the website you wish to delete , and then press the “Delete” button in the top-right hand corner . The website



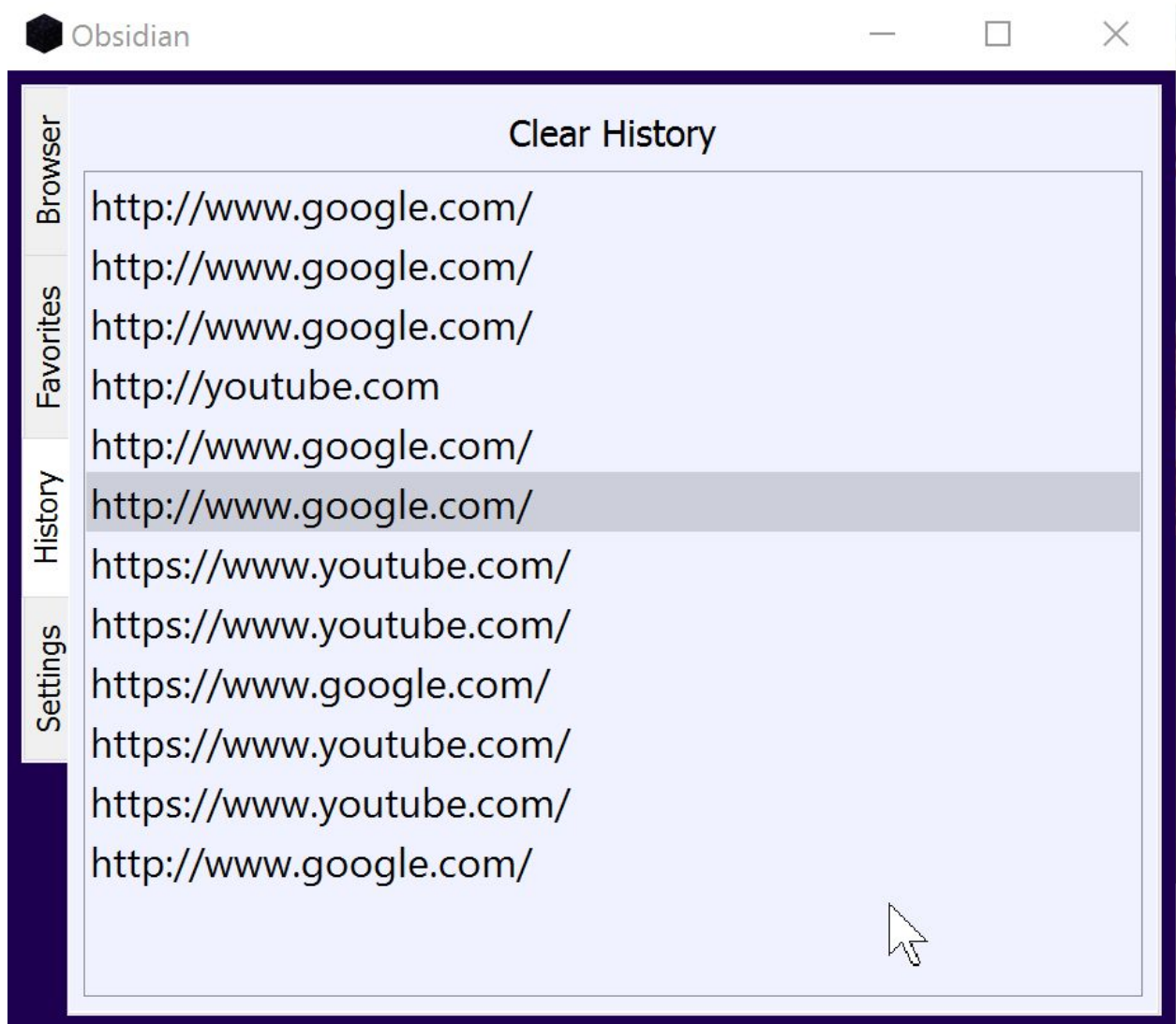
should now be deleted from Favorites.



Your favorites are saved, and the next time you open the browser, your saved favorites should

continue to be there. The next tab we will take a look at is the History tab . When you press

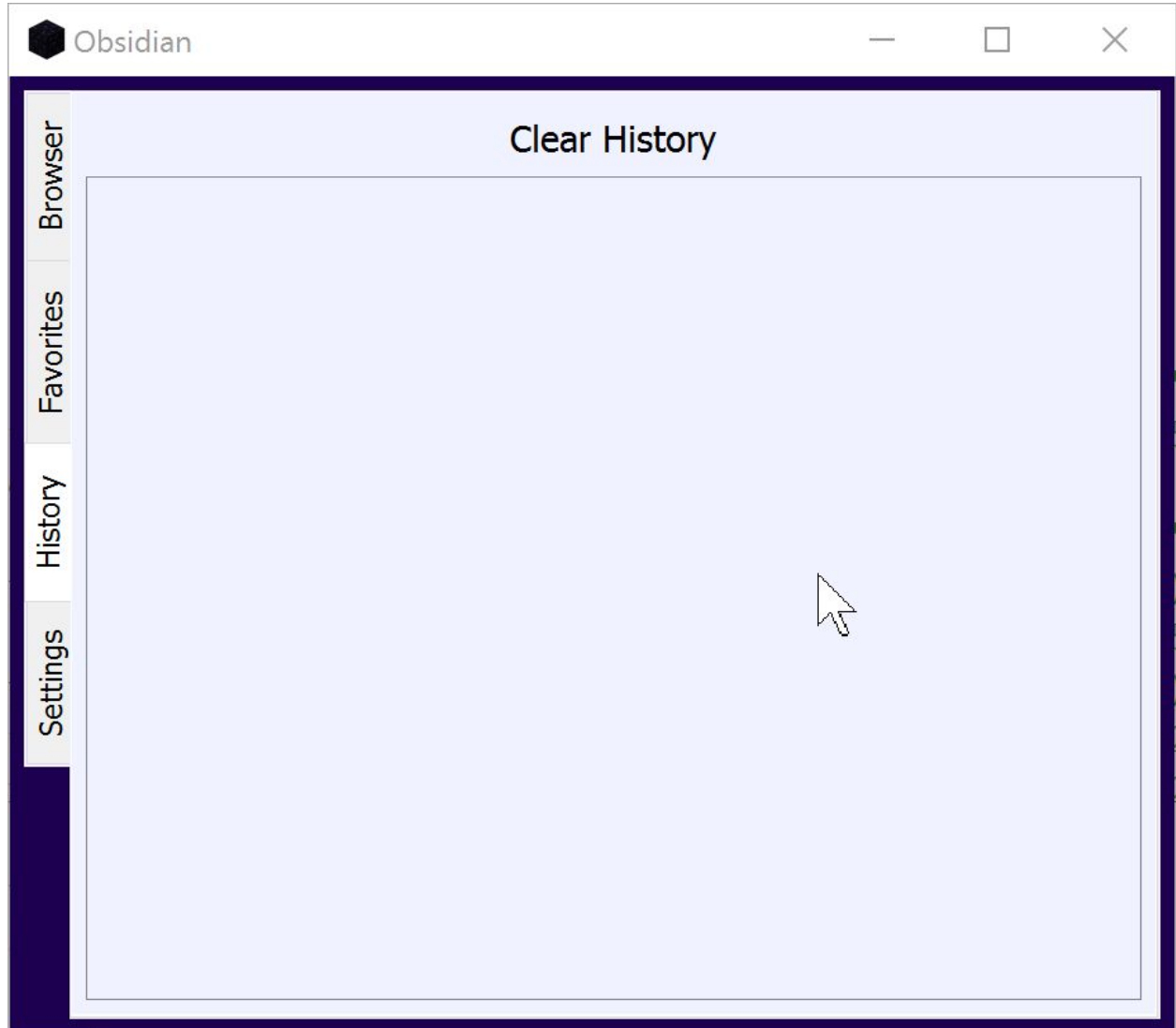
on the History tab, it should show the past websites you have visited.

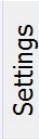


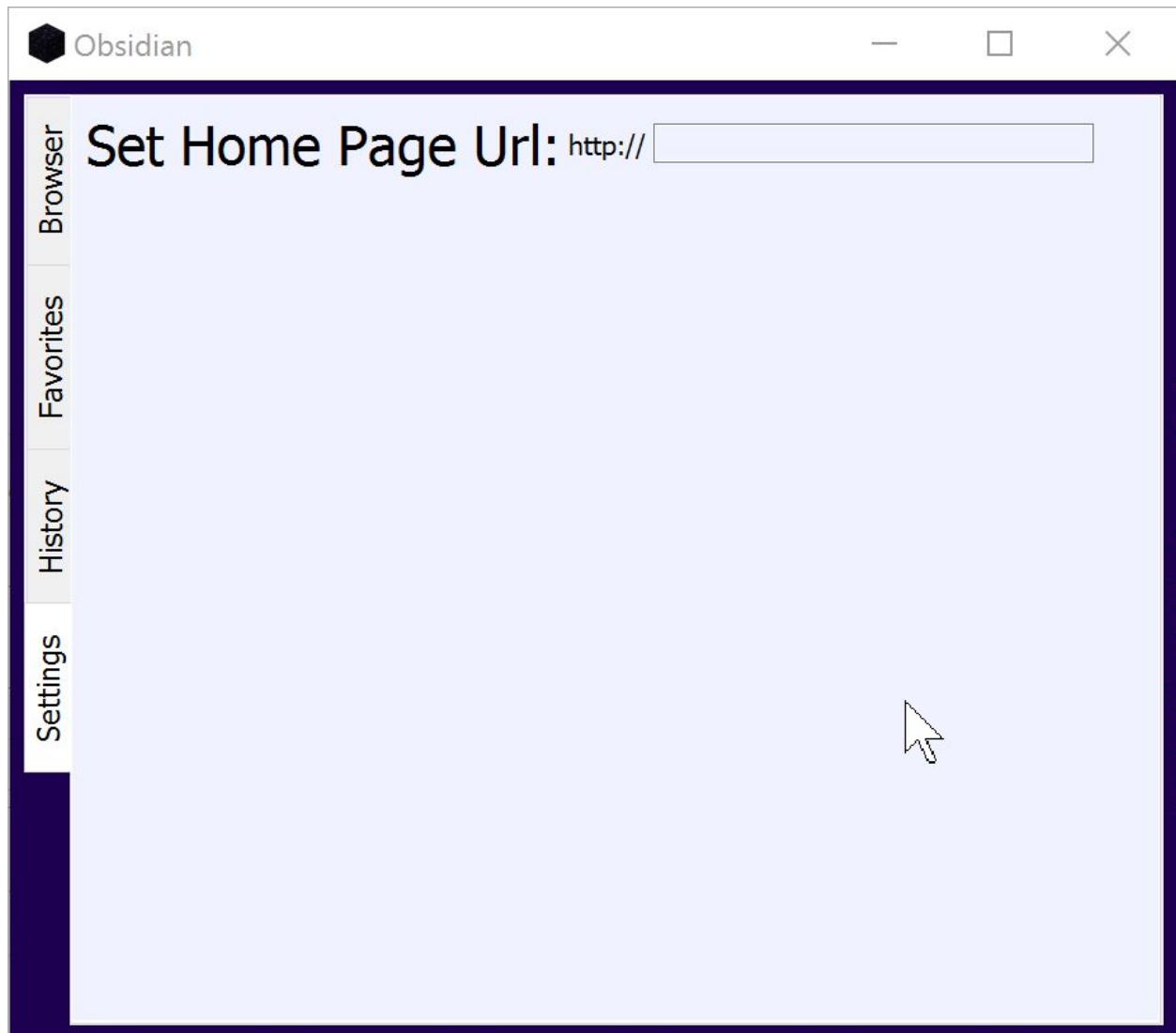
To navigate to one of these webpages, simply double click on them, and then go back to the Browser tab, just like you did with favorites. You can also clear your history, by pressing on the “Clear History” button at the top of the page

Clear History

This should remove all past websites you have visited, and your History tab should look like this.




The final tab on the left-hand side should be the Settings tab  , and the only thing you can do on this tab is change the home page URL.



Initially the Home Page URL should be blank, and thus navigate you to Google.com by default.


To change the home page URL, simply type it in the box provided

. The next time you start up Obsidian, or open a new tab, it will navigate you to the new Home Page.

Obsidian also allows the resizing of the browser just like you would any other application.



Simply navigate to the edge of the Web Browser until you see this icon , then click and drag to the desired size. Alternatively, you can resize the application by clicking the

“Maximize” button located at the top-right hand corner . To the left of the “Maximize” button, the “Minimize” button can be found, which will minimize the application



. Finally to the right of both of those buttons, can be found the “Close” button



. This button will close out of Obsidian.

## Developer Guide

The application interface is presented on a **QApplication** with a **QWindow**. We have a **MainWindow** that holds the entire browser and all of its Ulements. The outermost part of the interface is the central **tabWidget**. This widget has 4 tabs: **Browser**, **Favorites**, **History**, and **Settings**. Each tab holds its own set of widgets corresponding with that specific item. In the browser tab, there is another **QTabWidget** named **browserTabs**. Tabs added to **browserTabs** contain a **QWebEngineView** that is preloaded with a web url. When the application starts, a new tab is automatically created with the saved homepage url. The “setTabsClosable” flag is set to true on **browserTabs** allowing for the user to close each tab individually via a button on each tab. An add tab button was also created and put onto the tab widget itself; this button created a new tab and loads it to the saved home page. When the user types a web url into the search bar and hits return (enter), the application takes the information and correctly loads the requested URL. This process utilized the UTF-8 encoding to make sure each part of the URL entered is

property send to the **QWebEngineView**.

```
// Load the url in the search bar if enter is pressed
void MainWindow::on_lineEdit_returnPressed()
{
    loadUrl_on_currentTab(QUrl(ui->lineEdit->text().toUtf8()));
}
```

```
// Load a url on the current tab webPage
void MainWindow::loadUrl_on_currentTab(QUrl url)
{
    //get access to the current webView
    QWebEngineView* pWebView = NULL;
    QWidget* pWidget = ui->browserTabs->widget(ui->browserTabs->currentIndex());
    // You can use metaobject to get widget type or qobject_cast
    pWebView = (QWebEngineView*)pWidget;

    pWebView->load(url);

    //Add the web url to the history
    data->historyList << url;
    ui->historyListWidget->addItem(url.toString());
    data->save_historyData();
}
```

The forward and back buttons simply call `->forward();` and `->back();` on the current webview respectively. Finally, the browser tab contains an add favorite button presented with a “<3” label. This button creates a new Favorite object, populates the name and url based on the current webview information, and adds it to the list of Favorites located in the **BrowserDataManager** object.

The **BrowserDataManager** takes care of all the saving, loading, and accessing of user data. The **Favorites**, history, and homepage are located here. This object uses **QFile** to read lines from a text file and save lines as well. There are load and save methods for favorites, history, and the homepage. The data is saved using the ending “\n” to separate each item from another. For the favorites the items are listed as: “name”, “url”, “name2”, “url2”, etc. For the history: “url1”, “url2”, etc. The homepage will only ever be one url.

```

void BrowserDataManager::save_homePage() {
    // create a homepage.txt
    QFile file("homepage.txt");
    file.open(QIODevice::WriteOnly);

    // to output to file
    QTextStream qout(&file);

    // set the file text to the homepage url
    qout << homePage.toString();

    file.close();
}

```

Originally we intended on using an XML file to save the data; however, this proved to be somewhat difficult and the code still did not work after a few iterations. The **QSettings** class was also considered.

The next tab in the central tabWidget is the Favorites tab. This tab contains a **QListWidget** and some UI elements to alter the elements in the list. The **QListWidget** favoritesListWidget contains the names of each **Favorite** in the **BrowserDataManager** FavoritesList. When the application is run, the saved **Favorite** items are automatically added to the **QListWidget**. If the user double clicks on one of the list items, the item will be matched up with a **Favorite** and that information will be loaded into the current browsertab's webview. Each **Favorite** contains a pointer to its matching **QListWidgetItem** "listItem". The program then checks each **Favorite**'s listItem with the current **QListWidgetItem**. There are also UI elements on the top right of the tab to alter items in the list. An Add, Edit, and Delete button allow the user to change the list of **Favorites**. The Add and Edit button both create a new **QWindow** using the class and form: **FavoriteEdit**. Both of these button use the same method in **FavoriteEdit** slightly differently. For Edit: a pointer to the currently selected **Favorite** is given and the new window copies its information into its UI elements; it then changes that **Favorite's** variables using the pointer and the user input; name, url, and listItem. The Add button does the same

thing except instead of passing in an already existing **Favorite**, it creates a new one, adds it to the list of favorites, and edits the new **Favorite**. The Delete button simply removed the currently selected Favorite from the list and from the **QListWidget**. At the end of each process, the Favorites data is saved using the **BrowserDataManager**.

```
// set the edit item and get access to the browserData
void FavoriteEdit::setEditItem(Favorite& item, BrowserDataManager& data) {

    // fill window items with favorite information
    browserData = &data;
    editItem = &item;
    ui->nameEdit->setText(editItem->name);
    ui->urlEdit->setText(editItem->url.toString());
}

// If accept is chosen, add or edit the appropriate favorite
void FavoriteEdit::on_buttonBox_accepted()
{
    editItem->name = ui->nameEdit->text();
    editItem->url = QUrl(ui->urlEdit->text().toUtf8());
    editItem->listItem->setText(ui->nameEdit->text());
    browserData->save_favoritesData();
}

// If cancel is chosen, destroy the edit window
void FavoriteEdit::on_buttonBox_rejected()
{
    destroy();
}
```

The History tab is similar to the Favorites tab in design. It contains a **QListWidget** that itself contains the saved history list. When the application is run, the saved history data is loaded into the **QListWidget** from the **BrowserDataManager**. Whenever a web URL is opened using a **QWebEngineView** on any tab, it is added to the History list. If the user double clicks on any history item, that URL will be given to the current tab's webview. A clear history button is located at the top of the history tab. When this button is clicked, the method "clear\_historyData();" is called in the **BrowserDataManager**. This method deleted everything from the history list and saves the file.



The last tab in the central tabWidget is the Settings tab. There isn't much on this tab except for the option to change the home page. When return (enter) is hit on the homepage **QLineEdit**, the **BrowserDataManager** takes that URL, sets the homepage to it and saves the homepage file. More could be added to this tab if other features are desired. The clear history button could also be relocated to this tab.

### Testing

Testing the application was actually fairly easy and straightforward. To test the application, we each individually tested out every button that could be clicked, as well as tested to make sure we could navigate to a desired web page. We did this so that we could test to see if anything would break the application, as well as to make sure everything was working properly. Whenever an issue did arise, the `qDebug()` method was used to print out appropriate information to find a solution. In one test instance, there was an issue with the loading of saved favorites; the `listItem` pointer was not being saved when each favorite was loaded from the favorites file. It turned out that using a "foreach" loop to cycle through the Favorites did not actually allow for changing the favorites. So a regular for loop was used instead.

### Reflections

We used Qtwebengine to complete this project. Like stated we were originally going to use Qtwebkit to complete the project, but after further research, we concluded that Qtwebengine would be a better fit and easier way about going about it. One of the most helpful thing making the web browser was the fact that Qt has sample code already for a web browser using Qtwebengine. Because of this, we were able to look at the demo code and implement and change things to cater to our needs as well as if we got stuck, we could see how the demo code

did something. One of the major limitations of our code is the speed at which it does everything. It is not the slowest browser out there, but when compared with the major browsers of today such as Chrome or Firefox, it can be considered slow. The difference between our browser and these browsers is that they A. have larger teams working on the browser, and B. have had more time to develop and rework the browser than we have. Because of this, it would have been helpful to have even more time in order to enhance the performance of the browser.

Another limitation to making the web browser is the lack of tutorials or extra source code that we could use to make the program. Really the only code that we could specifically use for Qtwebengine came from the demo web browser. Other than that, we used other bits of code that we made for projects in this class, or bits of code we found online that didn't specifically have to deal with the Qtwebengine. All in all, the usability of the language was fairly simple once we got used to it. This was because although we had to learn new code and syntax, it was not an entirely new programming language to us, and we could use our prior experiences with it in order to help us with this project.

The final limitation and thing that we wish we had changed is the fact that one must go into Qt to run the program, and there is not just simply an executable. We tried researching how to do this, but no matter what we tried, it did not seem to work because of the the Qtwebengine widgets we were using.

## Conclusions

The thing we are most proud of in the browser is that it actually works, although not the fastest browser out there, it works. It has tabs, multithreading, favorites, a homepage, history, and most importantly, can navigate to websites just like a browser such as Google Chrome would. These things were all possible with just a group of Computer Science students. If we

were able to get the concept in less than one semester, then we have the confidence that we would be able to make a full browser and even apply to companies. Something that we would have worked better on would be time management. We could have broken up the project into smaller parts since the beginning so we wouldn't be so constraint on time. However, in the end we ended up finishing the project with a decent web browser on our hands, all in less than a semester.