# GREENFLASH BROWSER: A CUSTOM-MADE BROWSER INCORPORATING HASHING AND REGULAR EXPRESSIONS

*A project report submitted in fulfilment of the requirements for the Mid-Semester Small Project in*

**THEORY OF COMPUTATION (IT-301)**

*Submitted by*

**Anubhav Sharma and Arjun Choudhry**
**{2K18/IT/029, 2K18/IT/031}**

*Under the Guidance of*

**Ms. Sunakshi Mehra**
**Department of Information Technology**
**Delhi Technological University**
**New Delhi-110042**


**DELHI TECHNOLOGICAL UNIVERSITY (FORMERLY DELHI COLLEGE OF ENGINEERING)**

# ACKNOWLEDGEMENTS

The successful completion of this project wouldn't have been possible without the help and support of many, and we would like to extend our sincere gratitude to them.

We are highly indebted to Ms. Sunakshi Mehra for being a source of constant motivation, and for helping us improve the project to achieve its present shape. We would also like to thank her for giving us valuable insights on Regular Expressions and report writing.

It would be unfair, should we miss out on thanking our fellow classmates, for some wonderfully insightful discussions on Regular Expressions, for working around us, and for making the subject even more interesting and for sharing many unique ideas, both on and off the subject.

Last but not the least we would like to thank our parents for always being there, and guiding us through this journey.

# CONTENTS

# ABSTRACT

We are now in the age of information, computers and InfoTech. Information has become a multi-billion-dollar business, resulting in computer proliferation world-wide, which has revolutionized our lives. Internet is a collection of so many technologies, one which provides information as well as multimedia facilities at great speed.

By being hooked onto the internet, we can access all this information from the comfort of our homes. It offers a wealth of opportunities and advantages, and promises a whole new world. Thus, World Wide Web has become the de-facto lifeline of the current times.

The common user's interaction with the internet is using a web browser, an application that allows the users to access anything available on the web. But in recent times, these browsers have become heavy, bloated and inefficient, due to an increase in the number of features to accommodate the needs of a larger group of users.

# 1 INTRODUCTION

A web browser, in simple terms, is a software application that allows the users to access information on the World Wide Web and display it back on the user's device. These days, web browsers are used on a wide variety of devices, ranging from common devices like computers, laptops, tablets, smartphones, to less conventional devices like Smart TVs, gaming consoles, wristwatches, cars, speakers and even smart glasses.

The first web browser, called **WorldWideWeb**, which was later renamed to **Nexus** to reduce the confusion between the browser and the Web, was created by Sir Tim Berners-Lee in the year 1990. Nexus was capable of displaying basic style sheets. downloading and opening any file type with a MIME type that is also supported by the NeXT system (PostScript, movies, and sounds), browsing newsgroups, and spellchecking. This was our introduction to web applications, and since then, the world has seen thousands of more advanced web browsers to aid users in accessing the web in more efficient and user-friendly ways.

In recent years, the top web browsers have been adding a truckload of features to make the user experience better, and to cater to a wider userbase. Each browser has its own set of special features, extensions and themes, allowing the users to customize their browsing experience as per their requirements and likings.

At the same time, with an increase in the number of features, we see a major increase in RAM consumption by the browser, and a significant decrease in general performance. Machines with low processing power and less RAM are unable to handle a lot of these browsers, and these browsers also have a negative impact on the battery life of portable devices like laptops, tablets and smartphones.

The extra extensions and themes added to the browser by the user adds an additional load on the processor and memory, even if these improve the ease of use of the browsers. Also, these multiple extensions and themes widely available for browsers are sometimes malicious, making them a major threat to user privacy.

To combat some of these issues, we have created our custom browser by the name **GreenFlash** completely from scratch, with features like multi-user usage, built-in themes, a dedicated history section for every user, and the use of MD5 hashing technique to encrypt the user browsing data to prevent it from being accessed without consent.

# 2 PROBLEM STATEMENT

In this project, we have created a new internet browser from scratch, which is extremely fast, less resource-intensive, and requires less memory than the widely available browsers commonly used.

We have implemented the following features for enhanced privacy and an enhanced user experience:

1. Multi-user support
2. Individual browsing account for each user
3. Individual browsing history for each user
4. Support across multiple Operating Systems
5. Five built-in themes for a personalised browsing experience
6. Clean and user-friendly UI
7. Encryption of the browsing data and user account using MD5 Hashing Technique
8. Local storage of user details and browsing history, stored by each browsing session
9. Ability to open .html, .hbs and .pdf format documents within the browser by searching for the file
10. Instant screenshot button built into the UI to take a screenshot, and save it in the user-selected directory
11. Ability to open a previously used website directly from user browsing history

# 3 PROPOSED APPROACH

## 3.1 Approach Followed

Our browser, **GreenFlash**, has been created by incorporating **Hash Matching** and **Regex**[1] for safe and quick access of user account and user browsing history, and to ensure that the browsing history of each user is kept separate from other users, in separate browsing session files.

We have used the **QWebEngine**[2] class, which provides a widget to view and edit web documents, and the **QtWebEngine**, web browsing module, whose main widget is **web view**. The web site is loaded to the web view with the **load()** function, and the web view is displayed by invoking the **show()** function.

When the user signs up for the first time, a .txt database file is created for the user, which is encrypted using the **MD5** hashing algorithm, for which we use the **QCryptographicHash[3] library**, and the value 1 for the enum **'QCryptographicHash::Algorithm'**, which represents MD5 hashing algorithm.

To access the user history, we match the encrypted user account details with the ones generated by the values entered by the user, and if the match, the history is accessed. The data from different browsing sessions is kept under different hashed files names, which will never collide (no two browsing sessions can have the same hash as they use **time** as a parameter).

The matching of the current generated MD5 hashes to the previously stored MD5 hashes is done using the **Regex library**, by applying the **'regex_match()'** function. The pattern object is of the type **'std::regex'**, which is constructed from a string.

When the user signs up for the first time, the username and password are taken from the user, combined into one string, encrypted using MD5 hash, and this hash value is stored in the user.txt file. The user selects a default theme for the account, which is also stored in a .txt file, and applied every time a user signs in.

If the selected username already exists, using regex_match(), the application checks this, and conveys the feedback to the user.

On signing into an account, the input username and password are hashed and checked against the user.txt file using regex_match(), and if found to match, the user reaches the home page. If the hash value doesn't match, a pop-up window shows the message that the username or password did not match.

If the user does not sign in, the history details are stored in the 'Default' account, and a message is shown to the user regarding the same.

A variable stores the default directory for the screenshots, which can be edited from the GUI.

When the user changes the theme for the browser, the values of the **QPalette** elements are changed to match the values stored for the desired theme, which are stored in the resources.

## 3.2 Proposed Algorithms

Algorithm 1 History Extraction

Input:
Username

Output:

```
0 :  F ← QFile("db.txt")
1 :  Regex R ← "(username)(.*)"
2 :  T ← Textstream(F)
3 :  while NOT At_End(T) do
4 :        L ← ReadLine(T)
5 :        if Regex_match(L, R) do
6 :              LoadHistory(L[uersname.length() : L.length])
7 :        end if
8 :  end while
9 :  return
```

**QFile():** It is the constructor of the class QFile, which locates the file and enables data transfer. It has various open modes like QIODevice::ReadOnly, QIODevice::WriteOnly, etc.

**Textstream():** It is the object of QTextstream class which facilitates the transfer of text data from the file to the buffer.

**ReadLine():** It is used to read the current line and moves the pointer to the next line.

**Regex_match():** It matches the current regex object with the string, and returns a boolean value, indicating whether the parameters match or not.

**LoadHistory():** It is used to load the history from a given hash file in the vault (explained below in detail).

Algorithm 2 Loading History: LoadHistory()

Input:
Hash ← MD5 Hash code

Output:

```
0 :  G ← QFile("vault/" + Hash + ".txt")
1 :  W ← Textstream(G)
2 :  while NOT At_End(W) do
3 :        K ← ReadLine(W)
4 :        P, Q ← HistoryParser(K)
5 :        UI.AddToListWidget(P, Q)
6 :  end while
7 :  return
```

**QFile():** Constructor of the class QFile, and locates the file to enable data transfer.

**Textstream():** It is the object of QTextstream class which facilitates the transfer of text data from the file to the buffer.

**ReadLine():** Used to read the current line and moves the pointer to the next line.

**HistoryParser():** It parses the line from the history of a given browsing session, helping in seperating the URL from the title, thus enabling double-click reload.

Algorithm 3 History Parser: HistParser()

Input:
Histline

Output:

```
0 :  K ← Length(HistLine) - 1
1 :  while NOT (Histline[K]  = '>') do
2 :        K--
3 :  end while
4 :  URL ← Histline[K : Histline.length()]
5 :  Title ← Histline[0 : K]
6 :  return Title, URL
```

Algorithm 4 Sign In

Input:
Username, Password

Output:

```
0 :  Cmb_String ← Username + Password
1 :  hash ← Hash(Cmb_String, MD5)
2 :  D ← QFile("user.txt")
3 :  E ← QTextStream(D)
4 :  while NOT At_End(E) do
5 :        F ← Readline(E)
6 :        if regex_match(hash, F) do
7 :              SignIn()
8 :        end if
9 :  end while
10:  WrongDetails()
11:  return
```

**Hash():** Function of the QCryptographicHash library, which returns the hashed value of a string.

**SingIn():** Initiates the proceedings of a successful sign in.

**WrongDetails():** Shows a message box, indicating the input username or password was incorrect.

**QFile():** It is the constructor of the class QFile, which locates the file and enables data transfer. It has various open modes like QIODevice::ReadOnly, QIODevice::WriteOnly, etc.

**QTextstream():** It is the object of QTextstream class which facilitates the transfer of text data from the file to the buffer.

**ReadLine():** It is used to read the current line and moves the pointer to the next line.

# 4 IMPLEMENTATION DETAILS

| | |
|---|---|
| Programming Languages Used | C++ |
| Operating System | Created in Linux, but compatible with Windows, Linux and MacOS |
| Library, Packages or APIs Used | Qmake[4] Build<br>Qt[5] Libraries<br>QWebEngine<br>QCryptographicHash Library<br>Regex |
| Interface Design (GUI) | Created using QT Framework |

Table 1: Details of the implementation environment of the program, to ensure stability and compatibility across systems and operating systems.

We have programmed our whole project in **C++**, to ensure fast performance and loss resource utilization by the application. This is also why our application is supported across a wide range of devices.

The application does not also require any OS specific system call, thus it is compatible with not just Linux and Windows, but also MacOS.

**Qmake** is a build system, like CMake that uses existing functionality of CMake, and also incorporates Qt libraries widgets. It was used to write the build files for the program, linking libraries, compilers and other dependencies, for making the execution of the program possible.

**Qt Library** is a widely used set of libraries that was used to create the GUI of our application, and write multiple functionalities for the linking of each UI widget in our application.

**QWebEngine** comprises of web-based C++ tools, for efficient writing and processing of network-based requests. All the browser-based functions like back, forward, sending a request for a URL, are done using the QWebEngine, making it the backbone of the browsing process.

We have used the **QCryptographicHash** library to use the MD5 hashing algorithm to encrypt the user sign in details, history session files, and user preferences, so that this information cannot be accessed by anybody but the user. **Regex** is used for hash matching, and matching the strings with regex objects for successful sign in and history retrieval.

# 5 RESULTS

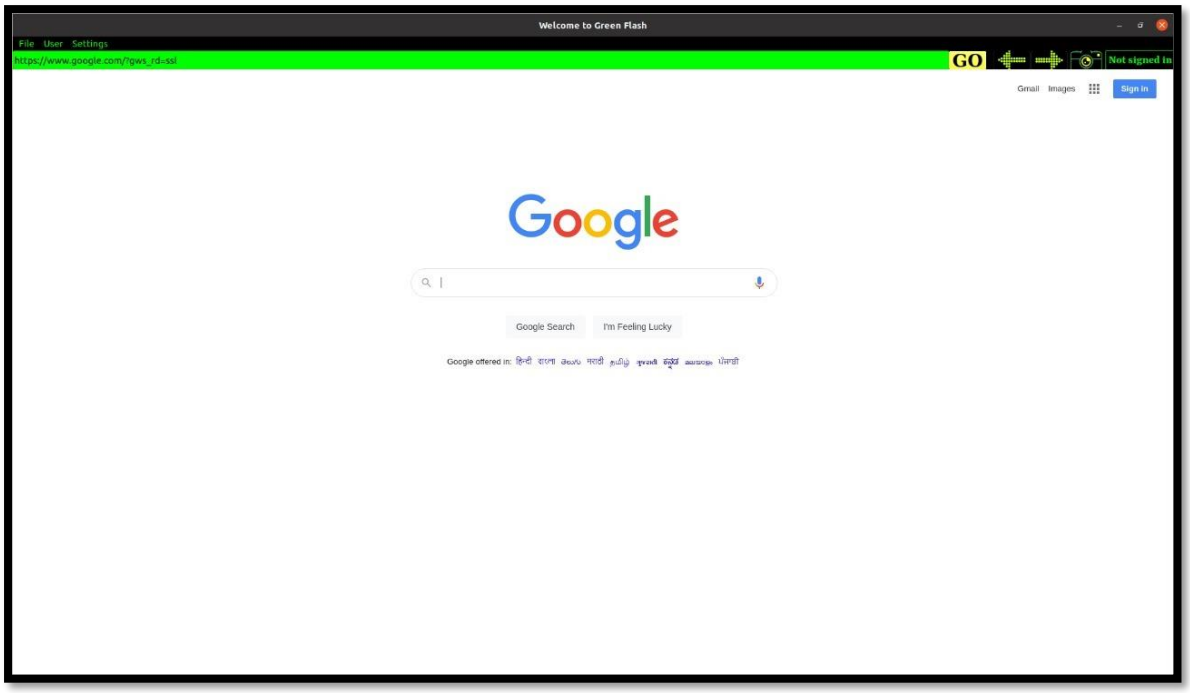


Fig 1: Home Page of the Application

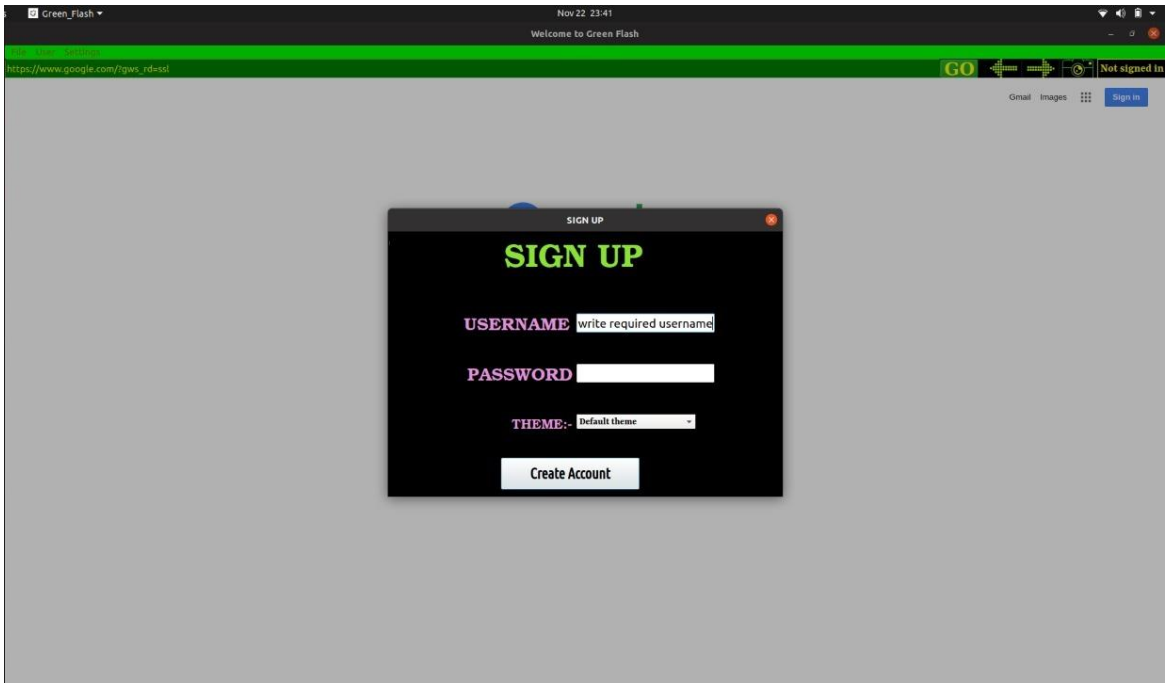


Fig 2: Sign Up Page of the Application

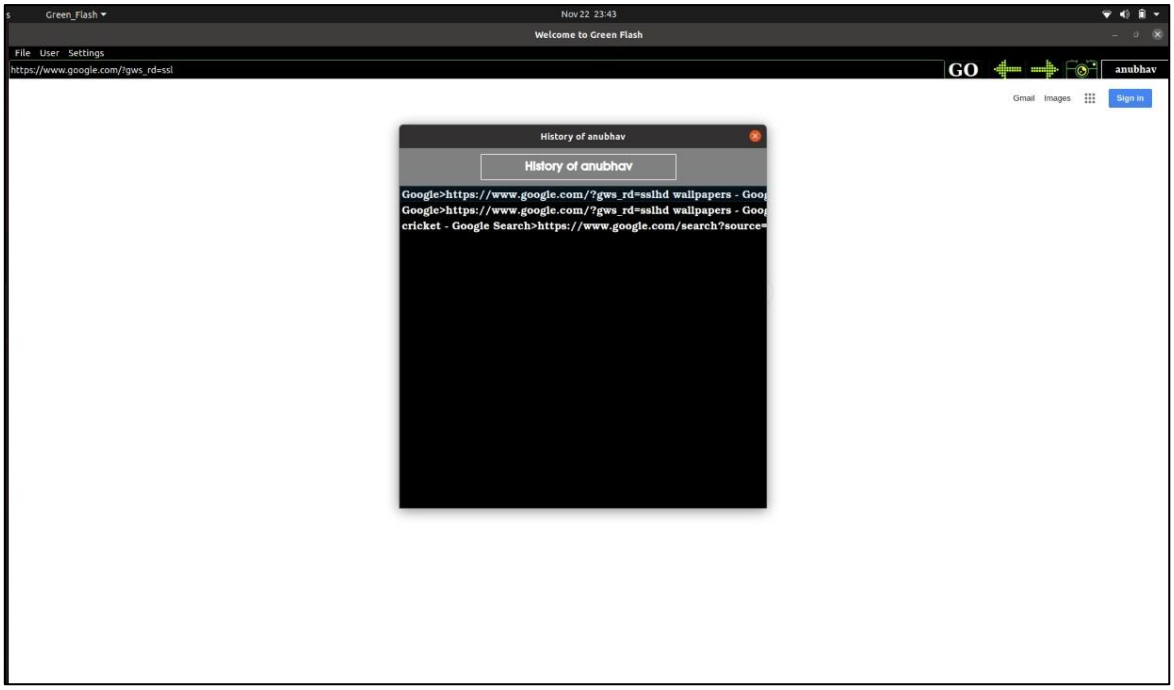


Fig 3: Individual History panel for a user

As it can be seen in the above images, GreenFlash browser offers all the main features that other browsers offer, without the bloatware that make other browsers bloated. The individual user history and privacy features like hashing and local storage of history are extremely useful to users whose focus is on privacy.

Our browser performs extremely quickly and smoothly, has a pleasant user interface, incorporated Regular Expressions for quick string matching, and has support for multiple themes.

When trying to detect which locally-stored browsing history file belongs to which user, we couldn't make out any difference which could help us figure out which browsing session belonged to which user. Also, we couldn't extract the user account details from the user.txt file, as all these are stored after being encrypted by the MD5 hashing algorithm, but the program could easily sign in to the correct account by matching the hash values to the strings using Regex functions.

# 6 CONCLUSION

We successfully made an internet browser from scratch, which uses Hash Matching and Regular Expressions for a more secure and protected browsing experience. Our browser performs as well or better than the commonly used browsers, and as all the user details and browsing history are stored locally, the experience is more secure. Also, we have included many user-friendly features like Instant Screenshot Button, configurable Screenshot directory, themes, to name a few, which further improve the browsing experience.

# 7 FUTURE WORK

We intend to take this project further, and some of the features on our roadmap include:

1. Bookmarks
2. A specific page for downloads
3. Multi-tabbing
4. Support for video plug-ins
5. In-Private Mode, where browsing history wouldn't be saved
6. Guest Mode
7. Password for the Browser
8. Stop Button
9. Integrated search using the user's preferred search engine
10. Omnibox support
11. Reading mode for documents
12. Password Management

The above features are not in any specific order, but will be determined by the usefulness and resource requirement, so as to ensure that the browser does not lose its speed, responsiveness and low memory requirements.

# 8 REFERENCES

1. Regex library, https://en.cppreference.com/w/cpp/regex

2. QWebEngine, https://doc.qt.io/qt-5/qtwebengine-index.html

3. Qt Documentation – QcryptographicHash Class, https://doc.qt.io/qt-5/qcryptographichash.html

4. Qmake, https://doc.qt.io/archives/qt-4.8/qmake-manual.html

5. Qt Tools - https://doc.qt.io/qt-5/topics-app-development.html