# DWIT COLLEGE

# DEERWALK INSTITUTE OF TECHNOLOGY


# DUPLICATE EMAIL DETECTION DURING REGISTRATION USING BLOOM FILTER

**A MICRO PROJECT REPORT**

**Submitted to**

**Department of Computer Science and Information Technology**

**DWIT College**

**Submitted By**

**Sushil Awale**

**Class of 2019**

**May 7, 2018**

# CERTIFICATION

This project titled ‘DUPLICATE EMAIL DETETCTION WHILE REGISTRATION USING BLOOM FILTER’ by Sushi Awale, under the supervision of Mr. Sudan Prajapati, Deerwalk Institutre of Technology, is hereby submitted for the partial fulfillment of the requirements for the Fifth Semester Micro Project.

Approved By:

---------- ----------

Signed Date

(Supervisor)

[Sudan Prajapati]

# ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my supervisor (Mr. Sudan Prajapati) as well as DWIT who gave me the golden opportunity to do this wonderful project on the topic (Duplicate Email Detection While Registration Using Bloom Filter), which also helped me in doing research and I came to know about so many things, I am really thankful to them.

Secondly, I would also like to thank my parents and friends who helped me a lot finalizing this project within limited time frame.

Regards,

Sushil Awale

Class of 2019

# ABSTRACT

Most software applications require users to enter an email or choose a username while registering into the application. In this process, it is critical that the email or username used for registration is a unique one. A primitive way of ensuring that the constraint is met is to check into the database whether the email/username already exists or not. However, this simple process consumes time and processing resources when the number of users grow exponentially.

In this project, space efficient probabilistic data structure called Bloom Filter is used to detect duplicate email during registration for a dummy email application. With the use of Bloom Filter membership query does not depend on the size of the set and even if the user size increases exponentially membership query is fast and inexpensive.

*Keyword: Bloom Filter, User Registration, Duplicate Email*

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| API: | Application Programming Interface |
| BSD: | Berkeley Software Distribution |
| CSS: | Cascading Style Sheet |
| HTML: | Hyper Text Markup Language |
| JQuery: | JavaScript Query |
| JS: | JavaScript |
| MIT: | Massachusetts Institute of Technology |
| MySQL: | My Structured Query Language |
| RAM: | Random Access Memory |

# CHAPTER 1: INTRODUCTION

## 1.1 Background

Registration is an integral part of many software applications. Most applications require the users to register with an email or select a unique username. In providing with this facility, the applications must implement a mechanism to check for duplicate username/email and disallow registration. The primitive way of checking for duplicate email/username is database query. However, as the number of users increase this simple process consumes a lot of time and processing resources, rendering the solution inefficient.

Bloom Filter is a space and time efficient probabilistic data structure that facilitates member lookup in a set without tapping into the resource heavy process of database query. Bloom Filter implements a simple bit array in which all bits are initialized to zero. When a member is added to the set, the bit value at corresponding index position is set to one. Next, when checking for membership, the bit value is checked and returns true if bit value is one else false. This simple solution can be used to detect duplicate email/username while registration.

## 1.2 Problem Statement

The goal of the project is to implement Bloom Filter data structure to facilitate time and space efficient email/username lookup while registration and notify the user of the availability of the email/username. The system should perform optimally regardless of the number of users registered into the system providing a scalable solution to any application that plans to implement it.

## 1.3 Objectives

The objectives of this project are:

- To develop a duplicate email checking mechanism using Bloom Filter data structure
- To make the application time and space efficient in order to solve the problem of checking duplicate emails in an application.

## 1.4 Scope

The project showcases the use of Bloom Filter to detect duplicate email during registration for a dummy email application. The implementation can be easily duplicated into any application that requires the function of detecting duplicate email/username during registration process. Moreover, it can also be implemented for any problem that requires fast membership check for a given element.

## 1.5 Limitation

1. Bloom filters are probabilistic data structure and may generate cases of false positive.
2. This project implementation is capable of storing only 1 billion users.
3. Once the user registers, the user cannot be deleted.

## 1.6 Outline of Document

The report is organized as follows:

**Preliminary Section:** This section consists of the title page, abstract, table of contents and list of figures and tables.

**Introduction Section:** In this section, the background of the project, problem statement, its objectives, scope and limitation are discussed.

**Requirement and Feasibility Analysis Section:** Literature review, Requirement analysis and Feasibility analysis make the bulk of this section.

**System Design Section:** This section consists of the methodology that were implemented in the project and the system design as well.

**Implementation and Testing Section:** In this section, the implementation, description of major methods and testing of the system are discussed.

**Conclusion and Recommendation:** This section consists of the final findings, and the recommendations that can be worked on in order to improve the project in the future.

# CHAPTER 2: REQUIREMENT AND FEASIBILITY ANALYSIS

## 2.1 Literature Review

### 2.1.1 Database Lookup

The primary method of detecting a duplicate email/username is to query into the database to see if the email/username already exists or not. This method, for example, can simply be implemented using a *SELECT ... WHERE* query in MySQL. Although this method is simple, querying into database is slow and expensive because the data is stored in secondary memory (like hard disk) rather than the primary memory (like RAM). As a result, this method renders inefficient when the database lookup needs to be fast and is conducted frequently.

### 2.1.2 Bloom Filter

A Bloom Filter is a space efficient probabilistic data structure which is used to represent a set and perform membership queries [1] i.e. query whether an element is a member of the set or not. The Bloom Filter data structure was introduced by Burton H. Bloom [2] in 1970.

There are many variations of Bloom Filters and the Standard Bloom Filter that is used in this project is described as follows:

A Bloom Filter for representing a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements is described an array of m bits, initially set to 0. A Bloom filter uses k independent hash functions $h_1, h_2, \ldots, h_k$ with range $\{1, \ldots, m\}$. Each hash function maps every item to some random number over the range $\{1, \ldots, m\}$. [3]

Standard Bloom filters have two operations:

**Add(x)** – Add x to the Bloom Filter

For each element x ∈ S, the bits $h_i$ (x) are set to 1 for 1 <= i <= k. [3]

**Contains(y)** – Check if y is in the Bloom Filter.

To check if an item y is in S, we check whether all $h_i$ (y) are set to 1. If any of $h_i$ (y) is 0 then clearly y does not belong to S. If all $h_i$ (y) are set to 1 then y may belong to S. [3]

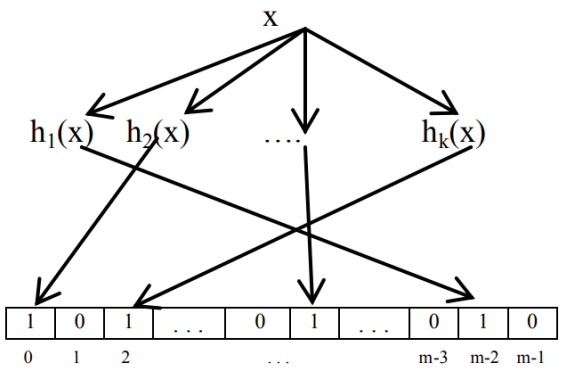



*Figure [1]: Set Operation in a Standard Bloom Filter*

A Bloom Filter occupies negligible space compared to the entire set. Space saving comes at the cost of false positives but this drawback does not affect the processing of information if the probability of an error is made sufficiently low. [3]

In this project, the username section of an email in the dummy email application is treated as element that must be unique. The username is passed through a hash function to generate an index and the bit value at that index in the bit array is set to 1.

When checking if the username exists or not, the username is passed through the same hash function which generates an index value. The bit value in the bit array for that index position is checked. If the bit value is set to 1, the username exists otherwise not.

There may be a cases that the same index value is generated for two or more usernames. In such case, false positives occur i.e. a username that does not exist is said to exist. However, we can minimize false positives by selecting a hash function with uniform distribution such as Murmur hash function or increasing the number of hash functions.

### 2.1.3 Murmur Hash Function

Murmur Hash is a very fast, non-cryptographic hash suitable for general hash-based lookup. [7] The hash function was created by Austin Appleby in 2008. [8] The name is composed of Multiply (MU) and rotate (R), used in it inner loop. [8]

Although Murmur Hash is not suitable for cryptographic purposes, it is an excellent choice for hash-based lookups. In a simple test conducted by Colin Pesyna [9], Murmur hash function was found to have no collision while testing with 42 million keys. Further tests resulted that Murmur Hash function had random uniform distribution, and good avalanche effect. [10] All these properties make Murmur hash an excellent choice for use in Bloom Filter.

## 2.2 Requirement Analysis

### 2.2.1 Functional Requirements

The functional requirements of this project are:

a. Bloom Filter data structure shall be implemented to detect duplicate email while registration.
b. A dummy registration form shall be provided for users with a required username field.
c. The users shall be notified immediately whether the username is available or not.
d. The system shall perform well irrespective of number of users.

### 2.2.1 Non-Functional Requirements

The non-functional requirements of this project are:

a. The application must have user-friendly interface.

## 2.3 Feasibility Analysis

After gathering of the required resource, whether the completion of the project with the gathered resource is feasible is checked using the following feasibility analysis.

### 2.3.1 Technical Feasibility

The project can be implemented using any high-level programming language along with an in-memory database. In this project, Node.js shall be used for the backend due to my expertise in JavaScript and easily available JavaScript libraries. The frontend module shall be implemented using HTML, CSS, JavaScript and JQuery.

For the in-memory database, Redis shall be used. Redis is a fast open-source in-memory database which shall be used for storing the bit array. MySQL database shall be used to store the user information.

### 2.3.2 Economic Feasibility

All the tools and technologies used for the project shall be open-source and freely available. The project can be easily completed with a dedicated time allocation of two hours per day for one week.

### 2.2.1 Operational Feasibility

The project meets all the requirements identified in the requirement analysis phase. The project can be implemented in any application that requires a duplicate email/username detection while registration. Hence, the project is operationally feasible.

# CHAPTER 3: SYSTEM DESIGN

## 3.1 Methodology

Methodology implemented while carrying out the project are discussed below:

### 3.1.1 Development Methodology

The waterfall development model was followed for this project because it is easy to understand and use. This is an individual project and the waterfall development model makes it easy to manage the project. In this project, each project module was developed sequentially. Moreover, the requirements are well understood at the beginning of the project, so, the waterfall model is helpful in this. In testing and validating the project, this model posed some difficulties such as it is very difficult to go back and change something that was not well thought out in the concept stage.

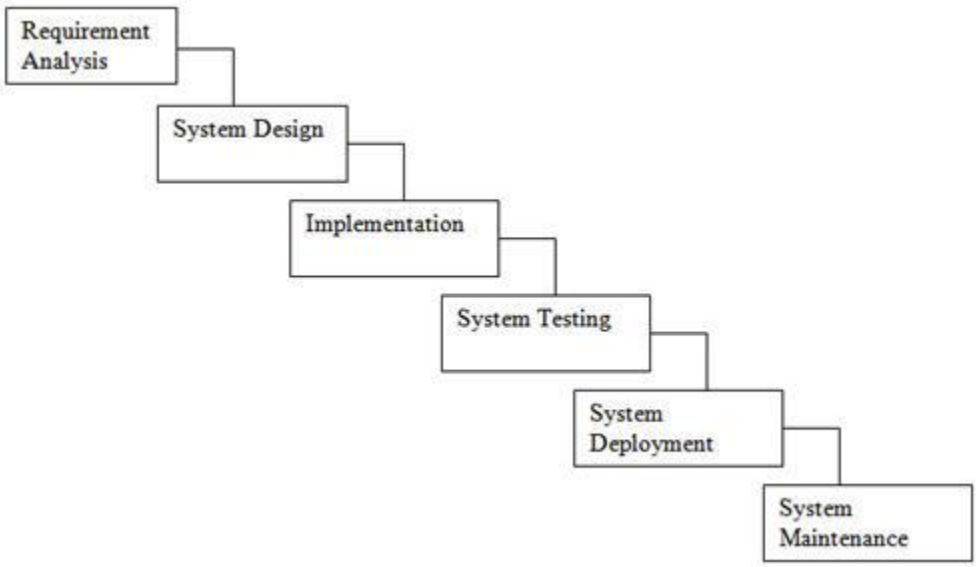



*Figure [2]: Development Phases in Waterfall Model*

## 3.2 System Design

### 3.2.1 Activity Diagram

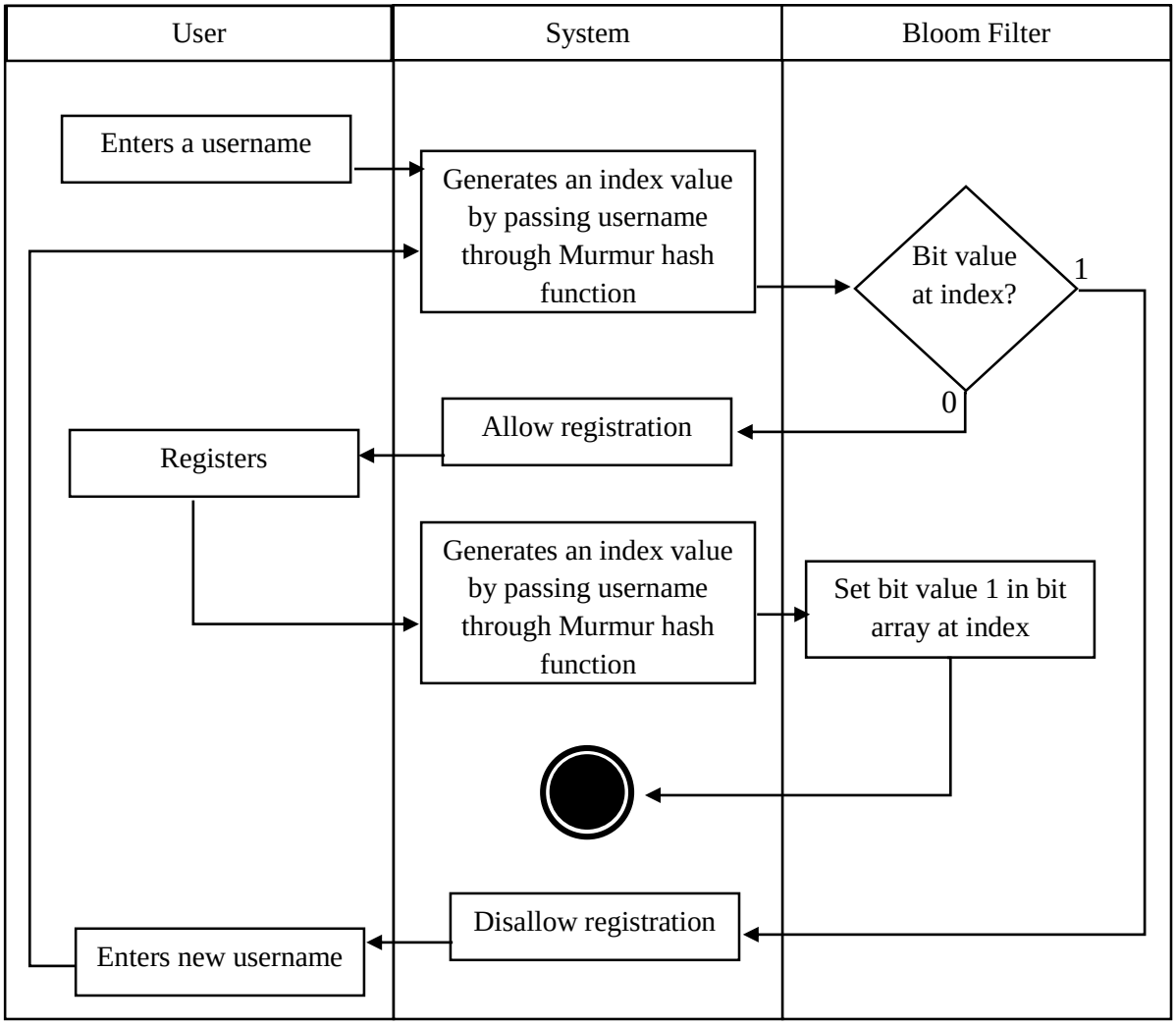



*Figure [3]: Activity Diagram of the System*

The above figure explains the activity of the system. First the user tries to register with a username of choice. For each character that the user enters into the username field, the system reads the string and passes it through the Murmur3 hash function which generates a 32-bit key value. Then the bit value of the bit array in Bloom Filter is checked at the index position corresponding to the key value. If the bit value is 1, the system disallows the user to register. Otherwise, the registration is allowed and the bit value and the generated index position is set to 1.

# CHAPTER 4: IMPLEMENTATION AND TESTING

## 4.1 Implementation

The dummy email application is implemented as a web application and can be accessed from the web browser. The application is developed in JavaScript using Node.js and Express Framework. The Bloom Filter data structure was implemented using Redis database and Murmurhash3js JavaScript library.

### 4.1.1 Tools Used

**Node.js and Express Framework**

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. [12]

Express.js, or simply Express, is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. [13]

The backend of this project is developed in Node.js using Express Framework.

**Redis Database**

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, and bitmaps with radius queries. [4]

Redis database’s bitmap data structure was used to implement the bit array for Bloom Filter.

**Murmurhash3js Library**

Murmurhash3js is a Node JS library is a JavaScript implementation of MurmurHash3's [6] hashing algorithms. [5] It was written by Karan Lyons and Sasscha Droste and is available under MIT License.

The library was used to implement the Murmurhash3 hash function to generate the index value to store and check bit value in the bit array.

**MySQL Database**

MySQL is an open-source relational database management system. The MySQL development project has made its source code available under the terms of GNU General Public License, as well as under a variety of proprietary agreements. [11]

MySQL database was used to store the user information.

## 4.2 Testing

The system was tested using a dataset containing 30,000 usernames. The datasets were downloaded from *Mockaroo* [14] by specifying the `users` table schema used in the project. Out of 30,000 data, around 21,000 were loaded into the Bloom Filter and the `users` table where as the rest was left for testing.

Next, the 30,000 data was tested against the system and following results were achieved:

| | |
|---|---|
| True Positive (tp) | 21277 |
| False Positive (fp) | 0 |
| False Negative (fn) | 0 |

*Table [1]: Test Results for Calculating Precision*

With the above data, a precision of 1.0 was achieved.

$$Precision = \frac{tp}{tp + fp}$$

With a dataset of more than 21,000 entries, the performance did not experience any lag or delay in generating results.

# CHAPTER 5: CONCLUSION AND RECOMMENDATION

## 5.1 Conclusion

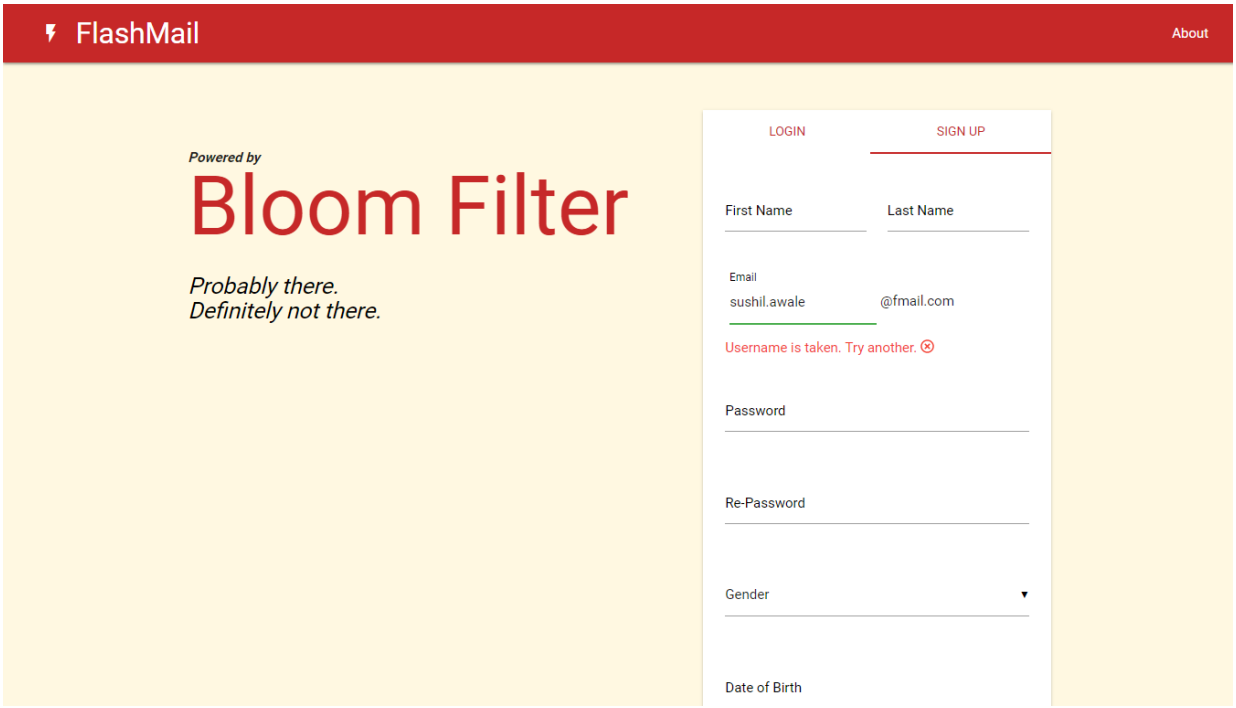
Bloom Filters is a fast and space-efficient model for membership lookup and is an excellent choice for detecting duplicate email/username lookup while registration in an application. The model can store billions of users' membership record while taking up relatively less memory. And the cost of looking up the membership is irrespective of the size of the data. In the above implementation, a dataset of more than 21,000 entries did not affect the speed and maintained a perfect precision level of 1.0.

However, due to lack of data, the model could not be tested to its full capability. Hence, more rigorous tests must be done in order to use this system for real-world applications.

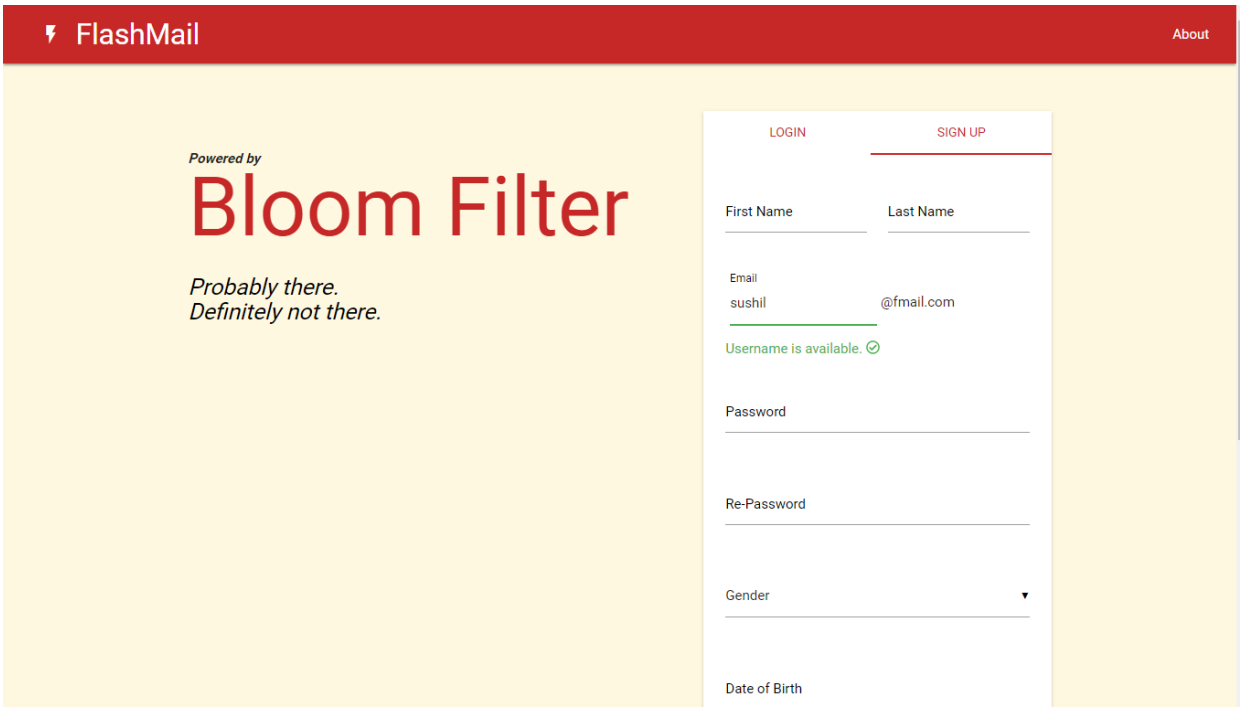
## 5.2 Recommendation

There are significant number of improvements that can be made to the system. First of all, a bit vector of larger size can be used to accommodate more than one billion users. Secondly, artificially intelligent systems can be implemented to recommend usernames in case of true positive results.

# APPENDIX




*Screenshot [1]: Case I Username is not available*




*Screenshot [2]: Case II Username is available*

# REFERENCES


[1] P. Brass, Advanced Data Structures, Cambridge University Press, 2008, pp. 402 – 405

[2] Burton H. Bloom, Space/time trade-offs in Hash Coding with Allowable Errors, Communications of the ACM, Volume 13, Issue 7, 1970, http://portal.acm.org/citation.cfm?doid=362686.362692

[3] S. K. Pal, P. Sardana, Bloom Filters & Their Applications, International Journal of Computer Applications and Technology, Volume 1, Issue 1, 2012, http://ijcat.com/archives/volume1/issue1/ijcatr01011006.pdf

[4] Redis Labs, Introduction to Redis, *Retrieved on April 12, 2018* https://redis.io/topics/introduction

[5] K. Lyons, S. Droste, mumurhash3js, *Retrieved on April 12, 2018* https://www.npmjs.com/package/murmurhash3js

[6] A. Appleby, SMHasher, *Retrieved on April 12, 2018* https://github.com/aappleby/smhasher

[7] Hadoop in Java, Hbase.apache.org, 2011, *Retrieved on April 12, 2018* https://web.archive.org/web/20120112023407/http://hbase.apache.org/docs/current/api/org/apache/hadoop/hbase/util/MurmurHash.html

[8] A. Appleby, MurmurHash, Google, 2008, *Retrieved on April 12, 2018* https://sites.google.com/site/murmurhash/

[9] C. Pesyna, Choosing a Good Hash Function, Part 2, Neustar, 2012, *Retrieved on April 20, 2018* https://research.neustar.biz/2011/12/29/choosing-a-good-hash-function-part-2/

[10] C. Pesyna, Choosing a Good Hash Function, Part 3, Neustar, 2012, *Retrieved on April 20, 2018* https://research.neustar.biz/2012/02/02/choosing-a-good-hash-function-part-3/

[11] What is MySQL? MySQL 5.1 Reference Manual. Oracle, *Retrieved on May 5, 2018* https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[12] What is NodeJs? NodeJS. Joyent, *Retrieved on May 5, 2018* https://nodejs.org/en/

[13] Express, Express, Strongloop, IBM, *Retrieved on May 5, 2018* https://expressjs.com/

[14] Mockaroo, Random Data Generator and API Mocking Tool, Mockaroo, LLC. https://mockaroo.com/