

# DUPLICATE EMAIL DETECTION DURING REGISTRATION USING BLOOM FILTER

## MICROPROJECT

### PROBLEM STATEMENT

In any web application where registration is required, it is of paramount importance that each user is assigned a unique identification key such as email, username etc. The primary way of assuring this constraint is not allowing the user to register with a duplicate identification key. During the registration process itself, the user must be notified about the availability of the username/email and it is ideal to notify the user instantaneously.

The solution to the problem seems trivial: query into the database and check if the username/email is available or not. However, when the number of users rise, database query becomes slow and expensive.

One of the optimal solution to the problem is use of Bloom Filter data structure.

### BLOOM FILTER

Bloom filter is a probabilistic data structure that checks membership of elements in a set and tells whether the element is a member or not.

Bloom filter has two parts: a bit vector and a hash function.

A bit vector is used to store the membership information of elements. If a member is in the set, the bit value at the index position corresponding to the element is set to 1 otherwise it is 0. The index position of the element is determined by the hash function.

### WHY BLOOM FILTER IS A GOOD SOLUTION FOR THE PROBLEM?

The main advantage of using bloom filter is its efficiency. It is both time and space efficient. The time complexity for bloom filter is  $O(1)$ . It instantly checks the bit value and index position and determines whether the element is in the set or not.

Bloom filter is also space efficient. Bloom filter uses only a single bit to store the membership of the element. Even though, we have billions of user, bloom filter only requires a few hundred MB of memory space to store the membership info.

## IMPLEMENTATION

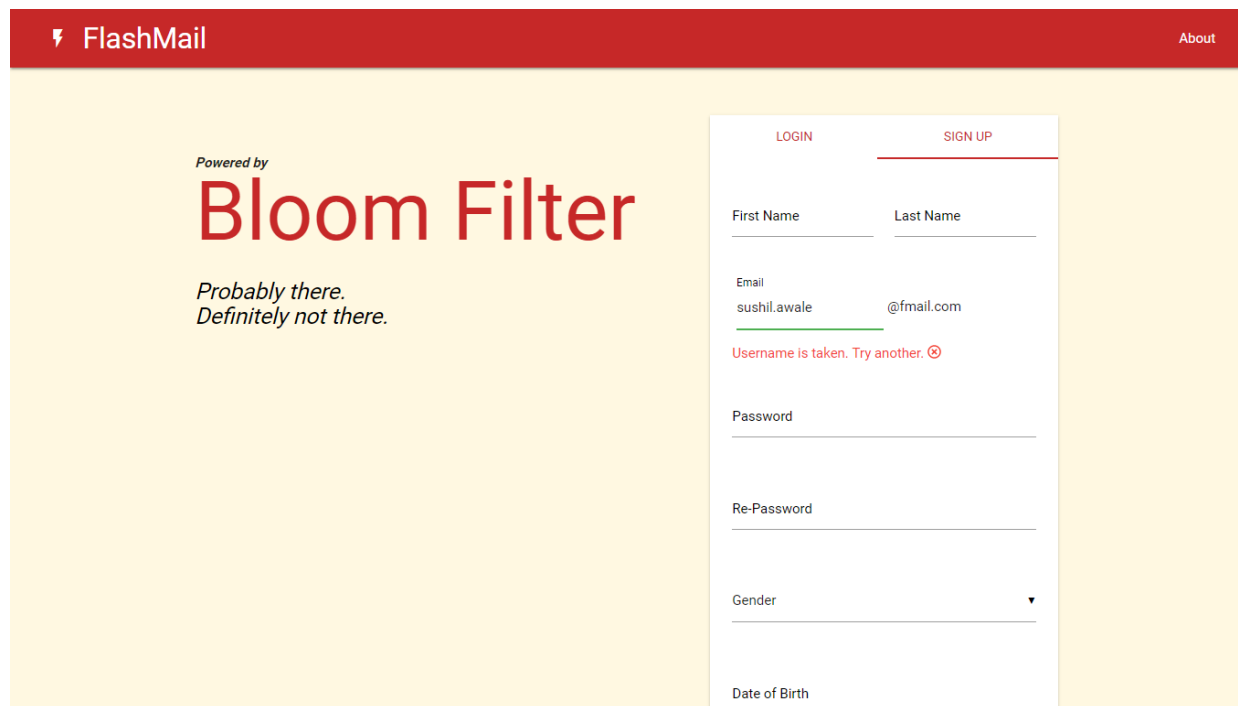
For my micro project, I created a mockup sign up page for an example email application 'Flash Mail'.

For my backend, I used NodeJS to implement the logic.

To create a bloom filter bit vector, I used Redis database, which is an open-source in-memory database and for my hash function I used 32-bit Murmur Hash function.


The project is currently capable of storing membership information of one billion user and takes less than 512 MB memory space.

It determines whether an email is taken or not in constant time of  $O(1)$ .



The screenshot displays the FlashMail application interface. At the top, a red header bar contains the 'FlashMail' logo on the left and an 'About' link on the right. The main content area has a light yellow background. On the left side of this area, the text 'Powered by' is above the large red title 'Bloom Filter'. Below the title, the phrase 'Probably there. Definitely not there.' is written in a smaller, italicized font. On the right side, there is a white sign-up form. The form has two tabs at the top: 'LOGIN' and 'SIGN UP', with 'SIGN UP' being the active tab. The form fields include: 'First Name' and 'Last Name' (both empty); 'Email' with the value 'sushil.awale' (underlined in green) and '@fmail.com' (to its right); a red error message 'Username is taken. Try another.' with a circular icon; 'Password' (empty); 'Re-Password' (empty); 'Gender' (a dropdown menu with a downward arrow); and 'Date of Birth' (empty).

*Case 1: Username is not available.*

FlashMail

About

Powered by

# Bloom Filter

*Probably there.  
Definitely not there.*

LOGIN

SIGN UP

First Name

Last Name

Email

sushil

@fmail.com

Username is available. ✓

Password

Re-Password

Gender

▼

Date of Birth

*Case 2: Username is available.*

## WORK LEFT

Next, I will be using NodeJS's selenium library to automate the registration of dummy users for 10,000 users and test efficiency and precision of the implementation.