#Overview

Buzzer is a microblogging service written in Go on which users socialize by posting messages known as *buzzes*.

Registered users can subscribe to another users posts which appear on their *buzz-feed* along with any message in which they were mentioned (@username).

Users can search for messages by tags (#topic) or for other users.

Buzzer uses Go's channels and goroutines to coordinate the asynchronous activity and expose the service via a WebSockets-based API for real-time, bidirectional communication with a web client.

The web client is written using HTML5 (including the WebSocket API), Facebook's React JavaScript framework, and uses some of the responsive-design elements of CSS3. It should be usable from any modern smartphone.

#Background

Goroutine

Lightweight thread managed by the Go runtime.

Execute concurrently with other functions.

Channel

Conduits to send and receive messages, which are typed values.

Message sent **happens-before** message received.

Sending and receiving is a synchronous operation.

Buffered channels allow for multiple messages to be sent or received before blocking.

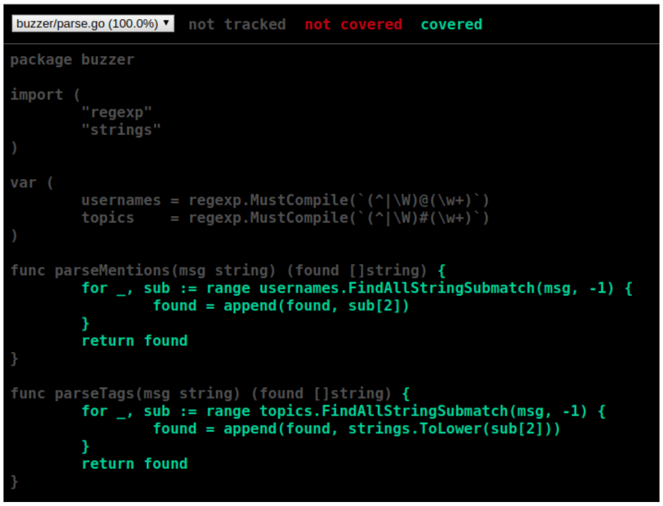
`select` statement operates on multiple channels.
`default` case makes communication non-blocking.

WebSockets

TCP-based protocol for bidirectional communication over ports 80 and 443 (for Transport Layer Security).

#Tools

Test Runner, Code Coverage, and Benchmarking

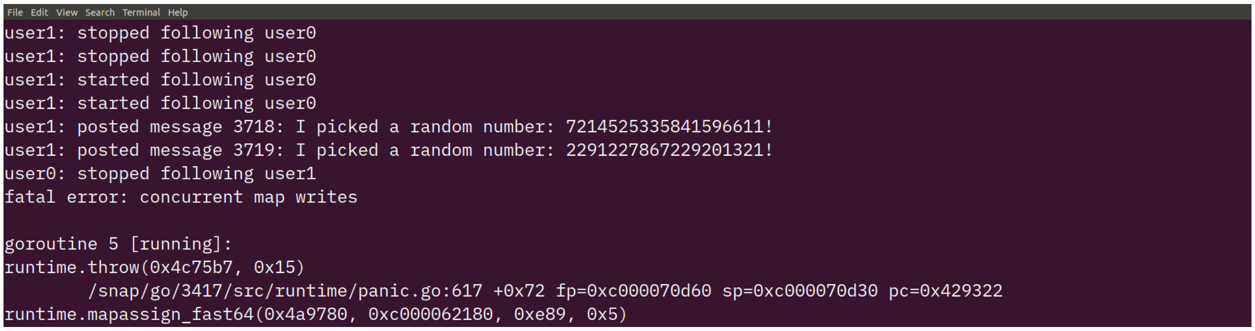


```
package buzzer

import (
	"regexp"
	"strings"
)

var (
	usernames = regexp.MustCompile(`(^|\W)@(\w+)`)
	topics    = regexp.MustCompile(`(^|\W)#(\w+)`)
)

func parseMentions(msg string) (found []string) {
	for _, sub := range usernames.FindAllStringSubmatch(msg, -1) {
		found = append(found, sub[2])
	}
	return found
}

func parseTags(msg string) (found []string) {
	for _, sub := range topics.FindAllStringSubmatch(msg, -1) {
		found = append(found, strings.ToLower(sub[2]))
	}
	return found
}
```

Code Coverage output

```
$ make benchmark
GOPATH=/home/taeber/code/cop5618-concurrent/project/lib:/home/taeber/code/cop5618-concurrent/project go test -benchmem -run=^$ buzzer -bench .
...
BenchmarkKernelPost-4           300000   4415 ns/op   457 B/op        0 allocs/op
BenchmarkChannelServerPost-4    200000   6450 ns/op   534 B/op        3 allocs/op
PASS
ok     buzzer2.744s
```

Deadlock and race detection



```
user1: stopped following user0
user1: stopped following user0
user1: started following user0
user1: started following user0
user1: posted message 3718: I picked a random number: 7214525335841596611!
user1: posted message 3719: I picked a random number: 2291227867229201321!
user0: stopped following user1
fatal error: concurrent map writes

goroutine 5 [running]:
runtime.throw(0x4c75b7, 0x15)
        /snap/go/3417/src/runtime/panic.go:617 +0x72 fp=0xc000070d60 sp=0xc000070d30 pc=0x429322
runtime.mapassign_fast64(0x4a9780, 0xc000062180, 0xe89, 0x5)
```

Data race detection in Go runtime

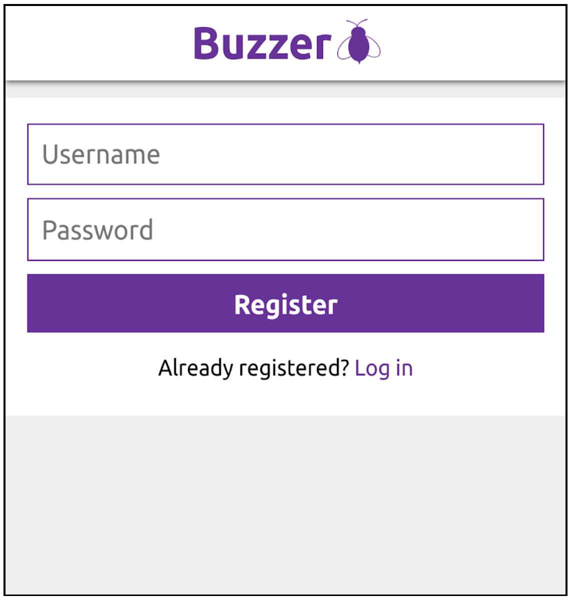
#FutureWork

Implement the backend using more traditional shared memory primitives like locks and conditional variables to compare performance.

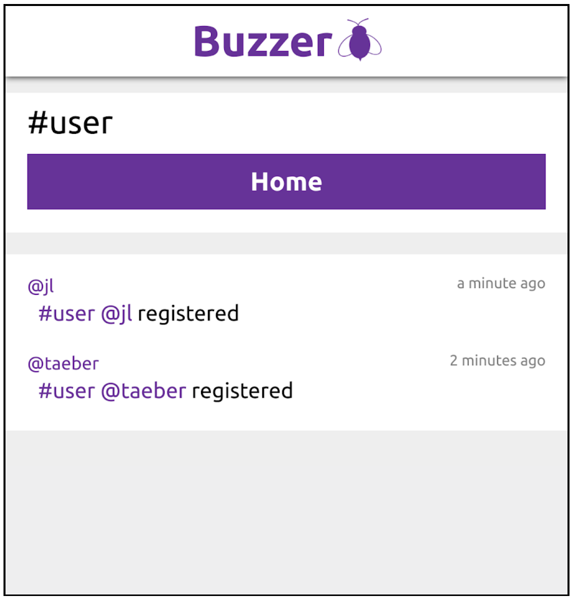
Write an Actor Model framework for Go or a language based on the Go library and runtime.

#Thanks

A Tour of Go, Golang.org Authors.
https://tour.golang.org/concurrency
WebSocket Protocol, Fette and Melnikov.
https://tools.ietf.org/html/rfc6455
Preparing and Presenting Effective Research Posters, Miller, Jane.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1955747/

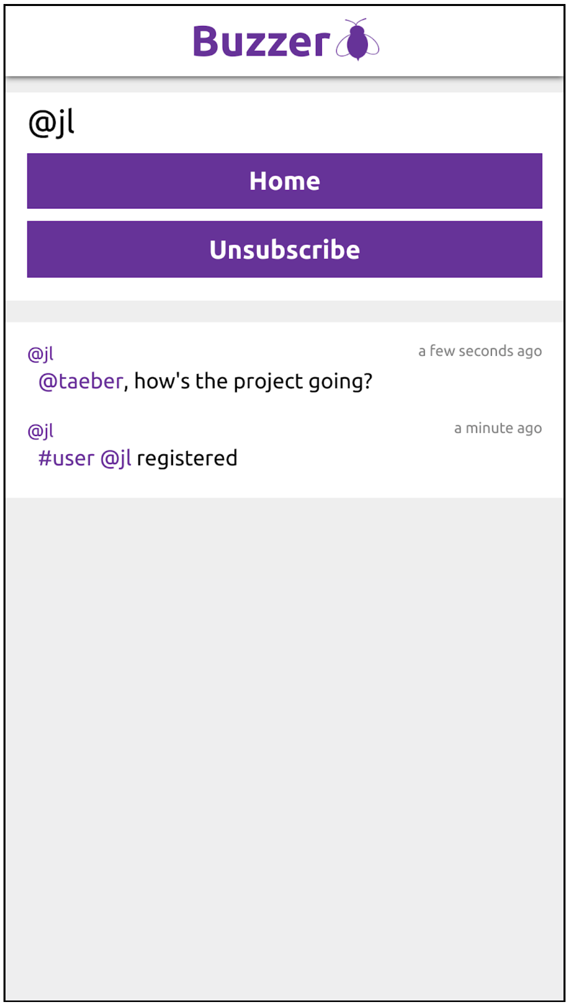


Already registered? Log in



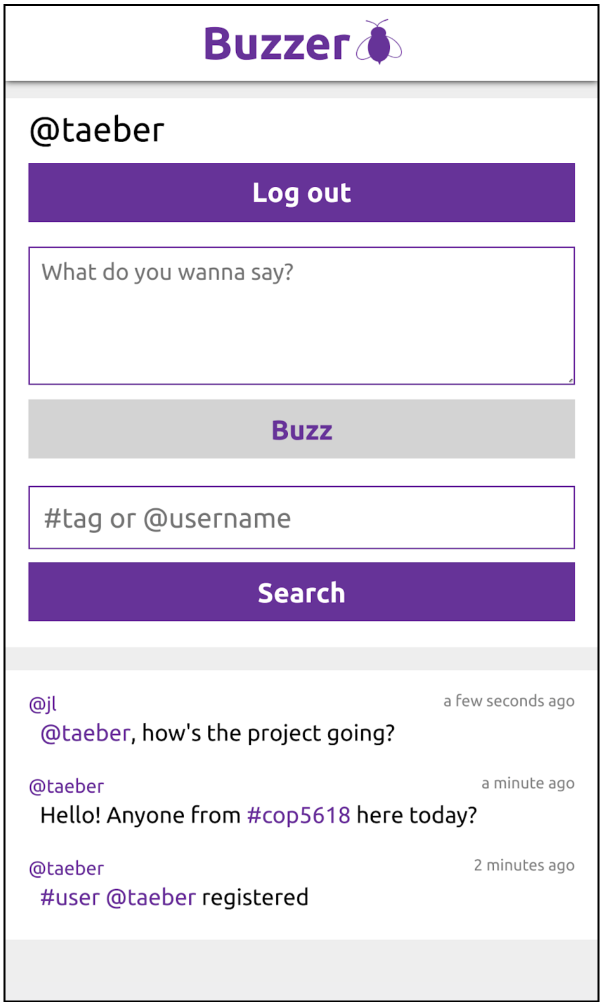
@jl a minute ago
#user @jl registered

@taeber 2 minutes ago
#user @taeber registered



@jl a few seconds ago
@taeber, how's the project going?

@jl a minute ago
#user @jl registered



@jl a few seconds ago
@taeber, how's the project going?

@taeber a minute ago
Hello! Anyone from #cop5618 here today?

@taeber 2 minutes ago
#user @taeber registered

#Locking

```
func newChannelServer(actual *kernel) *channelServer {
    // Use channels to sync access to actual kernel.
    return &channelServer{
        actual:   actual,
        post:     make(chan request, 100),
        follow:   make(chan request, 100),
        unfollow: make(chan request, 100),
        messages: make(chan request, 100),
...
        login:    make(chan request, 100),
        logout:   make(chan request, 100),
        shutdown: make(chan bool),
    }
}

func (server *channelServer) process() {
    for {
        select {
        case req := <-server.post:
            msgID, err := server.actual.Post(req)
            go respond(&req, response{data: msgID, error: err})
...
        case <-server.shutdown:
            return
        }
    }
}
```