Go Web Programming Bootcamp

Resources:

https://github.com/GoesToEleven/SummerBootCamp/tree/master/05_golang

Monday 7/6

1: Machine Setup

Instructions: www.golang-book.com/guides/machine_setup

2. Hello World

Outline:

- Files & Folders
- Terminal
- Text Editor
- Hello World
- godoc

Problems:

1. Create a program which prints “Hello World” to the terminal.
2. What is whitespace?
3. What is a comment? What are the two ways of writing a comment?
4. Our program began with package main. What would the files in the fmt package begin with?
5. We used the Println function defined in the fmt package. If we wanted to use the Exit function from the os package what would we need to do?
6. Modify the program we wrote so that instead of printing Hello World it prints Hello, my name is followed by your name.

3. Variables

Outline:

- Samples
- Blank Identifiers
- Short Variable Declarations
- Assignment
- Zero Values
- Scope
- Constants

Problems:

1. Create a program which prints “Hello <NAME>” with <NAME> replaced with your name to the terminal using a variable.
2. Use fmt.Scanf to read a user’s name and print “Hello <NAME>” with <NAME> replaced with the user’s name to the terminal.
3. Create a program which converts:
    a. Miles to Kilometers
    b. Fahrenheit to Celsius
    c. Pounds to Kilograms

4. Types

Outline:

- Integers
- Floating Point Numbers
- Strings
- Booleans

Problems:

1. How are integers stored on a computer?
2. We know that (in base 10) the largest 1 digit number is 9 and the largest 2 digit number is 99. Given that in binary the largest 2 digit number is 11 (3), the largest 3 digit number is 111 (7) and the largest 4 digit number is 1111 (15) what's the largest 8 digit number? (hint: 10^1-1 = 9 and 10^2-1 = 99)
3. Although overpowered for the task you can use Go as a calculator. Write a program that computes 32132 × 42452 and prints it to the terminal. (Use the * operator for multiplication)
4. What is a string? How do you find its length?
5. What's the value of the expression (true && false) || (false && true) || !(false && false)?

5. A Deeper Look

Outline:

- Memory
- Stack vs Heap

6. Control Structures

Outline:

- For
- If
- Switch

Problems:

- Write a program that prints out all the numbers evenly divisible by 3 between 1 and 100. (3, 6, 9, etc.)
- Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Tuesday 7/7

7. Strings Revisited

Outline:

- Immutability
- strings
- strconv
- Command line arguments

Problems:

1. Modify our miles to kilometers program to display in the following format:

   ```
   +-------------------------+
   | Miles: 50               |
   +-------------------------+
   | Kilometers: 80.47       |
   +-------------------------+
   ```

   Miles will be input by the user, and kilometers should be formatted to 2 decimal places.
2. Modify the above program so that it generates HTML instead of text. For example:

   ```
   <!DOCTYPE html>
   <html>
        <head></head>
        <body>
             Miles: 50<br>
             Kilometers: 80.47
        </body>
   </html>
   ```

   Feel free to use any HTML tags and CSS you may know.

3. Create a program which parses a query to do distance conversions. For example, from a terminal:

   ```
   $ distance_converter 50mi km
   ```

   Should produce:

   ```
   80.47km
   ```

   It should support miles (mi), kilometers (km), feet (ft) and meters (m).

4. One classic method for composing secret messages is called a square code. The spaces are removed from the english text and the characters are written into a square (or rectangle). For example, the sentence "If man was meant to stay on the ground god would have given us roots" is 54 characters long, so it is written into a rectangle with 7 rows and 8 columns.

   ```
   ifmanwas
   meanttos
   tayonthe
   groundgo
   dwouldha
   vegivenu
   sroots
   ```

   The coded message is obtained by reading down the columns going left to right. For example, the message above is coded as:

   ```
   imtgdvs fearwer mayoogo anouuio ntnnlvt wttddes aohghn sseoau
   ```

   In your program, have the user enter a message in english with no spaces between the words. Have the maximum message length be 81 characters. Display the encoded message. (Watch out that no "garbage" characters are printed.) Here are some more examples:

Input	Output
haveaniceday	hae and via ecy
feedthedog	fto ehg ee dd
chillout	clu hlt io

8. Arrays, Slices and Maps

Outline:

- Arrays
- Slices
- Maps

Problems:

1. How do you access the 4th element of an array or slice?
2. What is the length of a slice created using: make([]int, 3, 9)?
3. Given the array:

```
    x := [6]string{"a","b","c","d","e","f"}
```

what would x[2:5] give you?

4. Write a program that finds the smallest number in this list:

```
    x := []int{
        48,96,86,68,
        57,82,63,70,
        37,34,83,27,
        19,97, 9,17,
    }
```

5. Write a program that takes in a state code and returns the state’s name. (eg CA -> California)

9. Functions

Outline:

- Functions
- Multiple Return Values
- Variadic Functions
- Closure
- Recursion
- Panic & Recover

Problems:

1. **sum** is a function which takes a slice of numbers and adds them together. What would its function signature look like in Go?
2. Implement the sum function
3. Write a function which takes an integer and halves it and returns true if it was even or false if it was odd. For example half(1) should return (0, false) and half(2) should return (1, true).
4. Write a function with one variadic parameter that finds the greatest number in a list of numbers.
5. Using makeEvenGenerator as an example, write a makeOddGenerator function that generates odd numbers.
6. The Fibonacci sequence is defined as: fib(0) = 0, fib(1) = 1, fib(n) = fib(n-1) + fib(n-2). Write a recursive function which can find fib(n).
7. What are defer, panic and recover? How do you recover from a run-time panic?
8. Create a function which reverses a list of integers:

   ```
   reverse([]int{1,2,3}) → []int{3,2,1}
   ```

Wednesday 7/8

10. Review

Outline:

- Create Javascript Cheat Sheet: is.gd/XaJQal
- Pair-program on problems

Problems:

1. Create a program which reads user information (at least name and email) and creates a profile page in HTML. Also use or generate a profile image from gravatar.com.
2. Implement **WordCount**. It should return a map of the counts of each “word” in a string. (You might find strings.Fields helpful.)

   ```
   WordCount(“test test”) → map[string]int{ “test”: 2 }
   ```
3. Implement a centeredAverage function that computes the average of a list of numbers, but removes the largest and smallest values.

   ```
   centeredAverage([]float64{1, 2, 3, 4, 100}) → 3
   ```
4. Given a non-empty list, return true if there is a place to split the list so that the sum of the numbers on one side is equal to the sum of the numbers on the other side.

   ```
   canBalance([]int{1, 1, 1, 2, 1}) → true
   canBalance([]int{2, 1, 1, 2, 1}) → false
   canBalance([]int{10, 10}) → true
   ```
5. Say that a "clump" in a list is a series of 2 or more adjacent elements of the same value. Return the number of clumps in the given list.

   ```
   countClumps([]int{1, 2, 2, 3, 4, 4}) → 2
   countClumps([]int{1, 1, 2, 1, 1}) → 2
   countClumps([]int{1, 1, 1, 1, 1}) → 1
   ```

11. Pointers

Outline:

- * and &
- new
- Scan

Problems:

1. How do you get the memory address of a variable?
2. How do you assign a value to a pointer?
3. How do you create a new pointer?
4. What is the value of x after running this program:

   ```
   func square(x *float64) {
         *x = *x * *x
   }
   func main() {
         x := 1.5
         square(&x)
   }
   ```

5. Write a program that can swap two integers (x := 1; y := 2; swap(&x, &y) should give you x=2 and y=1).
6. Generalize swap into a rotate function so that it works with any number of variables:

   rotate(&x, &y, &z), with x=1, y=2, y=3, yields x=2, y=3, z=1

7. Create a rotateRight function which does the same thing as rotate but in the other direction.

12. Structs and Interfaces

Outline:

1. Structs
    a. Initialization
    b. Fields
2. Methods
3. Embedded Types

Problems:

1. What's the difference between a method and a function?
2. Why would you use an embedded anonymous field instead of a normal named field?
3. Add a new method to the Shape interface called perimeter which calculates the perimeter of a shape. Implement the method for Circle and Rectangle.

Thursday 7/9

13. Review

Outline:

- Add to Javascript Cheat Sheet
-

14. Files and IO

Outline:

- Opening, Closing, Reading, Writing
- Readers and Writers
- Copy
- Bufio, Scanner

Problems:

1. Write “Hello World” to a file
2. Read “Hello World” from that file and write it to stdout.
3. Generalize #2 and create your own version of cat which reads a file and dumps it to stdout. (my-cat file)
4. Create your own version of cp which reads a file and writes it to another file. (my-cp src dst)
5. Create a program which converts the first character of each line in a file to uppercase and writes it to stdout. (make-uppercase)
6. Remember WordCount? Use WordCount to count the frequency of words for a large book (eg http://www.gutenberg.org/files/2701/old/moby10b.txt).
7. Write a program which finds the longest word in the book.
8.
9. Base64 is often used to encode binary data safely for text transmission. Create a program which can base64 encode a file. (`my-base64 filename` should return something like ‘dGVzdA==’).
10. Modify #9 so that it supports decoding as well via a mode switch (`my-base64 MODE filename`, where MODE is ‘encode’ or ‘decode’)

15. CSV

Outline:

- Reading a CSV file

Problems:

1. Download a CSV file of state information from statetable.com and implement a program which takes in a state code (AL, CA, …) and returns a table of information for that state (name, region, etc…). Make sure to use a struct in your program.
2. Download this CSV file: https://data.illinois.gov/api/views/i97v-wnmd/rows.csv?accessType=DOWNLOAD, which contains information about monuments in the Champaign IL area. Write a program which finds the monument at the highest elevation and display infor mation about that monument as HTML.
3.

Friday 7/10

15. Review

Outline:

- io, csv

Problems:

1. Grab historical financial data from Yahoo (eg http://finance.yahoo.com/q/hp?s=GOOG+Historical+Prices) as a csv file, read that file and dump the contents as an HTML table. CHALLENGE: draw a line chart of the data.
2. Create a program which finds the md5 checksum of a file. (`my-md5 filename` should return a large hexadecimal number like 'd47c2bbc28298ca9befdfbc5d3aa4e65')
3. Modify #2 so that it computes all the md5 checksums for a directory

16. Concurrency

Outline:

- goroutines
- wait groups
- channels
- select

Problems:

1. Create a program which prints “Hello World” 5 times using 5 goroutines.
2. Create a ping-pong program with one goroutine writing “ping” to a channel, a second goroutine reading “ping” and then writing “pong” to another channel
3. Modify #1 so that you use a wait group to wait for all 5 hello world printers to finish
4. Modify #15.2 so that it computes all the md5 hashsums for a directory of files concurrently.

17. Time

Outline:

- Duration
    - Sleep
- Time
    - Parse
    - Format

Problems:

1. Create a program which sleeps for 10 seconds then prints “Hello World” to the screen
2. Create a program which can find the number of days between two dates: (`datediff 2015-07-10 2015-07-11` should give you 1)

Monday 7/12

18. Why Go?

- Simple, well-defined language
  - https://youtu.be/5kj5ApnhPAE
- Powerful, well-documented API
- Great Tools
  - Compiler speed, modularity built in
- Concurrency
- OO without hierarchy

19. JSON

Outline:

- Decode, Unmarshal
- Encode, Marshal

Problems:

1. Create a struct, encode it as JSON, and write it out to stdout.
2. Create a JSON file and a program to read that JSON file into a struct and write it to stdout.
3. Create a program which can read the csv file from #15.1 and write it as a JSON file.

20. Packages

Outline:

- Multiple packages
- go get
- godoc, go-search, etc…
- godep, gb, etc…

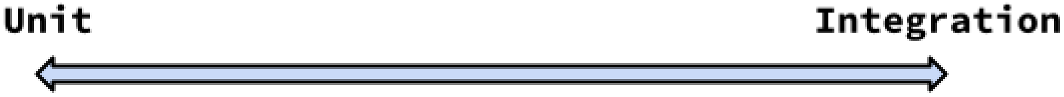
Problems:

1. Create a new package ‘week2/day1/hello’ with a function named ‘Hello’ which prints ‘Hello World’ to the screen. Use this package in a program and call this ‘Hello’ function.
2. Create a new package ‘week2/day1/converters’ with some of the conversion functions we created last week (miles to km, fahrenheit to celsius, etc…). Rewrite the conversion program to use this library.
3. Use a 3rd party library to add color to your program (for example: github.com/ttacon/chalk)

21. Testing

Outline:

- go test
- Test functions
- Test strategies
    - Unit Testing
    - Integration Testing



    - Mocking (File vs Reader)
- Benchmarks
- goconvey, race detector

Problems:

1. Create a new package ‘week2/day1/testing-example’ with the Sum function we wrote last week in a file ‘sum.go’. Add a ‘sum_test.go’ file which tests this function.
2. Add a benchmark function to benchmark the Sum function.

22. Networking

Outline:

- Protocols
- OSI Model
  - https://en.wikipedia.org/wiki/OSI_model#Description_of_OSI_layers
  - IP, TCP, HTTP
- An Exploration of a Web Request
  - DNS
  - HTTP: net tab (fiddler)
  - TCP: wireshark, traceroute
  - Packet Switched Routing
- Distributed Systems
  - Client / Server
  - Peer to Peer
- Web Architecture
  - DNS
    - nameservers, dig, whois
  - Stacks
    - Reverse proxies / load balancers (nginx, haproxy)
    - Web servers (nginx, apache, IIS, …)
    - Backend servers (databases, mail servers, …)
    - Cloud services
    - Queues

23. TCP Servers

Outline:

- Listen, Accept, Conn
- telnet (for windows: `pkgmgr /iu:"TelnetClient"`)
- Dial

Problems:

1. Create a TCP server that sends ‘Hello World’ to new connections and then closes them.
2. Create an ‘echo’ TCP server. It should accept a connection and write to the connection anything that’s sent to it.

Tuesday 7/14

24. Review

- json
- packages
- tests
- tcp listen and dial

Also: init

Problems:

1. Create a simplified redis clone which accepts GET, SET and DEL commands. ‘GET <KEY>’ should write the value of <KEY> followed by a new line. ‘SET <KEY> <VALUE>’ should set the value of <KEY>. ‘DEL <KEY>’ should remove the value. Data should be stored in memory.
2. Modify ‘echo’ so that it returns messages as rot 13.
3. Create a chat room server. A client can connect and send messages to the server. Those messages will be broadcast to any other currently connected clients.

Digression: On Redis

25. HTTP Servers (From Scratch)

Outline:

- HTTP Spec: http://www.w3.org/Protocols/rfc2616/rfc2616.txt
- Format:

```
Request = Request-Line
          *(header CRLF)
          CRLF
          [ message-body ]
Request-Line = Method Request-URI HTTP-Version CRLF
Header = Name: Value
Response = Status-Line
           *(header CRLF)
           CRLF
           [ message-body ]
Status-Line = HTTP-Version Status-Code Reason-Phrase CRLF
```

- Verbs / Methods (GET, POST, PUT, DELETE, HEAD)
- Headers
  - Accept
  - Connection
  - Content-Type
  - Location
  - Range
  - Referer
  - Transfer-Encoding
  - WWW-Authenticate
- Caching: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching

Problems:

1. Create a TCP server which can handle a simple HTTP request and return ‘Hello World’

2. Create a TCP server which can handle a simple HTTP request and return the URL that was passed into it

26. The net/http package

Outline:

- ListenAndServe, Serve
- Handler, Request, ResponseWriter
- ServeMux, DefaultServer

Problems:

1. Create a ‘Hello World’ http server using the net/http package
2. Create an echo http server that returns a plain text page contain the url the user accessed (going to localhost/whatever displays ‘/whatever’)
3. Create an http server which returns an html page with a picture of a cat for ‘/cat’ and a picture of a dog for ‘/dog’ using a ServeMux

27. Serving Files

Outline:

- Manually
- ServeContent
- ServeFile
- FileServer

Problems:

1. Create an http server that serves a file from the disk
2. Modify #1 to use ServeContent
3. Modify #1 to use ServeFile
4. Create a static http server (perhaps named static-http) that serves the contents of the current working directory.

Wednesday 7/15

28. Review

- http servers
- mux
- files

29. Templates

Outline:

- ParseFiles, ParseGlob
- Execute, ExecuteTemplate
- Actions

Problems:

1. Create an http application that returns Hello World using a template.
2. Modify #1 so that it returns the currently selected URL (use a variable in your template)
3. Modify the financial CSV program we wrote so that it outputs HTML using a template file instead of a string. Zebra stripe the table.

30. Forms and the Query String

Outline:

- Request.URL.Query().Get(KEY)
- Request.FormValue(KEY)
- Request.FormFile(KEY)

Problems:

1. Create an http application which has a page with a form that takes in first and last name and on submit displays “Hello <FIRST> <LAST>”
2. Create an http application with a form that accepts a file and uploads it to “/tmp”.

31. Cookies, Session and Context

Outline:

- Set-Cookie
- Request.Cookie
- Sessions

Problems:

1. Create an http application that tracks how many times a user accesses your web page with a cookie.
2. Create an http application that supports at least 2 endpoints: login and logout. login should accept a form and save a cookie. (with at least the username) logout should clear the cookie.

32. TLS and Certificates

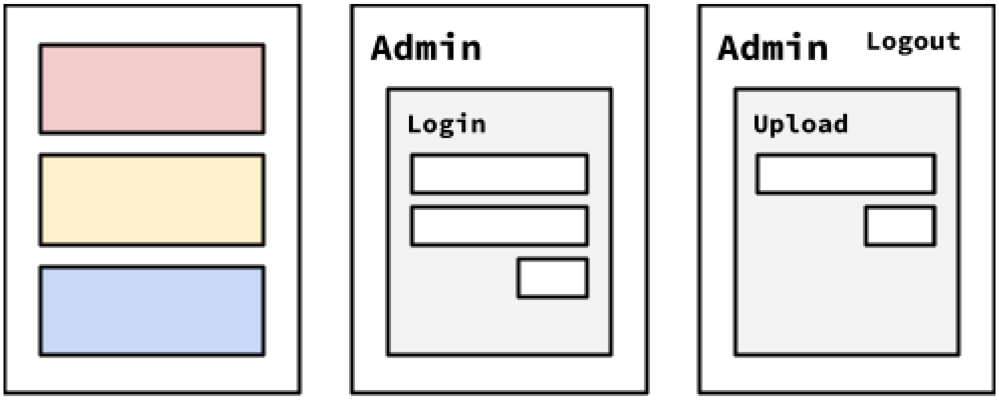
Outline:

- ListenAndServeTLS
- go run $(go env GOROOT)/src/crypto/tls/generate_cert.go --host=somedomainname.com

Problems:

1. Create a certificate and use it with ListenAndServeTLS in an http server.

33. Project: Photo Blog



Our goal for this project is to create a basic photo blog. It should have a few pages:

- The initial page (https://localhost:8080/) should list the most recent uploaded photos. Clicking a photo should download it. (hint: HTML5 download attribute or the Content-Disposition header)
- There should be an admin page (https://localhost:8080/admin) which allows a user to login (you can hardcode the username and password).
- Once logged in the admin user should be presented with a form to upload an image. The uploaded image should then be accessible on the main page.
- The admin section should require TLS. A self-signed certificate is ok. It’s ok to require TLS for the regular section of the site as well.
- Try to use the HTML/CSS skills you learned earlier to make the photoblog look nice. Make sure to upload some interesting photos (that perhaps reflect your personality)

Feel free to team up on this project. At the end of the day everyone will demo what they built and explain their code.

This project is intentionally open-ended. Using Google is not cheating.

Thursday 7/16

34. Review

- Routes with Serve Mux
- Templates
- Sessions
- Listing a Directory
- Uploading a File
- Setting up TLS

37. Manual Server Management

Outline:

- Virtual Machines
- SystemD
- Deploy
- Monitoring

38. App Engine Setup

Outline:

- https://cloud.google.com/appengine/docs/go/gettingstarted/introduction

Friday 7/17

39. App Engine Introduction

Outline:

- App Engine
  - Managed PAAS on Google Infrastructure
  - Free to Start, Affordable Limits
  - Autoscales
    - cloud.google.com/appengine/docs/scaling
  - Managed VMs, Compute Engine, Cloud SQL
- Project Layout
  - go stuff
  - app.yaml
  - instead of `goapp deploy`
    - appcfg.py update --oauth2 ./
- Console
  - APIs
  - Logs
  - Quotas

40. Users

Outline:

- Google Logins (cloud.google.com/appengine/docs/go/users)
    - user.LoginURL, user.LogoutURL
    - Through the YAML: https://cloud.google.com/appengine/docs/go/config/appconfig#Go_app_yaml_Requiring_login_or_administrator_status
    - OAuth?
    - Or Manual

Problems:

- Modify the photo-blog app so it uses Google logins

40.5: Custom Domains

41. Session Redux

Outline:

- ClearContext
- Other Routers
- Three Options:
    - Use Google logins and avoid session
    - Use Gorilla sessions with manual routing (either via http.ServeMux or some other router)
    - Use a Session ID and Memcache

42. Memcache

Outline:

- Get, Set, Delete
- Increment

Problems:

1. Use memcache to implement a counter every time a user visits the page and show the value of the counter.
2. Update the counter application so that it requires a login and add a per-user counter in addition to the global counter.

43. DataStore

Outline:

- Entities
  - Kind
  - Identifier (string or int)
  - Ancestors
  - Put, Get, Key, IncompleteKey
- Queries
  - Indexes (index.yaml, https://cloud.google.com/appengine/docs/go/config/indexconfig)
  - kind, filters, sort
  - KeysOnly
  - Project

Problems:

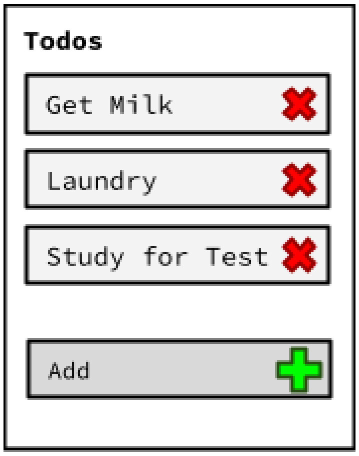
1. Create a user profile page that requires login (via Google) and save profile information in the data store.

44. AJAX

Outline:

- Building APIs

45. Project: ToDo



Create a google appengine application which implements a simple todo app. It should require login via Google and store a todo list for each user (one todo list per user). That todo list should only be visible to that user.

The todo list should support:

- A list of each item
- Adding a new item to the end of a list
- Removing an item from the list

This application should be implemented using Javascript and a JSON backend. For example to build the list view you should make a request to your server (perhaps GET /api/todos) which will return a JSON array. Similarly removing todo items can use (DELETE /api/todos/{TODO_ID}) and adding can use (POST /api/todos).

Monday 7/20

46. Todo Application Walkthrough

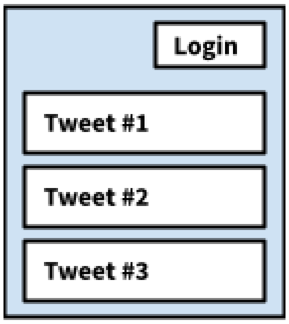
47. Project: Twitter Clone

For this project we are going to build a twitter clone. As we learn more about App Engine we will add features to our application.

To get started we want to support the following:

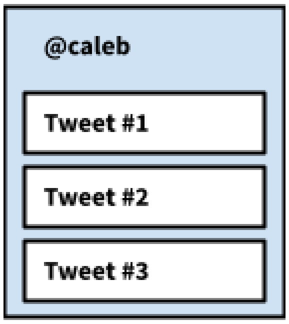
- The home page should display a login button and list recent tweets:

  twitter-clone.appspot.com/

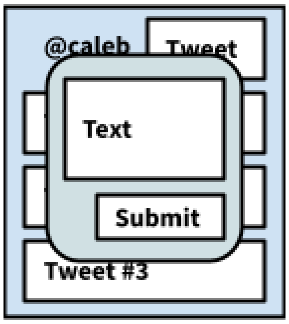
- User accounts can be done with standard Google Login. An account should not be required to view someone’s tweets but should be required to post tweets.
- The first time a user logs in they should be required to create a username. This username will be used when they tweet (@whomever). Usernames are required to be unique.
- Tweets should be visible on a user’s profile. For example we might have:

  twitter-clone.appspot.com/<USERNAME>



Which would list the most recent tweets for that user.

- For logged-in users the top right should have a tweet button which displays an overlay with a form inside of it. Submitting the form should create a new tweet.



- There should also be a logout button. Logout should log the user out of the twitter application but not out of their google account.
- Tweets should be limited to 140 characters. Use an Ajax API to implement tweet submission.

Tuesday 7/21

48. Logs

Outline:

- Infof, Errorf, Debugf, …

49. Testing

Outline:

- goapp test

50. Review

- Process
    - Github
    - Managing Complexity Using Issues
- Assessment
    - Git
    - Deployed
    - Home Page
        - Shows Tweets
        - Login Button
    - User Profile Page
        - Shows Username
        - Shows Tweets
        - Login or Tweet+Logout Buttons
    - Tweet Overlay
        - Submission Creates Tweet and Refreshes
    - ~~At least one test for data functions~~

51. Projection Queries

Outline:

- We can restrict the data returned with .Project():

  ```
      q := datastore.NewQuery("People").Project("FirstName",
  "LastName")
  ```

  Would only return the first and last name properties.

Wednesday 7/22

52. Twitter Clone: Followers

A user in Twitter has the ability to “Follow” another account. When a logged in user goes to the twitter clone they should only see the Tweets of their followers in their Timeline.

Add to our twitter app the ability to follow / unfollow another user and update the home page to show Tweets accordingly. Also add a page to a user’s profile page to show their followers.

53: Review

- Follower Implementation

54. Data Modeling without Joins

- FYI: http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html

55. Mail

Outline:

- Sending: http://godoc.org/google.golang.org/appengine/mail#Send

  -
    ```go
    func confirm(w http.ResponseWriter, r *http.Request) {
      c := appengine.NewContext(r)
      addr := r.FormValue("email")
      url := createConfirmationURL(r)
      msg := &mail.Message{
        Sender:  "Example.com Support <support@example.com>",
        To:      []string{addr},
        Subject: "Confirm your registration",
        Body:    fmt.Sprintf(confirmMessage, url),
      }
      if err := mail.Send(c, msg); err != nil {
        c.Errorf("Couldn't send email: %v", err)
      }
    }
    ```

56. Twitter Clone: Send an Email for @mentions

When a user tweets a message that contains another username (for example “hi @example”) send an email to the mentioned user with the content of the message.

CHALLENGE: write an incoming mail handler which can post a tweet on a user’s behalf via email.

57. Realtime Updates

Users should not be required to refresh their page to receive updates. Implement realtime updates by periodically polling an API endpoint and alerting the user when new tweets are available.

Thursday 7/23

59. Channels

Outline:

- Go: Create, Send
- Javascript:
    - ○ <script type="text/javascript" src="/_ah/channel/jsapi"></script>
    - ○ goog.appengine.Channel(token)
    - ○ open(optional_handler)

Problems:

1. Build a hello world app engine channel application.
2. Build a chat application using app engine channels.
    a. https://github.com/golang-book/bootcamp-examples/tree/master/week3/day4/chat-example

60. Search

Outline:

- Documents, Indexes, Queries and Results
- Open, Put, Get, Search, Delete

Problems:

1. Build a simple movie information site that has search capabilities. It should require login, any user can add a movie, and it should support searching.

Friday 7/24

61. Movie Search Implementation

Outline:

- Code
- Open Search

62. Google Cloud Storage

This is going to be painful...

Outline:

- https://cloud.google.com/appengine/docs/go/googlecloudstorageclient/getstarted
- ~/go_appengine/dev_appserver.py . --appidentity_email_address <your_app_email_address>@developer.gserviceaccount.com --appidentity_private_key_path pem_file.pem
- google.golang.org/appengine/file.DefaultBucketName(ctx)
- google.golang.org/cloud/storage, golang.org/x/oauth2/google
    - (https://github.com/calebdoxsey/tutorials/blob/master/appengine/cms/api.go#L194)
    - Bucket
    - Writer
    - Reader

Problems:

1. Create an http application with an upload form which uploads files to Google Cloud Storage
2. Add movie images to the movie information site

63. URLFetch

Outline:

- urlfetch.Client(ctx)
    - Do, Get, Head, Post, PostForm
- developer.github.com/v3/markdown/

Problems:

1. Add markdown support to the movie information site using Github’s API:
    a. curl -X POST -d '{

```
      "text": "Hello world github/linguist#1 **cool**, and #1!",
      "mode": "gfm",
      "context": "github/gollum"
    }' https://api.github.com/markdown
```

Monday 7/27

64. Review

Outline:

- Movie Markdown Walkthrough
- Movie Posters via GCS

65. Self-Destructing Messages

Build an application which allows sharing a message which after a certain amount of time expires.

Improve the security of the application by encrypting the message with a randomly generated private key which is never stored on the server.

66. Basic File Browser

Build an application which can access any GCS bucket using a provided JSON credentials file. It should support uploading, downloading and listing directories.

67. API Example: Financial CSV

Build an application which can take in two stock ticker symbols and calculate the covariance between their returns over the last 5 years. Add a chart of their two relative returns.

Tuesday 7/28

68. Review

- Financial CSV, charts, correlation

69. API Example: Stripe

Create an application which uses the Stripe API to allow site payments.

https://stripe.com/docs/tutorials/charges
https://github.com/stripe/stripe-go

70. API Example: 3rd Party OAuth

Create an application which uses a 3rd party for authentication rather than the built in Google provider.

Problem:

- Implement 3rd party login with another provider (google, twitter, facebook, etc…)

71. Task Queue

Outline

- cloud.google.com/appengine/docs/go/taskqueue
- google.golang.org/appengine/delay

72. Scheduled Tasks