

# Basic Go

---

USING GO ON THE WEB

[NXJ.ME/MITGOPRES](https://nxj.me/mitgopres)



# About Us

---

- Sam Skinner
  - Georgia Tech '15
- Christian Lemieux
  - RIT '15

Web Developers for Next Jump since 2015



- 
- eCommerce and loyalty product installed in the majority of Fortune 1000 companies
    - In other words, we run a large scale web application that handles lots of traffic and transactions
  - Developing suite of culture mobile apps for our clients
  - Majority engineers
    - Web developers
    - Database
    - Networking
  - Stack
    - Go, PHP, SQL Server, Apache, Linux

# Agenda

---

- Go overview
  - Why we use it and why you might want to use it
  - Why you might not want to use it
- Crash course
  - Scratching the surface
- Simple web application in Go
  - End to end example using Go to make a web application

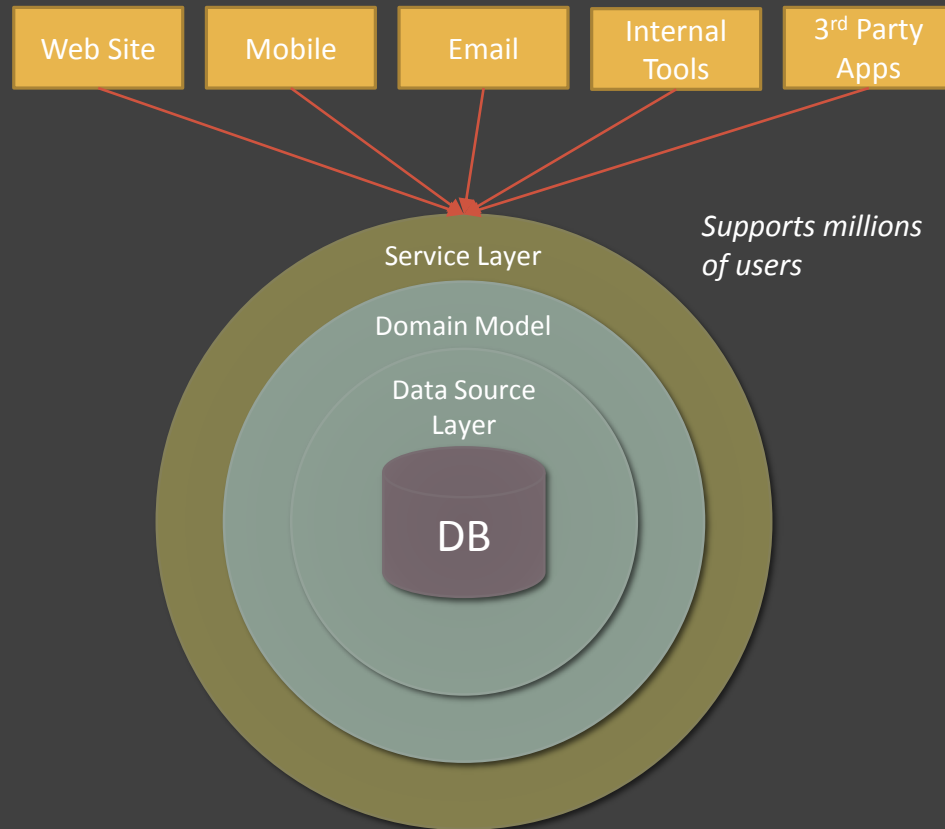
# Where Go fits in

Example Products / Services:

- Virtual Currency
- Travel reservation engine
- Transaction processing

We've implemented our service layer in Go

- Consistent data access
  - Logging
  - Security
- Smarter caching
- Shared definitions of objects



M. Fowler, *Patterns of Enterprise Application Architecture*, 1 edition. Boston: Addison-Wesley Professional, 2002.

# Important Characteristics

---

- Created in 2009
- Statically typed
- Compiled
- Similar to C

# Hello World

---

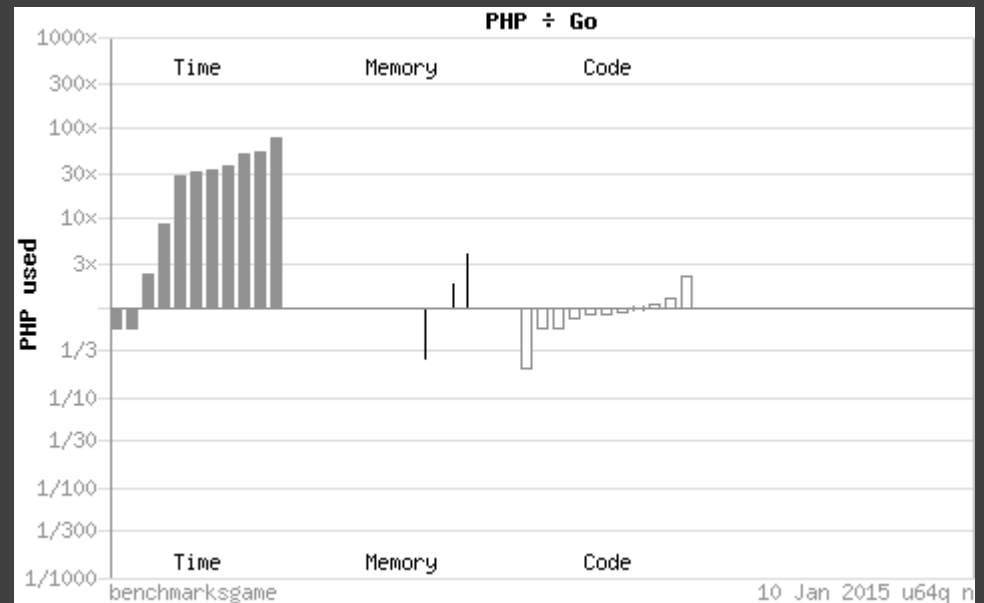
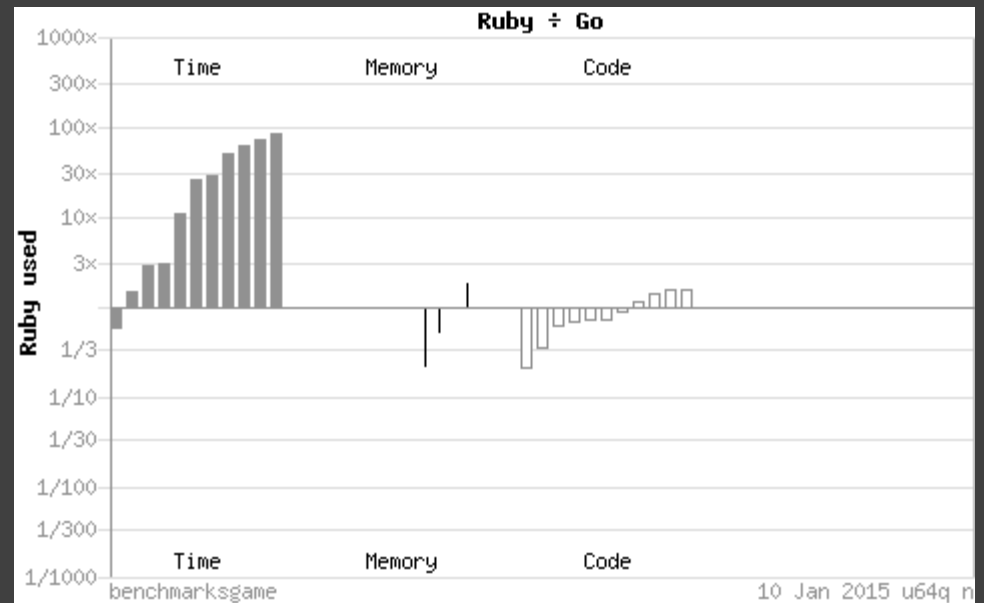
```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world")
}
```

# Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt



Note: These aren't web-specific benchmarks

<http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=yarv&lang2=go>



## Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

```
package main

import "fmt"

func main() {
    var a int64
    var b string
    a = 10
    b = "10"
    fmt.Println(a + b)
}
```

invalid operation: a + b (mismatched types int64 and string)

Compiler helps keep code clean:

- No unused variables
- Catches simple mistakes
- Safer refactoring

*Predictable code is important. No surprises.*

## Similar code in Java Script

```
var a;  
var b;  
  
a = 10;  
b = "10";  
  
console.log(a + b);
```

➔1010

## Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

*Predictable code is important. No surprises.*

# Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

## Go Looping

<http://golang.org/ref/spec>

```
package main

import "fmt"

func main() {
    var numbers []int64 = []int64{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    for index, value := range numbers {
        fmt.Printf("Index: %d Value: %d\n", index, value)
    }
}
```

## Java Script Looping

```
var numbers = [1,2,3,4,5,6,7,8,9,10];

var number;
for (number in numbers) {
    console.log("Index: " + number + " Value: " + numbers[number]);
}

var len = numbers.length;
var i = 0;
while (i < len) {
    console.log("Index: " + i + " Value: " + numbers[i]);
    i = i + 1
}

i = 0;
do {
    console.log("Index: " + i + " Value: " + numbers[i]);
    i = i + 1
} while(i < len)

numbers.map(function(value, index) {
    console.log("Index: " + index + " Value: " + value);
});
```

# Notable Features Missing From Go

- Inheritance
- Overloading
- Generics

## Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

## Other Examples

### - Prefix increment operator

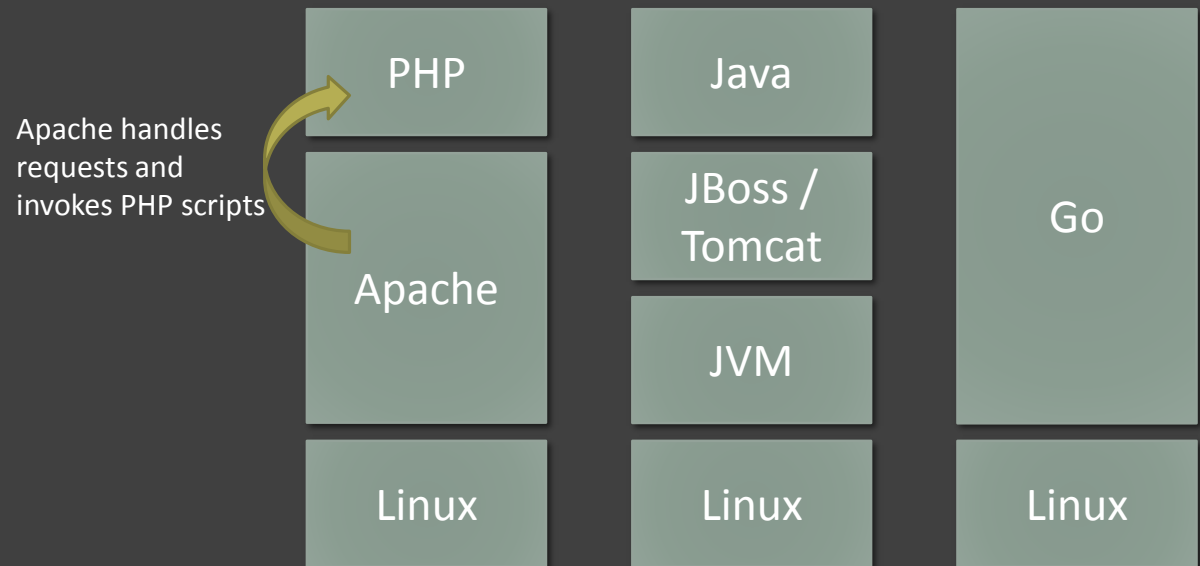
- ++i

### - Ternary form

- ([if] ? [true-value] : [false-value])

<http://golang.org/doc/faq>

# Configuration: Go vs PHP vs Java

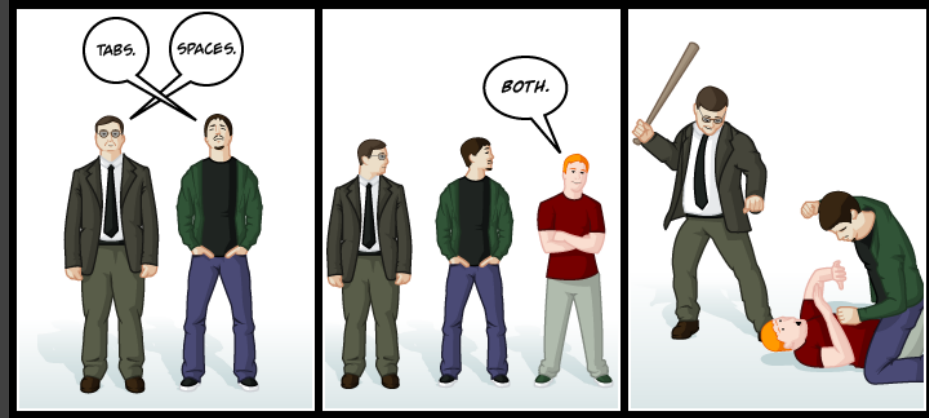


## Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

- Running a Go server has less overhead
- Less configuration (xml files)
- Tradeoff (you have to do more yourself)
  - Risk reinventing the wheel

# go fmt – no more code standards



<http://www.emacswiki.org/emacs/TabsSpacesBoth>

## go fmt your code:

- Speed
  - Static Typing
  - Simple Spec
  - Simple Stack
  - go fmt
- Easier to **write**: never worry about minor formatting concerns while hacking away
  - Easier to **read**: when all code looks the same you need not mentally convert others' formatting style into something you can understand
  - Easier to **maintain**: mechanical changes to the source don't cause unrelated changes to the file's formatting; diffs show only the real changes
  - **Uncontroversial**: never have a debate about spacing or brace position ever again!

## Choosing Go

- Speed
- Static Typing
- Simple Spec
- Simple Stack
- go fmt

## Summary

We use Go because:

- It's fast
- It's simple
  - Easy to learn
  - There aren't wildly different styles – no need to be clever
  - Readable
- It's predictable
  - Static typing gives confidence
- Our developers voted for it
  - Top contenders: Go, Scala, Java

# Why not Go?

---

## Random Internet Opinions

*“Go has the potential to become the next C, that is, to hold the computing world back for 4 (or more?) decades to come. I wish it never had come into existence.”*

- [fishface60](#)

“Seriously? Why don't we just go back to writing programs by using punch cards? I understand the need to reduce moving parts, but deliberately omitting a very useful abstraction mechanism is just insane.”

- [torquay](#)

Nobody's perfect...

- Lacks features that are common in many modern languages
- Strict types can be a burden
- Precise memory management is not easy
- New Language → new libraries
- Simple spec → more code (less clever)



# Go Crash Course

---

- Packages
- Variables
- Functions
- Flow Control
- Important Types

For a great intro visit: <http://tour.golang.org>

# Packages

---

- Packages are modules of code
  - Allows you to split code across multiple files and folders
- Packages are **imported** based on the path
  - The last part of the path is typically the package name
- The default package is named main
  - Starting point of your Go application is in the main package

```
package main

import (
    "fmt"
    "html"
)

func main() {
    var data string = "<h1> hi </h1>"
    fmt.Println(html.EscapeString(data))
}
```

Common types:

Bool, int, uint, byte, float, string

<https://golang.org/ref/spec#Types>

# Variables

---

- Can be declared at the function or package level
- := is a great shortcut inside functions
- Case matters for package level variables
  - First letter upper case: Public
  - First letter lower case: Protected
- Variables are typed
  - Can't assign to a string then assign to a number

```
package main

import (
    "fmt"
)

var (
    FirstName string // Accessible everywhere
    LastName  string // only accessible in the package
)

func main() {
    var age int // only accessible in this function
    FirstName = "John"
    LastName = "Hilliard"
    age = 28

    city := "Cambridge" // Declare and initialize automatically
    fmt.Printf("%s %s is %d years old and lives in %s.\n",
        FirstName, LastName, age, city)
}
```

# Functions

---

- Exported names work like variables
- Functions take zero or more arguments
- Functions can return zero or more results

```
package main

import "fmt"

func square(x int) int {
    return x * x
}

func main() {
    squaredNumber := square(42)
    fmt.Println(squaredNumber)
}
```

# Flow Control

---

- if, switch, and for
  - No parenthesis
  - No semicolons
- No do, or while
- range is a special function for iterating

```
package main

import "fmt"

func main() {
    numbers := []int{1, 2, 3, 4, 5}
    for index, value := range numbers {
        if index <= 1 {
            fmt.Printf("Beginning Number: %d\n", value)
        } else if index <= 3 {
            fmt.Printf("Middle Number: %d\n", value)
        } else {
            fmt.Printf("End Number: %d\n", value)
        }
    }
}
```

# Important Types

- There are a few other important types
  - Slices – growable arrays
  - Maps – like a dictionary or JS object
  - Structs – more rigid data type
- The make function is used to allocate maps and slices
- The new function is used to allocate a new struct

```
package main

import "fmt"

type (
    Person struct {
        Name string
        Age  int
    }
)

func main() {
    numbers := []int{1, 2, 3, 4, 5}
    numbers = append(numbers, 10)
    fmt.Println(numbers)

    // Create a map
    var numberMap map[int]string
    // Allocation with make
    numberMap = make(map[int]string)
    numberMap[1] = "one"
    numberMap[2] = "two"
    numberMap[3] = "three"
    fmt.Println(numberMap)

    // Declare a person
    var john *Person
    // Allocate with new
    john = new(Person)
    john.Name = "John Hilliard"
    john.Age = 28
    fmt.Println(john)
}
```

# Learn more

-There is a lot more to learn

- Pointers
- Concurrency
- Interfaces
- Methods
- Embedded types
- Higher order

-The standard library

-<http://tour.golang.org>

The screenshot shows the official Go Programming Language website. At the top, there's a navigation bar with links for 'Documents', 'Packages', 'The Project', 'Help', 'Blog', and a 'Search' button. The main content area is divided into several sections:

- Try Go**: A section with a code editor containing a Go program that prints "Hello, 世界". Below the editor is a text input field with "Hello, World!" and buttons for 'Run', 'Share', and 'Tour'. An orange arrow points from the 'Tour' button to the 'Featured articles' section.
- Featured video**: A video player showing a video titled "Andrew Gerrand - Go: a simple programming environment - Railsberry 2013".
- Featured articles**: A section with two articles:
  - Errors are values**: A common point of discussion among Go programmers, especially those new to the language, is how to handle errors. The conversation often turns into a lament at the number of times the sequence...
  - GothamGo: gophers in the big apple**: Last November more than two hundred gophers from all across the United States got together for the first full-day Go conference in New York City.
- Download Go**: A section with a cartoon gopher and text stating "Binary distributions available for Linux, Mac OS X, Windows, and more."

At the bottom, there's a footer with the text "Build version go1.4." and a license notice: "Except as noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code is licensed under a BSD license." There are also links for "Terms of Service" and "Privacy Policy".

# Example Website

---

End to end Example:

- I'll walk through my process
- Libraries that I used
- Product goals
  - View all champions from League of Legends
  - Ability to filter
  - View more details on a champion

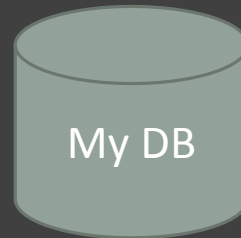


<http://paste.jfh.me:8888/home>



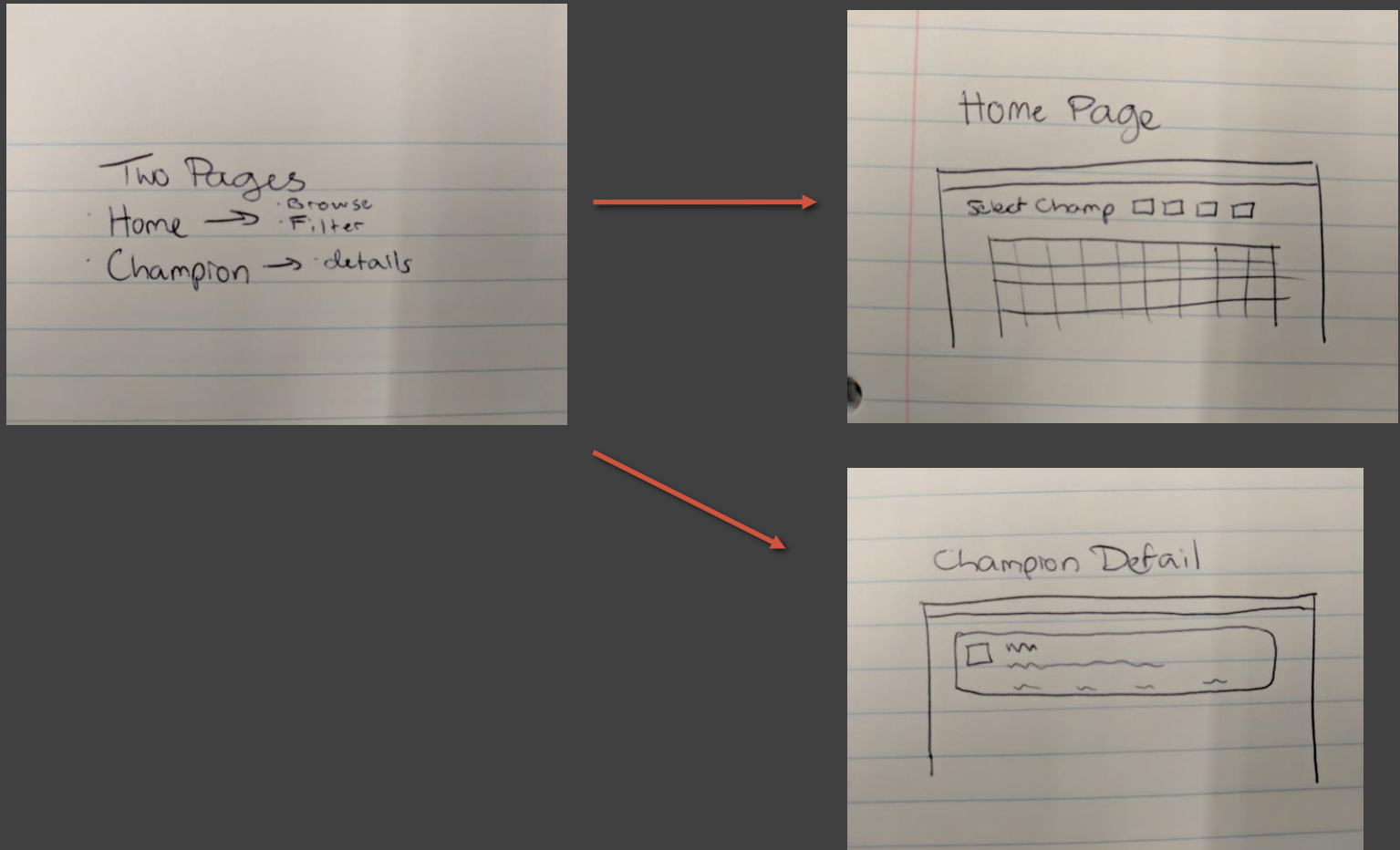
# Build Data

---



-Insert JSON data into SQLite Db

# Design



# Code

---

## Three Important Libraries

### -[net/http](#)

- HTTP server
- HTTP client

### -[database/sql](#)

- Generic interface for various SQL implementations

### -[html/template](#)

- Simple template library

# net/http

---

## Server

```
package main

import (
    "handlers"
    "log"
    "net/http"
)

func main() {

    // Handle all of the dynamic pages
    http.HandleFunc("/", handlers.Home)
    http.HandleFunc("/champion/", handlers.Champion)

    // Delegate static requests to http.FileServer. All of those
    // requests will look inside the /static folder
    http.Handle("/js/", http.FileServer(http.Dir("./static")))
    http.Handle("/css/", http.FileServer(http.Dir("./static")))
    http.Handle("/img/", http.FileServer(http.Dir("./static")))
    http.Handle("/favicon.ico", http.FileServer(http.Dir("./static/img")))

    log.Println("Starting Server")
    // Start the server on port 8888.
    log.Fatal(http.ListenAndServe(":8888", nil))
}
```

# database/sql

---

## 1. Create a connection

```
db, err := sql.Open("sqlite3", "./champions.db")
log.Printf("Opening connection to champions database")

if err != nil {
    log.Fatalf("Could not open connection to DB: %q", err)
}
```

## 2. Pull the data

```
func GetAllChampions() []*objects.Champion {
    e := data.GetQueryEngine()
    rows, err := e.Query("select * from champions", nil)
    champions := make([]*objects.Champion, 0)

    if err != nil {
        log.Printf("There was an issue fetching the list of champions: %q", err)
        return nil
    }
    defer rows.Close()

    for rows.Next() {
        champions = append(champions, scanChampion(rows))
    }
    return champions
}
```

# html/template

---

```
<h1> Select a Champion </h1>

<div class="container">
  <div class="row">
    {{ range .champions}}
      <div class="col-md-1 champion no-lr-padding">
        <a href="/champion/{{ .Id }}">
          
        </a>
        <input type="hidden" class="tag" value="{{ .Tag }}" />
      </div>
    {{end}}
  </div>
</div>
```

Define a template

```
baseLayout.Execute(rw, body)
```

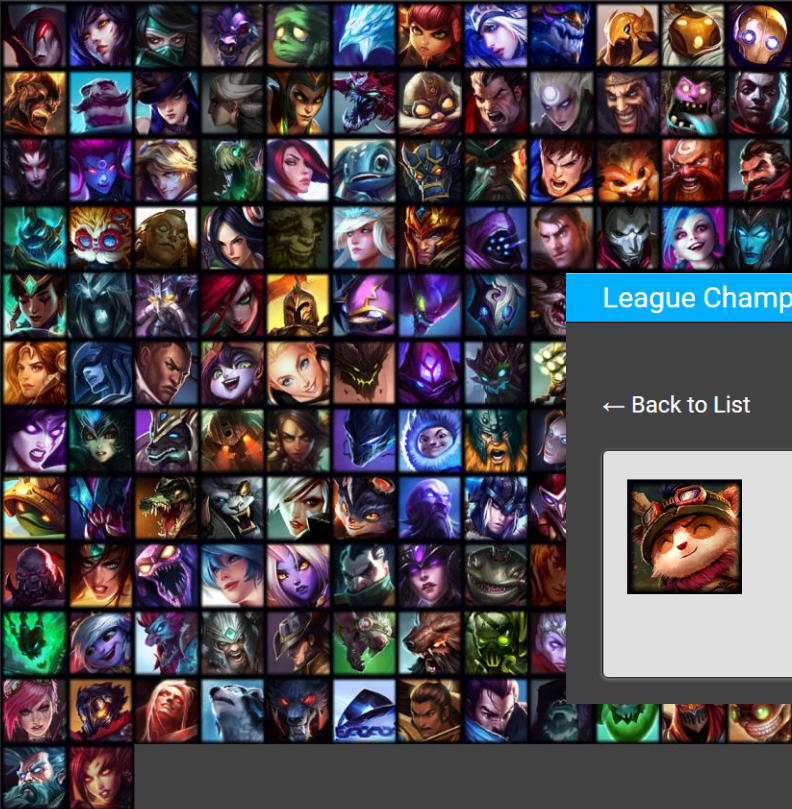
Execute

# Putting it all together

League Champions


Select a Champion

All Fighter Mage Assassin Tank Marksman Support



League Champions

[← Back to List](#)



Teemo

the Swift Scout

Teemo is a legend among his yordle brothers and sisters in Bandle City. As far as yordles are concerned, there is something just slightly off about him. While Teemo enjoys the companionship of other yordles, he also insists on frequent solo missions ...

ATTACK: 5 / 10

DEFENSE: 3 / 10

MAGIC: 7 / 10

DIFFICULTY: 6 / 10

# Thanks!

---

sskinner@nextjump.com

clemieux@nextjump.com

<https://github.com/ChristianLemieux/mit-lolchampions>