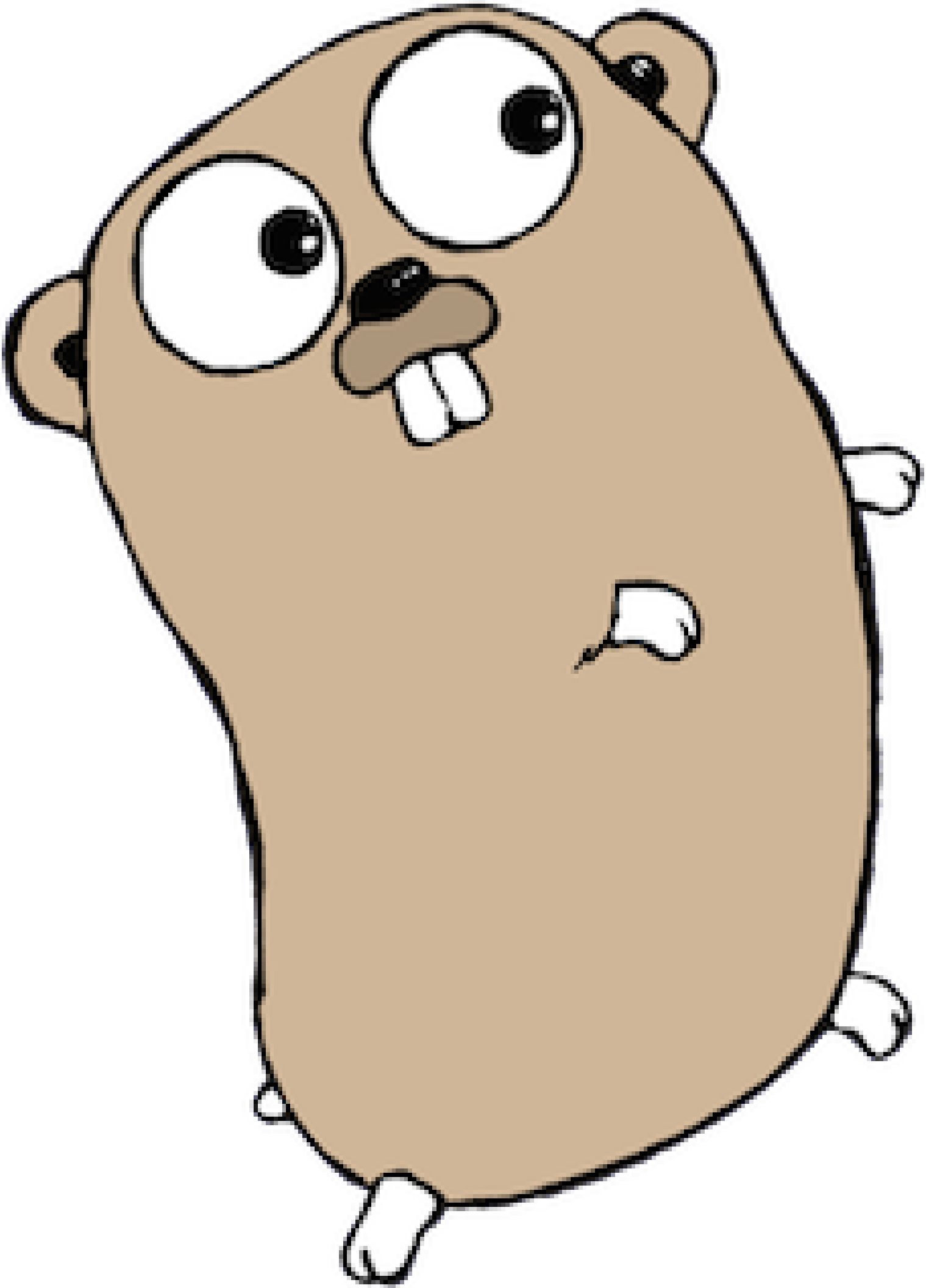


Go and MongoDB on MongoLab and Heroku

Satish Talim

This book is for sale at http://leanpub.com/goandmongodbonmongolabandheroku

This version was published on 2015-04-15

Also By Satish Talim

How Do I Write And Deploy Simple Web Apps With Go?

Building a package in Go

How do I use Sourcegraph with Go?

How do I use Sourcegraph with Ruby?

How to Deploy a Go Web App to the Google App Engine 101

How to Deploy a Go Web App to Heroku 101

How do I use the template package and handle forms?

New to Go? Want to know how to use Go with MongoDB On MongoLab and Heroku? This eBook quickly guides you to do exactly that.

Contents

Go and MongoDB on MongoLab and Heroku

What's NoSQL?

In the words of Karl Seguin[1]:

> NoSQL is a broad term that means different things to different people. Personally, I use it very broadly to mean a system that plays a part in the storage of data. Put another way, NoSQL (again, for me), is the belief that your persistence layer isn't necessarily the responsibility of a single system. Where relational database vendors have historically tried to position their software as a one-size-fits-all solution, NoSQL leans towards smaller units of responsibility where the best tool for a given job can be leveraged. So, your NoSQL stack might still leverage a relational database, say MySQL, but it'll also contain Redis as a persistence lookup for specific parts of the system as well as Hadoop for your intensive data processing. Put simply, NoSQL is about being open and aware of alternative, existing and additional patterns and tools for managing your data.

You might be wondering where MongoDB fits into all of this. As a *document-oriented database*, Mongo is a more generalized NoSQL solution. It should be viewed as an alternative to relational databases. Like relational databases, it too can benefit from being paired with some of the more specialized NoSQL solutions.

Also, most web apps will use either a relational or document-oriented database. A number of Platform-as-a-Service (PaaS) providers[2] allow you to use Go applications on their clouds but, as of this writing, only Heroku has a paid MongoDB add-on[3] or we could use Heroku and access our MongoDB database on MongoLab (later on, we shall see how).

What's MongoDB?

MongoDB is a high-performance, open source, schema-free, document-oriented database written in C++.

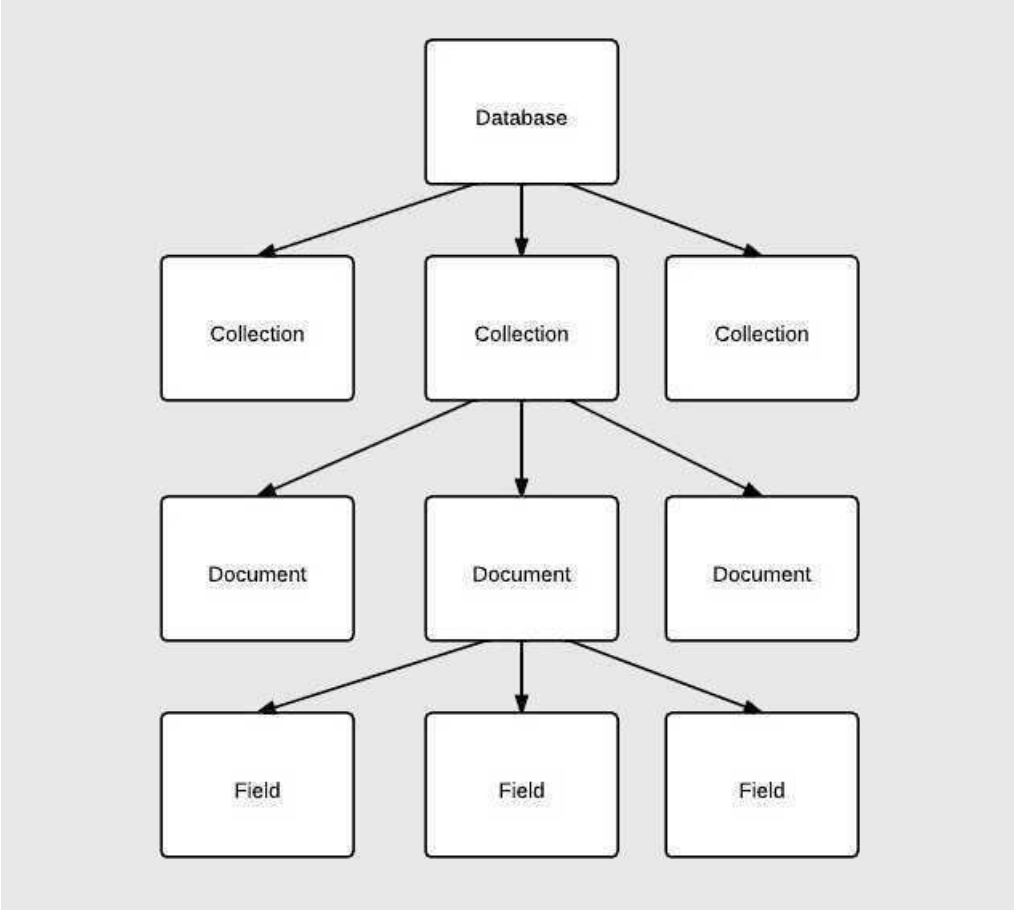
MongoDB Core Concepts

- MongoDB has the same concept of a "*database*" with which you are likely already familiar (or a schema for you Oracle folks). Within a MongoDB instance you can have zero or more databases, each acting as high-level containers for everything else.
- A database can have zero or more "*collections*". A collection shares enough in common with a traditional "table" that you can safely think of the two as the same thing.
- Collections are made up of zero or more "*documents*". Again, a document can safely be thought of as a "row".

[1] http://openmymind.net/2011/8/15/How-You-Should-Go-About-Learning-NoSQL/
[2] https://code.google.com/p/go-wiki/wiki/ProviderIntegration
[3] http://www.mongohq.com/heroku

- A document is made up of one or more "*fields*", which you can probably guess are a lot like "columns".
- "*Indexes*" in MongoDB function much like their RDBMS counterparts.
- "*Cursors*" are different than the other five concepts but they are important enough. The important thing to understand about cursors is that when you ask MongoDB for data, it returns a cursor, which we can do things to, such as counting or skipping ahead, without actually pulling down data.

To recap, MongoDB is made up of databases which contain collections. A collection is made up of documents. Each document is made up of fields. Collections can be indexed, which improves lookup and sorting performance. Finally, when we get data from MongoDB we do so through a cursor whose actual execution is delayed until necessary.



MongoDB

Note: The core difference between relational databases and document-oriented databases comes from the fact that relational databases define columns at the table level whereas a document-oriented database defines its fields at the document level.

MongoLab - The Fully-managed MongoDB-as-a-Service

MongoLab[4] is a platform for MongoDB hosting on the web.

Sign Up

Sign up[5] for a free account. When you fill up the online form, remember what you enter for default username and password - we will need this information later on. Next, click on "Plans & Features" and then select the free Sandbox plan[6].

Create a database

Next, create a free database. I have created a database named `godata` in my account. To connect using a driver via the standard URI:

```
1 mongodb://<dbuser>:<dbpassword>@ds031271.mongolab.com:31271/godata
```

Database User

A database user is required to connect to the above database. Click here[7] to create a new one. I entered "IndianGuru" for a Database username (you can enter whatever you want) and the requisite password.

mgo

Of all the Go drivers[8] available for MongoDB[9], mgo[10] is the most advanced and well-maintained.

To install `mgo`, in a command window type:

```
1 go get gopkg.in/mgo.v2
```

JSON Recap

JSON stands for JavaScript Object Notation.

JSON is syntax for storing and exchanging text information, much like XML. JSON is smaller than XML, and faster and easier to parse. JSON is language independent. Here's an example:

---

[4] https://mongolab.com/
[5] https://mongolab.com/signup/
[6] https://mongolab.com/plans/
[7] https://mongolab.com/databases/godata#users
[8] http://docs.mongodb.org/ecosystem/drivers/
[9] http://www.mongodb.org/
[10] http://labix.org/mgo

```
{
"employees": [
{ "firstName":"John" , "lastName":"Doe" },
{ "firstName":"Anna" , "lastName":"Smith" },
{ "firstName":"Peter" , "lastName":"Jones" }
]
}
```

The employee object is an array of 3 employee records (objects).

JSON data is written as name/value pairs. A name/value pair consists of a field name (in double quotes), followed by a colon and followed by a value:

```
"firstName" : "Satish"
```

JSON values can be:

- A number (integer or floating point)
- A string (in double quotes)
- A Boolean (true or false)
- An array (in square brackets)
- An object (in curly brackets)
- null

JSON objects are written inside curly brackets, Objects can contain multiple name/values pairs:

```
{ "firstName":"Satish" , "lastName":"Talim" }
```

package mgo

Usage: `import "gopkg.in/mgo.v2"`

The mgo[11] project (pronounced as “mango”) is a rich MongoDB driver for the Go language.

Usage of the driver revolves around the concept of sessions. To get started, obtain a session using the Dial[12] function:

```
session, err := mgo.Dial(url)
```

This will establish one or more connections with the cluster of servers defined by the url parameter. From then on, the cluster may be queried and documents retrieved with statements such as:

[11] http://gopkg.in/mgo.v2
[12] http://godoc.org/labix.org/v2/mgo#Dial

```
c := session.DB(database).C(collection)
err := c.Find(query).One(&result)
```

Once the session is not useful anymore, `Close` must be called to release the resources appropriately.

Function Close

```
func (s *Session) Close()
```

Close[13] terminates the session. It's a runtime error to use a session after it has been closed.

Function SetSafe

```
func (s *Session) SetSafe(safe *Safe)
```

SetSafe[14] changes the session safety mode. If the safe parameter is `nil`, the session is put in unsafe mode, and writes become fire-and-forget, without error checking. The unsafe mode is faster since operations won't hold on waiting for a confirmation.

The following statement will make the session check for errors, without imposing further constraints:

```
session.SetSafe(&mgo.Safe{})
```

Function Safe

```
func (s *Session) Safe() (safe *Safe)
```

Safe[15] returns the current safety mode for the session.

Function DB

```
func (s *Session) DB(name string) *Database
```

DB[16] returns a value representing the named database (in our case `godata`). If name is empty, the database name provided in the dialed URL is used instead. If that is also empty, "test" is used as a fallback.

Function C

```
func (db *Database) C(name string) *Collection
```

C[17] returns a value representing the named collection (in our case `user`).

[13] http://godoc.org/labix.org/v2/mgo#Session.Close
[14] http://godoc.org/labix.org/v2/mgo#Session.SetSafe
[15] http://godoc.org/labix.org/v2/mgo#Session.Safe
[16] http://godoc.org/labix.org/v2/mgo#Session.DB
[17] http://godoc.org/labix.org/v2/mgo#Database.C

Function Insert

```
func (c *Collection) Insert(docs ...interface{}) error
```

Insert[18] inserts one or more documents in the respective collection.

Function Find

```
func (c *Collection) Find(query interface{}) *Query
```

Find[19] prepares a query using the provided document. The document may be a map or a struct value capable of being marshalled with `bson`. The map may be a generic one using interface{} for its key and/or values, such as `bson.M`, or it may be a properly typed map. Providing `nil` as the document is equivalent to providing an empty document such as `bson.M{}`.

Further details of the query may be tweaked using the resulting Query value, and then executed to retrieve results using methods such as `One`, `For`, `Iter`, or `Tail`.

Function One

```
func (q *Query) One(result interface{}) (err error)
```

One[20] executes the query and unmarshals the first obtained document into the result argument. The result must be a struct or map value capable of being unmarshalled into by gobson. This function blocks until either a result is available or an error happens.

package bson

Usage: `import "labix.org/v2/mgo/bson"`

Package bson[21] is an implementation of the BSON specification for Go.

type M

```
type M map[string]interface{}
```

M[22] is a convenient alias for a `map[string]interface{}` map, useful for dealing with BSON in a native way. For instance:

```
bson.M{"a": 1, "b": true}
```

Program mongohqconnect.go

The program `mongohqconnect.go` connects to our database `godata` hosted on MongoLab and creates a collection `user`.

[18] http://godoc.org/labix.org/v2/mgo#Collection.Insert
[19] http://godoc.org/labix.org/v2/mgo#Collection.Find
[20] http://godoc.org/labix.org/v2/mgo#Query.One
[21] http://godoc.org/labix.org/v2/mgo/bson
[22] http://godoc.org/labix.org/v2/mgo/bson#M

Program mongohqconnect.go

```
1  package main
2
3  import (
4          "fmt"
5          "gopkg.in/mgo.v2"
6          "gopkg.in/mgo.v2/bson"
7          "log"
8          "os"
9  )
10
11 type Person struct {
12         Name string
13         Email string
14 }
15
16 func main() {
17         // Do the following:
18         // In a command window:
19         // set MONGOLAB_URL=mongodb://IndianGuru:password@ds031271.mongolab.com:31271/goda\
20 ta
21         // IndianGuru is my username, replace the same with yours. Type in your password.
22         uri := os.Getenv("MONGOLAB_URL")
23         if uri == "" {
24                 fmt.Println("no connection string provided")
25                 os.Exit(1)
26         }
27
28         sess, err := mgo.Dial(uri)
29         if err != nil {
30                 fmt.Printf("Can't connect to mongo, go error %v\n", err)
31                 os.Exit(1)
32         }
33         defer sess.Close()
34
35         sess.SetSafe(&mgo.Safe{})
36
37         collection := sess.DB("godata").C("user")
38
39         err = collection.Insert(&Person{"Stefan Klaste", "klaste@posteo.de"},
40                                 &Person{"Nishant Modak", "modak.nishant@gmail.com"},
41                                 &Person{"Prathamesh Sonpatki", "csonpatki@gmail.com"},
42                                 &Person{"murtuza kutub", "murtuzafirst@gmail.com"},
43                                 &Person{"aniket joshi", "joshianiket22@gmail.com"},
44                                 &Person{"Michael de Silva", "michael@mwdesilva.com"},
```

```
45                                  &Person{"Alejandro Cespedes Vicente", "cesal_vizar@hotmail.com"})
46          if err != nil {
47                  log.Fatal("Problem inserting data: ", err)
48                  return
49          }
50
51          result := Person{}
52          err = collection.Find(bson.M{"name": "Prathamesh Sonpatki"}).One(&result)
53          if err != nil {
54                  log.Fatal("Error finding record: ", err)
55                  return
56          }
57
58          fmt.Println("Email Id:", result.Email)
59  }
```

Deploying gomongohq.go on Heroku

In a previous free eBook[23], we have seen how to deploy web apps to Heroku.

The program `gomongohq.go` will reside on Heroku and will connect to our database `godata` hosted on MongoLab. The app will fetch information (the email id of say user Stefan Klaste) from the collection `user`.

Program gomongohq.go

```
1  package main
2
3  import (
4          "fmt"
5          "html/template"
6          "labix.org/v2/mgo"
7          "labix.org/v2/mgo/bson"
8          "log"
9          "net/http"
10         "os"
11 )
12
13 type Person struct {
14         Name string
15         Email string
16 }
17
18 func main() {
19         http.HandleFunc("/", root)
20         http.HandleFunc("/display", display)
21         fmt.Println("listening...")
22         err := http.ListenAndServe(GetPort(), nil)
23         if err != nil {
24                 log.Fatal("ListenAndServe: ", err)
25                 return
26         }
27 }
28
29 func root(w http.ResponseWriter, r *http.Request) {
30         fmt.Fprint(w, rootForm)
31 }
32
33 const rootForm = `
34   <!DOCTYPE html>
35     <html>
36       <head>
```

[23] https://leanpub.com/howtodeployagowebapptoheroku101/read

```
        <meta charset="utf-8">
        <title>Your details</title>
        <link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.4.2/pure-min.css">
      </head>
      <body style="margin: 20px;">
        <h2>A Fun Go App on Heroku to access MongoDB on MongoLab</h2>
        <p>This simple app will fetch the email id of a person, if it's already there in t\
he MongoDB database.</p>
        <p>Please enter a name (example: Stefan Klaste)</p>
        <form action="/display" method="post" accept-charset="utf-8" class="pure-form">
          <input type="text" name="name" placeholder="name" />
          <input type="submit" value=".. and query database!" class="pure-button pure-butt\
on-primary"/>
        </form>
        <div>
          <p><b>&copy; 2015 Go Challenge. All rights reserved.</b></p>
        </div>
      </body>
    </html>
`

var displayTemplate = template.Must(template.New("display").Parse(displayTemplateHTML))

func display(w http.ResponseWriter, r *http.Request) {
        // In the open command window set the following for Heroku:
        // heroku config:set MONGOHQ_URL=mongodb://IndianGuru:password@troup.mongohq.com:1\
0080/godata
        uri := os.Getenv("MONGOHQ_URL")
        if uri == "" {
                fmt.Println("no connection string provided")
                os.Exit(1)
        }

        sess, err := mgo.Dial(uri)
        if err != nil {
                fmt.Printf("Can't connect to mongo, go error %v\n", err)
                os.Exit(1)
        }
        defer sess.Close()

        sess.SetSafe(&mgo.Safe{})

        collection := sess.DB("godata").C("user")

        result := Person{}
```

```
82
83          collection.Find(bson.M{"name": r.FormValue("name")}).One(&result)
84
85          if result.Email != "" {
86                  errn := displayTemplate.Execute(w, "The email id you wanted is: " + result\
87  .Email)
88                  if errn != nil {
89                          http.Error(w, errn.Error(), http.StatusInternalServerError)
90                  }
91          } else {
92                  displayTemplate.Execute(w, "Sorry... The email id you wanted does not exis\
93  t.")
94          }
95  }
96
97  const displayTemplateHTML = `
98  <!DOCTYPE html>
99    <html>
100     <head>
101       <meta charset="utf-8">
102       <title>Results</title>
103       <link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.4.2/pure-min.css">
104     </head>
105     <body>
106       <h2>A Fun Go App on Heroku to access MongoDB on MongoLab</h2>
107       <p><b>{{html .}}</b></p>
108       <p><a href="/">Start again!</a></p>
109       <div>
110         <p><b>&copy; 2015 Go Challenge. All rights reserved.</b></p>
111       </div>
112     </body>
113   </html>
114 `
115
116 // Get the Port from the environment so we can run on Heroku
117 func GetPort() string {
118         var port = os.Getenv("PORT")
119         // Set a default port if there is nothing in the environment
120         if port == "" {
121                 port = "4747"
122                 fmt.Println("INFO: No PORT environment variable detected, defaulting to " + port)
123         }
124         return ":" + port
125 }
```

The program by now should be self explanatory.

- The `rootForm` uses Pure[24] a set of small, responsive CSS modules that you can use in every web project.
- The function `display` uses the `html/template` package and the `mgo` driver to access the database on MongoLab. If the name is found in the database the function throws a page to the user with the email id for that name.

You can visit this app here[25].

[24] http://purecss.io/
[25] https://lit-tor-9497.herokuapp.com/