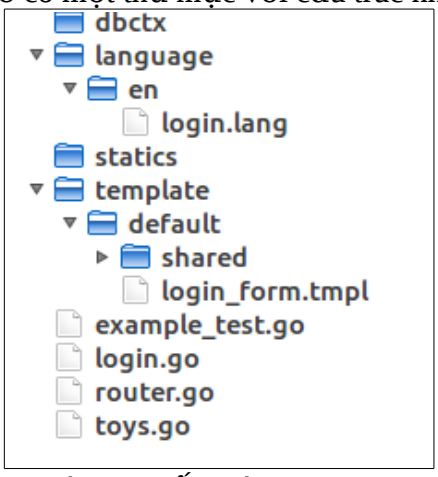
Cấu trúc một ứng dụng.

Sau đây là hướng dẫn tạo một project với toys framework.

Cho tới lúc này toys chỉ là một framework đơn giản với mô hình thiết kế ứng dụng MVC với một số ít ỏi các thư viện hổ trợ việc quản lý giao diện, đăng nhập và đa ngôn ngữ.
Các thư viện của toys sử dụng MongoDB với mgo driver để quả lý dữ liệu.

Hãy bắt đầu với việc download một project mẫu tại:
http://open-vn.org/toys/docs/tut/toysign/toysign.zip
Giải nén tập tin vừa tải về, bạn sẽ có một thư mực với cấu trúc như sau:



Hình 1. Cấu trúc thư mục.

Thự mục **dbctx**: là một sub-package đảm trách phần Model trong ứng dụng. Hiện tại chúng ta chưa dùng tới nó nhiều.
Thư mục **language**: chứa hệ thống hỗ trợ đa ngôn ngữ.
Thư mục **template**: chứ hệ thống giúp tối giản việc quản lý giao diện.
Tập tin **toys.go**: là tập tin mẫu bao gồm một số code được toys tự sinh ra. Bạn không cần quan tâm lắm về file này ở mức độ của bài viết này.
Tập tin **example_test.go**: nơi đây bạn khởi tạo một số thứ như gán giá trị đường dẫn tới các thư mục, thông tin đăng nhập vào MongoDB, bạn có thể dùng lệnh "go test" để chạy thử ứng dụng.
Tập tin **router.go**: trong tập tin này bạn ấn định hàm quản lý nào xử lý hành động nào trong hứng dụng.
Tập tin **login.go**: đây là một trong những tập tin do bạn tự tạo, trong đây khai báo những hàm quản lý (Action Controller).

Tiếp tục vào sâu trong nọi dung từng tập tin và thư mục.

Bắt đầu tới file example_test.go:

Nếu bạn cài đặt MongoDB tai localhost và không hề sửa chữa gì cấu trúc thư mục sau khi tải về bạn không cần phải chỉnh sửa bất cứ gì trong file này.
Trong thực tế ứng dụng này là một package (thư viện) không phải một command (chương trình thực thi) bạn có thể sửa nội dung file test này để tạo một ứng dụng bằng cách sau:
1. Tạo 1 file tên main.go, copy toàn bộ nội dung từ file
2. Đổi package name từ "toysign_test" sang "main".
3. Đổi tên function "ExampleHandler()" thành "main()"

4. Chạy lệnh: go run /path/to/your/main.go

Sau đây là một số gía trị cấu hình.

```
14          var (
15                  // cnnStr the connection string to MongoDB
16                  cnnStr = "localhost"
17                  // langRoot the path to language folder in file system
18                  langRoot = "language"
19                  // langDefaultSet the default language set
20                  langDefaultSet = "en"
21                  // tmplDefaultSet the path to template folder in files system
22                  tmplRoot = "template"
23                  // tmplDefaultSet the default template set
24                  tmplDefaultSet = "default"
25                  // rsrcRoot the path to static folder in file system
26                  rsrcRoot = "statics"
27                  // rsrcPrefix the URL path for static file server
28                  rsrcPrefix = "/statics/"
29                  //toysignPath the URL path for toysign
30                  toysignPath = "/user/"
31          )
```

router.go:
Đây là một tập tin tương đối ngắn, nội dung của cả tập tin như sau:

```
1   package toysign
2
3   func (h *handler) initSubRoutes() {
4           h._subRoutes = []route{
5                   route{"login", Login},
6                   route{"login2", Login2},
7           }
8   }
```

Dòng 5,6 là tất cả những gì bạn cần quan tâm.
route{“login”, Login} có nghĩa là nếu đường dẫn hiện tại của trang web trùng với ***toysignpath + “login”*** thì yêu cầu nãy sẽ được quản lý bởi hàm ***Login*** (hàm này được định nghĩa tại login.go). Một ví dụ dễ hiểu, nếu tên miền của bạn đang dùng là *example.com* và toysignpath có gí trị là *“/user/”* thì hàm Login sẽ quản lý việc tính toán, xuất mả cho yêu cầu tới từ *example.com/user/login*
Ngoài ra còn có môt số quy tắc đơn giản trong việc định tuyến này. Ví dụ, với:
route{“mem[0-9][0-9][0-9]”, Member}
Thì hàm Member sẽ được gọi khi yêu cầu tới từ *example.com/user/member123* cũng như *example.com/user/mem456* nhưng không phải là *example.com/user/mem23a*.
Việc so sánh này không dự trên thư viện regexp mà dựa trẹn quy tắc của hàm path.Math (xem thêm tại đây: http://golang.org/pkg/path/#Match)

login.go:
Đây là một file chứa các Action Controller. Hiện tại các Action vẫn chưa được xử lý gì nhiều cho nên nội dung của file rất ngắn.
Mỗi một Action Controller trong chương trình phải nhậm vào một tham số có kiểu *Controller.
Trong một file chỉ nên có những Action Controller liên quan mật thiết đến nhau.

Xem xét ví dụ về các Controller quản lý việc đăng nhập:

```
1  package toysign
2
3  func Login(c *controller) {
4          c.View("login_form.tmpl", nil)
5  }
6
7  func Login2(c *controller) {
8          c.Print("login2")
9  }
```

Code chưa được thực thi một các thật sự, nhưng ý đồ của mình là khi người dùng truy nhập vào trang example.com/user/login nếu chưa login thì sẽ hiện ra 1 form đăng nhập, khi nhấn nút đăng nhập thì sẽ được chuyển tới trang example/com/user/login2 (do hàm Login2 xử lý).
Controller là một kiểu "kế thừa từ" toys.Controller. Nó có một số hàm tiện lợi kế thừa từ http://godoc.org/github.com/openvn/toys#Controller và 2 hàm nữa là:
(*controller).View(page string, data interface{}) error
hàm này sẽ kết xuất mã từ "page" mà nó nhận vào kèm theo data (nếu có). Hiện tại ta không dùng tới bất kỳ dữ liệu gì cho việc này. Một error sẽ được trả về nếu trong quá trình xuất mã gặp trục trặc.
(*controller).ViewData(title string) map[string]interface{}
Hàm này đơn giản trả về 1 map[string]interface{} có kèm theo trường Title. Tương tự như việc chúng ta viết code như thế này:
data := map[string]interface{} {"Title": title}
Ngoài ra chúng ta còn có thể truy xuất (*controller).auth là đối tượng quảng lý việc chứng thực người dùng (http://godoc.org/github.com/openvn/toys/secure/membership), hoặc (*controller).sess đối tượng quán lý session (tương tự $_SESSION trong PHP)
http://godoc.org/github.com/openvn/toys/secure/membership/session)

Đây sẽ là luc thích hợp để nói về hệ thống template.

thư mục template:
Xét trong thư mục template có 1 folder con tên là **default**. default gọi là một template set, trong một ứng dụng sự dụng hệ thống giao diện phải có ít nhất set default này, còn lại chúng ta có thể có thêm các set như newyear, noel... để thay đổi dễ dàng cho từng mùa.
Quay trở lại file example_test.go, chúng ta hãy xem cách khởi tạo một gói giao diện:

```
40         //multi language support
41         lang := locale.NewLang(langRoot)
42         if err := lang.Parse(langDefaultSet); err != nil {
43                 fmt.Println(err.Error())
44         }
45
46         //template for cms
47         tmpl := view.NewView(tmplRoot)
48         tmpl.SetLang(lang)
49         tmpl.HandleResource(rsrcPrefix, rsrcRoot)
50         if err := tmpl.Parse(tmplDefaultSet); err != nil {
51                 fmt.Println(err.Error())
```

```
52              }
```
Gói giao diện hỗ trợ sự dụng hệ thống đa ngôn ngữ, điều này không bắt buộc nhưng chúng tôi khuyên bạn sử dụng chúng.
Việc cấu hình hỗ trợ đa ngôn ngữ như thế nào chúng ta sẽ bàn sau. Ở đây quay lại với hệ thống template.
Chúng ta bắt đầu khởi tạo hệ thống quảng lí template tại dòng 47 và sau đó hoàn tất mọi việc tại dòng 50 khi chúng ta gọi hàm (*View).Parse(set string).
Để sử dụng 1 set, trước tiên chúng ta cần Parse chúng. Set được Parse sau cùng sẽ là set mặt định.
Nếu muốn chuyển đổi một set ta gọi (*View).SetDefault(set string). Tr6en lý thuyết chúng ta có thể sự dụng thư viện này để xây dựng 1 ứng dụng có thể thay đổi giao diện mà không cần khởi động lại.
hãy xem cấu trúc của một set gồm những gì.
Mỗi set bắt buộc phải có 1 thư mục tên là **shared** và những file tmpl do bạn tự tạo (tương tự như login_form.tmpl)-gọi là Action View, trong thư mục shared bắt buộc phải có 1 file tên là **layout.tmpl**.
Tất cả những file giao diện trong ứng dụng phải có tên đuôi kết thúc bằng .tmpl.

Nội dung file layout.tmpl:
```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>{{if .Title}}{{.Title}}{{end}}</title>
  <meta name="description" content="toys template project - demon layout.tmpl">
  <meta name="author" content="nguyen@open-vn.org">
  <!--[if lt IE 9]>
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
</head>
<body>
          {{template "page" .}}
</body>
</html>
```

Đúng như tên gọi, nó đơn thuần là một file HTML quảng lí giao diện. Nó đơn thuần là một file HTML. Nội dung của nó chỉ đặc biệt ở dòng 5 và dòng 13.
Tại dòng 5 {{if .Title}}{{.Title}}{{end}}: đoạn này có nghĩa là nếu trong dữ liệu truyền vào có trường Title thì in nội dung trường Title ra.
Tại dòng {{template "page" .}}: đây là nơi bạn quyết định những Action View của mình được hiện chèn vào.
Những đoạn code {{…}} là những đoạn lệnh thực thi, để hiểu thêm có những đạon lệnh nào và cách dùng của chúng, hãy xem tài liệu tại: http://golang.org/pkg/text/template/#pkg-overview

Hãy xem xét một Action View bao gồm những gì.
login_form.tmpl:
```
{{define "page"}}
Login {{lang "login.lang" "hello"}}
{{end}}
```

Đây là 1 Action View đơn giản, nó có sử dụng tới hệ thống đa ngôn ngữ (sẽ bàn tới ngay sau đây)

trước khi kết thúc phần này chúng ta hãy xem qua mã HTML được kết xuất khi một người dùng lướt tới trang example.com/user/login.
Nên nhớ chúng ta không hề truyền bất kỳ dữ liệu nào vào Action View này, nó hầu như là một trang tỉnh ngoại trừ phần đa ngôn ngữ.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title></title>
  <meta name="description" content="toys template project - demon layout.tmpl">
  <meta name="author" content="nguyen@open-vn.org">
</head>
<body>

Login Hello!

</body>
</html>
```

Xet thư mục **language** có chứa một thư mục con tên là **en**, en là một Language Set. Tương tự như template set, bạn cần phải Parse một ngôn ngữ trước khi dử dụng chúng (example_test.go dòng 42). Set cuối cùng được Parse sẽ là set mặt định.
Trong mỗi language set là một tập hợp các file .lang, nội dung của file là những dòng theo cấu trúc:

```
key1=content1
key2=something else
….............................
keyn=contentN
```

Hãy xem nội dung thự tế của file login.lang (chỉ bao gồm 1 dòng)

```
hello=Hello!
```

Bên trong login_form.tmpl tôi đã gọi {{lang "login.lang" "hello"}} và kết quả là nội dung “Hello!” được thay thế tại ví trí đó.