Easy REST with AngularJS and Go

V. Glenn Tarcea

November 2nd, 2014

Outline

Speaker

- Glenn Tarcea
- Senior Developer at University of Michigan
- Current Project: Materials Commons

What this talk is about

What this talk doesn't cover

- How to setup angularjs
    - A
    - B
    - C

AngularJS Setup - Imports

- We are going to import the required modules
  - AngularJS router doesn't allow sub-views so we'll use ui-router
  - Restangular provides a nice REST interface
- We don't technically need the extensions but they will make our lives easier

```
<script src=".../angularjs/1.3.1/angular.min.js">
</script>
<script src=".../angular-ui-router.min.0.2.11.js">
</script>
<script src=".../restangular.min.js">
</script>
```

AngularJS Setup - Setup our app

- To turn your app into an AngularJS app you need to add ng-app.
- Here we set up a name of our name. We'll see more about this.

```
<html ng-app="myapp" lang="en">
  <head>...</head>
  <body>
```

AngularJS Setup - View

- ui-view is where we'll load page content.
- ui-router allows sub views. Basically we can have a tree of views and states.

```
    <div class="main-content">
      <!-- Setup location for our main view -->
      <div ui-view>
      </div>
    </div>
  </body>
</html>
```

AngularJS Setup - Putting it all together

- So here is what our index.html html looks like

```
<html ng-app="myapp" lang="en">
  <head>...</head>
  <body>
    <div class="main-content">
      <div ui-view>
      </div>
    </div>
    <script>...</script>
  </body>
</html>
```

- To configure our App we need to set up our routes and module references.
  - Routes control which pages to display
  - Module references give us an easy way to reference the different pieces of our project
    - Controllers
    - Filters
    - Services
    - Directives

Module References

- Set references to our app modules.
  - We break our app into different modules for the models in AngularJS.

```
var App = App || {};
App.Services = angular.module('app.services', []);
App.Controllers = angular.module('app.cntrlrs', []);
App.Filters = angular.module('app.filters', []);
App.Directives = angular.module('app.directives', []);
var app = angular.module('myapp', [
    "ui.router", "restangular",
    "app.services", "app.cntrlrs", "app.filters",
    "app.directives"
]);
```

Interlude: Dependency Injection

- AngularJS makes extensive use of dependency injection
- It does inject based on the name
    - This doesn't work when minimizing your code
- You have 2 options when you want to minimize
    - You can use a plugin that will rewrite your code
    - Or you can write your code so it can be minimized
        - I use this option throughout the example code

```
App.Controllers.controller("name-of-controller",
                          ["dependency1Name", "...",
                           controllerFunction]);
function controllerFunction (dependency1Name) {
    // ...
}
```

Configure our Routes

- We set up 2 routes and a default route

```
app.config(["$stateProvider", "$urlRouterProvider", ap
function appConfig($stateProvider, $urlRouterProvider)
    $stateProvider
        .state("users", {
            url: "/users",
            templateUrl: "app/users.html",
            controller: "usersController"
        })
        .state("users.add", {
            url: "/add",
            templateUrl: "app/add.html",
            controller: "addUserController"
        });
    $urlRouterProvider.otherwise("/users");
}
```

- Now well configure a Go server
- We'll use this server for our REST services and to serve our web pages
  - Go has an HTTP interface that makes writing web servers and services very easy
    - This is one of the nicest pieces of using Go

Go Web Server Setup

- We'll point our web server at our apps directory
- This will be our default route
  - The server will automatically pick up the index.html file

```
webdir := ...
dir := http.Dir(webdir)
http.Handle("/", http.FileServer(dir))
addr := "localhost:8081"
fmt.Println(http.ListenAndServe(addr, nil))
```

REST Setup

- We'll use a nice REST extension package: go-restful
- Because this package uses HTTP interfaces we can use standard Go http to setup

```
container := ...

// All REST calls come through a /api/... route.
// We strip off /api before sending on to our
// container this way the container doesn't
// care about the prefix.
http.Handle("/api/", http.StripPrefix("/api",
        container))
```

```
ws := new(restful.WebService)
ws.Path("/users").
        Consumes(restful.MIME_JSON).
        Produces(restful.MIME_JSON)

ws.Route(ws.GET("").To(rest.RouteHandler(r.getAllUsers
        Doc("Retrieves all users").
        Writes([]schema.User{}))
```

Service Implementation

```
func (r *usersResource) createUser(request *restful.Re
        response *restful.Response, user schema.User)

        var req userReq
        if err := request.ReadEntity(&req); err != nil
                return err, nil
        }
        u, err := r.users.CreateUser(req.Email, req.Fu
        return err, u
}
```