AngularJS and Go

V. Glenn Tarcea

November 15th, 2014

Outline

About Me

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

Materials Commons

- Materials Commons is an online collaborative space for Metals Researchers
- We have open sourced all the code for Materials Commons:
  - Go, Javascript, Java, Python, Erlang, C
- You can find our code at:
  - `https://github.com/materials-commons`
  - `https://github.com/prisms-center/materialscommons.org`
- There are alot of nice (if sometimes a bit rough) packages:
  - Erlang: gen stomp, resource discovery, process monitoring, OS interfaces
  - Go: Utilities, config, file transfer, FlowJS server
  - Javascript: AngularStomp
  - Java: DM3 Parser for Tika (not touched in a while)

What this talk is about

- This talk will cover creating a website using
    - Go and AngularJS
    - Websockets
    - REST
    - JWT
- The site will allow for simple "collaboration"
    - By using broadcasts to keep each site in sync

What this talk doesn't cover

- This talk is not a Go or AngularJS tutorial
    - We will go over some aspects of both but will not spend a lot of time on the basics
- It won't cover all aspects of the application
    - We will elide some details but you can refer to the sample app to get all the details

Where to get the app

- I've set up a Github repo that contains the working app
    - `https://github.com/gtarcea/1DevDayTalk2014`
- The README.org goes over getting it running
    - In a nutshell:
        - Install go
        - Install godep (go get github.com/tools/godep)
        - make run
- The intent of this app is to give you a nice starting point
    - It gives you a working JWT, Websocket, REST based application
    - With client side authentication
    - Reconnect
    - Broadcast to keep all connected clients updated
- It looks simple but there is a lot going on

Overview

- We'll cover the basics of setting up an angular app and configuring the needed packages
- We use a few client libraries to make our lives easier
    - ui-router to give us multiple state based routes
    - ng-websocket for websocket communication
    - angular-jwt for easy JWT integration
    - Restangular for REST communication
- We will cover configuring and integrating these packages

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>
```

Overview

- 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"
]);
```

Configure our Routes

- We set up 2 routes and a default route

```
app.config(["$stateProvider", "$urlRouterProvider",
            appConfig]);
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");
}
```

Configure Authentication

- To configure authentication we need to
  - Control access to protected areas of our app
  - Track user authentication
  - Setup JWT Headers for all REST calls

Controlling Access

```
app.run(["$rootScope", "User", "$state", appRun]);
function appRun($rootScope, User, $state) {
    // $stateChangeStart is fired when a route change
    // is starting. Here we check if the user is already
    // authenticatd. If they aren't then we redirect
    // them to the login page.
    $rootScope.$on('$stateChangeStart', stateChange);

    function stateChange(event, toState, toParams) {
        if (!User.isAuthenticated()) {
            if (toState.url !== "/login") {
                // Cancel whatever route we were going
                // to and instead go to the login page.
                event.preventDefault();
                $state.go("login");
            }
        }
    }
}
```

Configuring JWT

- The following code is also in appConfig (where we also configured the routes)

```
// The JWT token is stored in sessionStorage. When our
// app starts up we explicitly clear the previous token.
sessionStorage.setItem("token", null);

// This interceptor will set the Authorization field
// in the header with the JWT token.
jwtInterceptorProvider.tokenGetter = function() {
    var token = sessionStorage.getItem("token");
    return token ? token : "";
};
$httpProvider.interceptors.push("jwtInterceptor");
```

Configure Websockets

Overview

Overview

Overview

- 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))
```

Overview

```
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
}
```