

ANGULARJS AND GO

V. GLENN TARCEA

GTARCEA@UMICH.EDU

NOVEMBER 15TH, 2014

TABLE OF CONTENTS

- Introduction
- AngularJS Setup
- REST using Restangular
- Services Overview
- Go Routes Setup
- Go REST Service
- Go Websockets
- Conclusion

1 INTRODUCTION

1.1 ABOUT ME

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

1.2 MATERIALS COMMONS

- Materials Commons is an online collaborative space for Metals Researchers
- All the code for Materials Commons is open sourced:
 - Go, Javascript, Java, Python, Erlang, C
- You can find the 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)

1.3 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 browser in sync

1.4 WHAT THIS TALK DOESN'T COVER

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

1.5 WHERE TO GET THE APP

1.6 DEMO

- Login/Logout
- Reconnect/Disconnect
- Multiple browsers staying in sync

2 ANGULARJS SETUP

2.1 OVERVIEW

- Cover the basics of setup and configuration for AngularJS
- Uses the following external packages
 - ui-router to give us multiple state based routes
 - ng-websocket for websocket communication
 - angular-jwt for easy JWT integration
 - Restangular for REST communication

2.2 MODULE REFERENCES

- Set references to app modules

```
var App = App || {};  
  
App.Constants = angular.module('app.constants', []);  
App.Services = angular.module('app.services', []);  
App.Controllers = angular.module('app.controllers', []);  
App.Filters = angular.module('app.filters', []);  
App.Directives = angular.module('app.directives', []);  
  
// Load our module dependencies.  
var app = angular.module('myapp', [  
    "ui.router", "restangular",  
    "ngWebsocket", "angular-jwt",  
    "app.constants", "app.services",  
    "app.controllers", "app.filters",  
    "app.directives"  
]);
```

2.3 CONFIGURE ROUTES

- Set up routes to pages and views in system

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

  // If the route isn't recognized goto /users
  $urlRouterProvider.otherwise("/users");
}
```

2.4 CONFIGURE AUTHENTICATION

- Required steps
 - Control access to protected areas of app
 - Track user authentication
 - Setup JWT Headers for all REST calls

2.5 CONTROLLING ACCESS

```
// appRun allows us to intercept different events while our
// application is running. Here it is used to control access
// to the application by requiring the user to login.
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 authenticated. If they
    // aren't then we redirect them to the login page.
    $rootScope.$on('$stateChangeStart', function(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");
            }
        }
    });
}
```


2.6 CONFIGURING JWT

- The following code is also in appConfig (where routes were configured)
- It configures \$http (and Restangular) to include the JWT token in all REST calls

```
// 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");
```


2.7 CONFIGURE WEBSOCKETS

- WebSockets uses events
- Would like to connect to WebSocket after authentication
 - Unfortunately the WebSocket spec doesn't support custom headers
 - Work around: pass the token as argument in the URL and in each message
 - For simplicity this wasn't implemented

```
// Connect the socket
$websocket.$new({
  url: ws.url(),
  reconnect: true,
  reconnectInterval: 500
});

// Wait on events
var s = ws.get();
s.$on("addeduser", function(user) {
  $timeout(function() {
    Users.add($scope.users, user);
  });
});
```

3 REST USING RECTANGULAR

3.1 OVERVIEW

- Restangular makes REST easy by
 - Providing Promises
 - Restangularizing your objects
 - Methods are attached to the returned object
 - You don't have to remember the route
 - Easy to use API

3.2 EXAMPLE

This example demonstrates retrieving and updating a user

```
var user = Restangular.all("users", 123).get();  
// change their name  
user.fullname = "New Name";  
user.put()
```

3.3 RECALL SENDING JWT

Recall that we configured the underlying \$http service to include Authorization in the header with the JWT Token. Just to review:

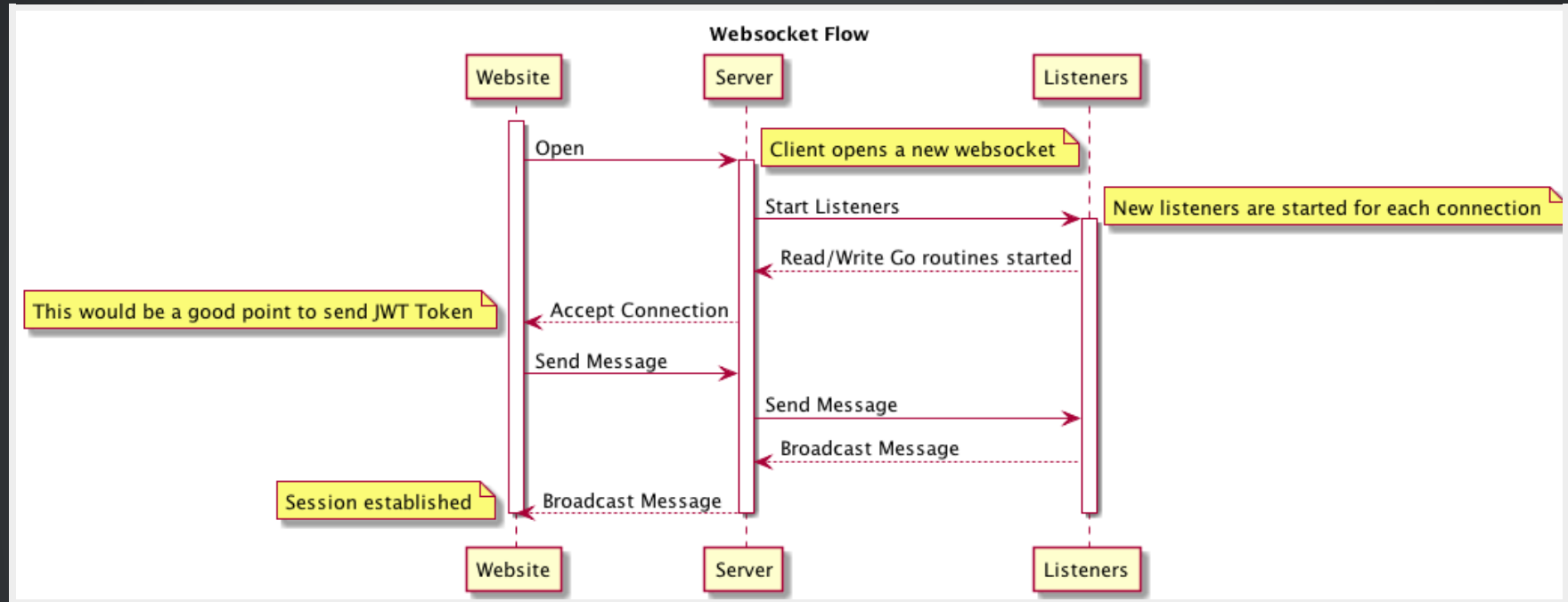
```
// The JWT token is stored in sessionStorage. When our
// app starts up we explicitly clear the previous token.
sessionStorage.setItem("token", null);

// 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");
```

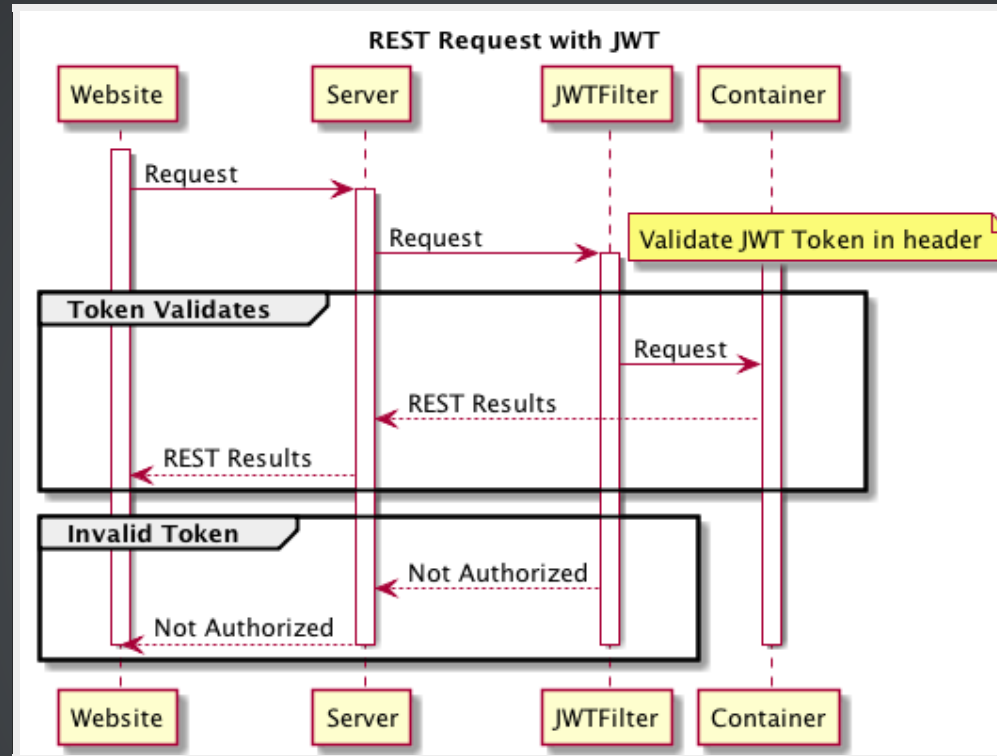
Now whenever we make a Restangular call the header is automatically included

4 SERVICES OVERVIEW

4.1 WEBSOCKETS



4.2 REST AND JWT AUTHENTICATION



5 GO ROUTES SETUP

5.1 OVERVIEW

- Configure Go HTTP server to handle:
 - Serve website content
 - REST Calls
 - Websocket connections and broadcast
- Go has an HTTP interface that makes developing servers/services easy

5.2 GO WEB SERVER SETUP

- Point web server at website directory
- Set as default route
 - Server will automatically pick up index.html

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

5.3 REST SETUP

- Use REST extension package: go-restful
 - <https://github.com/emicklei/go-restful>
- Common theme: Package support standard HTTP interface

```
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))
```

5.4 WEBSOCKET SETUP

- WebSocket is also handled through HTTP handler

```
s := events.NewServer(hub)
http.Handle("/ws", websocket.Handler(s.OnConnection))
```

6 GO REST SERVICE

6.1 OVERVIEW

- Configure our REST service to handle REST verbs
- Example shows configuring GET verb
- go-restful also supports SWAGGER
 - See:
 - Website: <http://swagger.io/>
 - Demo: <http://petstore.swagger.wordnik.com/>

```
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{}))
```


6.2 JWT TOKEN CREATION

- Tokens need a private and public key
 - Generate key files
 - Uses private key to create token

```
# These commands were run to create our public/private files
openssl genrsa -out app.rsa 1024
openssl rsa -in app.rsa -pubout > app.rsa.pub
```

```
// At this point we have read the public and private keys
// Create the JWT Token
token := jwt.New(jwt.GetSigningMethod("RS256"))
token.Claims["ID"] = req.Username
token.Claims["exp"] = time.Now().Add(time.Hour * 72).Unix()
tokenStr, err := token.SignedString(r.privateKey)
if err != nil {
    return err, nil
}

auth := schema.Auth{
    Username: req.Username,
    Token:    tokenStr,
}
```


6.3 JWT TOKEN VERIFICATION

- Use intercept filter to verify the token
- Abort request if token doesn't validate
- Handle login path as special case

```
// Setup the filter for the container
```

```
f := filters.NewJWTFilter(publicKey, "/users/login")
container := restful.NewContainer()
container.Filter(f.Filter)
```

```
// Verify the token on each rest call
```

```
func (f *jwtFilter) Filter(req *restful.Request, resp *restful.Response,
                           chain *restful.FilterChain) {
    // if the user is logging in for the first time then the
    // path will be f.loginPath. If that is the case then we just
    // go to the next filter because there is no token to
    // authenticate against.
    if req.Request.URL.Path != f.loginPath {
        token, err := jwt.ParseFromRequest(req.Request, f.getKey)
        if err != nil || !token.Valid {
            fmt.Printf("invalid token for url %s: %s\n ", req.Request.URL.Path, err)
            resp.WriteString(http.StatusUnauthorized, "Not authorized")
            return
        }
    }
    chain.ProcessFilter(req, resp)
}
```

6.4 SERVICE IMPLEMENTATION

```
// userReq is sent when a client creates a new user.
type userReq struct {
    Email    string `json:"email"`
    Fullname string `json:"fullname"`
}

func (r *usersResource) createUser(request *restful.Request,
    response *restful.Response, user schema.User) (error, interface{}) {

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

7 GO WEBSOCKETS

7.1 OVERVIEW

- WebSockets are long lived connections
 - Setup 2 go routines for reading/writing
 - For our purposes: register with broadcaster (EventHub)

```
// Recall setup
http.Handle("/ws", websocket.Handler(s.OnConnection))

// OnConnection is called when a new websocket connection is made.
// It creates a persistent client connection and registers that
// connection with the hub. It is meant to be called by the
// websocket.Handler method.
func (s *Server) OnConnection(ws *websocket.Conn) {
    defer func() {
        ws.Close()
    }()

    client := NewClient(ws, s.hub)
    s.hub.Register(client)
    client.Listen()
}
```

7.2 READ HANDLING

- The read handler waits in an event loop
- The write side is similar(ish)

```
// readListener processes messages on the websocket.
func (c *Client) readListener() {
    for {
        select {
        case <-c.done:
            c.hub.Unregister(c)
            c.done <- true
            return
        default:
            var msg Message
            err := websocket.JSON.Receive(c.ws, &msg)
            switch {
            case err == io.EOF:
                c.done <- true
                return
            case err != nil:
                c.done <- true
                return
            default:
            }
        }
    }
}
```

8 CONCLUSION

- AngularJS and Go work well together
- The large number of standard libraries for each means you can easily create complex applications
- I've seen a lot of questions on the web on:
 - AngularJS client side authentication
 - JWT with AngularJS (and Go)
 - How to use Websockets
- This talk and the example app at <https://github.com/gtarcea/1DevDayTalk2014> will help get you started
- If you have questions please contact me at gtarcea@umich.edu
 - Or send me a pull request with a fix :-)