**Details**

The rise of SPA frameworks is changing the way we architect our applications. The frontend and backend are seen as two independent components. In this talk we'll look at how .Net Core allows us to build powerful APIs on the backend, and have VueJS consume these API's on the frontend. Then, we'll go through the process of securing this API on both the frontend and backend as well as cover common security challenges like XSS, CSRF and ways to address them.  
  
Speaker  
  
Ado Kukic  
  
Ado is a full-stack developer, advocate, and evangelist at Auth0. Mixing his passion of programming and education, he creates tutorials, courses, and other educational content focusing on security, authentication, and much more. On the front-end, he prefers Angular, while on the backend he flip-flops between Node.js and Golang.

[chris@kinsman.net](mailto:chris@kinsman.net)

technical speaker workshop

[ted@neward.com](mailto:ted@neward.com)

Ado Kukic

@kukicado

Our Application

**GifBattle**

Using the Giffy API

3 components

* Creative Gif component
* Caption it
* Vote the funnier Gif

Disclaimer: list of gif is static.

**.NET Core API**

**The Brains**

services.AddCors(options =? {

options.AddPolict(“AllowSpecificOrigin”) …..

}

Random endpoint

Create endpoint

Versus endpoint

Vote endpoint

Leaderboard endpoint

Claims endpoint

**VueJS**

**The Interface**

Why VueJS? Just passed the number of stars React has.

React VueJS Angular

Library Framework

🡨------------------------------------------------------🡪

Angular has their own http client, but has nuances to learn.

Vue is so simple it doesn’t have a build.

Vue.js is Angular 1.x but better under the hood.

*v-if=”randomGif”*

* Show this UI if true

v-for=”(gif,index )in leaderboard”

<div id=”app”>

<!-- instantiate Vue -->

<script> const app = new Vue({

el: ‘#app’,

data ….

})

**Typical security model**

https://gifbattle.com

* Lives in the server
* User makes a request, server would fulfill the request or not (i.e. permission).
* If ok, render the page on server, then send to the user.
* Security is ok if all the UI, front-end and back-end layers is at the server

On the client,

* After authenticating a user, creates a session cookie id.

**Modern security model**

https://gifbattle.com

* React/Angular/VueJS is at https://app.gifbattle.com
* Why not have them talk in the same language all the time?
* Secure the transport layer only.

JSON Web Tokens

* JWT’s (RFC 7519) for creating access tokens that assert some number of claims. For example, a server could generate a token that has the claim "logged in as admin" and provide that to a client.
* Understanding token based.
* Driver’s license is like a JSON Web Token

Header

* Drivers License – New York State
* JSON Web Token (green part) - fnksdnfklsflsnlkdsfnksdfnsdlk.321ji21j3i12n30i21n3012.n2n321n30i210312 { “alg”: “HS256”, “typ”:”JWT”}

Payload

* Drivers License – Picture, Name, Address, Demographics, Restrictions
* JSON Web Token (blue part) – fnksdnfklsflsnlkdsfnksdfnsdlk.321ji21j3i12n30i21n3012.n2n321n30i210312

{“sub”:”123456890”, “given\_name”:”Thor”,”family\_name”: “Odinson”, “admin”: true}

Public claims

Private claims

* “admin”: true – only in back-end

Signature (of the token)

* Driver’s license: UV Light, Hologram
* JSON Web Token (black part) – fnksdnfklsflsnlkdsfnksdfnsdlk.321ji21j3i12n30i21n3012.n2n321n30i210312

HMACSHA256(header + “.” + payload, “lokisucks”)

*Validity (time) of token depends on implementation*

Token needs password to check if token is valid.

Password = authentication server sign in key.

How token-based authentication work in the real-world

* Person -> go to DMV -> employee would verify -> gets valid Driver’s license (or token).
* Person goes to the club -> can present the driver’s license to the security (validated by the machine, etc.).

3-pronged approach

1 – api.battle.com (server)

2 – app.gifbattle.com (React/Angular/VueJS)

3 – login.gifbattle.com (Auth0)

**Case 1**

Log in -> goes to Auth0 to authenticate -> authentication server (auth0) sends back a token) -> web application stores this token.

Then goes web token go to back-end, If not authenticated, you get status 401.

If match all 3 criteria (permission, authorization, etc.), access is granted, and given a JSON data.

Typically companies store these tokens in local storage / session storage.

* Susceptible to cross-script hacking through JavaScript.

**Case 2**

When user authenticates in the authentication server, we set a cookie session at app.gifbattle.com -> store the token in the webpage so that the token is destroyed upon refresh.

Cookie on the authorization server does nothing.

**Case 3**

Once you refreshed the page, create an iframe.

*Hey do you have a valid cookie session id? If yes, issue new token. (cookie persistent on the auth server not on the app)*

User doesn’t see this happen.

This maintains the user experience.

In code,

Add JwtBearerDefaults

Authorization server will send a public key.

Add a decorator in a route.