* Course Overview
* Intro
  + Json Web Tokens(JWT)
* Why Use an Auth Provider
  + Authentication: Who are you?
    - Login with email and password
  + Authorization: What are you allowed to do?
    - Check user rights
  + Auth0, Okta
    - OAuth provider
* OAuth 2.0
  + Auth0 is security provider
  + OAuth is security protocol
  + OAuth 2.0 is about authorization
  + Scope is permission
    - String they denotes type of access
    - Like read only, write, etc
  + OAuth
    - Authorization protocol
    - Authorize a user without their password
    - Get info from a third party
  + OAuth setup
    - Register with service(google, facebook, etc)
    - Give app name, website, callback url
    - Auth0 handles this for us
  + OAuth roles
    - Resource owner: user
    - Client: app that want to access user account
    - Auth Server: auth logic, provides access token
    - Resource Server: api application want to access, user data
* Choosing an OAuth Flow
  + Client ap sent auth request to user
  + User accepts request and sends back auth grant to app
  + App sends grant to auth server
  + Auth server sends app access token
  + App use access token to call api in resource server
  + Resource server returns data
  + Grant(aka flow)
    - A way to receive an access token
  + Implicit Flow
    - Your app directs the browser to the Auth0 sigin-in
    - Auth0 redirects your app, at the callback url you chose
    - Your app reads the tokens from the URL
  + OAuth is for authorization not authentication
  + Use OpenID Connect for authentication
* OpenID Connect
  + OAuth is for authorization
    - No standard for scopes and user info requests
  + Nice to avoid managing passwords ourselves
  + OpenID Connect
    - Authenticate users without managing passwords
    - Sits on top of OAuth2.0
    - OAuth2.0 sits on top of http
    - Adds ID Token(JWT), userinfo endpoint, standard scopes(permissions)
  + Using Auth0 we don’t have to implement OpenID connect from scratch
* JSON Web Tokens
  + JWT
  + Access token
  + Used for authorization and info exchange
  + Often contains user info
  + Pronounced “JOT”
  + Digitally signed
  + Can be encrypted
  + JSON is less verbose than XML
  + Smaller when encoded
  + Easier to sign than SAML
  + JSON is easy to parse on the client
  + JWT Parts
    - Header
      * Type, hash algorithm, key id
    - Body
      * User identity claims
    - Signature
      * Verify the sender assure content is legit
  + JWT looks like a bunch of random letters
  + Use <http://jwt.io> to decode JWTs
  + OpenID Connect uses identity token
  + OAuth2.0 uses access token
  + JWTs can’t be revoked
    - So make their lifespan short
    - Auth0 default to 10 hours
* Summary
  + Login and receive JWT identity token(OIDC)
  + Authorize app and receive access token(OAuth 2.0)
    - For api calls
  + Include your access token in API calls for authorization
  + Use identity tokens for authentication
    - Contains user information
  + Use access tokens to access APIs
    - Contains scopes to describe user permissions
* Intro
  + create-react-app
  + react router
* Tools We’re Using in this Cour..
  + Github.com/facebook/create-react-app
  + Install latest version of node.js
  + Install code editing tool like visual studio code
* Create App via create-react-a…
  + running at least Node 6 and npm 5.2
  + in command line type to check
    - node -v
    - npm -v
  + on older versions
    - npm install -g create-react-app
    - create-react-app react-auth0
  + in v6 and above type
    - npx create-react-app react-auth0
  + cd in the folder created
  + type to start react app
    - npm start
  + open the app in vs code
  + install reactjs code snippets package
  + create a ‘Home’ file in src folder
    - type rcc and hit enter to have component snippet generated
    - add <h1>Home</h1> in between the <div> in the return ()
  + create a ‘Profile’ files in src folder
    - remove <div> and add <h1>Profile</h1>
  + in App.js
    - remove everything inside of <div classNampe=”App”> but leave the tags
  + open the visual studio code terminal
  + install dependency(bit.ly/ps-auth0 to get the updated npm install line)
    - npm install auth0-js@9.13.4 auth0-lock@11.25.1 express@4.17.1 express-jwt@5.3.1 express-jwt-authz@1.0.0 jwks-rsa@1.3.0 npm-run-all@4.1.5 react-router-dom@5.2.0
* Add Routing vis React Router
  + user react router to navigate between pages
  + in index.js
    - import {BrowserRouter as Router} from ‘react-router-dom’;
  + then wrap app with router component
  + declare a single route that will render the app component
    - ReactDOM.render(
    - <Router>
    - <Route component={App} />
    - </Router>,
    - document.getElementById(“root”)
    - );
  + in App.js, declare routes
    - import {Route} from “react-router-dom”;
    - import Home from “./Home”;
    - import Profile from “./Profile”;
    - …
    - render() {
    - return(
    - <>
    - <Route path=”/” exact component={Home} />
    - <Route path=”/profile” component={Profile} />;
    - </>
    - );
    - }
    - …
* Add Navigation Bar
  + Add new file to src folder ‘Nav.js’
    - Use rcc
    - Import { Link } from ‘react-router-dom’;
    - …
    - Change <div> to ul
    - <nav>
    - <ul>
    - <li>
    - <Link to=”/”>Home</Link>
    - </li>
    - <li>
    - <Link to=”/profile”>Profile</Link>
    - </li>
    - <ul>
    - </nav>
    - …
  + In App.js
    - Import nav component
    - import Nav from “./Nav”;
    - ..
    - <>
    - <Nav />
    - <Route ….
    - …
* Style App
  + Copy this into index.css
    - nav li a {
    - display: block;
    - color: white;
    - text-align: center;
    - padding: 14px 16px;
    - text-decoration: none;
    - }
    - nav li a:hover {
    - background-color: #d9390d;
    - }
    - nav button {
    - margin-top: 13px;
    - }
* Summary
* Intro
* Auth0 Version Used in the C..
* Sign up for Auth0
  + Tenant: logical isolation unit
    - No tenant can access data in another tenant
    - Think of tenant like a development environment
* Key Auth Decisions
  + Pick a flow
    - Implicit flow for client side web apps
  + Implicit flow
    - App directs browser to auth0 sign in
    - Auth0 redirects to your app at the callback URL you chose
    - Your app reads the tokens from the URL
    - App can use token to call resource server, for example API
  + Pick Login/Sign up
    - Universal
    - Embedded lock
    - Custom UI
  + Universal and embedded use lock widget
    - Easily integrates with Auth0
    - Adapts to your config
    - Looks great on any device
    - Remembers the last used connection
    - Automatic internationalization
    - Detailed password policy check
    - Customizable
  + Universal is hosted by Auth0
    - Avoids cross domain issues
  + Embedded Lock
    - Embedded in app
    - Places login form in app
    - Less secure because it requires cross domain calls
  + Custom UI
    - Build your own interface
    - Use Auth0 APIs
* Create an App in Auth0 D..
  + Click on new application
  + Click Single Page Web Applications
    - Configures auth0 to use the implicit grant
  + Click on app in application to go into settings
    - Specify the callback url
    - <http://localhost:3000/callback>
    - Need specify same callback url in application
* Configure Environment Variable
  + Use auth0.js package
  + Create a .env file
    - Common file to specifying environment specific variables
    - Set the variables
    - REACT\_APP\_AUTH0\_DOMAIN=[domain for app in auth0]
    - REACT\_APP\_AUTH0\_CLIENTID=[client id for app in auth0]
    - REACT\_APP\_AUTH0\_CALLBACK=[callback url]
  + Using REACT\_APP\_ as a prefix
    - Create-react-app automatically exposes environment variables that start with REACT\_APP to our app
* Create Auth Object
  + Create a new file in new folder Auth
    - Create file Auth.js
    - Import auth0 from ‘auth0-js’
    - export default class Auth {
    - constructor(history) {
    - domain: process.env.REACT\_APP\_AUTH0\_DOMAIN,
    - clientID: process.env.REACT\_APP\_AUTH0\_CLIENT\_ID,
    - redirectUri: process.env.REACT\_APP\_AUTH0\_REDIRECT\_URL,
    - responseType: “token id\_token”,
    - scope: “openid profile email”
    - {
    - }
  + Pass react router history in so auth can perform redirects
  + Id\_token option gives us an JWT token to authenticate the user when they login
  + Token gives us an access token so the user can make API calls
  + Scope is where we specify permissions
    - openid: we’ll get back jwt with openid claims
    - profile: receive user profile date, depends on how user signs in based on identity provider
* summary
* Intro
  + Implement login
* Setup Login
  + In Auth.js add method
    - login = () => {
    - this.auth0.authorize();
    - }
    - This will redirect the browser to the Auth0 login page
  + In App.js, instantiate instance of authentication object for it to be used by components
    - …
    - import Auth from “./Auth/Auth”;
    - class App extends Component {
    - constructor(props) {
    - super(props);
    - this.auth = new Auth(this.props.history);
    - }
    - ..
  + In App.js pass auth object to components
    - <Route path=”/” exact render={props => <Home auth={this.auth} {…props} />
  + Add a login button to Home.js
    - <button onClick={this.props.auth.login}> Log in </button>
  + npm install to install all the packages in package.json
  + add back in later
    - <Route
    - path="/callback"
    - render={props => <Callback auth={this.auth} {...props} />}
    - />
* Review Callback URL and JWT
* Parse Callback URL in Callbac..
  + create a new file Callback.js
  + add route to callback in App.js and import Callback
    - ..
    - <Route path=”/callback” render={props => <Callback auth={this.auth} {...props} />} />
  + in Callback.js
    - handle reading the url, handle authentication if expected values are int url
    - componentDidMount() {
    - if(/access\_token|id\_token|error/.test(this.props.location.hash)) {
    - this.props.auth.handleAuthentication();
    - } else {
    - throw new Error(“Invalid callback URL.”);
    - }
  + then need to implement handleAuthentication()
* Pick a Token Storage Approach
  + can store JWT token in LocalStorage, SessionStorage, Cookie, In Memory
    - tradeoffs involved
    - if your react app has a dedicated server to then use cookies
    - HttpOnly cookie prevents cross site scripts
  + Cross-site scripting
    - attacker injects client-side script onto your page
    - Risk: Mishandling user content
  + Don’t store tokens in local storage
    - susceptible to cross site scripting
  + Handle token in backend if possible
  + for Single page applications use in memory storage
    - use silent auth0 to avoid relogging
* Implement Handle Authentication
  + in Auth.js
    - …
    - handleAuthentication = () => {
    - this.auth0.parseHash((err, authResult) => {
    - if (authResult && authResult.accessToken && authResult.idToken) {
    - this.setSession(authResult);
    - this.history.push(“/”);
    - } else if (err) {
    - this.history.push(“/”);
    - alert(`Error: ${err.error}. Check the console for further details.`);
    - console.log(err);
    - }
    - });
    - };
    - setSession = authResult => {
    - console.log(authResult);
    - const expiresAt = JSON.stringify(
    - authResult.expiresIn \* 1000 + new Date().getTime()
    - );
    - localStorage.setItem(“access\_token”, authResult.accessToken);
    - localStorage.setItem(“id\_token”, authResult.idToken);
    - localStorage.setItem(“expires\_at”, expiresAt);
    - };
  + run page then login
    - right click inspect app
    - click ‘Application’
    - check local storage
    - the access\_token, id\_token, expires\_at should be present
* Check if User is Authenticated
  + In Auth.js add isAuthenticated()
    - isAuthenticated() {
    - const expiresAt = JSON.parse(localstorage.getItem(“expires\_at”));
    - return new Date().getTime() < expiresAt;
    - }
  + in Home.js add logic to show profile link instead of login button if user is authenticated
    - <h1> Home </h1>
    - {this.props.auth.isAuthenticated() ? (
    - <Link to=”/profile”>View profile</Link> ) :
    - ( <button onClick={this.props.auth.login)>Log In </button> )}
    - ….
* Summary
* Intro
* Implement Logout
  + in auth.js implement logout
    - logout = () => {
    - localStorage.removeItem(“access\_token”);
    - localStorage.removeItem(“id\_token”);
    - localStorage.removeItem(“expires\_at”, expiresAt);
    - }
  + add logout button to NAV.js
    - …
    - <button onClick={isAuthenticated() ? logout : login}>
    - {isAuthenticated() ? “Log Out” : “Log In”}
    - </button>
  + if you don’t logout of auth0 server and just wipe local storage, its considered a soft logout
    - useful for single-sign on scenarios so you session stays valid for other apps using this Auth0 tenant
    - auth0 checks the session cookie on your Auth0 domain to determine if you’re logged in
  + in logout
    - add
    - this.auth0.logout({
    - clientID: process.env.REACT\_APP\_AUTH0\_CLIENT\_ID,
    - returnTo: “<http://localhost:3000>”
    - });
    - and remove this.history.push(“/”);
  + in Auth0 dash board, click on application
    - in Allowed Logout URLs add the logout url
* Review Signup Process
* Get User Profile
  + In Auth.js add code to display profile information
    - getAccessToken = () =>
    - const accessToken = localStorage.getItem(“access\_token”);
    - If(!accessToken) {
    - throw new Error(“No access token found.”);
    - }
    - return accessToken;
    - };
    - getProfile = cb => {
    - if(this.userProfile) return cb(this.userProfile);
    - this.auth0.client.userInfo(this.getAccessToken(), (err, profile) => {
    - if(profile) this.userProfile = profile;
    - cb(profile, err);
    - });
    - };
  + Add this.userProfile = null; to the constructor
  + Add this.userProfile = null; to the logout function
* Configure Profile Page Route
* Display User Profile
* Summary
* Intro
  + Create APIs via Node and Express
  + Integrate APIs with create-react-app
  + Configure express to parse JWTs
  + Create multiple API endpoints
    - Public
    - Private(login required)
* Create Environment Variables
  + In .env add variables
    - REACT\_APP\_AUTH0\_AUDIENCE=http://localhost:3001
    - REACT\_APP\_AUTH0\_URL=http://localhost:3001
* Create API with Node and Exp..
  + Create a new file in source directory. Server.js
    - const express = require(‘express’);
    - require(‘dotenv’).config();
    - const app = express();
    - app.get(‘/public’, function(req, res) {
    - res.json({
    - message: “Hello from a public API!”
    - });
    - });
    - app.listen(3001);
    - console.log(“API server listening on “ + process.env.REACT\_APP\_AUTH0\_AUDIENCE);
* Start Express API Server via n..
  + Create script to automate some work
  + In package.json
    - Add new script in scripts
    - “scripts”: {
    - ….
    - “start:server”: “node server.js”,
    - …..
  + Then in terminal type ‘npm run start:server’
  + In browser go to <http://localhost:3001/public>
  + To get it to run on npm start, in server.js
    - “scripts”: {
    - “start”: “run-p start:client start:server”,
    - “start:client”: “react-scripts start”,
    - “start:server”: “node server.js”,
    - ….
* Call Public API via React
  + Create a new page in src, public.js
    - class Public extends Component {
    - state = {
    - message: “”
    - };
    - componentDidMount() {
    - fetch(“/public”)
    - .then(response => {
    - if(response.ok) return response.json();
    - throw new Error(“Network response was not ok.”);
    - })
    - .then(response => this.setState({message: response.message}))
    - .catch(error => this.setState({message: error.message }));
    - }
  + Add to Nav.js and App.js links
  + In package.json add above “eslintConfig”
    - “proxy”: <http://localhost:3001>
* Create Auth0 API
  + In Auth0 dashboard, need tell about it
  + Click on APIs
    - Click ‘create API’
    - Fill in and click create
  + Use a separate Auth0 tenant for each environment and set the API identifier to the API’s environment for QA and prod
  + In Auth.js, specify the audience
    - ….
    - redirectUrl: process.env.REACT\_APP\_AUTH0\_CALLBACK\_URL,
    - audience: process.env.REACT\_APP\_AUTH0\_AUDIENCE,
    - ..
  + The audience is the identifier
* Configure Express to Parse J..
  + In server.js, support reading a JWT
    - …
    - const jwt = require(“express-jwt”); //validate JWT and set req.user
    - const jwtRsa = require(“jwks-rsa”); //retrieve RSA keys from a JSON web key set(JWT) endpoint
    - const checkJwt = jwt({
    - secret: jwksRsa.expressJwtSecret({
    - cache: true,
    - rateLimit: true,
    - jwsRequestPerMinute: 5,
    - jwksUri: ‘https://${process.env.REACT\_APP\_AUTH0\_DOMAIN}/.well-known/jwks.json’
    - }),
    - audience: process.env.REACT\_APP\_AUTH0\_AUDIENCE,
    - issuer: ‘https://${process.env.REACT\_APP\_AUTH0\_DOMAIN}/’,
    - alogrithms: [“RS256”]
    - });
  + Validating a JWT
    - Verify signature
      * Can see jwt at https://YOUR\_AUTH0\_DOMAIN/.well-know/jwks.json
    - Validate Claims
      * Exp(expiration), confirm it hasn’t expired
      * Iss(Issued by) confirm it matches your Auth0 domain
      * Aud(Audience) confirm it matches your clientID
  + Json web key set(JWKS)
    - Json object that represent a cryptographic key
    - Member are properties of the key
* Call Private API
  + create a new file Private.js in src, similar to Public.js
    - ….
    - componentDidMount(){
    - fetch(‘/private’, {
    - headers: { Authorization: `Bearer ${this.props.auth.getAccessToken()}`}...
  + add route to Private.js in App.js
  + add link in Nav.js
* Summary
* Intro
* Intro to OAuth Scopes
  + Give permission without sharing credentials
  + Each permission you grant is called a scope
* Create a New Scope
  + Scopes delegate permission
  + Scopes specify the actions an app can perform on behalf of a user
  + In Auth0 dashboard, click on your api
    - Click on ‘Permissions’/’Scopes’
    - Add a permission
      * read:course
      * give a description
  + in Auth.js, request the scope we created
    - this.requestedScopes = “opendid profile email read:courses”;
    - …
    - responseType: “token id\_token”,
    - scope: this.requestedScopes
  + Then in setSession in Auth.js
    - ….
    - const scopes = authResult.scope || this.requestedScopes || ‘’;
    - ….
    - localStorage.setItem(“scopes”, JSON.stringify(scopes));
    - };
  + Then in logout in Auth.js add code to remove scopes
  + Then in Auth.js add function to check scope
    - userHasScopes(scopes) {
    - const grantedScopes = (
    - JSON.parse(localStorage.getItem(“scopes”)) || “”
    - ).split(“ “);
    - return scopes.every(scope => grantedScopes.includes(scope));
    - }
  + Return scopes.every…
    - Iterate over each scope and returns true if everyone of the scopes passed in the function are found in the list of scopes in local storage
* Configure Express to Validate ….
  + In server.js, validate jwt scopes
    - …
    - const checkScope = require(‘express-jwt-authz’); //validate jwt scopes
  + the add another endpoint in server.js
    - app.get(“/courses”, checkJwt, checkScope([“read:courses”]), function(req, res) {
    - res.json({
    - courses: [
    - { id: 1, title: “Building Apps with React and Redux” },
    - { id: 2, title: “Creating Reusable React Components” },
    - ]
    - });
    - });
* Create React Page That Valid…
  + Create course.js page
  + Then in app.js create a route and add client side checks
    - <Route path=”courses” render={props =>
    - this.auth.isAuthenticated() &&
    - this.auth.userHasScopes([“read:courses”]) ? (
    - <Courses auth={this.auth} {…props} />
    - ) : ( this.auth.login() )
    - } />
  + The app.js check are merely for user experience, not security
  + It’s the server’s job to validate the user is authorized when an API call is made
  + Then add in Nav.js
    - {isAuthenticated() && userHasScopes([“read:courses”]) && (
    - <li> <Link to=”/courses”>Courses</Link></li>)}
* Assign Admin Role via a Rule
  + Customize authentication pipeline behavior using rules
  + Written in java script
  + Click add rules in auth0 dash board
    - Choose ‘set roles to a user’
* Debug Rule
* Validate Roles in Access Tokens
  + Role is a custom claim
    - Custom claims aren’t added to access tokens by default
  + In Rules in auth0 dashboard
    - Add an empty rule
    - Add custom claim to access token
    - function (user, context, callback) {
    - if(user.app\_metadata && user.app\_metadata.roles) {
    - context.accessToken[‘http://localhost:3000/roles’] = user.app\_metadata.roles;
    - }
    - callback(null, user, context);
    - }
  + In server.js add api to check roles
    - ..
    - app.get(“/admin”, chekcJwt, checkRole(“admin”), function(req, res){
    - res.json({ message: “Hello from an admin API!” }); });
* Authentication Approach Opt….
  + Session Cookie
    - Store cookie in browser
    - Random characters tie a user to a session that’s stored on the server
    - Simple,secure
    - No authorization data included
    - Impacts performance because have to look up privileges
  + JWT with scopes
    - Scopes were designed to specify what an application is allowed to do with a third party on a user’s behalf
    - Can assign different scopes to different users upon login using auth0 rules
    - Performance
    - Bloated jwts
    - You can end up with a bloated jwt with dozens of scopes that you need to send for every request
    - Doesn’t scale well
  + JWT with roles
    - Roles group user by permissions
    - Simple, scalable
    - Avoids jwt bloat
    - Fast
  + Use scopes when interacting with third parties
  + Use roles for handling your app’s permissions
* Summary
* Intro
* Redirect to Previous Page Up..
* Create PrivateRoute Component
* Create AuthContext
  + with React’s context we can eliminate the need for passing the same prop down to every component
  + steps to configure context
    - declare context in new file
    - declare provider(provides data/funcs)
    - declare consumer(consumes data/funcs)
* Declare AuthContext Provider
  + typically the provider is declared near the app’s entry point so all child components can consume the data and functions it provides
* Consume AuthContext
* Store Tokens in Memory
* Silent Authentication and Tok…