Migrating Applications from AngularJS to Angular

Schedule

Intro - 30 minutes

Preparing your ng1 app - 1.5 hours

- Steps to Prepare
  - 1: follow the style guide
    - **Tag Step2 - converting adminLogin controller to controllerAs**
      - Mention splitting each object into a separate object. Look at components.js for a bad example.
  - 2. Update to the latest version of Angular 1
    - **Update to angular 1.5.5 in index.html**
  - 3. All new dev with components
  - 4. Switch controllers to components
    - **Tag Step 3 - converting a controller to a component**
      - Convert the adminLoginCtrl.js to a component
      - Rename the file, remove Ctrl
      - Change the name in index.html
      - Change the route to be a component route
      - EXERCISE
        - Convert admin/results to component
        - Show how to handle the routing with resolve
  - *5. Remove incompatible features from Directives*
    - *Compile*
    - *Terminal*
    - *Priority*
    - *Replace*
  - 6. Switch Component Directives to 1. 5 Components
    - **Tag step4 - Convert unreviewedTalk to a component**
      - Change the js file & the html file
      - Optional: discuss new features in 1.5 components
      - **EXERCISE**
        - Convert the nav.js to a component
  - 7. Switch to Manual Bootstrapping
    - **Tag step5 - change to manual bootstrapping**
      - Remove ng-app from index.html

        - Explain what ng-app does
        - Add the bootstrap call in app.js
            - You can bootstrap document, or document.body
- 8. Add TypeScript & a build
    - **Tag step6 - install typescript & build**
        - Npm i typescript -D (don't actually run this, should already be on their machine if they ran "npm i" after cloning
        - Add "tsc" command to package.json file
        - Add tsconfig.json file. Explain important pieces
        - Rename app.js to app.ts
        - Npm run tsc
        - Notice how the js & map file were produced
        - Talk about using typescript - **SLIDES**
            - Do we mass rename all js files to ts?
            - Do we not? If so we might edit the js file accidentally
            - Do we build to a separate directory?
            - If so then relative paths to template files may no longer work
            - Recommendation: rename all js to ts, ignore js & map files from source control & editor
        - Note that some files may give the TSC fits (warnings). Show this by renaming toastr.js to toastr.ts and npm run tsc.
            - Fix this by adding declare var toastr; in the file
        - Just update to step6, don't walk through the steps to do all this
- 9. Start using ES6
    - If time allows:
    - Mention arrow functions
    - Mention multiline strings
    - Mention string interpolation
    - Mention classes
    - Mention destructuring
- 10. Switch Controllers to ES6 Classes
    - **Tag step7 - change login.ts to use a class**
        - Rename file to login.ts
        - Update index.html
        - Change login.ts to use class
        - Update login.html
- 11. Switch Services to ES6 Classes
    - **Tag step8 - change auth.ts to use a class**
        - Update the file to be a class
    - **Tag step9 - everything prepared**

Adding ng2 - 30 minutes

- Installing ng2
  - **Update package.json & npm install**
- Migration Step 12 & 13: Add Angular 2 & Bootstrap
  - **Tag step10 - bootstrap ng2**
  - Explain each piece
  - Update index.html
    - We now include all the shims to use systemjs and es6 etc.
    - We use systemjs to begin loading our angular 2 code with es6 modules
  - Update tsconfig.json to use node moduleresolution
    - This allows typescript to find the modules in the right place
  - Update app.ts
    - We no longer bootstrap angular 1
  - Create config/systemjs.config.js
    - App: './' is because the index.html is in the app directory. So app is now in the same dir as index.html
    - Explain the @angular/?? Maps
    - Explain the packages app & rxjs
  - Update expressConfig.js to serve up config & node_modules directories.
  - Create main.ts,
    - This is the first file requested by systemjs.
    - We bootstrap here using the ngUpgrade bootstrapper
    - We bootstrap our ng2 module, and then we bootstrap the ng1 module
  - Create rxjsOperations.ts
    - This has the operations from rxjs we need. Explain rxjs requests
  - Create app.module.ts
    - The ng2 module imports the important modules, and declares out app component & marks it to be the bootstrapped component
  - Create app.component.ts,
    - We define the template for our root component here, which uses an ngView
    - We can add a router outlet for when we start routing using the ng2 router. That's not necessary yet

The Migrating Process - 3 hours 30 minutes

- Migrate & downgrade a service
  - **Tag step11 - Convert nameParser service to ng2**
    - Convert the service
    - Rename the file to .service.ts
    - Remove from index.html
    - In main.ts

            - Import downgradeInjectable from angular upgrade static
            -
        -
            - We have to add providers to the module before its bootstrapped with ng1
        - Add the new service to the providers list of the module in app.module
        - We also cleaned up some erroneous stuff in the app.module.ts file from the old upgrade adapter
    -
- Migrate a sub - component
    - **Tag step12 - convert unreviewedTalk to ng2**
        - Rename the file & template to .component.*
        - Convert file to ng2
            - Mention inputs & outputs
        - Convert html to ng2
            - Remove pipe for now
        - Downgrade the component in main.ts
            - Set the inputs & outputs
        - Add the component to app.module declarations & entryComponents
        - Remove script from index.html
        - Adjust home.html
            - Note how we use ng2 bindings inside the node. It's owned by ng2
- Migrating a Pipe
    - **Tag step13 - convert a pipe to ng2**
        - We can't upgrade/downgrade a pipe, so we duplicate it until it's only used by ng2. Keeping track of when it's safe to delete can be a trick.
        - Create the common directory
            - Silly to use the components directory. Components are overused in ng2
        - Create duration pipe file in common
        - Implement it
        - Add it to the app.module in the declarations
        - Add it to the unreviewedTalk.component.html file
- Migrate a top level component
    - **Step14 - convert profile to ng2**
        - Rename profile to .component
        - Rename profile html to .component
        - Convert profile component
        - Convert profile template
        - Remove from index.html
        - Add to app.module declaration & entryComponents
        - Downgrade in main.ts
        - Fix route in routes.ts

        - Fake it since we know that we don't have the services
- upgrade a service
    - **Step15 - upgrade the toastr, location, & currentIdentity services**
        - Start with $location
        - Add the provide statement in the app module
        - Add the @Inject in the profile component
        - Explain string tokens
        - Next do currentIdentity. Same as $location
        - Now do toastr. For this we could just use the ng1 toastr service, but instead let's migrate it, since we don't have to rewrite any functionality. Just wrap it differently
        - Create the toastr.service.ts file
        - Implement the toastr.service file with the interface and an opaque token
        - Add toastr as a service in the app.module file, include the declare var
        - Add the @Inject in the profile component
- upgrade a sub component
    - **Step16 - upgrade the nav component.**
        - Discuss possibilities: migrate it, or upgrade it.
        - Create nav.component.ts and the UpgradeComponent
        - Add it to the app.module.ts file
        - Run, note the error about async tempaltes
        - Discuss remedies wrap it, inline template, precache template
        - Inline it and show that it works fine.
        - If short on time, write a wrapping control.
    - **Step 17 - wrapping control - OPTIONAL**
        - Create nav-wrapper.comopnent.ts, impllement it
        - Import the nav wrapper in the app.module
        - Change profile.html to use nav-wrapper
        - Change nav.ts to go back to templateUrl instead of template
- Migrate a service that uses http
    - RxJS vs Promises
    - **Step18 - Migrate Sessions service to ts**
        - We've now upgraded & downgraded components and services. Time to look a bit deeper
        - Migrating a large service can be difficult.
        - It can be used by lots of consumers
        - It's complicated by http wrapping services which change the interface from promises to observables
            - **Look at slide in slide deck on http vs observables.**
        - If the service is stateless there's a technique for migrating a method at a time. If not, it's just going to be a pain in the butt.
        - Create sessions.service.ts
        - Add sessions to app.module

        - Downgrade sessions in main.ts, naming it sessions_v2
        - Migrate just getSessionsByUser to the ng2 sessions service.
        - Implement it in the routes.ts for the userSessions resolve
        - Verify it works, then remove it from the v1 service.
        - This ONLY works for stateless services, which is a huge advantage of statelessness. Good engineering practices are showing their benefits!
    - **User Exercises**
        - Migrate more methods to ng2
- Content projection/transclusion
    - **Step19 - Migrating DetailPanel to ng2**
        - Create common/detailPanel.component.ts & html
        - Convert the code
        - Remove detailPanel script from index.html
        - Add to main.ts & downgrade it, adding the inputs
        - Add to app.module declarations & entryComponents
        - //
        - Change the binding syntax in sessionDetail.html
            - We can either leave it alone, since it’s just strings and they’re one-time. They don’t need to actually bind. Or we use ng2 syntax
                - title="{{$ctrl.session.title}}"
                - OR
                - [title]="$ctrl.session.title"
                - Can’t leave it alone with objects. Have to do [] then.
            - We can fix the type coercion by using a binding in sessionDetail.ts of either “=” or “<”. Discuss how < is the same now as @Input
            - It’s always annoying when we mix, going back and forth between ng1 and ng2
        - EXERCISE
            - Fix binding syntax in sessionDetailWithVotes.html
- Dealing with resolved data and guards
    - **Step20 - migrate admin results page**
        - Rename files to .component
        - Remove from index.html
        - Migrate sessionDetailWithVotes (to make the whole thing pretty much ng2)
        - Change <nav> to <nav-wrapper>
        - Downgrade the results component in main.ts, using the inputs array
        - Add both components it to app.module & results to entryComponents
        - Change the binding syntax in routes (mix of ng1 and ng2)
- Migrating a decorator directive
- Best Practices
    - Don't go back & forth from ng1 to ng2
- Migrate routing

- don't use .otherwise
- don't nest components
- **Step21 - Migrate routing for admin/results**
- Create router-outlet in the app component
- Import routermodule
- Add urlHandling Strategy
- Add it as a provider
- Define routes with forRoot()
- Remove resultsComponent as an entrycomponent
- Remove the otherwise route
- Provide a redirect from / to /home
- Remove /admin/results from ng1 route list
- Provide $scope in app.module (only because nav is still ng1)
- Test. no data for the sessions
- Create AllSessions resolver.ts file
- Update the allSessions method of sessions.service
- Import it in the app.module, add it as a resolve on the route
- Add it as a provider
- Note that we can’t refresh, because some of the page is ng1. Really this should be done when everything is converted to ng2

Final Thoughts