Advanced Angular

Toby Dussek

framework training
business value through education

Course times each day

- 9:30 start
- 11:00 coffee break
- 12:30 lunch break
- 3:00 tea break
- 4:30 end

Welcome and introductions

- What you currently know
- What you need to know

Check Setup

- Node 14.x or better
  - node -v
- npm install -g @angular/cli
  - npm install -g @angular/cli@latest
  - @angular/cli@11.0.7

Change Detection

framework training
business value through education

OnPush Change Detection

- When using OnPush detectors, the framework will check an OnPush component:
  - When any of its input properties changes
  - When it fires an event
  - When an Observable fires an event
  - https://blog.angular-university.io/onpush-change-detection-how-it-works/

- It is an anti-pattern to expose your subject to the outside world
  - Always expose the observable by using the asObservable() method
  - https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/

Angular Ivy

framework training
business value through education

Ivy will replace current rendering engine

- Keynote 2018: Miško Hevery (24-29 mins)
  - https://www.youtube.com/watch?v=dIxknqPOWms&feature=youtu.be&t=1359
- Locality: compile one file at a time
  - Ivy can only use component and it's decorators
    - No need for metadata.json
    - Less likely compilation needs to build all files
    - Enables meta programming: create components, modules, directives dynamically at run time E.g. higher-order components
  - Current compiler requires global knowledge of all dependencies
- To use it now
  - ng new myIvy --enable-ivy

Ivy and NgModule

- Ivy will enable us to bootstrap or render components to the DOM without any Angular modules
- We will be able to use components, directives, and pipes in component templates without Angular modules to resolve them
- https://blog.angularindepth.com/angular-revisited-tree-shakable-components-and-optional-ngmodules-329a4629276d

Ivy Rendering Code Video Demo

- Tree-shakeable by design (Kara Erickson 29-38 mins)
  - Rather than using static analysis, Ivy can determine actual retainment needs
  - Leads to smaller generated code, easier to read and debug
- New renderComponent bootstrapping API Demo (38-41 min)
- Debugging improvements Demo (inc. break-points) (41-44 mins)

Ivy Design Documentation

- Compiler Architecture
- https://github.com/angular/angular/blob/master/packages/compiler/design/architecture.md

Track How Ivy is Progressing

- Is Angular Ivy ready
    - https://is-angular-ivy-ready.firebaseapp.com/#/status
- Implementation Status
    - https://github.com/angular/angular/blob/master/packages/core/src/render3/STATUS.md

Angular Ivy change detection execution

- Preparing for Ivy
  - https://blog.angularindepth.com/angular-ivy-change-detection-execution-are-you-prepared-ab68d4231f2c
- Animated Visualization of Ivy Rendering Process
  - https://alexzuza.github.io/ivy-cd/

In-Depth Ivy

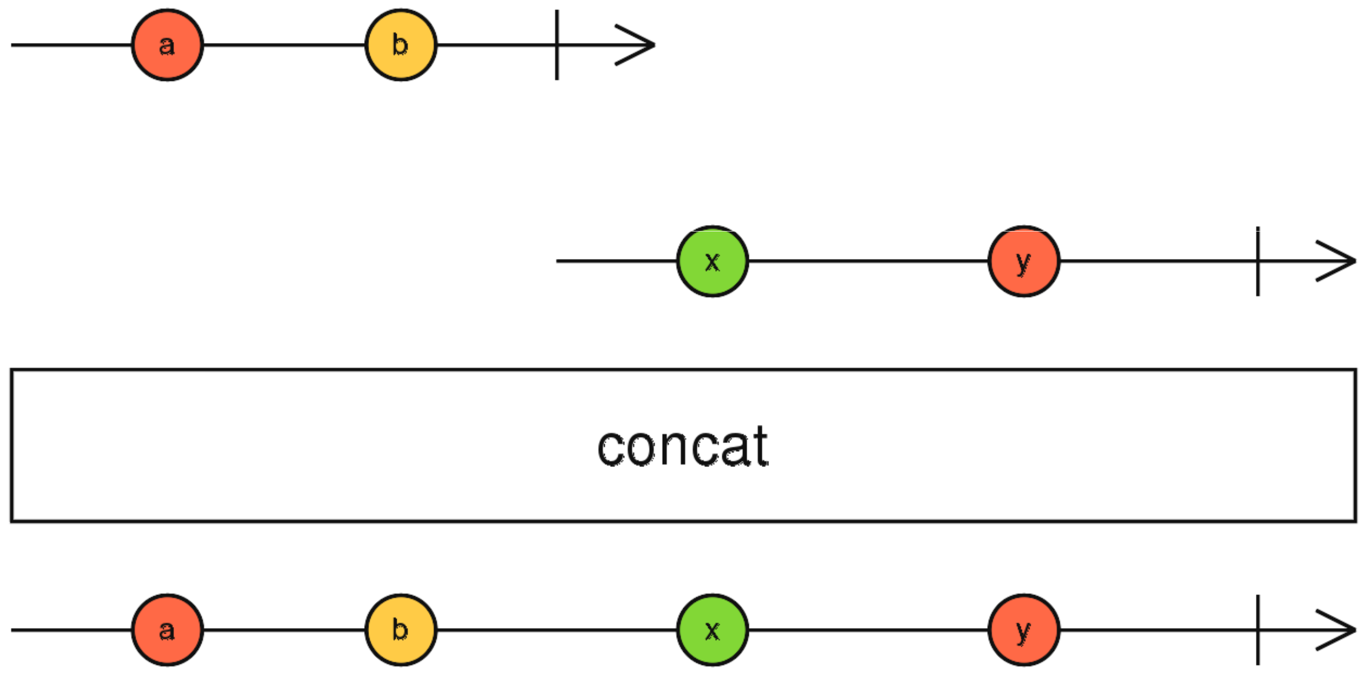
- https://blog.angularindepth.com/ivy-engine-in-angular-first-in-depth-look-at-compilation-runtime-and-change-detection-876751edd9fd
- In Ivy NgOnChanges is not a real lifecycle any more
- It is no longer right to say Angular is based solely on dirty check

Higher-Order RxJs Mapping Operators

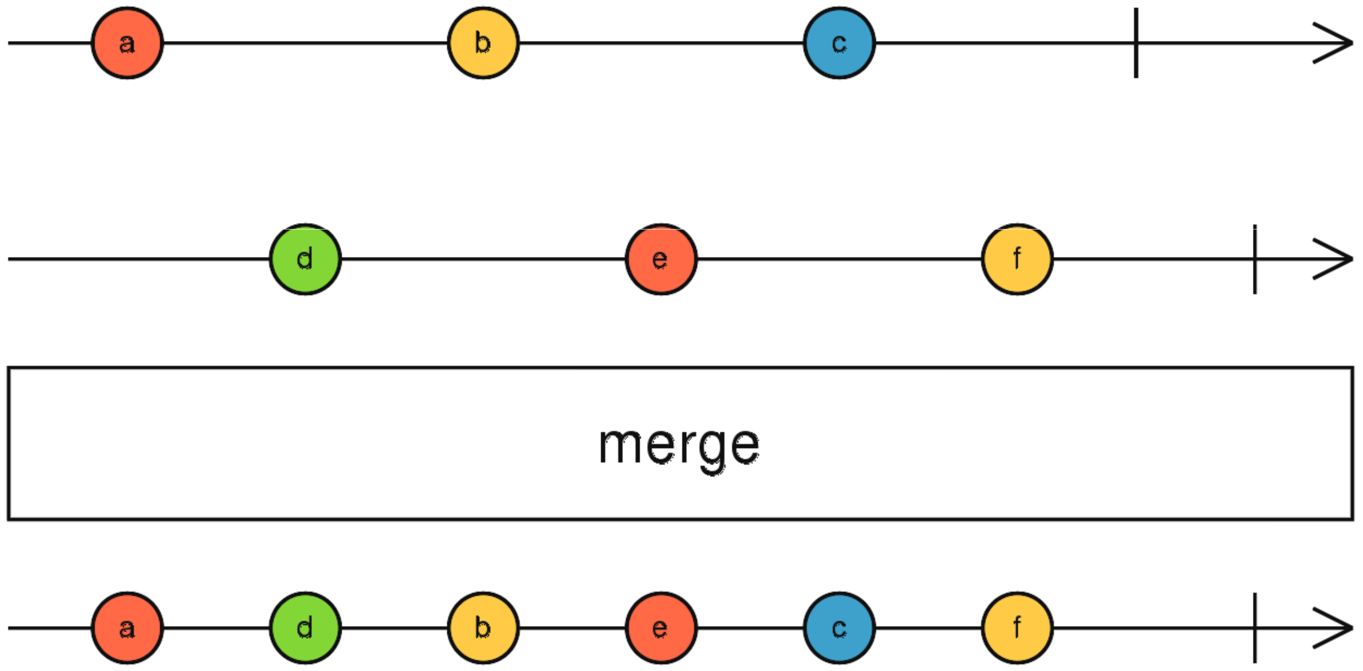
https://blog.angular-university.io/rxjs-higher-order-mapping/

framework training
business value through education

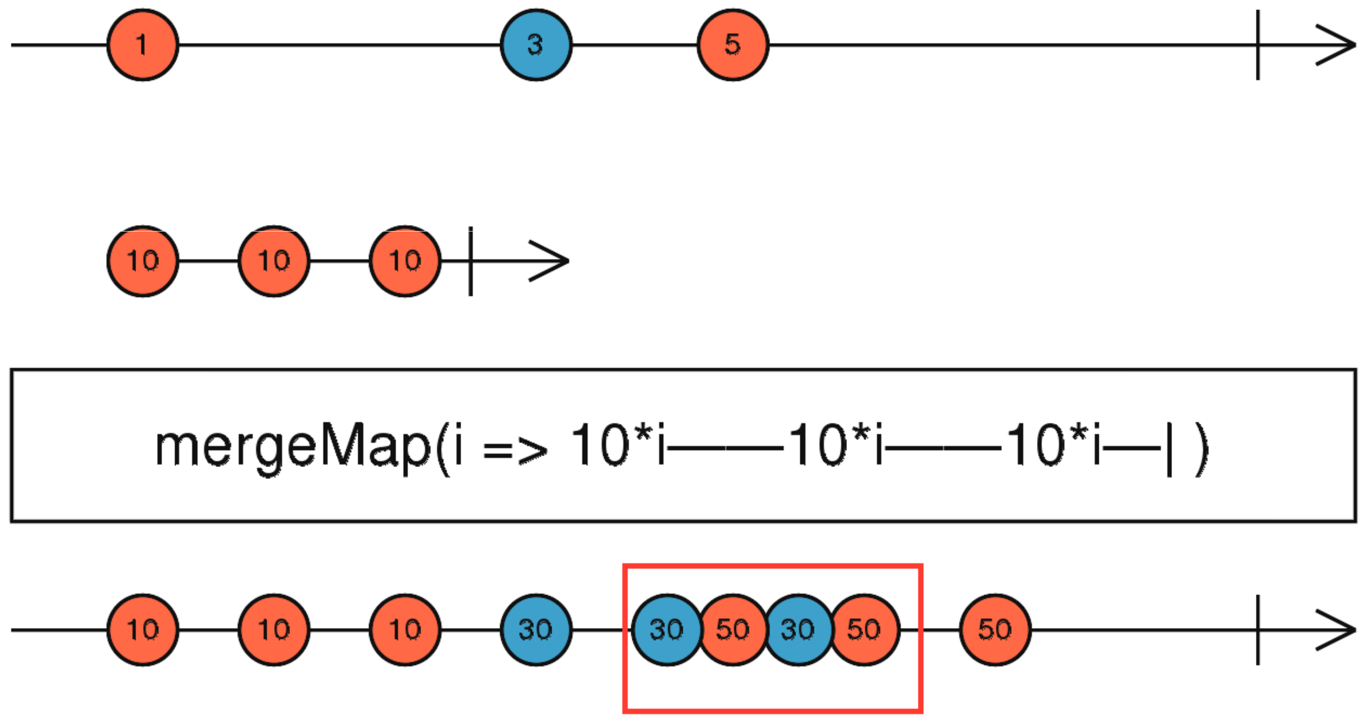
Concat



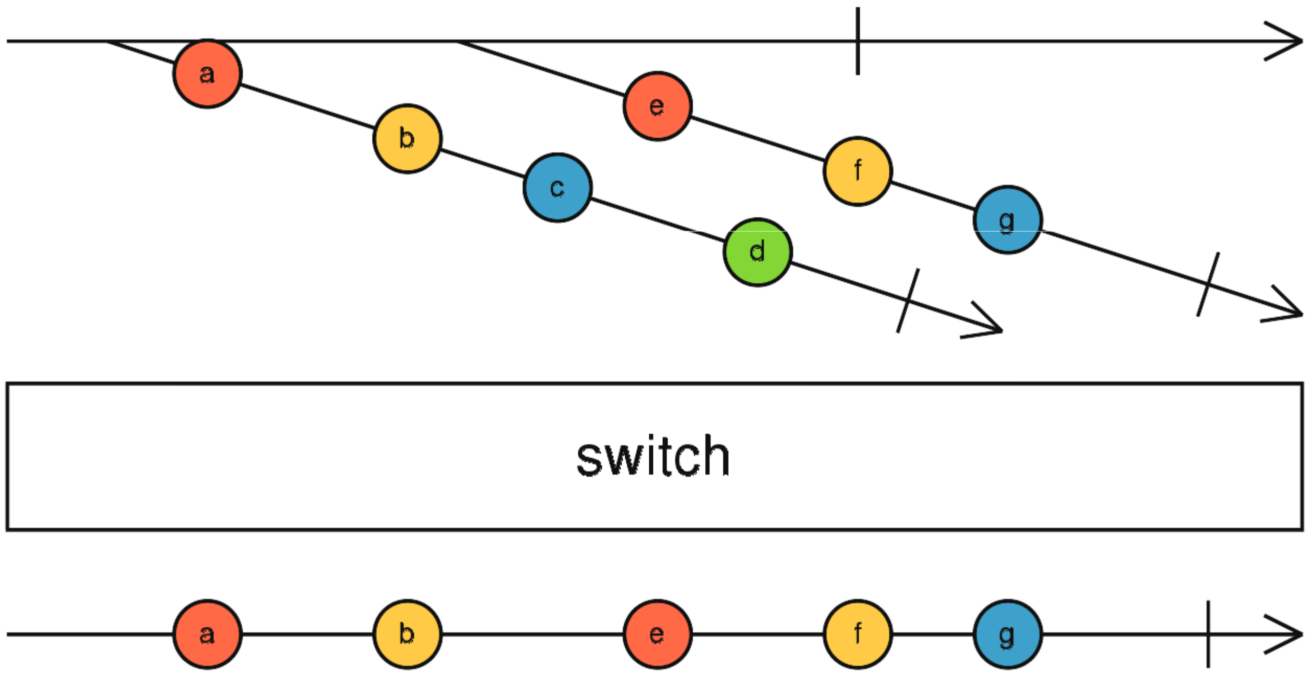
Merge



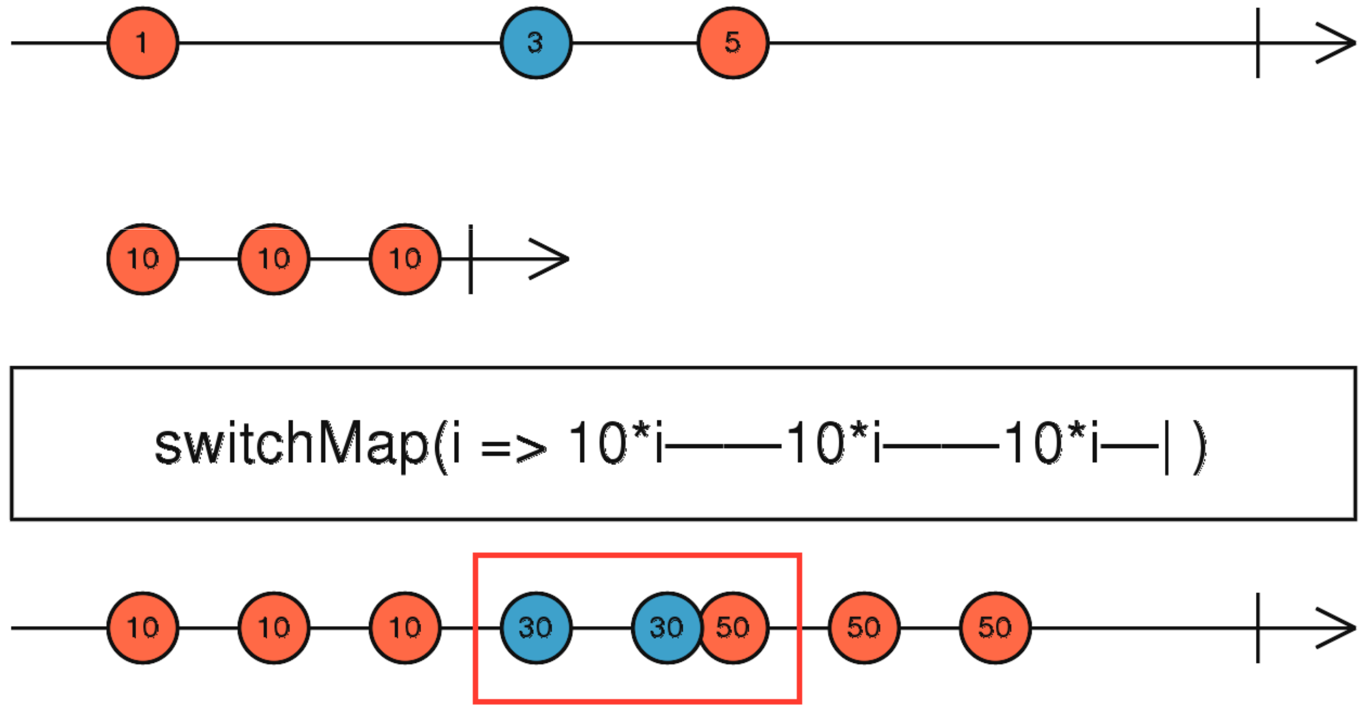
MergeMap



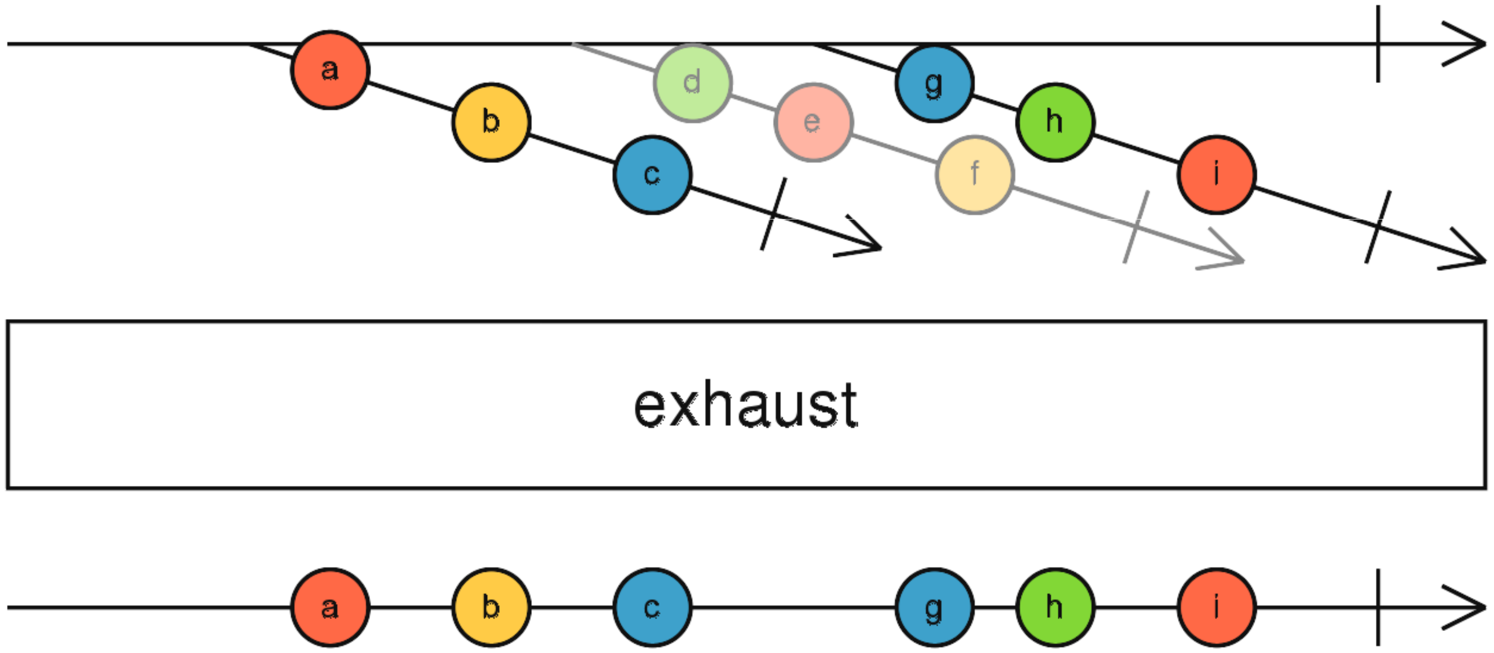
Switch



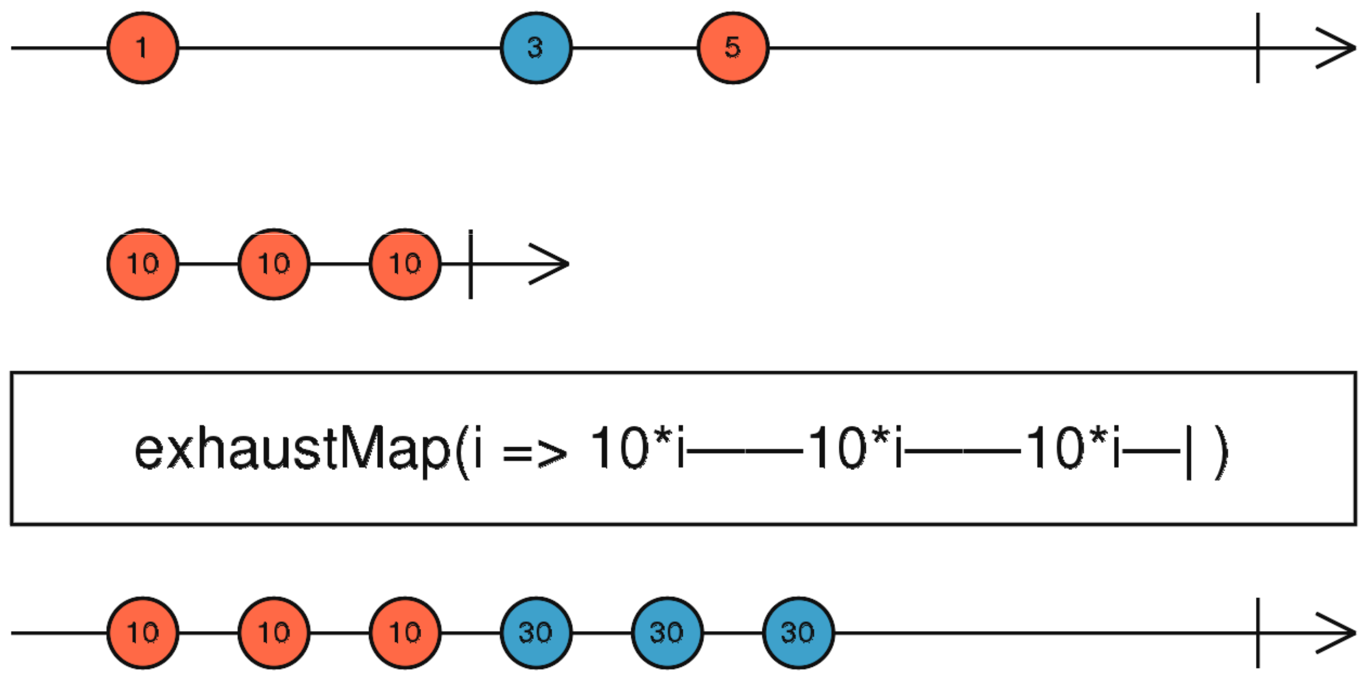
SwitchMap



Exhaust



ExhaustMap



Angular Security

framework training
business value through education

Built In security by default

- The DOM sanitizer, cross-site scripting attack protection etc. are all enabled by default
- You should consider very carefully If you are thinking you need to disable any of them
- Angular Security Best Practices
  - https://ordina-jworks.github.io/angular/2018/03/30/angular-security-best-practices.html
- HttpClient Security: XSRF Protection
  - https://angular.io/guide/http#security-xsrf-protection

Node Module Security

- Very common for vulnerabilities to be found in open source modules
- npm audit fix
- Implications for long-term projects

Authentication With JSON Web Tokens (JWT)

- Increasingly common for authentication and authorization
  - https://blog.angular-university.io/angular-jwt-authentication/
  - https://jwt.io/

Recent Changes to Angular CLI

- The Angular CLI configuration file angular-cli.json has been replaced with angular.json
- Angular CLI now generates a workspace that directly supports multiple projects
- ng update <package> will update dependencies to the latest versions
  - Including core packages in the dependencies and devDependencies and CLI
  - Due to its recursive action, the dependencies including zone.js, rxjs and typescript also get updated automatically
- By default polyfills are now loaded with <script nomodule> so earlier browsers will still load polyfills but module-aware browsers will not
  - Estimated to save ~56KB on es2015+ browsers

Angular Universal

framework training
business value through education

More than just SEO friendly

- Consider if the user starts interacting with the page immediately
  - There is a lag when the plain HTML being rendered until Angular kicks and take over
  - In that lag the user might click something or start using a form field
  - Via preboot functionality, Angular Universal records the events the user is triggering and plays them back once Angular is ready
  - Content is rendered on the server by the express Angular Universal rendering engine expressEngine (there is also a Hapi engine available)
- Server Side Rendering Development is not like coding for the client only
  - We want our application to do the exact same thing on the server as on the client
    - Not depending directly on browser APIs
  - This is not always possible
    - There are certain things that we want to do differently on the client and on the server
- https://blog.angular-university.io/angular-2-universal-meet-the-internet-of-the-future-seo-friendly-single-page-web-apps/

Angular Universal: Authentication

- Use Dependency Injection to implement Authentication
- On the client, we need the authentication identity of the user to be available as a token
  - e.g. on a cookie or in local storage
- On the server we need the identity token to be read from an HTTP request header

- https://blog.angular-university.io/angular-2-universal-meet-the-internet-of-the-future-seo-friendly-single-page-web-apps/

Routing

framework training
business value through education

Routing in Angular

- Angular Router: Child Routes, Auxiliary Routes, Master-Detail
- https://blog.angular-university.io/angular2-router/

- Angular Router: A Complete Example (build a Bootstrap Navigation Menu)
- https://blog.angular-university.io/angular-2-router-nested-routes-and-nested-auxiliary-routes-build-a-menu-navigation-system/

Libraries

framework training
business value through education

Create Library

- New feature in cli is createApplication=false
  - ng new myLib --create-application=false
- Then create library in this project
  - Change directory into myLib
  - Recommendation is to use a Prefix (default is lib)
  - ng generate library fooLib --prefix=foo
- Add as many libraries and applications as you like to the workspace
- Ensure a unique library name if planning to publish to npm
  - npm only allows a specific name and version combination once
  - You will need to change the version each time you publish updates

Generate Harness App and e2e Testing

- We can generate a harness app and e2e to call and test the Library
  - ng generate application fooTester
  - Angular CLI also adds fooTester-e2e
- Then include the library in the app
- Add the library to the imports array in app.module
  - import { FooLibModule } from ‘fooLib’
    - Remember to also add to imports array
  - Careful not to import as ./
  - Angular CLI looks in tsconfig.json paths then in node_modules
  - In app.component.html, include a reference to the library component
    - <foo-fooLib></foo-fooLib>

Build and Test the Library

- Build the Library
  - ng build fooLib
  - Angular CLI always builds libraries in production mode
- Build and serve the harness application
  - ng build fooTester --prod
  - ng serve fooTester
- Run Unit Tests for Angular Library and harness application
  - ng test fooLib
  - ng test fooTester
- Watch for library changes
  - ng build fooLib watch
- No need to publish to npm
  - Instead create an npm package and use it locally

Generating more library components

- Generate a component my-comp.component in my-comp folder in lib
  - ng generate component myComp --project=fooLib
  - Angular CLI added it to the declarations array in library module file
  - We must manually add new components to the exports array, to make the element visible (unless it’s an internal-use component)
  - Also add it to the public-api.ts file to make the class visible
- Could copy in from another project
  - Remember to import any required dependencies into library module
- Rebuild the library
  - ng build fooLib
- Or just watch the library
  - ng build fooLib --watch

Caution: many package.json files

- There are likely to be several package.json files
- Root package.json
  - This is the main package.json file for the library
  - Lists dependencies for the library (and any harness application)
  - Any package needed to build the application or the project must be in here
- Library project package.json
  - Tells ng-packagr what goes in the distribution package.json for the library
- Library distribution package.json
  - Generated by ng-packagr when we build the library
  - It is the package.json that gets released with our library
  - Developers who use this library will use npm install which will look at this package.json to determine what other dependencies need to be installed
  - Should not edit this file directly

Automation Scripts for Library

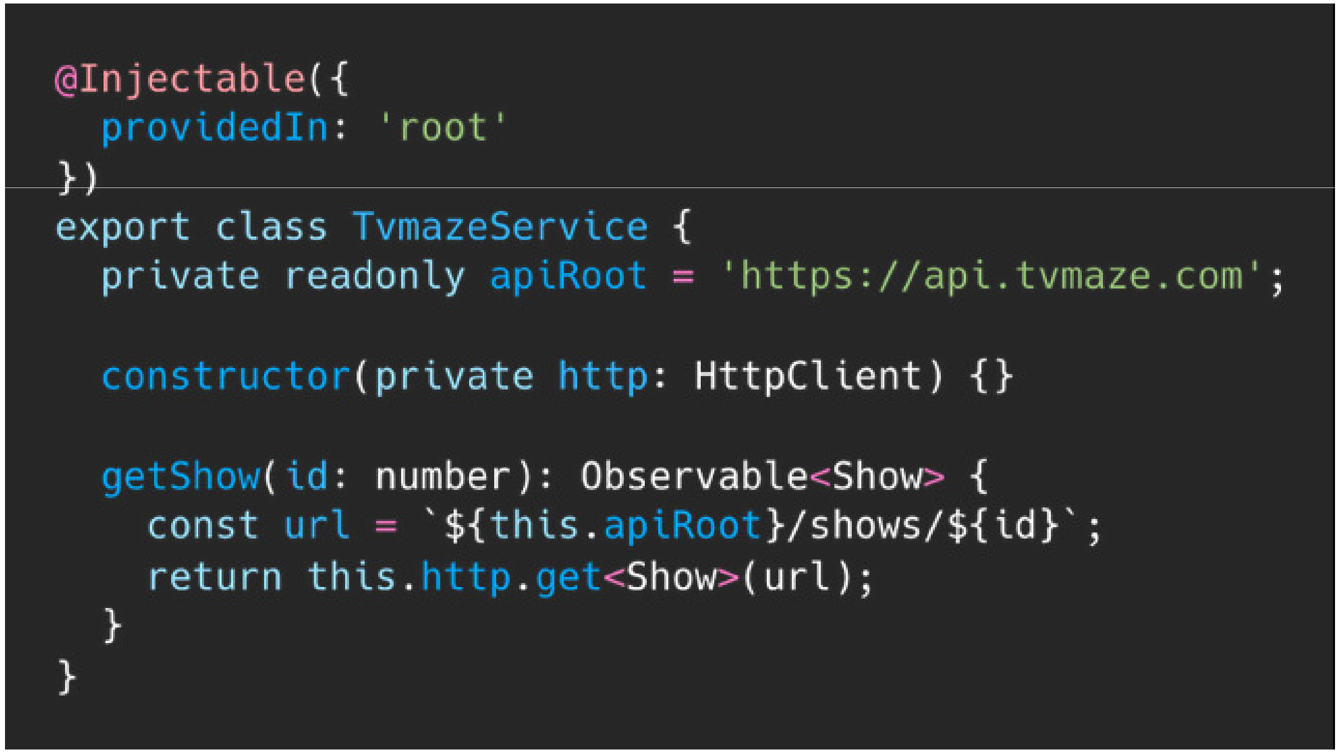
- Consider adding scripts to root package.json
  - "build_lib": "ng build fooLib",
  - "npm_pack": "cd dist/foo-lib && npm pack",
  - "package": "npm run build_lib && npm run npm_pack"
- Then we can generate fooLib-0.0.1.tgz
  - npm run package
  - Zip file is generated in root of fooLib folder
- Note: any application in same project as library can use the library since its path will be declared in tsconfig.json for the project

Using Library in another project

- Always install the library’s .tgz package and not the directory
  - This is in the dist/foo-lib/ folder (dist/ is a sibling of projects/ folder)
- Import library into a project
  - npm install ..\fooLib\dist\foo-lib\f*.tgz
  - The library is added to package.json of the project using file: url
  - Important: we no longer need the path to be specified in tsconfig
- Import into app.module as usual
  - Unless providedIn:root was specified for service inside a library
- Updates to library
  - Use semantic versioning
  - Re-generate library
  - npm install it into projects

providedIn:root (for library services)

- Provide a service without explicitly registering it in any NgModule
  - Since Angular 6
  - Enables service to be tree-shaken

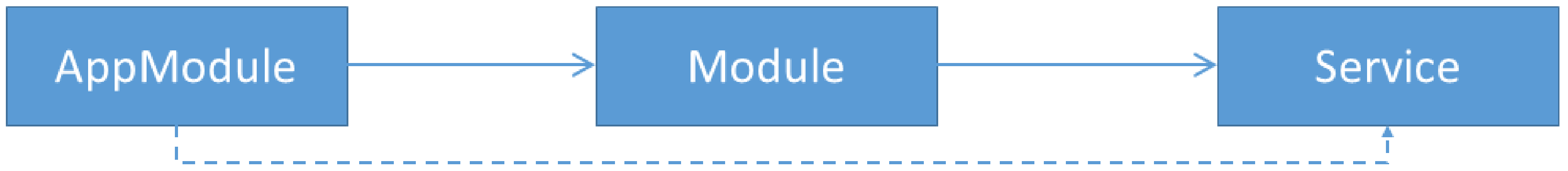


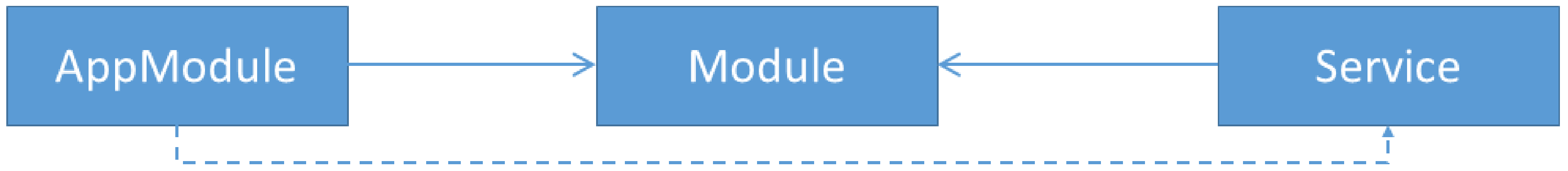
```
@Injectable({
  providedIn: 'root'
})
export class TvmazeService {
  private readonly apiRoot = 'https://api.tvmaze.com';

  constructor(private http: HttpClient) {}

  getShow(id: number): Observable<Show> {
    const url = `${this.apiRoot}/shows/${id}`;
    return this.http.get<Show>(url);
  }
}
```

Tree-Shakeable
– turn the dependency arrow around





Metadata for library

- ngc will report syntax errors immediately if you set the strictMetadataEmit option in tsconfig

```
"angularCompilerOptions": {
  ...
  "strictMetadataEmit" : true
}
```

- Angular libraries have this option by default
  - Ensures all Angular .metadata.json files are clean
- It is a best practice to do the same when building your own libraries
  - https://angular.io/guide/aot-compiler

Schematics

- Schematics are generators i.e. they are functions
- They are descriptive i.e. they only describe what they wish to achieve e.g. add a module
  - Schematics are not a way to access the filesystem and modify the files directly
- https://blog.angular.io/schematics-an-introduction-dc1dfbc2a2b2
- https://itnext.io/building-better-angular-libraries-part-1-217b3af7a3a1

Layout with ngContainer

framework training
business value through education

NgContainer

- Reduce number of nested DOM elements
  - Instead of nesting elements, use ng-container

```
<div *ngIf="lessons">
    <div class="lesson" *ngFor="let lesson of lessons">
        <div class="lesson-detail">
            {{lesson | json}}
        </div>
    </div>
</div>
```

```
<ng-container *ngIf="lessons">
    <div class="lesson" *ngFor="let lesson of lessons">
        <div class="lesson-detail">
            {{lesson | json}}
        </div>
    </div>
</ng-container>
```

https://blog.angular-university.io/angular-ng-template-ng-container-ngtemplateoutlet/

OnPush

framework training
business value through education

OnPush

- Note: the term ‘DOM update’ is actually interpolation update
    - For the DOM <span>some {{name}}</span> only the {{name}} part will be rendered during checks

Internationalization

framework training
business value through education

Internationalization and Localization

- https://angular.io/guide/i18n
- Angular follows the Unicode LDML convention
  - uses stable identifiers based on BCP47
  - Follow this convention since Angular i18n tools use locale id to find correct locale data

Template translations

- Configure the app for Internationalization and Localization via the cli, then
  - Mark static text in component templates for translation
    e.g. <h1 i18n>Hello</h1>
  - Create a translation file with the Angular CLI xi18n command
    - Extracts marked text into translation source file
  - Edit the generated translation file, translating extracted text
  - Merge the completed translation file into the app with the Angular CLI build command, choosing a locale-specific configuration
  - This generates a new version of the app in the target language
  - Build and deploy a separate version of the app for each supported language

Help translator with description and meaning

- Add a description of the text message as the value of the i18n attribute
  - <h1 i18n=“greeting">Hello</h1>
- Add context with a meaning separated from the description
  - <h1 i18n="site header|greeting">Hello</h1>
- All text message that have the same meaning will have the same translation
- A text message associated with different meanings can have different translations
  - Text messages with different descriptions but the same meaning are extracted only once

Default and Custom id

- Set a custom id for persistence and maintenance
- By default the tool assigns each translation unit a unique id
- If you change the translatable text the extractor tool generates a new id for that translation unit
  - E.g. <trans-unit id="ba0cc104d3d69bf669f97b8d96a4c5d8d9559aa3" datatype="html">
  - You must update the translation file with the new id
- Alternatively, specify a unique custom id with the prefix @@
  - <h1 i18n="@@greeting">Hello</h1>
- The tools then generate a translation unit using this persistent custom id
  - <trans-unit id=“greeting" datatype="html">

Attribute Translations

- To mark an attribute for translation
  - Write the attribute in the form i18n-x
  - E.g. <img [src]="logo" i18n-title title=“logo image" />
  - Assign a meaning, description, and id
    - i18n-x= "meaning|description@@customid"

Plurals

- Use regex to determine plural translation
-
```
<ng-container i18n>
        Updated {
                        minutes, plural,
                        =0 {just now}
                        =1 {one minute ago}
                        other {{{minutes}} minutes ago}
                    }
</ng-container >
```

Create a translation source file

- Use the Angular CLI to extract marked text into a translation source file
  - ng xi18n
  - This creates a file named messages.xlf in src folder
- Supply command options to change the format, name, location, and source locale of the extracted file
  - ng xi18n --output-path src/locale

Translate the source text

- Create a locale folder under src to store internationalization files
- Create the translation files for a locale
- Copy the generated messages.xlf into locale folder
- Rename it to messages.fr.xlf (for French)
- Give the messages.fr.xlf file to a translator to enter the translations

Merge the translated files into the app

- Compile the app with the completed translation file
- Provide the compiler with the translation-specific info
  - Which translation file to use
  - The translation file format
  - The locale (such as fr)
- Currently this is different if you compile with JIT or AOT

Compiling with AOT

- To internationalize with AOT you must pre-build a separate application package for each language
  - Serve the appropriate package based on server-side language detection or url parameters
- To tell the AOT compiler to use your translation configuration set "i18n" build configuration options in angular.json
  - i18nFile: path to translation file
  - i18nFormat: format of the translation file
  - i18nLocale: locale id
- Direct the output to a locale-specific folder by setting the outputPath configuration option

Compiling with JIT

- To support translation with the JIT compiler
  - Import the language translation file as a string constant
  - Create corresponding translation providers for the JIT compiler
  - Bootstrap the app with those providers
- Three providers tell the JIT compiler how to translate the template while compiling the app
  - TRANSLATIONS is a string containing the content of the translation file
  - TRANSLATIONS_FORMAT is the format of the file
  - LOCALE_ID is the locale of the target language
- The Angular bootstrapModule method has a second compilerOptions parameter
  - Use it to specify the translation providers

Missing Translations

- If a translation is missing the build succeeds but generates a warning
  - E.g. Missing translation for message "foo“
- Configure the level of warning that is generated by the Angular compiler
  - Error: throw an error
    - For AOT compilation the build will fail
    - For JIT compilation the app will fail to load
  - Warning (default)
    - Show a 'Missing translation' warning in the console
  - Ignore
    - do nothing

Testing

framework training
business value through education

Aim for

- More isolated testing
  - Test the services and components not their dependencies
- Strongly typed template checking
  - If a child component definition changes, ensure dependent unit tests also break
- Less coupling between components and services
  - Dependency Injection makes it easy to inject services
  - But if those services are being used widely refactoring is difficult
- Prefer dumb components over smart components
  - Dumb components that rely on inputs and outputs are much easier to test than components dependent on services

Aside: Smart or Dumb Components

- Divide components into Smart and Dumb
  - A Dumb Component works like a pure function
  - A Smart Component is more like an impure function
- Keep components as Dumb as possible
  - Data is injected via @Input(), not dependant on external services
  - Emit with @Output(), and do not produce any side effects
  - Do not mutate inputs
- Decide when a component should be Smart instead of Dumb
  - If it can be Dumb, make it Dumb
  - What cannot be Dumb, make it Smart

- https://medium.com/@jtomaszewski/how-to-write-good-composable-and-pure-components-in-angular-2-1756945c0f5b

Testing Services

- Prefer spies as they are usually the easiest way to mock services
- https://angular.io/guide/testing#service-tests

- Testing HttpClient Requests in Angular
- https://alligator.io/angular/testing-httpclient/

- Unit testing Angular service with HttpClient
  - Includes routing tests and login test
  - Combination test@example.com, testpassword is correct and any other combination is wrong
  - https://skryvets.com/blog/2018/02/18/unit-testing-angular-service-with-httpclient/

Mocking Child Components

- Options are: (least recommended to most recommended):
  - Add the child component to declarations array
    - Test is not isolated: Changes to the child component shouldn’t require us to modify the parent component unit tests
  - Use NO_ERRORS_SCHEMA to ignore the child component
    - Test doesn’t alert us to broken templates, even simple component instance typos
  - Manually mock/stub the child component
    - Tests become verbose, and requires changes to the stub for every change to component
  - **Use ngMocks to automatically mock the child component**
    - npm install ng-mocks --save-dev
    - Produces a strongly typed mock for components

- https://medium.com/@abdul_74410/towards-better-testing-in-angular-part-1-mocking-child-components-b51e1fd571da

Exploring Transclusion

framework training
business value through education

Content Projection

- Content Projection is a very important concept in Angular
- Developers can build reusable components and make applications more scalable and flexible
- https://angular-2-training-book.rangle.io/handout/components/projection.html

Template-Ref

- Present thinking prefers template-refs over ng-content for transclusion
  - Life-cycle events fire when template is used
  - Ng-content is created in different life-cycle and may not be destroyed as expected
- https://medium.com/@unrealprogrammer/angulars-content-projection-trap-and-why-you-should-consider-using-template-outlet-instead-cc3c4cad87c9

Performance

framework training
business value through education

Compiling AOT or JIT

- The Ahead-of-Time (AOT) compiler (documentation)
  - Important info about controlling compilation
- https://angular.io/guide/aot-compiler#phase-3-binding-expression-validation

shareReplay

- Use shareReplay when there are taxing computations that you do not wish to be executed across multiple subscribers
- Also where there wil be late subscribers to a stream that need access to previously emitted values
- This ability to replay values on subscription differentiates share from shareReplay
- http://jiodev.com/angular/learn-rxjs/shareReplay

The @Attribute Decorator

- The @Attribute decorator returns the value of the specified attribute from the host
- Angular will evaluate it once and forget about it
- As a general rule use the @Attribute() approach when the string is a fixed value that never changes

Summary

- Architecture matters
- Agree coding style
- Look for optimal strategies
- Angular will evolve rapidly

Благодаря!

We hope you enjoyed your training course
and will find it useful in the future.

Please could you complete your feedback form before you leave.

Many thanks from Toby and everyone at Framework Training

framework training
business value through education