View Techniques

1. Pipes
2. Asynchronous data
3. Forms

- Custom pipes

```
Demo app:    AngularDev/Demos/04-ViewTechniques/DemoApp
To install:  npm install
To run:      ng serve
```

Section 1: Pipes

- Overview
- Simple pipes
- Simple pipes example
- Additional pipes
- Additional pipes examples

Overview

- Pipes can format output in an HTML template

```
{{ expression | pipeName : param1 : param2 : ... paramN }}
```

- Examples of some simple built-in pipes in Angular:

```
{{ myfullname | uppercase }}
{{ mysalary   | currency }}
{{ coffeetime | date : 'hh:mm:ss a' }}
```

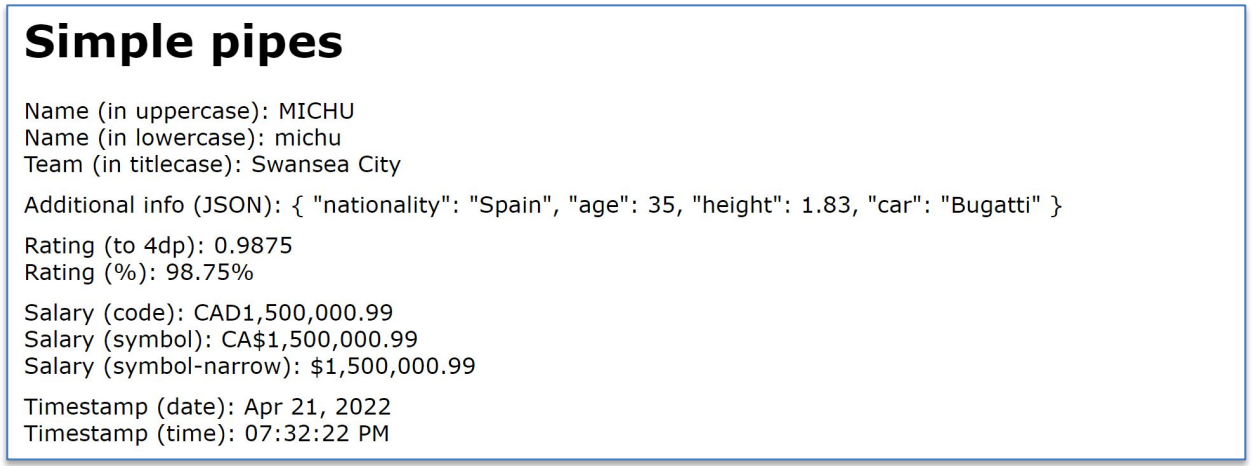
- There are 13 built-in pipes, as we'll see...
  - Each pipe is implemented as a class
  - For details, see https://angular.io/api/common#pipes

Simple Pipes

Pipe name	Pipe class	Description
`lowercase`	`LowerCasePipe`	Transforms text to lower-case
`uppercase`	`UpperCasePipe`	Transforms text to upper-case
`titlecase`	`TitleCasePipe`	Transforms text to title-case
`json`	`JsonPipe`	Converts a JavaScript object into a JSON string
`number`	`DecimalPipe`	Formats number as text
`percent`	`PercentPipe`	Formats number as a percentage
`currency`	`CurrencyPipe`	Formats number as a currency
`date`	`DatePipe`	Formats a date as a string
`keyvalue`	`KeyValuePipe`	Transforms an object or a map into an array of key-value pairs

Simple Pipes Example

- Let's see an example of simple pipes
  - In the demo app, click the **Simple pipes** link



  - See `simple-pipes.component.html`

Additional Pipes

Pipe name	Pipe class	Description
`slice`	`SlicePipe`	Creates a new List or String containing a subset (slice) of the elements
`i18nPlural`	`I18nPluralPipe`	Maps a numeric value into suitable plural string, e.g. 'mouse' or 'mice', 'child' or 'children', etc.
`i18nSelect`	`I18nSelectPipe`	Formats a string value into a suitably selected string, e.g. 'Dear Sir', 'Dear Madam', etc.
`async`	`AsyncPipe`	Accepts a Promise or Observable as input, and updates the view with the appropriate value(s) when the promise is resolved or the observable emits a new value.

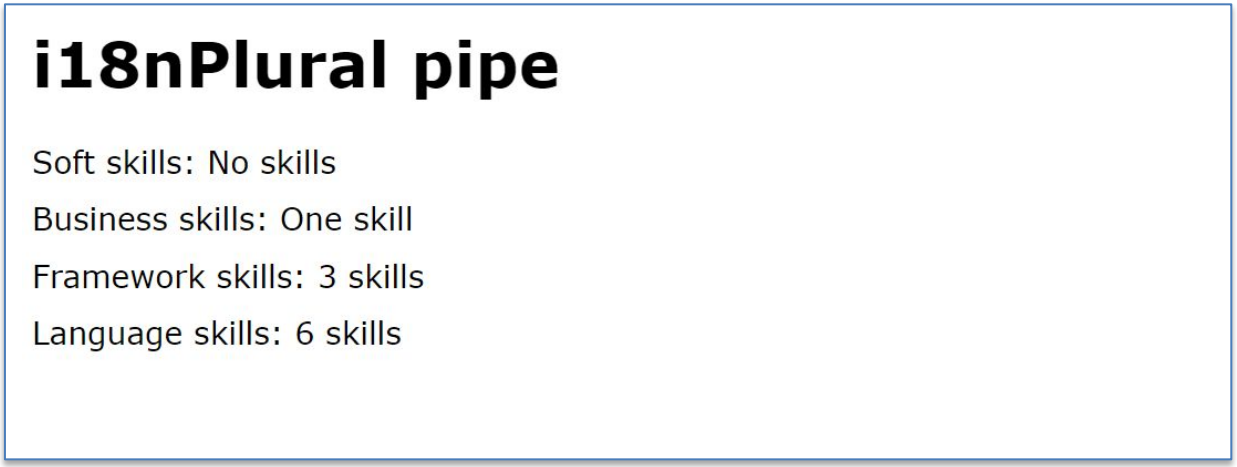
Additional Pipes Examples (1 of 3)

- Let's first see an example of the slice pipe
  - In the demo, click **Slice pipe**

  - See `slice-pipe.component.html`

Additional Pipes Examples (2 of 3)

- Now let's see an example of the `i18nPlural` pipe
  - In the demo, click **i18nPlural pipe**



  - See `i18n-plural-pipe.component.html`

Additional Pipes Examples (3 of 3)

- Now let's see an example of the `i18nSelect` pipe
  - In the demo, click **i18nSelect pipe**

  - See `i18n-select-pipe.component.html`

Section 2: Asynchronous Data

- Overview
- Promises
- Observables
- How to display asynchronous data
- Asynchronous data example

Overview

- Web apps often have to deal with asynchronous data
  - E.g. the result of a long calculation in a Web Worker
  - E.g. the result of a REST service
  - E.g. incoming data messages from a Web Socket

- There are 2 ways to handle async data in a web app:
  - `Promise` - One-off asynchronous result
  - `Observable` - Stream of data from some source

Promises

- `Promise` is a standard JavaScript class
  - Represents a one-off asynchronous result

```
var aPromise = new Promise( (resolve, reject) => {
  … … …
  if (someTest)
    resolve(someResult);
  else
    reject(someError);
});
```

  - The `Promise` constructor takes 1 argument - a callback function with two parameters (`resolve`, `reject`)
  - Do some async work in the callback, then call either `resolve()` or `reject()`

Observables (1 of 2)

- `Observable` is an RxJs (Reactive Ext^{n}s for JS) class
  - Represents an observable stream of data from a source

- You need the `rxjs` dependency in `package.json`
  - Note, Angular CLI adds this automatically

```
"dependencies": {
  "rxjs": "~7.5.0",
  …
}
```
package.json

For details about the `Observable` class, see:
http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html

Observables (2 of 2)

- `Observable` example

```
import {interval} from 'rxjs';
import {map} from 'rxjs/operators';

let anObservable = interval(3000).pipe( map(() => new Date()) );
```

- `interval()`
  - Factory function, creates an `Observable` object that emits a sequential number every time interval

- `pipe()`
  - Pipes the result into the `map()` operator function

How to Display Asynchronous Data

- If a component has an asynchronous data property, you can display it via the `async` pipe

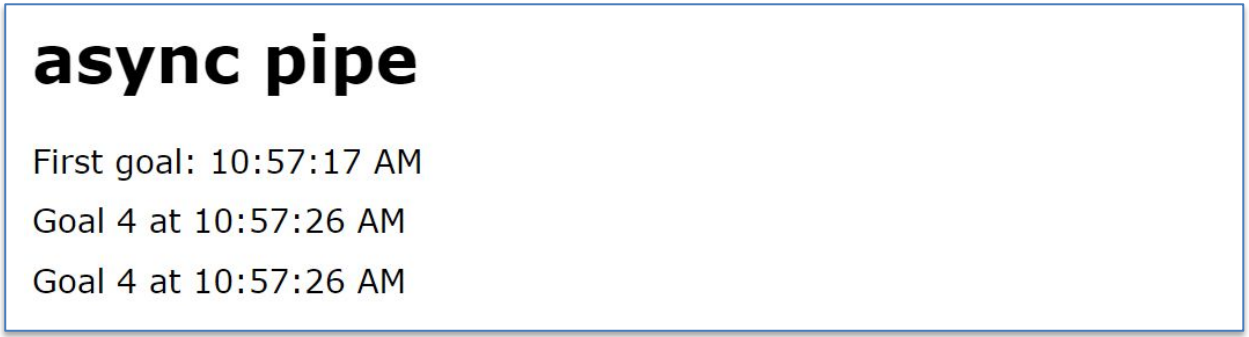
```
{{ aPromise | async }}

{{ anObservable | async }}
```

- What does the `async` pipe do?
  - Waits for a `Promise` object to be resolved
  - Subscribes to data published by an `Observable` object

Asynchronous Data Example (1 of 3)

- Let's see an example of asynchronous data
  - In the demo app, click the **async pipe** link



Asynchronous Data Example (2 of 3)

- Here are the relevant bits of the TypeScript code

```
import {Observable, interval} from 'rxjs';
import {map} from 'rxjs/operators';
…
@Component(…)
export class AsyncPipeComponent {

  firstGoal:      Promise<Date>;
  mostRecentGoal: Observable<Goal>;

 constructor() {
    this.firstGoal = new Promise((resolve, reject) => {
      setTimeout(() => resolve(new Date()), 3000);
    });
    this.mostRecentGoal = interval(3000).
                          pipe(map(n => new Goal(n+1, new Date())));
  }
}
```

async-pipe.component.ts

Asynchronous Data Example (3 of 3)

- Here are the relevant bits of the HTML template

```
<div>
    First goal:
    {{ firstGoal | async | date:'hh:mm:ss a' }}
</div>

<div>
    Goal {{ (mostRecentGoal | async)?.goalNumber }}
    at   {{ (mostRecentGoal | async)?.goalTimestamp | date:'hh:mm:ss a' }}
</div>

<div *ngIf='mostRecentGoal | async as mrg'>
    Goal {{ mrg.goalNumber }}
    at   {{ mrg.goalTimestamp | date:'hh:mm:ss a' }}
</div>
```

async-pipe.component.html

Section 3: Forms

- Strategies for implementing forms
- Adding support for forms
- Examples

Strategies for Implementing Forms

- Template-driven forms
  - The HTML template specifies form semantics
- Reactive forms (a.k.a. model-driven forms)
  - The component class defines form semantics in code

Adding Support for Forms

- Your app dependencies must include Angular Forms

```
"dependencies": {
  "@angular/forms": "~14.1.0",
  …
}
```
package.json

- Your app module must bundle either `FormsModule` or `ReactiveFormsModule`

```
import { FormsModule --or-- ReactiveFormsModule } from '@angular/forms';
…
@NgModule({
  imports: [FormsModule --or-- ReactiveFormsModule, …],
  …
})
export class AppModule {}
```
app.module.ts

Pearson

Examples

- The demo app has examples of the forms strategies...

- Click the appropriate link in the demo app:
  - **Template-driven forms**
  - **Reactive forms**

Summary

- Pipes
- Asynchronous data
- Forms

Annex: Custom Pipes

- Overview
- Generating a custom pipe
- Implementing a custom pipe
- Bundling a custom pipe
- Using a custom pipe
- Running the demo application

Overview

- You can define custom pipes
  - Define a class and decorate with `@Pipe`
  - Implement the `PipeTransform` interface

```
import {Pipe, PipeTransform} from '@angular/core';
…

@Pipe({name: 'nameOfMyPipe'})
class MyPipeClass implements PipeTransform {

  transform(value: any, ...args: any[]): string {
    // Add code here, to transform the value and return the result.
    …
  }
}
```

Generating a Custom Pipe

- You can use Angular CLI to generate a custom pipe
- For example, this is how we generated a custom pipe in the demo application:

```
ng g pipe custom-pipes
```

```
CREATE src/app/flexi-title-case.pipe.spec.ts (221 bytes)
CREATE src/app/flexi-title-case.pipe.ts (233 bytes)
UPDATE src/app/app.module.ts (1222 bytes)
```

Implementing a Custom Pipe

- Here's the implementation of our custom pipe class:

```
@Pipe({name: 'flexititlecase'})
export class FlexiTitleCasePipe implements PipeTransform {

  transform(value: string, upperOrLower: string): string {
    if (upperOrLower !== 'upper' && upperOrLower !== 'lower')
      return value;

    let words:string[] = value.split(' ');
    for (var i = 0; i < words.length; i++)
      if (upperOrLower === 'upper')
        words[i] = words[i].charAt(0).toUpperCase() +
                   words[i].slice(1).toLowerCase();
      else
        words[i] = words[i].charAt(0).toLowerCase() +
                   words[i].slice(1).toUpperCase();
    return words.join(' ');
  }
}
```

flexi-title-case.pipe.ts

Bundling a Custom Pipe

- You must bundle custom pipes into your app module
  - Angular CLI does this for you automatically

```
import { FlexiTitleCasePipe } from './flexi-title-case.pipe';
…

@NgModule({
  declarations: [FlexiTitleCasePipe, … ],
  imports: [BrowserModule, AppRoutingModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.module.ts

Using a Custom Pipe

- You can use the custom pipe in the HTML template for a component, as follows:

custom-pipes.component.html

```
<h1>Custom pipes</h1>

<div>
    Name: {{ name }}
</div>

<div>
    Name (flexititlecase:'upper'): {{ name | flexititlecase:'upper' }}
</div>

<div>
    Name (flexititlecase:'lower'): {{ name | flexititlecase:'lower' }}
</div>
```

Running the Demo Application

- Let's see the custom pipe in action
  - In the demo app, click the **Custom pipes** link