Angular Advanced Reusable Components

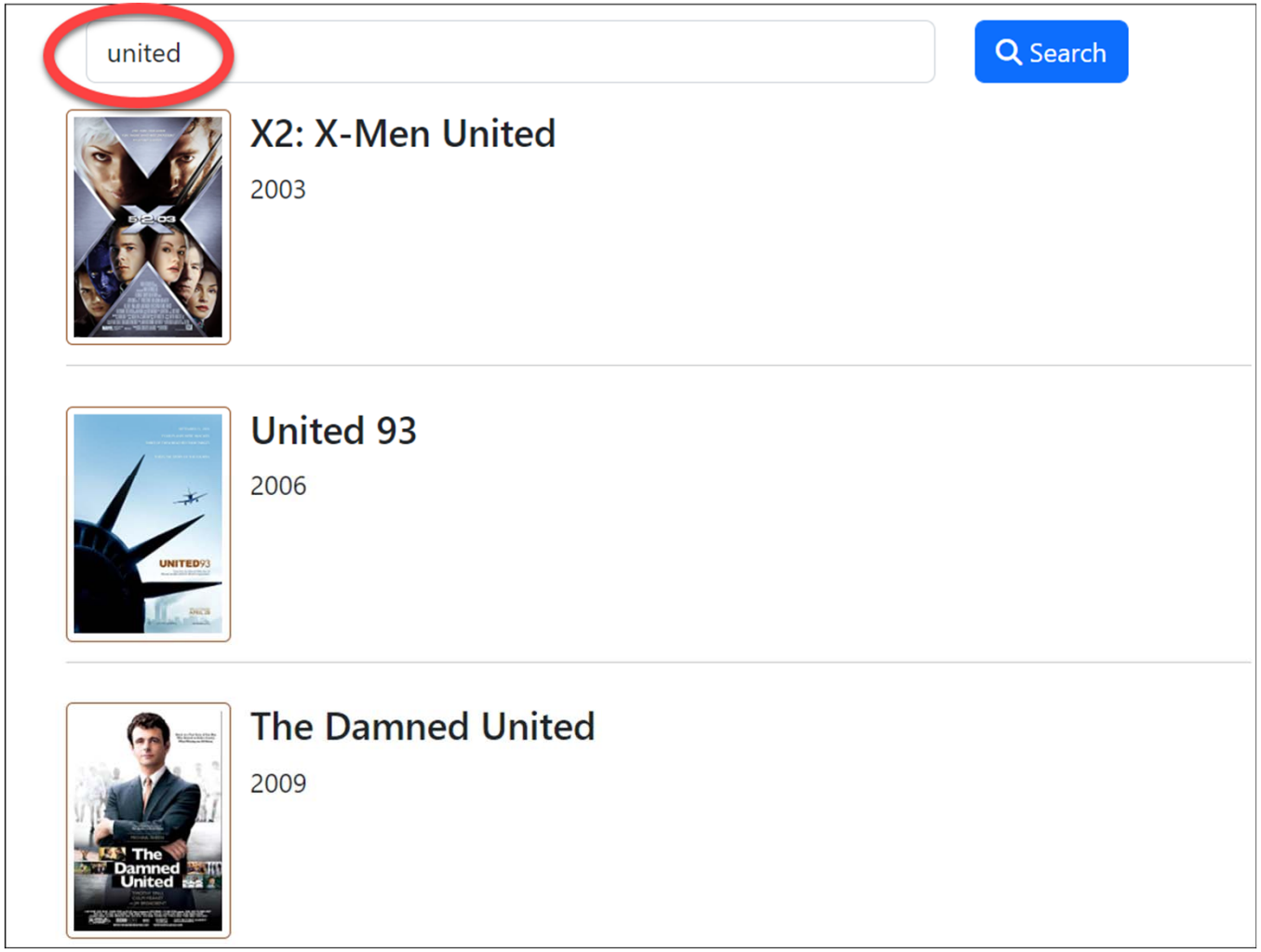
Peter Kassenaar –
info@kassenaar.com

What are Reusable Components?

Define a generic component, specific templates, and load on demand

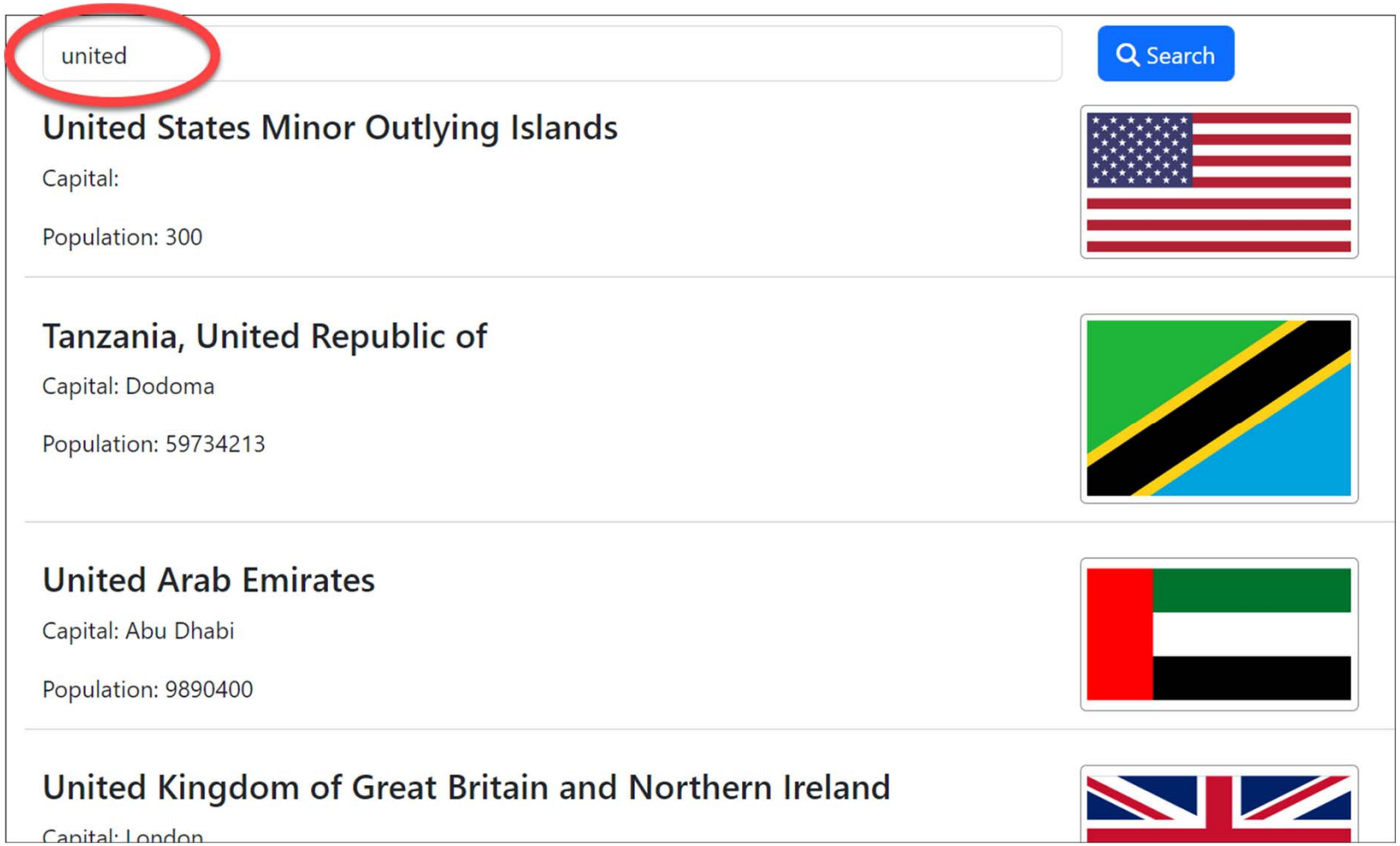
What is the problem?

Let's say we have a `ListComponent`, displaying a search result voor movies:



…

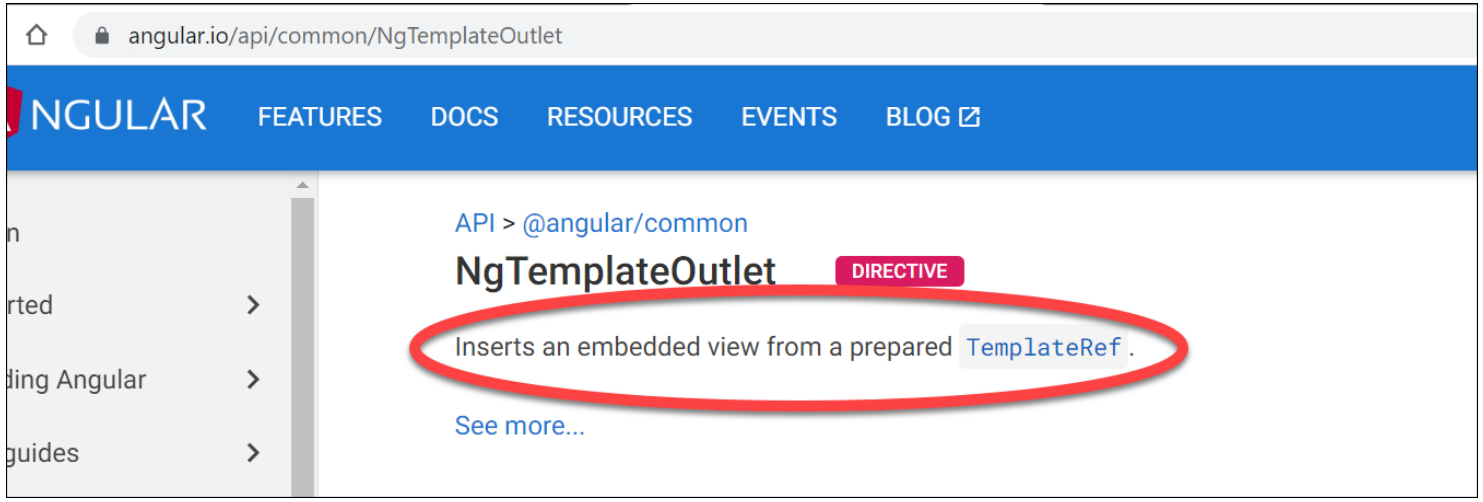
But we want to be able to present other search results, with – possibly – other formatting. For instance, countries:



We can then…

- Modify the existing component with a bunch of if-else clauses to address various usages
- Create a new list by duplicating the code and updating where necessary
- Both of which are overcomplicated and don't scale well

`ngTemplateOutlet` to the rescue!



What is ngTemplateOutlet?

"Instead of having a selection list that explicitly defines the component of its items, we use an `ng-container` as a placeholder. The placeholder has an `ngTemplateOutlet` that receives the reference of a template, `itemTemplateRef`"

Creating a GenericListComponent

```
<div *ngFor="let item of items">
  … Generic items for every list…
  <ng-container
    [ngTemplateOutlet]="itemTemplateRef"
    [ngTemplateOutletContext]="{ $implicit: item }"
  ></ng-container>
</div>
```

```
export class GenericListComponent {

  // the items to display are coming from the outside and
  // can be of any kind, or they can be null.
  @Input() items!: unknown[] | null;

  // Refence to the template passed in into GenericList
  @ContentChild('itemTemplate', {static: false})
  itemTemplateRef!: TemplateRef<unknown>;
}
```

Using `ngTemplateOutletContext`

In the previous code we also use `ngTemplateOutletContext`. We can then pass context to this template. In this case, we pass the context of each `item`, so that the template is aware of which item it displays each time.

We can then run code for that specific item, instead of for the list as a whole

Defining the template(s)

- Next, define the template that replaces the placeholder.
- We create a `ListComponent` for movies, for countries, and so on.
- Example: `MovieListComponent`:

```
<!-- The Generic list component, it takes app-movie-template as its template-->
<app-generic-list [items]="movie$ | async">
  <ng-template #itemTemplate let-item>
    <app-movie-template [movie]="item"></app-movie-template>
  </ng-template>
</app-generic-list>
```

- This component gets a list of movies and passes them on to a single `<app-movie-template>` component

The MovieTemplateComponent

- A simple view component that just displays the passed in movie
- We create a similar CountryTemplateComponent and possibly other templates

```
@Component({
  selector: 'app-movie-template',
  templateUrl: './movie-template.component.html',
  styleUrls:['./movie-template.component.css']
})
export class MovieTemplateComponent {
  @Input() movie! : Movie;
}
```

```
<div class="row mb-2 mt-2">
  <img
    class="col-md-3 poster"
    [src]="movie.Poster"
    alt="Poster for {{ movie.Title }}" width="90">
  <div class="col-md-9">
    <h4>{{ movie.Title }}</h4>
    <p>{{movie.Year }}</p>
  </div>
  <hr>
</div>
```

Last steps

- Lastly, look again at the `itemTemplate` reference passed to the `ngTemplateOutlet` directive.
- This is done by using the `@ContentChild` decorator.
- In `GenericListComponent`:

```
// Refence to the template passed in into this GenericList
@ContentChild('itemTemplate', {static: false})
itemTemplateRef!: TemplateRef<unknown>;
```

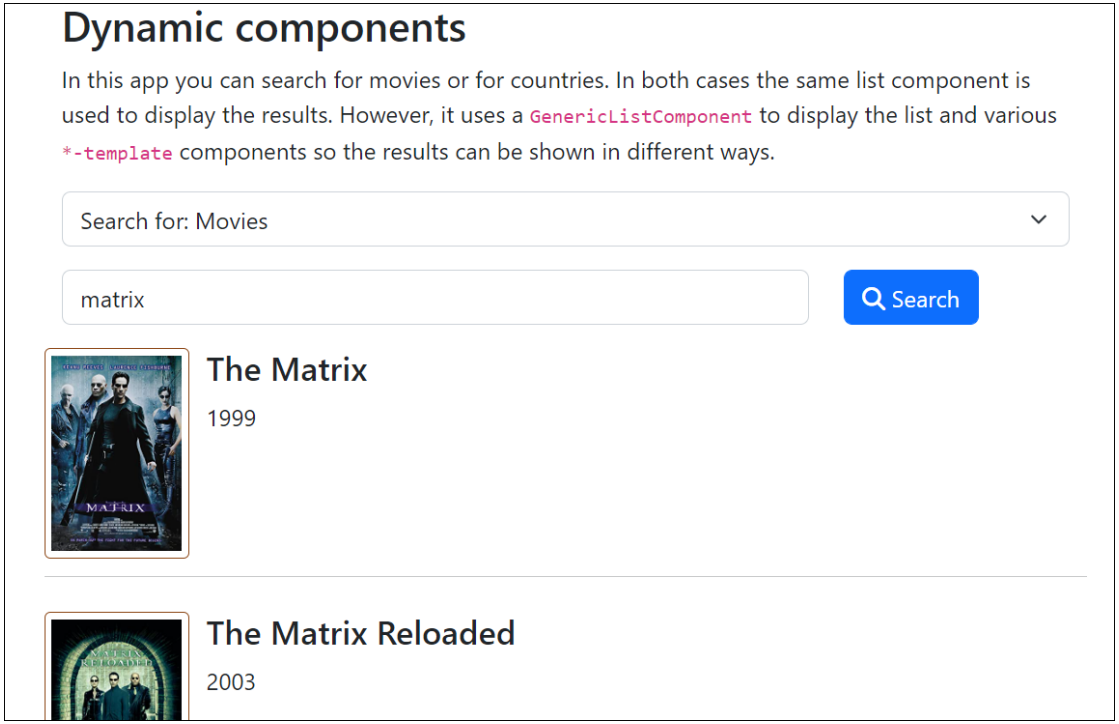
Generic-list.component.ts

```
<app-movie-list *ngIf="selected === 'movies'"></app-movie-list>
<app-country-list *ngIf="selected === 'countries'"></app-country-list>
```

app.component.html

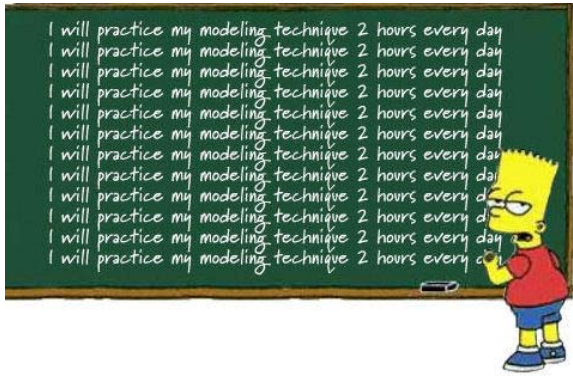
Creating additional Components

- The heavy lifting is done! We can now reuse the `GenericListComponent` to display countries, movies, or whatever we like.
- Per template we can use different properties, lay-out, and so on
- We also created a `SearchComponent` to emit a keyword to search for in the given APIs

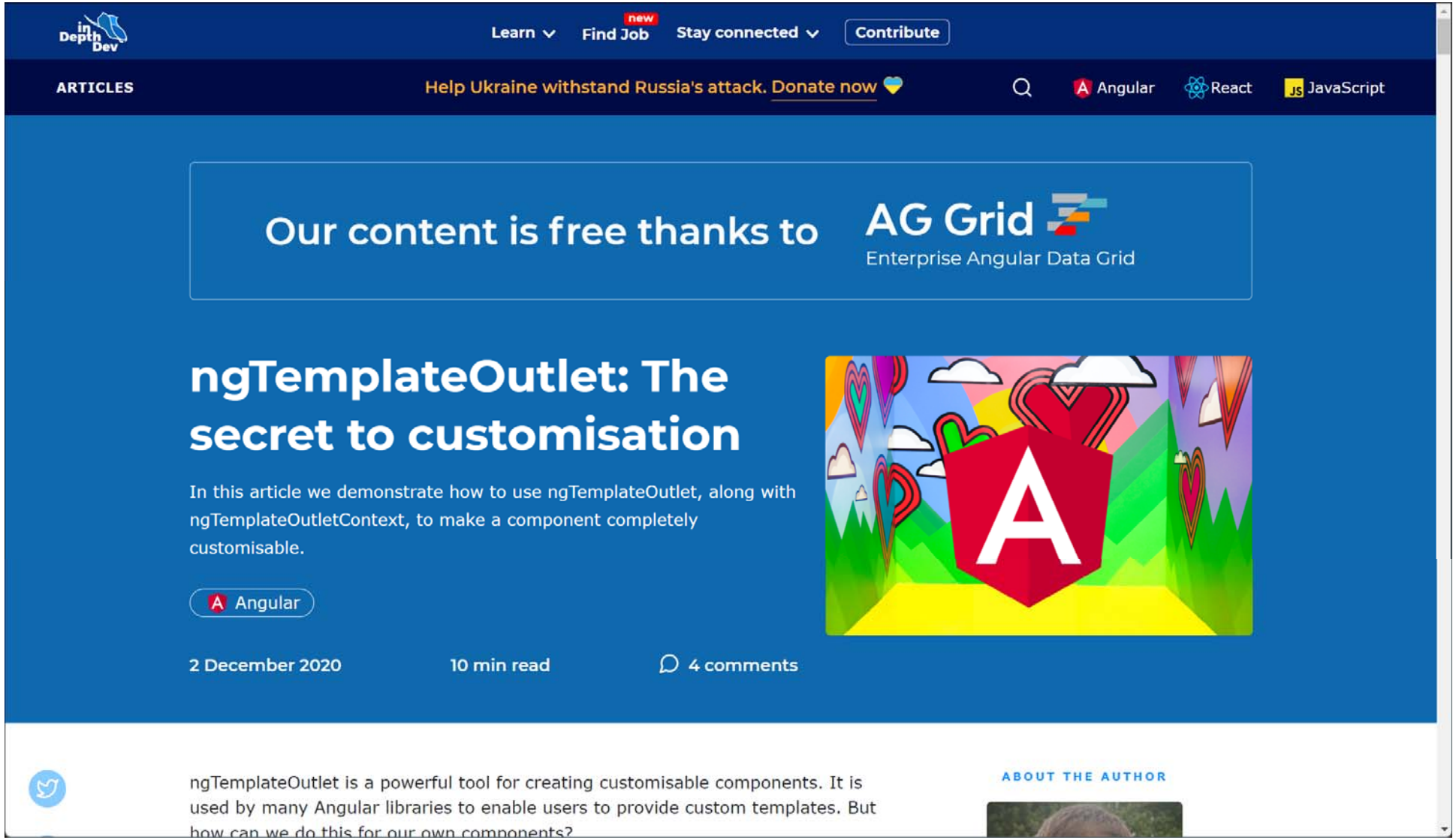


Workshop

1. Study the example code in `../783-reusable-components`
   - See how the data flows through the `GenericListComponent` to the various lists and templates
   - A module system is used to separate `Movies` from `Countries` and so on
   - The data is fetched via Services and APIs
   - In `app.component.html|ts` we switch between movies and countries.
2. Create a new module, displaying a list of random user data from https://jsonplaceholder.typicode.com/users
   - Add it to the existing example
   - Make sure we can switch between `users|countries|movies`
3. Create a new, blank application, using the reusable approach outlined in this presentation
4. Read the following blogs & documentation.

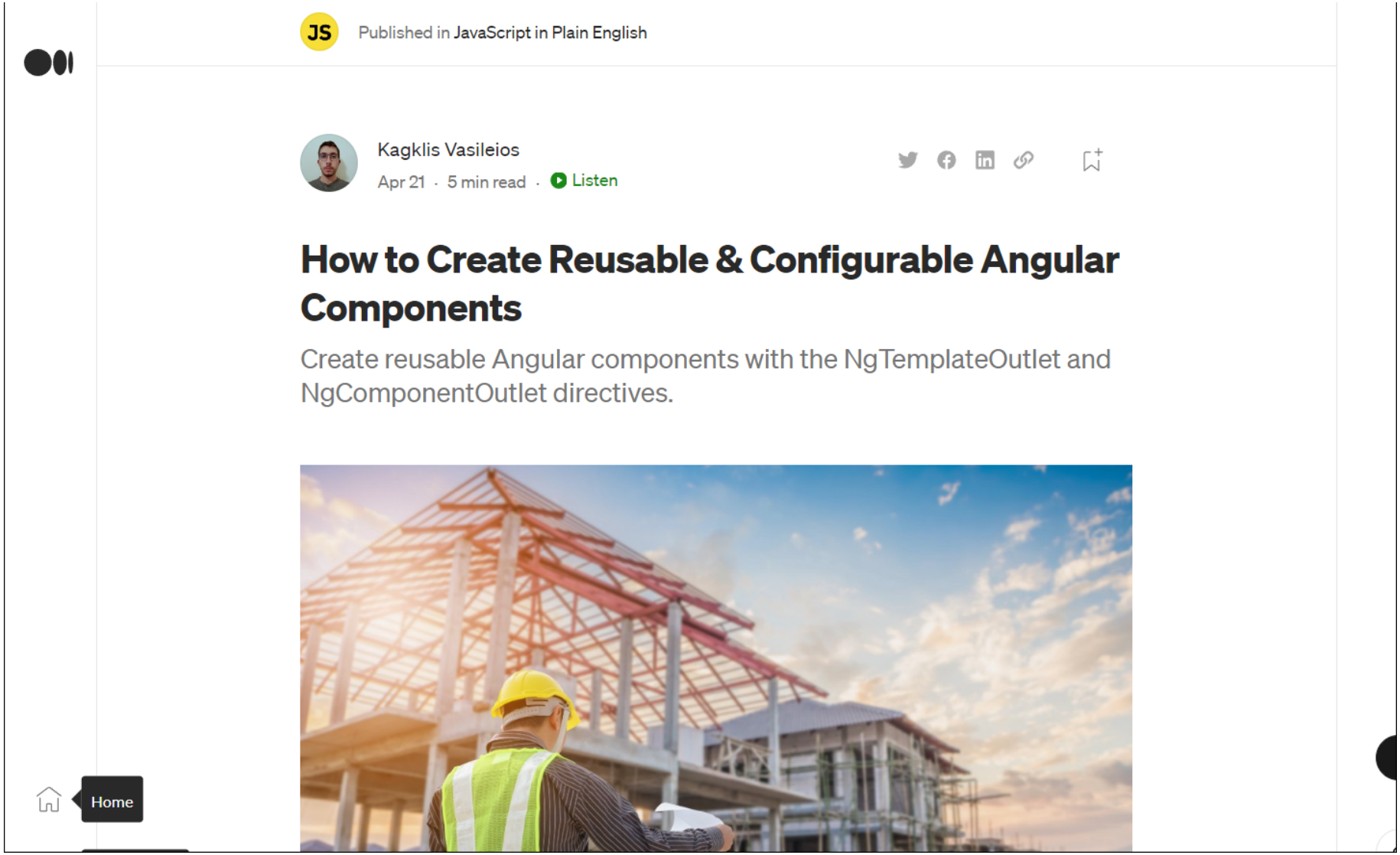


More info



https://indepth.dev/posts/1405/ngtemplateoutlet

More info



https://javascript.plainenglish.io/creating-reusable-configurable-angular-components-b7fcba2f5f38

Official Documentation

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG

Search /

- Introduction
- Getting started
- Understanding Angular
- Developer guides
- Best practices
- Angular tools
- Tutorials
- Feature preview
- Release Information
- Reference
  - Conceptual reference
  - CLI Command Reference
  - API reference
  - Error reference
  - Extended diagnostic reference
  - Example applications
  - Angular glossary
  - Angular coding style

API > @angular/common

NgTemplateOutlet

DIRECTIVE

Inserts an embedded view from a prepared `TemplateRef`.

See more...

Exported from

- `CommonModule`

Selectors

`[ngTemplateOutlet]`

Properties

Property	Description
`@Input()` `ngTemplateOutletContext: Object | null`	A context object to attach to the `EmbeddedViewRef`. This should be an object, the object's keys will be available for binding by the local template `let` declarations. Using the key `$implicit` in the context object will set its value as

- NgTemplateOutlet
  - Exported from
  - Selectors
  - Properties
  - Description
    - Example

https://angular.io/api/common/NgTemplateOutlet