



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:


Search



X2: X-Men United
2003



United 93
2006


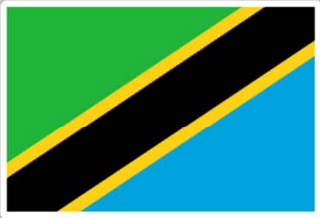




The Damned United
2009

...

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

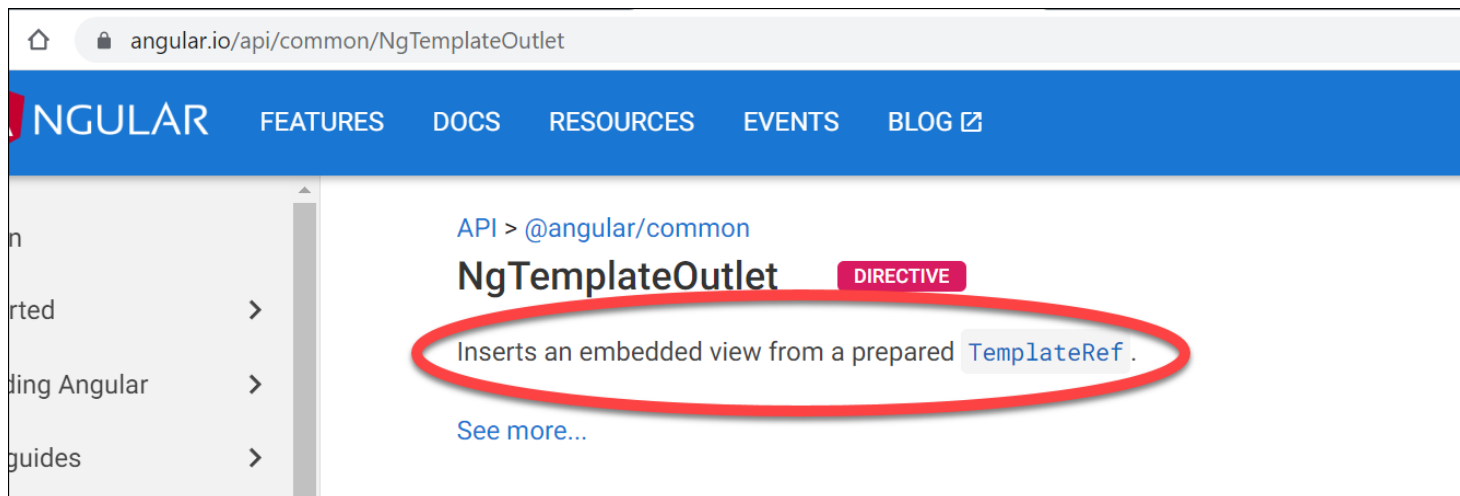
Search

United States Minor Outlying Islands Capital: Population: 300	
Tanzania, United Republic of Capital: Dodoma Population: 59734213	
United Arab Emirates Capital: Abu Dhabi Population: 9890400	
United Kingdom of Great Britain and Northern Ireland Capital: London	

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


Dynamic components

In this app you can search for movies or for countries. In both cases the same list component is used to display the results. However, it uses a `GenericListComponent` to display the list and various `*-template` components so the results can be shown in different ways.

Search for: Movies


matrix

Search



The Matrix

1999

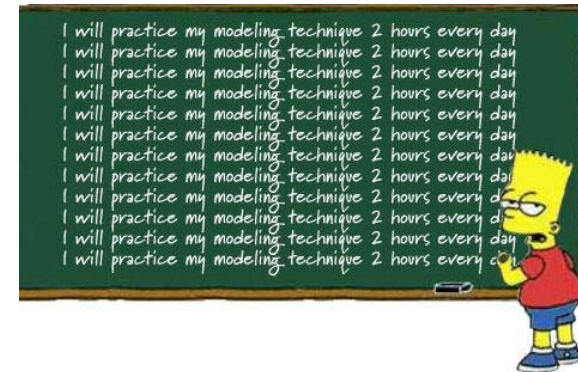


The Matrix Reloaded

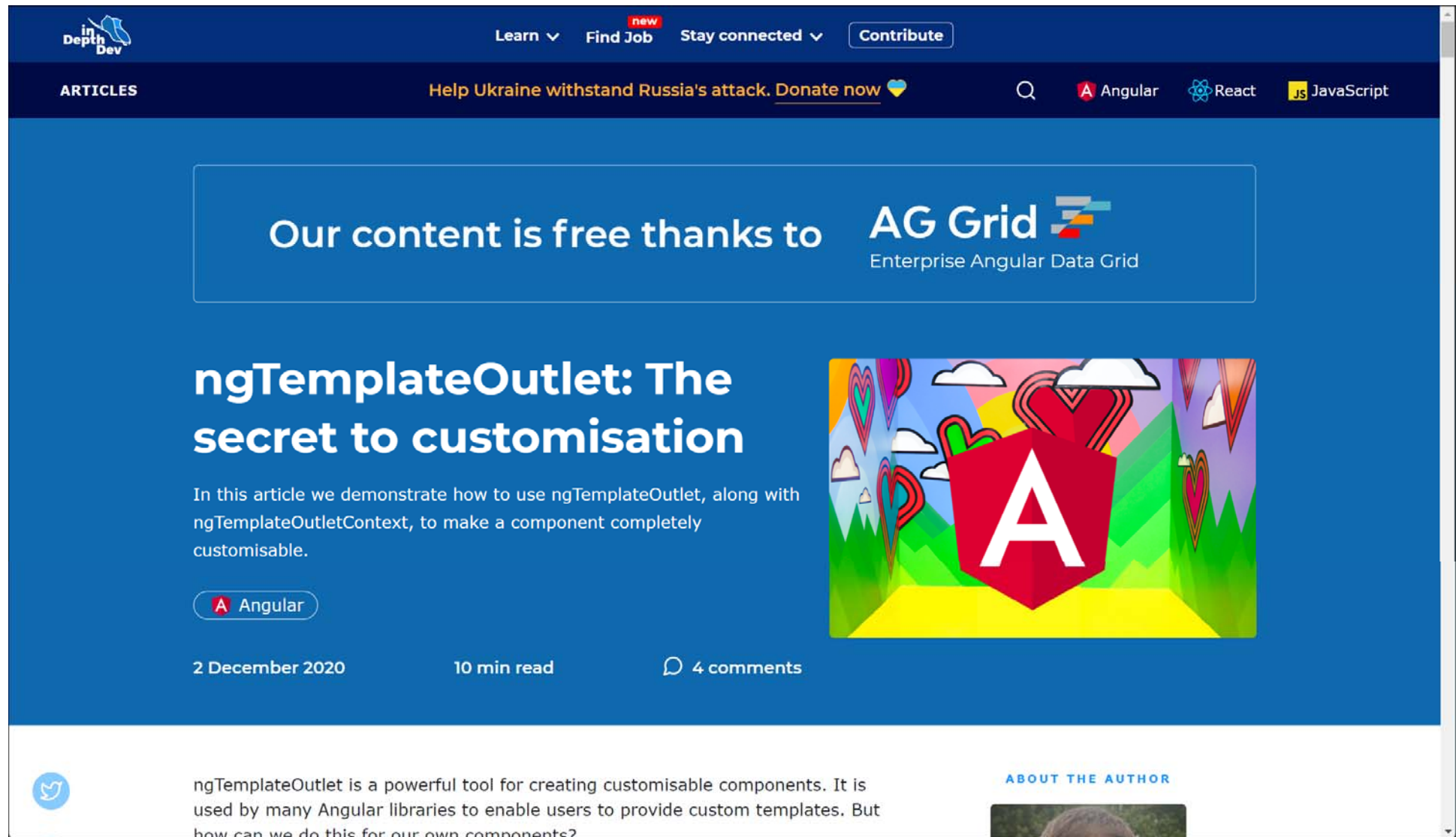
2003

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



The screenshot shows the inDepth.dev website interface. At the top, there's a dark blue header with the site logo, navigation links (Learn, Find Job, Stay connected), and a 'Contribute' button. Below the header, a secondary bar features a call to action about helping Ukraine, a search icon, and links to Angular, React, and JavaScript. The main content area has a blue background. A white-bordered box at the top of this area contains a sponsorship message for AG Grid. Below this, the article title 'ngTemplateOutlet: The secret to customisation' is displayed in large white text. A short introductory paragraph follows, explaining the article's focus on ngTemplateOutlet and ngTemplateOutletContext. To the right of the text is a colorful illustration of a red Angular logo in a stylized landscape. Below the text, there's a tag for 'Angular', the date '2 December 2020', the estimated reading time '10 min read', and a comment count '4 comments'. At the bottom, there's a Twitter icon, a snippet of the article text, and a section titled 'ABOUT THE AUTHOR' with a small profile picture.


inDepth Dev

Learn ▾ Find Job ^{new} Stay connected ▾ Contribute

ARTICLES


Help Ukraine withstand Russia's attack. [Donate now](#) 💙

🔍 Angular React JS JavaScript


Our content is free thanks to **AG Grid** 
Enterprise Angular Data Grid


ngTemplateOutlet: The secret to customisation

In this article we demonstrate how to use ngTemplateOutlet, along with ngTemplateOutletContext, to make a component completely customisable.

 Angular


2 December 2020 10 min read 4 comments

 ngTemplateOutlet is a powerful tool for creating customisable components. It is used by many Angular libraries to enable users to provide custom templates. But how can we do this for our own components?

ABOUT THE AUTHOR 


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

More info









JS

Published in JavaScript in Plain English




Kaglis Vasileios


Apr 21 · 5 min read ·  Listen



How to Create Reusable & Configurable Angular Components

Create reusable Angular components with the NgTemplateOutlet and NgComponentOutlet directives.



 Home

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

Official Documentation

The screenshot shows the Angular official documentation website. The top navigation bar is blue with the Angular logo, a search bar, and links for FEATURES, DOCS, RESOURCES, EVENTS, and BLOG. A left sidebar contains a table of contents with categories like Introduction, Getting started, Understanding Angular, Developer guides, Best practices, Angular tools, Tutorials, Feature preview, Release Information, Reference (expanded), Conceptual reference, CLI Command Reference, API reference, Error reference, Extended diagnostic reference, Example applications, Angular glossary, and Angular coding style. The main content area is titled 'API > @angular/common' and features the 'NgTemplateOutlet' directive, marked with a red 'DIRECTIVE' tag. It includes a description: 'Inserts an embedded view from a prepared TemplateRef.' and a 'See more...' link. Below this are sections for 'Exported from' (listing CommonModule), 'Selectors' (listing [ngTemplateOutlet]), and 'Properties'. The 'Properties' section contains a table with two columns: 'Property' and 'Description'.

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

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