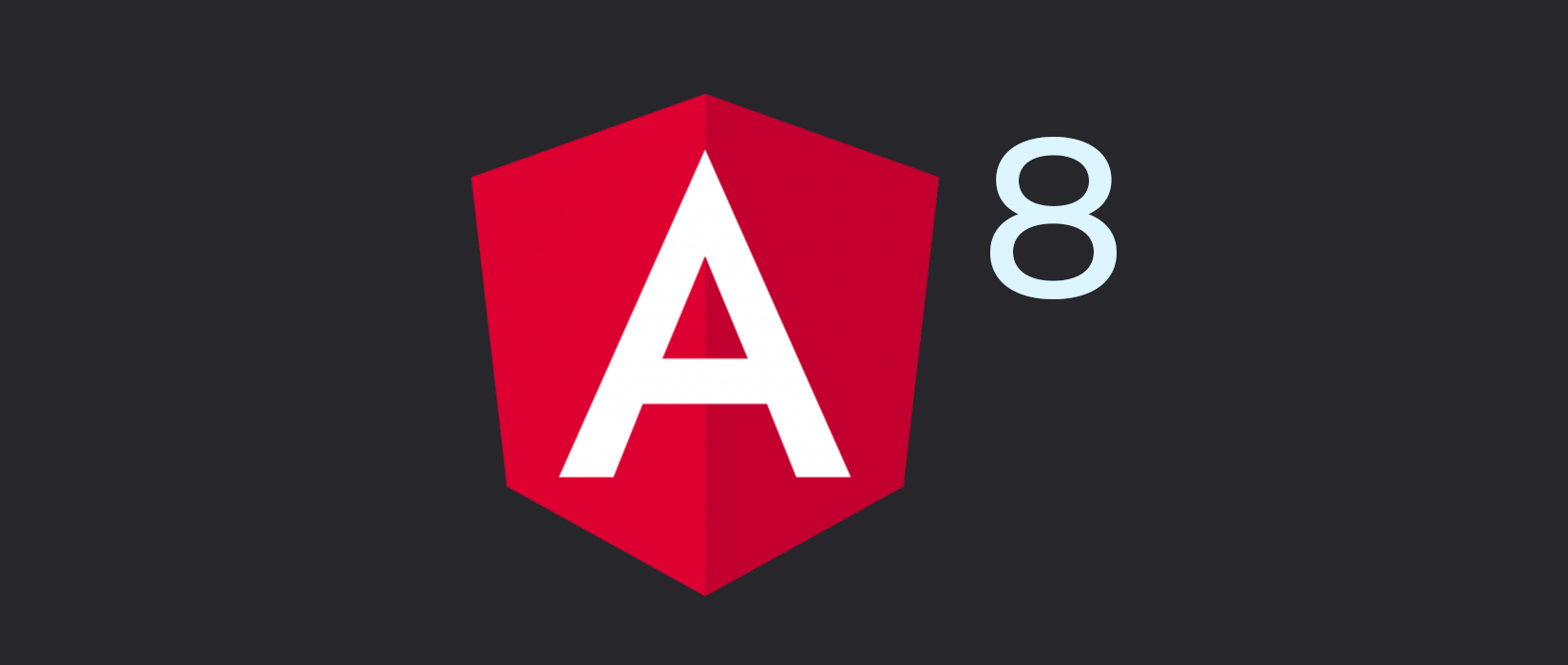
# Angular 8 Tutorial & Crash Course

ԳԱՐԻ ՍԻՄՈՆԻ ԿՈՂՄԻՑ - 14 ՀՈՒԼԻՍԻ, 2019 Թ



Անկյուն 8-ը դուրս է եկել դրա հետ միասին, գալիս է իմ անկյունային 8 ձեռնարկը (ավելի շատ նման է վթարի դասընթացին), որը սկսնակների համար հենց այն է, թե ինչպես պետք է արթնանալ և գործել այս հզոր frontend javascript շրջանակներում:

Angular- ը թույլ է տալիս ստեղծել SPA- ի (Single Page Apps), SSR- ի (Server Side Rendered) և PWA- ի (Progressive Web Apps): Այս ձեռնարկի համար մենք կենտրոնանալու ենք պարզապես SPA կառուցելու հիմունքների վրա:

Ձեզ հարկավոր է հարմարավետ լինել HTML, CSS և JavaScript հիմունքներով, նախքան Angular- ին անցնելը:

Եկեք սկսենք!

**[[](https://coursetro.com/courses/19/Learn-Angular-5-from-Scratch---Angular-5-Tutorial?utm_source=in_article&utm_campaign=article&utm_medium=Learn+Angular+5+from+Scra)](https://coursetro.com/courses/19/Learn-Angular-5-from-Scratch---Angular-5-Tutorial?utm_source=in_article&utm_campaign=article&utm_medium=Learn+Angular+5+from+Scra)**

**[Սովորեք անկյունային 5-ը քերծվածքից - անկյունային 5 ձեռնարկ](https://coursetro.com/courses/19/Learn-Angular-5-from-Scratch---Angular-5-Tutorial?utm_source=in_article&utm_campaign=article&utm_medium=Learn+Angular+5+from+Scra)**

**[[](https://coursetro.com/courses/18/Build-a-Beautiful-CryptoCurrency-App-using-Ionic-3?utm_source=in_article&utm_campaign=article&utm_medium=Build+a+Beautiful+CryptoC)](https://coursetro.com/courses/18/Build-a-Beautiful-CryptoCurrency-App-using-Ionic-3?utm_source=in_article&utm_campaign=article&utm_medium=Build+a+Beautiful+CryptoC)**

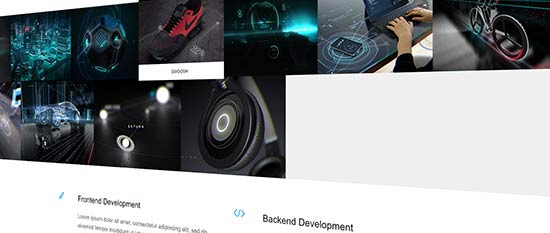
**[Կառուցեք գեղեցիկ CryptoCurrency հավելված ՝ օգտագործելով Ionic 3](https://coursetro.com/courses/18/Build-a-Beautiful-CryptoCurrency-App-using-Ionic-3?utm_source=in_article&utm_campaign=article&utm_medium=Build+a+Beautiful+CryptoC)**

**[[](https://coursetro.com/courses/13/Create-a-MEAN-App-Called-CodePost---Full-Stack?utm_source=in_article&utm_campaign=article&utm_medium=Create+a+MEAN+App+Called+)](https://coursetro.com/courses/13/Create-a-MEAN-App-Called-CodePost---Full-Stack?utm_source=in_article&utm_campaign=article&utm_medium=Create+a+MEAN+App+Called+)**

**[Ստեղծեք MEAN ծրագիր, որը կոչվում է CodePost - Full Stack](https://coursetro.com/courses/13/Create-a-MEAN-App-Called-CodePost---Full-Stack?utm_source=in_article&utm_campaign=article&utm_medium=Create+a+MEAN+App+Called+)**

**[[](https://coursetro.com/courses/12/Learn-Angular-4-from-Scratch?utm_source=in_article&utm_campaign=article&utm_medium=Learn+Angular+4+from+Scra)](https://coursetro.com/courses/12/Learn-Angular-4-from-Scratch?utm_source=in_article&utm_campaign=article&utm_medium=Learn+Angular+4+from+Scra)**

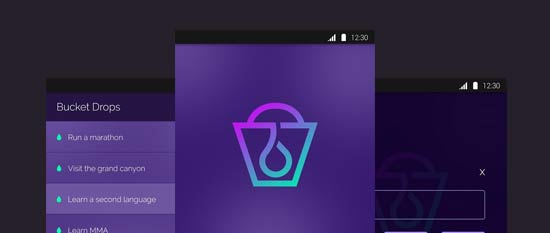
**[Սովորեք անկյունային 4-ը Քերծվածքից](https://coursetro.com/courses/12/Learn-Angular-4-from-Scratch?utm_source=in_article&utm_campaign=article&utm_medium=Learn+Angular+4+from+Scra)**

**[[](https://coursetro.com/courses/10/Create-a-Personal-Portfolio-using-Angular-2-&-Behance?utm_source=in_article&utm_campaign=article&utm_medium=Create+a+Personal+Portfol)](https://coursetro.com/courses/10/Create-a-Personal-Portfolio-using-Angular-2-&-Behance?utm_source=in_article&utm_campaign=article&utm_medium=Create+a+Personal+Portfol)**

**[Ստեղծեք անձնական պորտֆոլիո ՝ օգտագործելով Angular 2 & Behance](https://coursetro.com/courses/10/Create-a-Personal-Portfolio-using-Angular-2-&-Behance?utm_source=in_article&utm_campaign=article&utm_medium=Create+a+Personal+Portfol)**

**[[](https://coursetro.com/courses/5/Getting-Started-with-Java?utm_source=in_article&utm_campaign=article&utm_medium=Getting+Started+with+Java)](https://coursetro.com/courses/5/Getting-Started-with-Java?utm_source=in_article&utm_campaign=article&utm_medium=Getting+Started+with+Java)**

**[Ինչից սկսել Java- ն](https://coursetro.com/courses/5/Getting-Started-with-Java?utm_source=in_article&utm_campaign=article&utm_medium=Getting+Started+with+Java)**

**[[](https://coursetro.com/courses/2/Design-+-Code-an-Android-App-from-Scratch?utm_source=in_article&utm_campaign=article&utm_medium=Design+%2B+Code+an+Androi)](https://coursetro.com/courses/2/Design-+-Code-an-Android-App-from-Scratch?utm_source=in_article&utm_campaign=article&utm_medium=Design+%2B+Code+an+Androi)**

**[Դիզայն + Քերծվածքից ծածկագրեք Android հավելված](https://coursetro.com/courses/2/Design-+-Code-an-Android-App-from-Scratch?utm_source=in_article&utm_campaign=article&utm_medium=Design+%2B+Code+an+Androi)**

## Եթե ​​նախընտրում եք տեսանյութ դիտել ..

Վստահ լինել,  [**Բաժանորդագրվեք պաշտոնական Coursetro Youtube ալիք**](http://youtube.com/user/designcourse)  է ավելի տեսանյութեր.

## Տեղադրում

Նախ, ձեզ հարկավոր է Node.js- ը `Angular CLI- ն տեղադրելու համար: Անցեք դեպի [**http://nodejs.org**](http://nodejs.org/) , ներբեռնեք և տեղադրեք այն: Լռելյայն պարամետրերով տեղադրումից հետո բացեք ձեր հրամանի տողը / վահանակը և գործարկեք հետևյալ հրահանգը.

> npm -v

Սա պետք է թողարկի տարբերակի համարը: Եթե ​​այո, պատրաստ եք տեղադրել Angular CLI (հրամանի տողի միջերես), որը հրամանային տողի գործիք է, որը թույլ է տալիս ստեղծել և կառավարել ձեր անկյունային 8 նախագծերը:

Անկյունային CLI- ն տեղադրելու համար թողարկեք հետևյալ հրահանգը.

> npm install -g @angular/cli

 Հիանալի է, հիմա եկեք ստեղծենք նոր անկյունային 8 նախագիծ.

> ng new myapp

Սա ձեզ հուշելու է մի քանի հարցով: Պատասխանեք նրանց ՝ համաձայն ստորև բերված պատասխանների.

? Would you like to add Angular routing? Yes

? Which stylesheet format would you like to use? SCSS]

Անկյունային երթուղին թույլ է տալիս երթուղիներ ստեղծել ձեր ծրագրի բաղադրիչների միջև: Մենք կօգտագործենք նաև Sass- ը (SCSS), այնպես որ դա նույնպես ավելացնում ենք:

Եկեք մտնենք այն թղթապանակը, որտեղ պահվում է մեր նոր նախագիծը.

> cd myapp

Այս պահին ես սովորաբար արձակում եմ հրամանը ՝ կոդ: որը բացում է Visual Studio Code- ը (իմ կողմից օգտագործվող կոդերի խմբագրիչը) ընթացիկ պանակում:

Հիանալի է, մենք պատրաստ ենք ռոքս անել հիմա:

## Runրագրի գործարկում

Երբ ձեր անկյունային 8 ծրագիրը զարգացնում եք, դուք կցանկանաք տերմինալում թողարկել հետևյալ հրամանը.

> ng serve -o

-O դրոշը պարտադիր չէ, բայց այն բացում է ձեր նախնական զննարկիչը զարգացման վայրում [**http: // localhost: 4200**](http://localhost:4200/)

Այժմ, երբ ձեր անկյունային հավելվածն եք մշակում, ամեն անգամ ֆայլը թարմացնելիս զննարկիչը ավտոմատ կերպով վերաբեռնում է (տաք վերբեռնում), այնպես որ դուք կարող եք տեսնել ծրագիրը և շտկել այն իրական իրական ժամանակում:

**Նշում** . Երբ ուզում եք տեղադրել ձեր անկյունային ծրագիրը, դուք կօգտագործեք այլ հրաման: Մենք դրան կանդրադառնանք ավելի ուշ:

## Թղթապանակի կառուցվածք

Արժե մի փոքր ժամանակ հատկացնել այն կարևոր ֆայլերն ու պանակները ուրվագծելու համար, որոնցում սովորաբար աշխատելու եք, և նաև հասկանալ որոշ ներքևի մասերից, որոնք անկյունային 8-ը աշխատում են:

Թղթապանակը և ֆայլի կառուցվածքը նման է անկյունային 8 նախագծում.

> e2e

> node\_modules

> src

> app

> assets

> environments

..index.html

..styles.scss

* Որ **e2e** թղթապանակը վերջ մինչեւ վերջ փորձարկման. Այս դասընթացում մենք չենք լուսաբանելու թեստավորումը, բայց ես դրա վերաբերյալ կկատարեմ առանձին ձեռնարկ:
* **node\_modules-** ն այն թղթապանակն է, որը դուք երբեք չեք դիտելու դիպչելու համար, քանի որ այն պարունակում է նախագծի կախվածությունները:
* / **src** պարունակում է ձեր ծածկագրի մեծ մասը:
* / **հավելվածն** այն **վայրն** է, որտեղ դուք ձեր ժամանակի մեծ մասը կանցկացնեք ձեր Անկյունային 8 կոդը գրելու համար: Այն ներառում է երթուղին, բաղադրիչները և այլն:
* / **index.html-** ը ծրագրի մուտքի կետն է, և դուք, ընդհանուր առմամբ, չեք շոշափում այս ֆայլը:
* / **styles.scss- ը** այն **վայրն** է, որտեղ բնակվելու են ձեր CSS գլոբալ կանոնները:

## Անկյունային 8 բաղադրիչ

Ձեր անկյունային հավելվածի հիմնական կառուցվածքային բաղադրիչները բաղադրիչներն են: Բաղադրիչները բաղկացած են 3 տարրերից.

* **Ներմուծումը**
* **Բաղադրիչի զարդարանք** , որոնք տարբեր հատկություններ են ձեր բաղադրիչի համար: Բաղադրիչի զարդարանքը ներառում է ձեր բաղադրիչի ձևանմուշի և CSS- ի տեղադրության վայրեր:
* **Բաղադրիչի տրամաբանությունը** , որտեղ գտնվում է ձեր կոդը:

Եկեք նայենք այն անկյունային CLI- ի ստեղծած բաղադրիչին, որպեսզի տեսնենք այս 3 ոլորտները գործողության մեջ:

Բացեք **/src/app/app.component.ts** :

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.scss']

})

export class AppComponent {

title = 'myapp';

}

Ինչպես տեսնում եք, վերևում մենք ունենք մեկ ներմուծում, որն անհրաժեշտ է բոլոր անկյունային բաղադրիչների համար: Մենք ունենք նաև **@Component ({})**  դեկորատոր, և ներքևի **մասի** բաղադրիչի տրամաբանությունը `մեկ **վերնագրի**  հատկությամբ:

Առաջընթացի ընթացքում մենք կաշխատենք այս 3 հասկացությունների հետ էլ ՝ հավելվածը ստեղծելու համար:

## Նավիգացիա ստեղծելը

Եկեք մեր հավելվածի վերևում ավելացնենք նավանշան ՝ պատկերանշանով և նավիգացիայով:

Բացեք **/src/app/app.component.html- ը**  և հեռացրեք բոլոր ընթացիկ ծածկագրերը: Փոխարինեք այն հետևյալով.

<header>

<div class="container">

<a routerLink="/" class="logo">CoolApp</a>

<nav>

<ul>

<li><a href="#" routerLink="/">Home</a></li>

<li><a href="#" routerLink="/list">List</a></li>

</ul>

</nav>

</div>

</header>

<div class="container">

<router-outlet></router-outlet>

</div>

Երկու անկյունային 8-ին հատուկ երկու կարևոր ոլորտներն են.

1. **routerLink**  - այսպես եք կապում ձեր ծրագրի տարբեր էջերը: Դուք չեք օգտագործում href :
2. **երթուղիչի ելք**  - այստեղ է, որ երթուղային բաղադրիչները ցուցադրվում են ձևանմուշի ներսում: Դուք կտեսնեք, թե ինչպես է դա գործում շուտով:

Հաջորդը, եկեք այցելենք գլոբալ **/app/styles.scss**  ֆայլ ՝ այն հետևյալ կանոններով ապահովելու համար.

@import url('[**https://fonts.googleapis.com/css?family=Nunito:400,700&display=swap**](https://fonts.googleapis.com/css?family=Nunito:400,700&display=swap)');

$primary: rgb(111, 0, 255);

body {

margin: 0;

font-family: 'Nunito', 'sans-serif';

font-size: 18px;

}

.container {

width: 80%;

margin: 0 auto;

}

header {

background: $primary;

padding: 1em 0;

a {

color: white;

text-decoration: none;

}

a.logo {

font-weight: bold;

}

nav {

float: right;

ul {

list-style-type: none;

margin: 0;

display: flex;

li a {

padding: 1em;

&:hover {

background: darken($primary, 10%);

}

}

}

}

}

h1 {

margin-top: 2em;

}

Այստեղ ոչ մի հուզիչ բան տեղի չի ունենում: Փրկելուց հետո ձեր հավելվածն այժմ պետք է ունենա ոճավորված նավիգացիոն գոտի:

## Երթուղի

Let's use the Angular CLI to generate a couple components for the pages in our app.

Issue the following commands from the console:

> ng generate component home

> ng generate component list

This will generate several files for each component.

Next, we need to visit **/src/app/app-routing.module.ts** and add the following code:

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component'; // Add this

import { ListComponent } from './list/list.component'; // Add this

const routes: Routes = [

{ path: '', component: HomeComponent }, // Add this

{ path: 'list', component: ListComponent } // Add this

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

We first import the components that were generated, and then we add them as an object in the **Routes** array. We left the **path:** property blank, which signifies the home component that will load by default when the app loads.

If you click on the **List** and **Home** links in the navigation, they will now display the component template associated with the clicked component!

Simple!

## One way data binding

When you want to communicate data **from** the component logic **to** the template (or vice versa), this is called one-way data binding.

Open up the **/src/app/home/home.component.html** file and replace it with the following:

<h1>Welcome!</h1>

<div class="play-container">

<p>You've clicked <span (click)="countClick()">this</span> {{ clickCounter }} times.</p>

</div>

We have a few things happening here:

* **(click)** - This is a click event, which means if the element is clicked, it will call the function countClick() which doesn't yet exist.
* **{{ clickCounter }}** this is interpolation. clickCounter is a property (not yet defined) that will display data that's retrieved from the component.

Visit the **home.component.ts** file and add the following code:

export class HomeComponent implements OnInit {

clickCounter: number = 0;

constructor() { }

ngOnInit() {

}

countClick() {

this.clickCounter += 1;

}

}

We've defined the property (with TypeScript) and we've set it to 0.

Next, we created the function which will increment the clickCounter property by 1.

Before we give it a shot, let's give this some style. Visit the **home.component.scss** file and specify:

span {

font-weight: bold;

background: lightgray;

padding: .3em .8em;

cursor: pointer;

}

.play-container {

padding: 3em;

border: 1px solid lightgray;

margin-bottom: 1em;

input {

padding: 1em;

margin-bottom: 2em;

}

}

Save all of the files you just modified, and give it a shot!

First, the template is retrieving the clickCounter property from the component. Then, if you click on the span element, it is communicating data from the template to the component!

## Two way data binding

The best way to demonstrate the concept of data binding is to do it with a form element.

Visit **home.component.html** and add the following code:

<div class="play-container">

<p>

<input type="text" [(ngModel)]="name"><br>

<strong>You said: </strong> {{ name }}

</p>

</div>

In order for ngModel to work correctly, we need to import it into our **/src/app/app.module.ts**:

// other imports

import { FormsModule } from '@angular/forms';

@NgModule({

...

imports: [

BrowserModule,

AppRoutingModule,

FormsModule // add this

],

providers: [],

bootstrap: [AppComponent]

})

Next, we have to define the **name** property in the **home.component.ts** file:

clickCounter: number = 0;

name: string = ''; // add this

If you save it and begin to type within the textfield, you will see that it displays in the line beneath it in real time. This is two-way data binding because it's both setting and retreiving the property to and from the component/template!

## ng-template

What about working with **if and else** within your templates? We can use **ng-template** for that.

Add the following code at the end of **home.component.html**:

<div class="play-container">

<ng-template [ngIf]="clickCounter > 4" [ngIfElse]="none">

<p>The click counter <strong>IS GREATER</strong> than 4.</p>

</ng-template>

<ng-template #none>

<p>The click counter is <strong>not greater</strong> than 4.</p>

</ng-template>

</div>

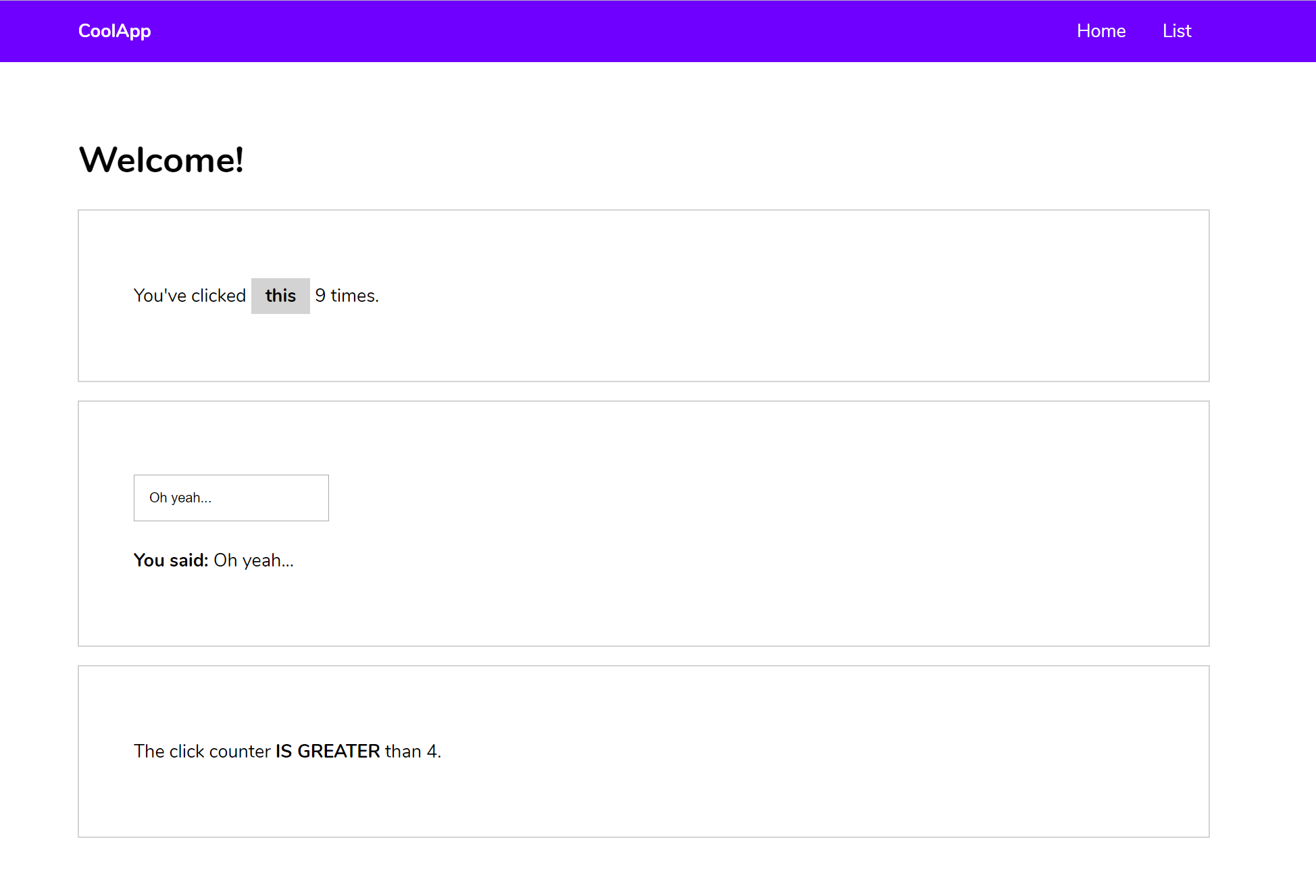
First, we use property binding **[ngIf]** and bind it to an expression clickCounter > 4.

If that expression isn't true, it will call upon a template called none with **ngIfElse**.

If that expression is true, it will show the HTML within the initial **ng-template** block.

If not, it shows the template defined by **#none** beneath it!

Give it a shot by clicking the span element until it reaches 5 or more and you will see it work in action.



Awesome!

## Style Binding

Sometimes, you want to modify the appearance of your UI based on events that occur in your app. This is where class and style binding come into play.

Modify the last **play-container** class in our HTML like so:

<div class="play-container" [style.background-color]="clickCounter > 4 ? 'yellow' : 'lightgray'">

With inline style binding, you wrap it in brackets (property binding) and specify **style.** and then the name of the CSS property. You bind them to an expression (we're using **clickCounter > 4**, or this could be a boolean value too) and then a ternary operator ? where the first value is used if it's true, and the second value after the colon is used for false.

If you save, it will initially show the play container block as light gray. If you click our span button a few times, it will turn yellow.

What if you wanted to specify multiple CSS properties?

Modify the code like this:

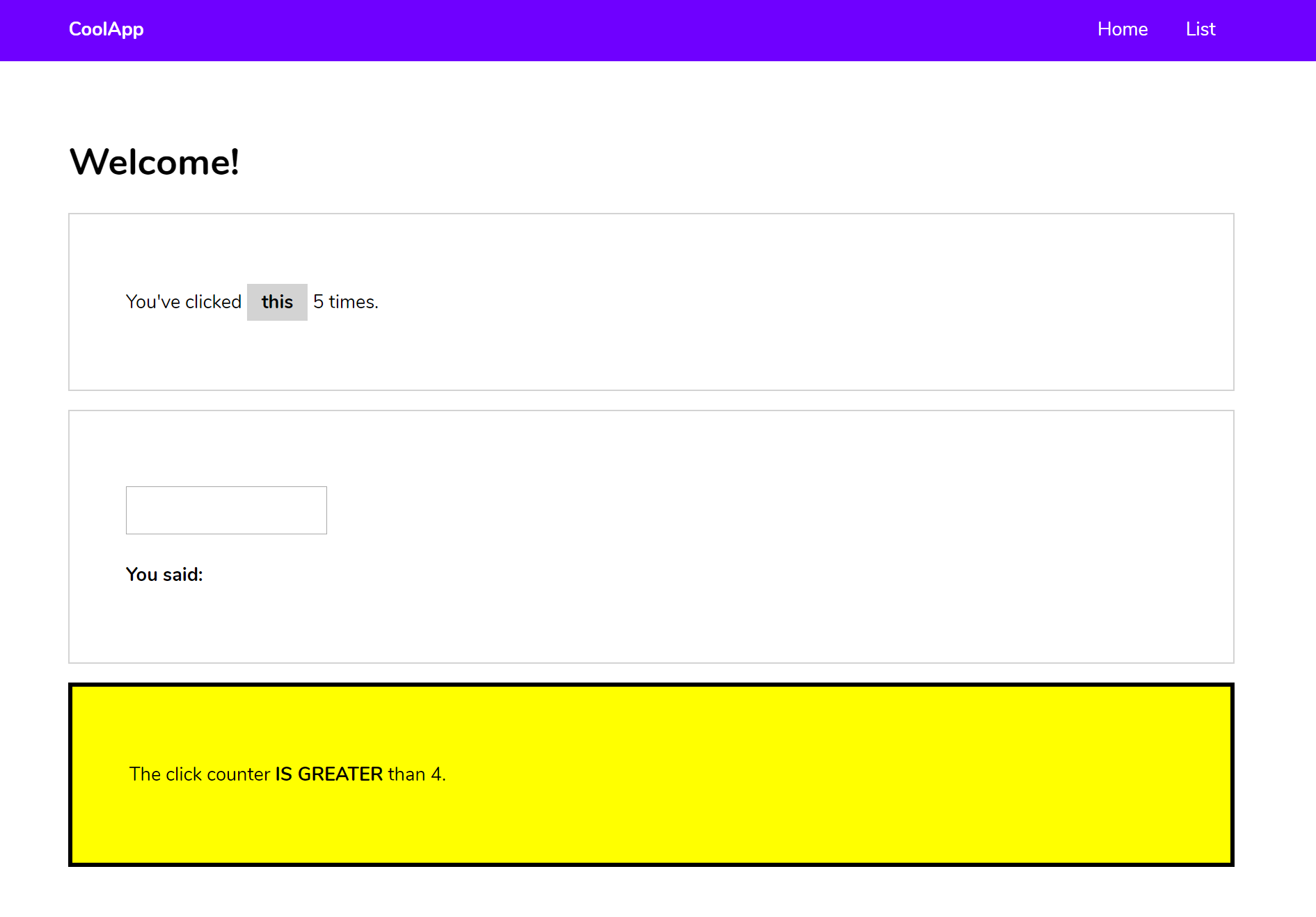
<div class="play-container" [ngStyle]="{

'background-color': clickCounter > 4 ? 'yellow' : 'lightgray',

'border': clickCounter > 4 ? '4px solid black' : 'none'}

">

Try it out now, and you will notice both CSS properties change.



**Note**: You can specify **[ngStyle]="someObject"**instead, if you wish to specify that logic in the component instead of the template.

## Class Binding

If you wish to add or remove entire classes that are defined in your CSS, you can do this with class binding.

Modify the current **.play-container** we've been working with, to the following:

<div class="play-container" [class.active]="clickCounter > 4">

Visit the **home.component.scss** and add this ruleset:

.active {

background-color: yellow;

border: 4px solid black;

}

Give it a shot! It works!

We can also set multiple classes with ngClass.

Modify the template as shown below:

<div class="play-container" [ngClass]="setClasses()">

Let's visit the component file and add the following:

setClasses() {

let myClasses = {

active: this.clickCounter > 4,

notactive: this.clickCounter <= 4

};

return myClasses;

}

We added the **notactive** class here, so we should define it in the component's CSS file as well:

.notactive {

background-color: lightgray;

}

Give it a shot! Awesome stuff!

## Services

Services are special components that are reusable throughout your app. We're going to create a service for the purpose of communicating with an API to fetch some data and display it on our lists page.

Let's generate the service with the Angular CLI:

ng g s http

Notice "g s", these are just shorthand terms for "generate service". The name we're giving this service is "http".

Let's visit the new service file located at **/src/app/http.service.ts**:

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class HttpService {

constructor() { }

}

It looks similar to a component, except the import is an Injectable instead of a Component, and the decator is based on this @Injectable.

Let's create a custom method that other components can access:

export class HttpService {

constructor() { }

myMethod() {

return console.log('Hey, what is up!');

}

}

Next, in **/src/list/list.component.ts:**

export class ListComponent implements OnInit {

constructor(private \_http: HttpService) { }

ngOnInit() {

this.\_http.myMethod();

}

}

ngOnInit() is a lifecycle hook that is fired when the component loads. So, we're saying, run our **.method()** from the service when the component loads.

If you click to the **list** link in the navigation and view your console in the web developer tools, you will see "Hey, what is up!" output.

## Angular HTTP Client

We need to integrate the HTTP client within our http service, which will allow us to communicate with a public API.

Visit our **http.service.ts** file and add the following:

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Injectable({

providedIn: 'root'

})

export class HttpService {

constructor(private http: HttpClient) { }

getBeer() {

return this.http.get('[**https://api.openbrewerydb.org/breweries**](https://api.openbrewerydb.org/breweries)')

}

}

First, we import the HttpClient, then we create an instance of it through dependency injection, and then we create a method that returns the response from the API. Simple!

We have to import the HttpClientModule in our **/src/app/app.module.ts** file:

import { HttpClientModule } from '@angular/common/http'; // Add this

@NgModule({

imports: [

BrowserModule,

AppRoutingModule,

FormsModule,

HttpClientModule // Add here

],

Next, open up the **list.component.ts** file and add the following:

export class ListComponent implements OnInit {

brews: Object;

constructor(private \_http: HttpService) { }

ngOnInit() {

this.\_http.getBeer().subscribe(data => {

this.brews = data

console.log(this.brews);

}

);

}

}

 The service returns an observable, which means we can subscribe to it within the component. In the return, we can pass the data to our brews object.

Next, visit the list template file and add the following:

<h1>Breweries</h1>

<ul \*ngIf="brews">

<li \*ngFor="let brew of brews">

<p class="name">{{ brew.name }}</p>

<p class="country">{{ brew.country }}</p>

<a class="site" href="{{ brew.website\_url }}">site</a>

</li>

</ul>

First, we add an **\*ngIf** to only show the UL element if **brews** exists.

Then, we iterate through the array of objects with **\*ngFor**.

After that, it's a simple matter of iterating through the results with interpolation!

Let's style this with CSS real quickly in this component's .scss file:

ul {

list-style-type: none;

margin: 0;

padding: 0;

display: flex;

flex-wrap: wrap;

li {

background: rgb(238, 238, 238);

padding: 1em;

margin-right: 10px;

width: 20%;

height: 200px;

margin-bottom: 1em;

display: flex;

flex-direction: column;

p {

margin: 0;

}

p.name {

font-weight: bold;

font-size: 1.2rem;

}

p.country {

text-transform: uppercase;

font-size: .9rem;

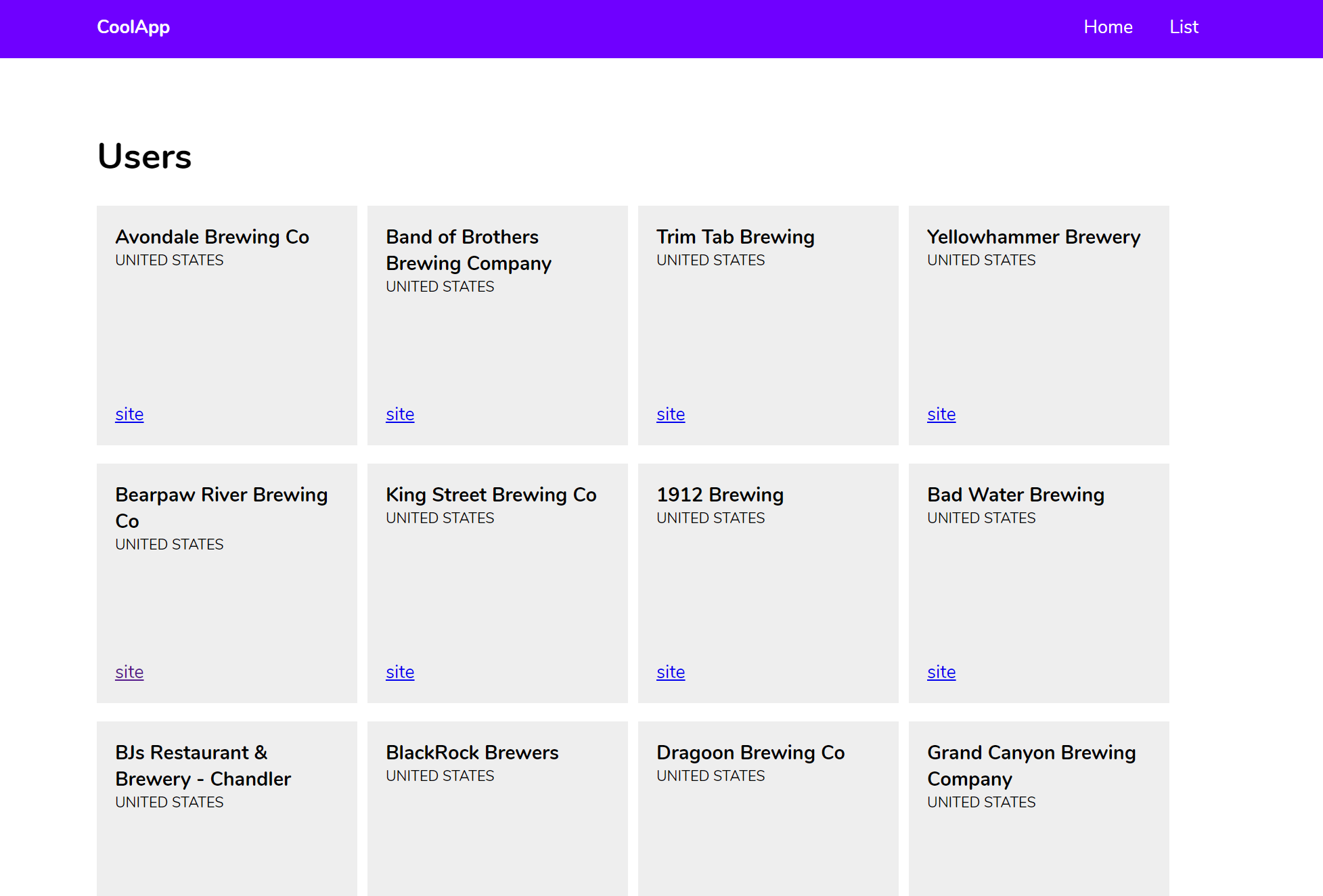
flex-grow: 1;

}

}

}

And here's the result!



## Deployment

Ասենք, որ մենք գոհ ենք մեր հավելվածից և ցանկանում ենք տեղակայել այն:

Մենք նախ պետք է արտադրական կառուցվածք ստեղծենք Angular CLI- ով: Այցելեք մխիթարիչ և թողարկեք հետևյալ հրահանգը.

> ng build --prod

Սա կստեղծի **/ dist**  պանակ: Մենք կարող ենք նույնիսկ այն տեղական եղանակով գործարկել lite-server- ի նման մի բանի միջոցով: Lite- սերվեր տեղադրելու համար.

> npm i -g lite-server

**Ներխուժեք** պանակ ՝ **myapp \ dist \ myapp \**  և գործարկեք ՝

> lite-server

Սա կսկսի արտադրության կառուցվածքը զննարկչում:

Այս պահին դուք ունեք դրա տեղակայման մի շարք տարբերակներ (Github Pages, Netlify, ձեր սեփական հոսթինգ և այլն:):

## Փակում

Մենք պարզապես քերել ենք մակերեսը այստեղ, բայց դուք նաև սովորեցիք **տոննան** : Ես առաջարկում եմ վերստեղծել մեկ այլ ծրագիր `օգտագործելով այն ամենը, ինչ սովորել եք այստեղ, նախքան թեմաները առաջ մղելու ավելի շատ միջանկյալի անցնելը: Այդ կերպ, դուք կարող եք իսկապես կրկնել հիմնական հիշողությունը հիշողությանը կրկնության միջոցով:

Վայելեք