Cookbook Angular

CodeWise

formations pour développeur web

Sommaire

Introduction

Vous souhaitez démarrer un projet Angular ?

Dans ce document, retrouvez toutes les étapes de création et de configuration d'une application Angular de qualité entreprise.

Tout est répertorié de façon à vous laisser guider : Installation, initialisation, configuration des environnements, des tests, du proxy, installation des librairies incontournables, mise en place du lazy loading, etc.

Suivez ce guide, étape par étape, pour vous garantir de ne rien oublier !

Installation NPM, Node.js et Angular CLI

Avant de démarrer votre projet, assurez-vous d'avoir installé Node.js, npm et la CLI d'Angular. Si ce n'est pas fait, suivez ces deux étapes.

1. Node.js et NPM

Installeur Windows et Mac :

https://nodejs.org/en/download/

Commande d'installation Linux :

```
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs
```

2. Angular CLI

Pour installer la CLI d'Angular, il vous suffit de lancer cette commande.

```
npm install -g @angular/cli
```

Démarrer un projet (via la CLI)

Une fois votre environnement installé, lancez la commande suivante via la CLI Angular pour générer une base de projet :

```
ng new <nom du projet>
```

ou

```
ng n <nom du projet>
```

La CLI vous demandera :

- Le nom du projet si vous ne l'avez pas renseigné dans la commande ci-dessus
- Si vous voulez intégrer le routing (navigation dans votre application)
- Le format de style que vous souhaitez utiliser (CSS, SCSS, Sass ou Less)

Quand vous aurez répondu à ces questions, la génération du projet s'exécutera. Pour plus de détails, consultez la documentation de la commande ng new : https://angular.io/cli/new#options

Choisir les librairies

Prochaine étape : **le choix des librairies !**

Il vous faut désormais choisir les librairies dont vous aurez besoin lors du développement de votre application. Voici une liste des librairies les plus utilisées et leur commande d'installation.

1. Framework CSS

Vous pouvez choisir d'utiliser un framework/librairie CSS pour gagner du temps lors de la mise en place du design de votre application. Les frameworks recommandés pour une application Angular sont :

ng-boostrap

```
ng add @ng-bootstrap/ng-bootstrap
```

Angular Material

```
ng add @angular/material
```

Nebular

```
ng add @nebular/theme
```

2. Icônes

Si vous souhaitez utiliser des icônes dans votre application :

```
ng add @fontawesome/angular-fontawesome
```

3. Utils

Voici une liste de librairies qui peuvent vous être utiles pendant le développement :

ngneat : permet de détruire automatiquement les souscriptions aux observables RxJs (vivement recommandé)

```
npm i @ngneat/until-destroy
```

mobile-device-detect : librairie pour détecter le type d'appareil mobile

```
npm i mobile-device-detect
```

webfontloader : permet de précharger des polices depuis le code Angular.

```
npm i webfontloader
```

prettier : Le meilleur linter JS actuellement

```
npm i prettier
```

4. Tests

Il est vivement recommandé de tester votre application. Ces libraires ont été conçues pour vous aider dans cette étape :

ng-mocks : Outil de mocking pour les composants, pipes, directives etc.

```
npm i --save-dev ng-mocks
```

ts-auto-mock : Librairie très utile pour générer des objets aléatoirement pour vos tests.

```
npm i --save-dev ts-auto-mock
```

jest : Exécuteur de test puissant et récent, alternative à Karma

```
npm i --save-dev jest jest-preset-angular @types/jest
```

Configuration de jest : https://www.xfive.co/blog/testing-angular-faster-jest/

5. State Management

Si votre projet est voué à se complexifier, vous devriez sûrement envisager l'utilisation d'une librairie de state management.

ngxs : Librairie de state management pour Angular (recommandée)

```
npm i @ngxs/store
```

ngrx : Adaptation de redux pour Angular

```
npm i @ngrx/store
```

Configurer le thème de couleurs

Afin de garder un design uniforme sur votre site, il est recommandé de configurer un thème.

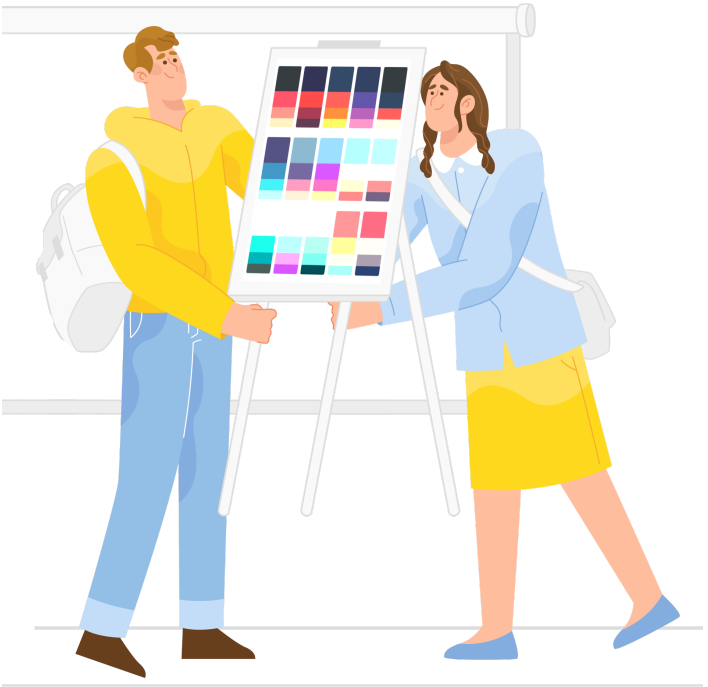
Chaque framework a sa manière de gérer les thèmes CSS. Voici les documentations des frameworks cités plus haut :

Angular Material : https://material.angular.io/guide/theming

ng-bootstrap : https://getbootstrap.com/docs/5.0/customize/color/

Nebular :
https://akveo.github.io/nebular/docs/design-system/create-custom-theme#create-custom-theme

Vous pouvez également créer votre propre thème, indépendant du framework, en créant un fichier global contenant des variables CSS/Sass/Less que vous utiliserez dans votre application.



Configuration générale

Rentrons maintenant dans les détails de la configuration de votre projet Angular. Un projet bien configuré grandit plus sereinement !

1. Typescript

Voici le fichier *tsconfig.json* recommandé pour démarrer un projet :

```
{
 "compileOnSave": false,
 "compilerOptions": {
   "noUnusedLocals": true,
   "noUnusedParameters": true,
   "noImplicitReturns": true,
   "noFallthroughCasesInSwitch": true,
   "skipLibCheck": true,
   "forceConsistentCasingInFileNames": true,
   "resolveJsonModule": true,
   "esModuleInterop": true,
   "baseUrl": "./",
   "strict": true,
   "outDir": "./dist/out-tsc",
   "sourceMap": true,
   "declaration": false,
   "downlevelIteration": true,
   "experimentalDecorators": true,
   "moduleResolution": "node",
   "importHelpers": true,
   "target": "es2015",
   "module": "es2020",
   "lib": [
     "es2018",
     "dom"
   ]
 }
}
```

Il est fortement recommandé d'utiliser l'option `strict: true` pour que votre projet évolue correctement. Activer cette option plus tard dans le développement sera très coûteux.

2. Proxy

Quel que soit votre projet, il est recommandé d'utiliser un proxy. Cela vous évite, entre autres, les erreurs CORS en environnement local, et permet d'avoir des redirections claires et centralisées.

Si vous voulez configurer un proxy avec Angular, créez un fichier *proxy.conf.json* à la racine du projet :

```
{
  "/api": {
    "target": "http://localhost:3000",
    "secure": false,
    "pathRewrite": {
      "^/api": ""
    }
  }
}
```

Plus d'infos : https://angular.io/guide/build#proxying-to-a-backend-server

3. I18n

Si vous souhaitez rendre votre application multilingue, il est recommandé d'utiliser le module ngx-translate :

```
npm install @ngx-translate/core --save
```

Pour plus d'infos : https://github.com/ngx-translate/core

Sachez qu'il existe différentes alternatives. Si celle ci ne convient pas à votre projet, consultez la documentation d'Angular au sujet de l'i18n : https://angular.io/guide/i18n-overview

Routeur

Si votre projet est voué à contenir plusieurs pages, vous allez devoir configurer un routeur. Vous devriez vous en soucier dès le début du développement.

L'étape 1 est obligatoire, les 3 autres sont optionnelles en fonction de vos besoins.

1. Configurer le routing

Cette documentation vous guidera pas à pas dans la mise en place du routing.

https://guide-angular.wishtack.io/angular/routing/mise-en-place-du-routing

2. Configurer le lazy-loading

Le lazy-loading vous permet de réduire le temps de chargement initial de votre application. Principalement utilisé pour les sites référencés sur Google.

https://guide-angular.wishtack.io/angular/routing/lazy-loading

3. Ajouter des Guards

Les Guards permettent de sécuriser l'accès à vos différentes pages.

https://guide-angular.wishtack.io/angular/routing/route-guards

4. Ajouter des Resolvers

Les Resolvers vous permettent de précharger le contenu d'une page pendant son affichage.

https://makina-corpus.com/front-end/routing-angular-optimisez-le-rendu-au-changement-de-page

Optimiser le build de production

Afin d'optimiser la taille de vos fichiers déployés et vos performances, il est recommandé de configurer les options de build.
Selon votre projet vous devriez également configurer différents environnements (dev, release, prod, etc.)

1. Configurer la compilation AOT

Angular implémente par défaut le compilateur AOT (Ahead of time, compilateur recommandé).

Si vous voulez le configurer ou le désactiver, dans votre fichier *ts.config.json* définissez "*angularCompilerOptions*" comme ceci :

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    // ...
  },
  "angularCompilerOptions": {
    "enableI18nLegacyMessageIdFormat": false,
    "strictInjectionParameters": true,
    // ...
  }
}
```

La liste des options du compilateur :

https://angular.io/guide/angular-compiler-options#template-options

2. La configuration prod

En plus d'utiliser vos variables d'environnement de production, le build de prod optimise votre projet en réalisant les actions suivantes :

- Tree shaking
- Suppression du code inutilisé

Pour lancer un build en mode prod :

```
ng build --configuration=prod
```

3. Build Optimizer

Le build optimizer permet de désactiver les décorateurs et paramètres de constructeur à la compilation (gain de performance et d'espace disque) :

```
ng build --configuration=prod --buildOptimizer=true
```

4. Configuration des environnements

Dans cette section, retrouvez les étapes à suivre pour créer un environnement de dev.

- Dans votre dossier *environments*, créez un fichier *environment.dev.ts* :

```
export const environment = {
 production: false,
 title: 'Dev Environment Heading',
 apiURL: 'http://dev.example.com'
};
```

- Ensuite dans votre fichier *angular.json*, ajoutez l'environnement dev :

```
...
"configurations": {
   "production": {
     "fileReplacements": [
       {
         "replace": "src/environments/environment.ts",
         "with": "src/environments/environment.prod.ts"
       }
     ],
...
},
   "dev": {
     "fileReplacements": [
       {
         "replace": "src/environments/environment.ts",
         "with": "src/environments/environment.dev.ts"
       }
     ],
       "optimization": false,
     "outputHashing": "all",
     "sourceMap": false,
     "extractCss": true,
     "namedChunks": false,
     "extractLicenses": true,
     "vendorChunk": false,
     "buildOptimizer": true,
```

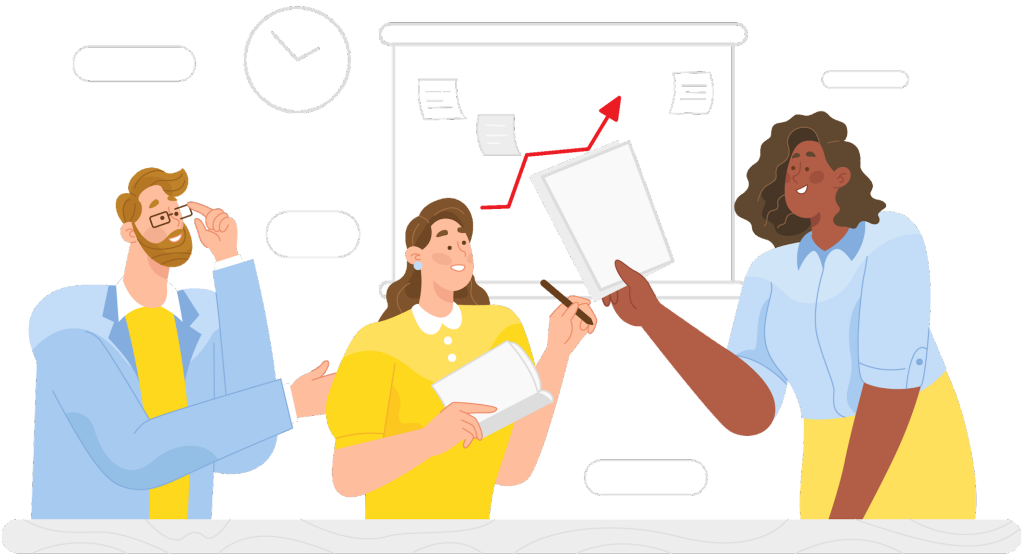
```
      "budgets": [
        {
          "type": "initial",
          "maximumWarning": "2mb",
          "maximumError": "5mb"
        },
        {
          "type": "anyComponentStyle",
          "maximumWarning": "6kb",
          "maximumError": "10kb"
        }
      ]
    }
  }
...
"serve": {
...
  "configurations": {
        "production": {
          "browserTarget": "appEnv:build:production"
        },
        "dev": {
          "browserTarget": "appEnv:build:dev"
        }
      }
    }
...
"e2e": {
      "builder": "@angular-devkit/build-angular:protractor",
      "options": {
        "protractorConfig": "e2e/protractor.conf.js",
        "devServerTarget": "appEnv:serve"
      },
      "configurations": {
        "production": {
          "devServerTarget": "appEnv:serve:production"
        },
        "dev": {
          "devServerTarget": "appEnv:serve:dev"
        }
      }
    }
```

Remarquez les configurations “`fileReplacements`“.

Et voilà ! Vous pouvez maintenant utiliser vos variables d'environnement :

```
import { Component } from '@angular/core';
import { environment } from './../environments/environment';
 @Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 title = environment.title;
 apiURL = environment.apiURL;
}
```

Source : https://www.itsolutionstuff.com/post/how-to-use-environment-variable-in-angularexample.html



Scaffolding

La configuration est terminée, il est temps de commencer à coder !

Pour générer un schéma avec la CLI :

```
ng generate <schema> [option]
```

ou

```
ng g <schema>
```

La liste des schémas disponibles :

- app-shell
- application
- class
- component
- directive
- enum
- guard
- interceptor
- interface
- library
- module
- pipe
- resolver
- service
- service-worker
- web-worker

Conclusion

Vous venez de voir les différentes étapes de création d'un projet Angular. Plus de syndrôme de la page blanche maintenant, vous n'avez qu'à suivre les étapes et prendre ce qui est pertinent pour votre projet.

Nous espérons vivement que ce cookbook vous sera utile ! Pour toute suggestion ou question : contact@codewise.fr