# zenika-formation-angular2

# Travaux Pratiques

zenika
ARCHITECTURE INFORMATIQUE

# Pré-requis

## Installation

Installer les technos demandées :

- NodeJS / NPM
- GIT

# TP 1 : Démarrer une application Angular

Dans ce premier TP, nous allons initier notre première application **Angular**, qui sera réutilisée dans les TPs suivant.

L'initialisation de cette application se décomposera en plusieurs étapes :

- Création d'un projet Angular avec `angular-cli`
- Implémentation de la page principale
- Création du composant principal
- Lancement du serveur afin de tester

## Création du projet

L'application, que nous allons implémenter, sera initialisée via l'outil `angular-cli`. Cet outil va automatiser :

- la création et la configuration du squelette de l'application
- la gestion des dépendances
- Téléchargez `angular-cli` en utilisant `NPM`. Ce module nécessite une version récente de *NodeJS*
- Depuis votre console, créez un nouveau projet via la commande `ng new Application`
- Regardez la structure de l'application tout juste créée
    - dépendances installées
    - configuration TypeScript
    - les différents fichiers TypeScript
- Une fois cette étape terminée, vous pouvez à présent lancer votre application en

exécutant la commande `ng serve`. Cette commande va prendre en charge la compilation de vos sources et le lancement d'un serveur.

### Modification de l'application

Même si nous n'avons pas encore abordé les concepts du framework, nous allons faire des petites modifications afin de prendre en main la structure de notre application.

- Le composant principal devra contenir le code HTML suivant :

```
<h1>This is my first component</h1>
```

- La chaîne de caractère ci-dessus pourra être modifiée par la variable `title` de la classe `Application`. Pour afficher cette variable dans le template, nous utiliserons la même syntaxe que AngularJS : `{{title}}`
- Vérifiez que vous obtenez bien la toute dernière version de votre application dans le navigateur.

## TP 2 : Les Tests

Dans ce second TP, nous allons écrire nos premiers tests unitaires. Ce TP vient au tout début de la formation, afin de vous laisser la possibilité d'écrire de nouveaux tests pour les fonctionnalités que nous allons implémenter dans les TPs suivants.

Lors de la mise en place initiale de l'application dans le TP précédent, toute la configuration nécessaire à la bonne exécution des tests unitaires a été automatiquement réalisée.

Nous allons vérifier que tout fonctionne correctement.

- Lancez les tests via `angular-cli` et `karma`

```
ng test
```

- Vous pouvez jeter un coup d'oeil aux tests générés par angular-cli. Ils seront expliqués plus en détails dans les parties suivantes.

## TP3 : Composants

L'application que nous allons développer tout au long de cette formation, est une application d'e-commerce.

Après avoir reçu du formateur le template principal de l'application, veuillez suivre les étapes suivantes :

- Modifiez le fichier index.html créé dans les TPs précédent, afin d'intégrer le template envoyé par le formateur.
- Tout le code HTML situé entre les balises `body` doit être défini dans le template du composant `Application`
- Créez un nouveau composant `menu\menu.component.ts` dans lequel vous allez implémenter le menu principal de l'application. Pour créer un nouveau composant Angular, nous allons utiliser a commande `ng generate component menu`
- Remplacez dans le composant `Application` le menu initial par le composant que vous venez de créer.
- Le total de votre panier sera défini dans une variable `total` que nous allons initialiser à 0 dans le constructeur du composant.
- Dans un nouveau fichier `model\product.ts`, créez une nouvelle classe `Product` ayant les propriétés suivantes:
  - title de type `string`
  - description de type `string`
  - photo de type `string`
  - price de type `number`
- Dans le constructeur du composant `Application`, instancier un nouveau tableau de `Product`. Et ajoutez les produits utilisés dans le template.
- Modifier le template pour utiliser ce tableau pour l'affichage des différents produits. Comme nous n'avons pas encore vu la directive `ngFor`, vous êtes obligé de copier/coller le template pour chaque élément du tableau.
- Nous allons à présent externaliser le template utilisé pour afficher un produit dans un nouveau composant `ProductComponent`. Ce composant aura un paramètre `data` correspondant à un objet de type `Product`. Ajoutez ce composant dans le template.
- Nous allons à présent émettre un évènement `addToBasket`, via le composant `ProductComponent`, lorsque l'utilisateur cliquera sur le bouton `Ajoutez au panier`. Cet évènement sera utilisé par le composant `Application` pour mettre à jour la variable `total` créée précédemment.
- Pour terminer ce TP, nous allons créer un dernier composant, `footer` qui sera utilisé à la place de l'élément HTML `<footer>`. Ce composant devra avoir un point d'insertion (via le composant `ng-content`), grâce auquel nous spécifierons le texte à afficher.

# TP4 : Les directives Angular

Dans ce TP, nous allons utiliser les directives `ngFor`, `ngIf` et `ngClass` pour dynamiser notre page. Les autres directives d'Angular seront présentées lors du chapitre sur les formulaires.

- Grâce à la directive `ngFor`, itérez sur la liste des `products` afin d'afficher autant de composants `ProductComponent` qu'il y a d'éléments dans ce tableau.
- Dans la classe `Product`, ajoutez une propriété `stock` de type `number`.
- Initiez cette propriété pour tous les produits définis dans le composant `AppComponent`. Nous vous conseillons de mettre une valeur différente pour chaque produit, afin de pouvoir tester les différents cas définis ci-dessous.
- Grâce à la directive `ngIf`, affichez un produit, seulement si sa propriété `stock` est supérieure à 0. Vous serez peut-être obligé de revoir l'utilisation du `*ngFor` du point précédent.
- Grâce à la directive `ngClass`, ajoutez une classe CSS `last`, sur l'élément utilisant la classe `thumbnail`, si la propriété `stock` d'un produit atteint 1. Cette classe ne sera utilisée que pour modifier la couleur de fond (`background-color: rgba(255, 0, 0, 0.4)`)

# TP5 : Injection de Dépendances

Nous allons à présent aborder les services et l'injection de dépendances dans une application Angular.

Nous allons créer deux services :

- ProductService : qui sera en charge de la gestion des produits,
- CustomerService : qui sera en charge du panier de l'utilisateur.
- Veuillez créer un service `service\ProductService.ts` dans lequel vous allez définir :
  - un tableau `products` avec les valeurs définies dans le composant `AppComponent.ts`
  - une méthode `getProducts()` : retournera le tableau `products`
  - une méthode `isTheLast(produitTitle)` : retournera `true` si le stock d'un produit est égal à 1
  - une méthode `isAvailable(produitTitle)` : retournera `true` si le stock d'un produit n'est pas égal à 0

    - une méthode `decreaseStock(produitTitle)` : mettra à jour la propriété `stock` du produit spécifié en paramètre

- Veuillez créer un service `service\CustomerService.ts` dans lequel vous allez définir :
    - une méthode `addProduct(product)` : ajoutera le nouveau produit dans un tableau
    - une méthode `getTotal()` : calculera le montant total du panier.
- Importez ces deux services dans votre composant `Application`, et modifiez l'implémentation de ce composant afin d'utiliser les différentes méthodes implémentées précédemment.
- Pour terminer ce TP, nous allons externaliser le titre "Bienvenue sur Zenika Ecommerce" dans une variable injectable de type `String` en utilisant un provider de type `Value`

# TP6 : Les Pipes

Nous allons à présent utiliser les `pipes`, afin de formatter le contenu de notre application.

Dans un premier temps, nous allons utiliser les `pipes` disponibles dans le framework : `uppercase` et `currency`.

- Dans le template du composant `product`, utiliser le `pipe` `currency` afin d'afficher le prix d'un produit avec la devise *euro* et avec deux chiffres après la virgule.

Pour insérer le sigle de la devise après le prix d'un produit, vous allez devoir indiquer que votre application doit utilise la locale française. Pour cela, veuillez surcharger le provider `LOCALE_ID` de `@angular/core` dans la configuration des providers de votre module.

```
{ provide: LOCALE_ID, useValue: 'fr-FR' }
```

- Dans le constructeur du service `ProductService`, injecter le pipe `uppercase` afin de transformer les propriétés `title` de chaque produit.

Nous allons à présent créer notre propre `pipe`, qui va nous permettre de trier une collection de produit par sa propriété `title`.

- Créer un nouveau `pipe` grâce à *angular-cli*
- Implémenter la méthode de transformation, dans laquelle nous allons trier un tableau via la méthode `sort` du prototype `Array`

- Utiliser votre `pipe` dans le template du `ng-for`
- Nous allons à présent ajouter un paramètre à notre `pipe`. Ce paramètre permettra de définir la propriété sur laquelle le tri doit s'effectuer.

# TP7 : Communication avec une API REST

Après avoir reçu de la part du formateur un serveur REST développé en NodeJS, nous allons manipuler cette API pour récupérer la liste des produits, et persister le panier de l'utilisateur.

Pour lancer le serveur REST, vous devez exécuter la commande suivante :

```
cd server
npm install
node server.js
```

Le serveur sera disponible via l'URL `http://localhost:8080/rest/`.

Cette API propose plusieurs points d'entrée :

- `GET` sur `/products` retournera la liste des produits
- `GET` sur `/basket` retournera le panier de l'utilisateur
- `POST` sur `/basket` pour ajouter un nouveau produit au panier de l'utilisateur
- Nous allons tout d'abord modifier le service `ProductService`. Dans la méthode `getProducts`, nous allons envoyer une requête `HTTP` vers l'API correspondante. Lors de la reception de la réponse, vous devez :
  - convertir en `json`,
  - mettre en majuscule les propriétés `title` de chaque produit
  - stocker le résultat dans la propriété `products` de la classe
- Modifier le composant `AppComponent` en conséquence.
- Il se peut que vous ayez des des erreurs dans votre navigateur à cause de la méthode `map`. *RxJS* n'inclut pas tous les opérateurs afin de réduire au maximum la taille de la librairie. Pour résoudre ce problème, vous devez importer les opérateurs que vous souhaitez utiliser.
  - Créez un fichier `rxjs-operators.ts` à côté du fichier `app.module.ts`
  - Importez-y l'opérateur

```
import 'rxjs/add/operator/map';
```

  - Importez ce fichier dans `app.module.ts`

```
import './rxjs-operators';
```

- Vous devrez faire de même pour tous les opérateurs *RxJs* que vous utiliserez ensuite.
- Nous allons à présent modifier, de la même façon, le service `CustomerService`.
  - Créez une méthode `getBasket` avec le même fonctionnement que le point précédent
  - Implémentez une méthode `addProduct` dans laquelle nous allons envoyer une méthode `POST` pour ajouter un produit au panier de l'utilisateur.
- Modifiez le composant `AppComponent` afin d'utiliser la nouvelle version des services `CustomerService` et `ProductService`.

# TP8 : Router

Nous allons intégrer dans notre application le routeur proposé par défaut dans le framework.

- Créez deux composants : `home` et `basket`
  - le composant `home` aura la charge d'afficher le contenu de la page que nous avons implémenté dans les TPs précédents
  - le composant `basket` permettra d'afficher, pour l'instant, le contenu du panier de l'utilisateur (via le pipe `json`)
- Ajoutez à votre application la configuration nécessaire pour le fonctionnement du router. Pour cela, nous allons utiliser la méthode `forRoot` mis à disposition par le module `@angular/router`
- Dans le template du composant `Application`, nous allons utiliser la directive `router-outlet` afin d'indiquer le point d'insertion des différentes pages de l'application.
- Ajoutez la directive `routerLink` dans le composant `menu` afin de rediriger l'utilisateur vers les deux composants que nous venons de créer.

# TP9 : Gestion des Formulaires

Nous allons créer une nouvelle vue qui permettra de valider la commande.

Pour ce faire, nous allons commencer par créer une classe et un service pour gérer la validation.

- Dans un nouveau fichier `model\customer.ts`, créez une nouvelle classe `Customer` ayant les propriétés suivantes :
  - name de type `string`
  - address de type `string`
  - creditCard de type `string`
- Dans le service `service\CustomerService.ts` rajouter une méthode `checkout(customer)` qui doit :
  - faire un `POST` sur `/basket/confirm` pour persister la commande d'un client côté serveur

Pour interagir avec ces nouvelles fonctionnalités, nous allons utiliser le composant `basket` créé précédemment. Il affichera :

- le panier de manière simplifiée (une liste avec le nom et le prix de chaque produit)
- un formulaire permettant de saisir les informations du client.

Ajoutez un lien dans le composant `Home` qui pointe vers la page `/basket`.

Ce formulaire devra respecter les contraintes suivantes :

- Exécution de la méthode `checkout` lorsque l'évènement `ngSubmit` est émis. Après avoir reçu la réponse du serveur, redirigez l'utilisateur sur la page `home`
- un champ `input[text]` pour saisir le nom du client qui devra
  - être lié sur la propriété `name` de l'objet `Customer`
  - être requis (grâce à l'attribut *required*)
  - avoir la class CSS `has-error` s'il n'est pas valide
- un champ `textarea` pour saisir l'adresse du client qui devra
  - être lié sur la propriété `address` de l'objet `Customer`
  - être requis (grâce à l'attribut *required*)
  - avoir la class CSS `has-error` s'il n'est pas valide
- un champ `input[text]` pour saisir un code de carte bleue factice qui devra

    - être lié sur la propriété `creditCard` de l'objet `Customer`
    - être requis (grâce à l'attribut *required*)
    - respecter le pattern `^[0-9]{3}-[0-9]{3}$` qui correspond par exemple à `123-456`
    - avoir la class CSS `has-error` s'il n'est pas valide
    - afficher le message `Invalid pattern. Example : 123-456` si le pattern est incorrect
- un bouton `button[submit]` pour valider la saisie qui devra :
    - être désactivé si tout le formulaire n'est pas valide

Pour information, voici le template à utiliser pour ajouter un champ de formulaire dans votre page :

```
<div class="form-group has-error">
    <label class="control-label" for="name">Name</label>
    <input type="text" id="name" class="form-control">
</div>
```