**Laboratoire #3 : Partie 1**

Plusieurs composants

**Étape 1 - Parc du sud ⛄🏘**

* Réinstallez les dépendances du projet **Angular✨** fourni à l’aide de PowerShell.
* Ouvrez le projet avec Visual Studio Code et exécutez l’application Web.

**Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement**

**Étape 2 - Composants déroutés** 💥🚗

* Présentement, trois composants existent : **app**, **index** et **list**.
* Pour le moment, lorsque l’application Web est exécutée, on peut voir le composant **app**.(C’est le composant racine de l’application. Son HTML est affiché en permanence et il sert de squelette : il contient le header, le nav et le footer) Étant donné que le composant **app** contient le composant **index**, on voit le html du composant **index** également.

Une image contenant texte, Police, blanc, conception

Description générée automatiquement

(HTML dans app.component.html)

* **2**  Avant de jouer avec le routage, créez un nouveau composant nommé **details**. Pour le moment il sera à peu près vide, bien entendu.
* **9-13**  Nous aimerons que le <main></main> du composant **app** affiche un composant différent selon la route actuelle. Créez des **règles de routage** et faites les modifications nécessaires au template HTML du composant « **app** » pour que ...
  + Le composant **index** soit affiché par défaut **et** avec la route **localhost:4200/index**
  + Le composant **list** soit affiché avec la route **localhost:4200/list**
  + Le composant **details** soit affiché avec la route **localhost:4200/details**
  + De plus, assurez-vous que les trois boutons du **menu de navigation** fonctionnent et permettent de changer de route.



* + De plus, dans le composant index, il faut que cliquer sur « Voir une liste de personnage » mène vers la route **/list** et il faut que cliquer sur le bouton « Chercher » mène vers la route **/details** (Ne vous préoccupez pas de l’input à côté pour le moment !)
    - Autant pour le texte que pour le bouton, l’attribut **[routerLink]** peut faire le travail, même si ce ne sont pas des éléments <a>.
  + Normalement, vous venez de créer 4 règles de routage dans **app.module.ts** et vous avez modifié le template HTML de **app** et **index**.

**Étape 3 - Jeunes et innocents** 😇

Une image contenant Visage humain, dessin humoristique, capture d’écran, texte

Description générée automatiquement

* Nous allons travailler sur le composant **list** pour cette étape. Ci-dessus, nous avons le rendu visé.
* **3**  Créez un nouveau composant nommé **card**. Il servira à afficher un personnage (son nom et son image)
  + Voici le HTML et le CSS à utiliser pour le composant **card** :

<div class="card" title="Afficher les détails de ???">

    <h4>???</h4>

    <img src="/assets/img/???.png" alt="???">

</div>

.card{

    background-color:white; text-align: center; padding:5px;

    border-radius:5px; overflow: hidden; cursor:pointer; min-width:100px;

}

.card img{ height:100px; }

.card h4{ margin:5px 0px; }

* + Ce composant **ne sera pas accessible via le menu de navigation** ! Il sera intégré dans le composant **list** dans les étapes qui suivent. (Méthode des poupées russes)
  + **4-8**  Dans le TypeScript du composant **list**, remarquez qu’il y a une liste de personnages. Le but est, **pour chaque personnage dans cette liste**, d’afficher **un** composant **card** auquel nous enverrons un nom de personnage, via un paramètre (un **@Input**) nommé **characterName**.
    - En gros, grâce à un **\*ngFor**, vous allez afficher un **card** qui reçoit "bebe", puis un **card** qui reçoit "butters", puis un **card** qui reçoit "clyde", etc.
  + Finalement, dans le HTML de **card**, remplacez tous les **???** par la variable **characterName** servant d’@Input qui contient le nom du personnage à afficher. Cela complètera l’affichage.
  + Le composant **card** peut sembler un peu « inutile » dans cette situation. (On aurait pu tout mettre directement dans **list** …) Cela dit, il suffit qu’on réutilise **card** ailleurs dans le projet et cela contribuerait déjà à éviter de répéter du code.
    - En gros, dès que du HTML / CSS risque d’être réutilisé dans plusieurs composants, n’hésitez pas à créer des petits composants à réutiliser un peu partout.

**Étape 4 - Détails sans intérêt** 🔍

****

* Pour cette étape, nous allons principalement travailler sur le composant **details**. Ci-dessus, il y a le rendu visé.
* Voici le HTML et le CSS pour le composant **details** :

<h3>Détails sur ???</h3>

<img src="/assets/img/???.png" alt="???">

<div \*ngIf="false">

  <table>

    <tr><td><b>Nom complet</b> : </td><td>???</td></tr>

    <tr><td><b>Âge</b> : </td><td>???</td></tr>

    <tr><td><b>Occupation</b> : </td><td>???</td></tr>

    <tr><td><b>Grade</b> : </td><td>???</td></tr>

    <tr><td><b>Nombre d'épisodes</b> : </td><td>???</td></tr>

  </table>

</div>

td:first-child{text-align:right;}

td:last-child{text-align: left;}

* Dans le TypeScript du composant **details**, créez deux variables :
  + **characterName**, qui est de type **string | null**. Donnez-lui la valeur null.
  + **characterDetails**, qui est de type **Character | null**. Donnez-lui la valeur null.
* Ensuite, dans la fonction ngOnInit() :
  + Affectez la valeur "kenny" à la variable **characterName**. (Ceci est temporaire)
  + Envoyez la requête suivante et tentez de créer un objet de type **Character** pour le mettre dans la variable **characterDetails**.

"https://spapi.dev/api/characters?search=" + this.characterName

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

(Pour le nombre d’épisodes… vous aurez besoin de **.length** !)

* À l’aide des variables **characterName** et **characterDetails**, vous allez pouvoir remplacer tous les ??? dans le HTML.
  + Le nom et l’image au début utiliseront **characterName**.
  + Toutes les données dans le tableau utiliseront **characterDetails.propriété**.

**Étape 5 - Brancher quelques fils 🔌**

* **14-18**  Présentement, le composant **details** affiche toujours les informations sur Kenny. Nous allons faire quelques modifications pour pouvoir choisir le personnage dont on affiche les détails.
  + Ajoutez une règle de routage qui permet d’envoyer un **paramètre de route** au composant **details**. (Il faut encore pouvoir charger le composant **details** sans paramètre de route cependant !)
  + Ce paramètre de route servira à contenir le nom d’un personnage comme « tolkien », « wendy », etc.
    - Dans le composant **details**, dans **ngOnInit()**, on doit vérifier si un paramètre de route a été reçu. Si oui, on le met dans la variable **characterName**. (Qui est utilisée pour la requête et l’affichage) Si non, on met "Kenny" dans la variable **characterName**, comme avant. (Pour que la variable ne reste pas null)

Une image contenant dessin humoristique, clipart, Dessin animé

Description générée automatiquement

* + Dans le composant **card**, ajoutez un attribut **[routerLink]** sur le <div> pour rediriger vers le chemin **/details/nom\_du\_personnage\_cliqué**. (Par exemple, si on clique sur le <div> de Wendy, cela va rediriger vers **details/wendy** et donc on verra ses détails dans le composant **details**)

Une image contenant texte, capture d’écran, Police, Bleu électrique

Description générée automatiquement

* + Dans le composant **index**, faites les modifications nécessaires pour qu’on puisse écrire le nom d’un personnage dans l’input et que ce nom soit utilisé comme **paramètre de route** quand on appuie sur le bouton « Chercher » qui mène vers la route **/details**. (Donc si j’écris « eric » dans l’input, le bouton va rediriger vers la route **/details/eric**)
    - Ce n’est pas grave si le composant **details** fonctionne mal si on choisit un nom invalide. Vous pouvez vous contenter de tester avec un nom valide.

**Laboratoire #3 : Partie 2**

Services

**Étape 1 - Réouvrir votre projet Angular de la partie 1 📦**

* Réinstallez les dépendances du projet **Angular✨** à l’aide de PowerShell.
* Ouvrez le projet avec Visual Studio Code et exécutez l’application Web.

**Étape 2 - Rendez-vous service : utilisez les services 🛠🧰**

* **19-25**  Si on jette un coup d’œil aux composants **list** et **index**, on remarque qu’il y a une liste absolument identique qui est présente dans les deux classes... (characters) C’est de la répétition de données honteuse.
  + Créez un **service** nommé **data** dans lequel cette liste sera intégrée.
  + Injectez ce service aux deux composants qui utilisaient ces données et supprimez les données répétitives dans les deux composants.
  + Finalement, vous devrez modifier un peu les templates HTML de ces deux composants pour les adapter à la nouvelle façon d’accéder à la liste accessible via l’injection du service que vous avez créé.
  + Bien entendu, grâce à cette amélioration, si on voulait ajouter des personnages dans la liste, nous n’aurions à le faire qu’à un endroit plutôt que deux désormais, par exemple.

**Étape 3 - La grande migration** 🦆

* **26-30**  Lorsqu’on affiche les détails d’un personnage, une requête est lancée à un API pour obtenir des données. Vous pouvez retrouver cette requête dans le composant **details**.
  + Vous devez déplacer cette requête dans le service **data** plutôt que la laisser dans **details**.
    - Cela signifie que vous allez créer une fonction dans le service **data**. Elle va recevoir en paramètre le nom d’un personnage et elle retournera **Promise<Character>**.

**Étape 4 - Tout tient en place ?** ⚖

* Si l’application Web compile, toutes les images s’affichent et qu’on peut toujours vérifier les détails d’un personnage, c’est signe que vous avez bien tout transféré les données et la requête HTTP dans un service.
  + Bon travail ! Avec les services, on évite la répétition de code et on allège les composants pour qu’ils ne contiennent que le strict nécessaire pour l’affichage des données et la gestion des inputs.
  + Bien entendu, dans ce cas, les services ont peut-être l’air très petits et peu pertinents. C’est dans un projet d’ampleur que les services deviennent plus critiques.