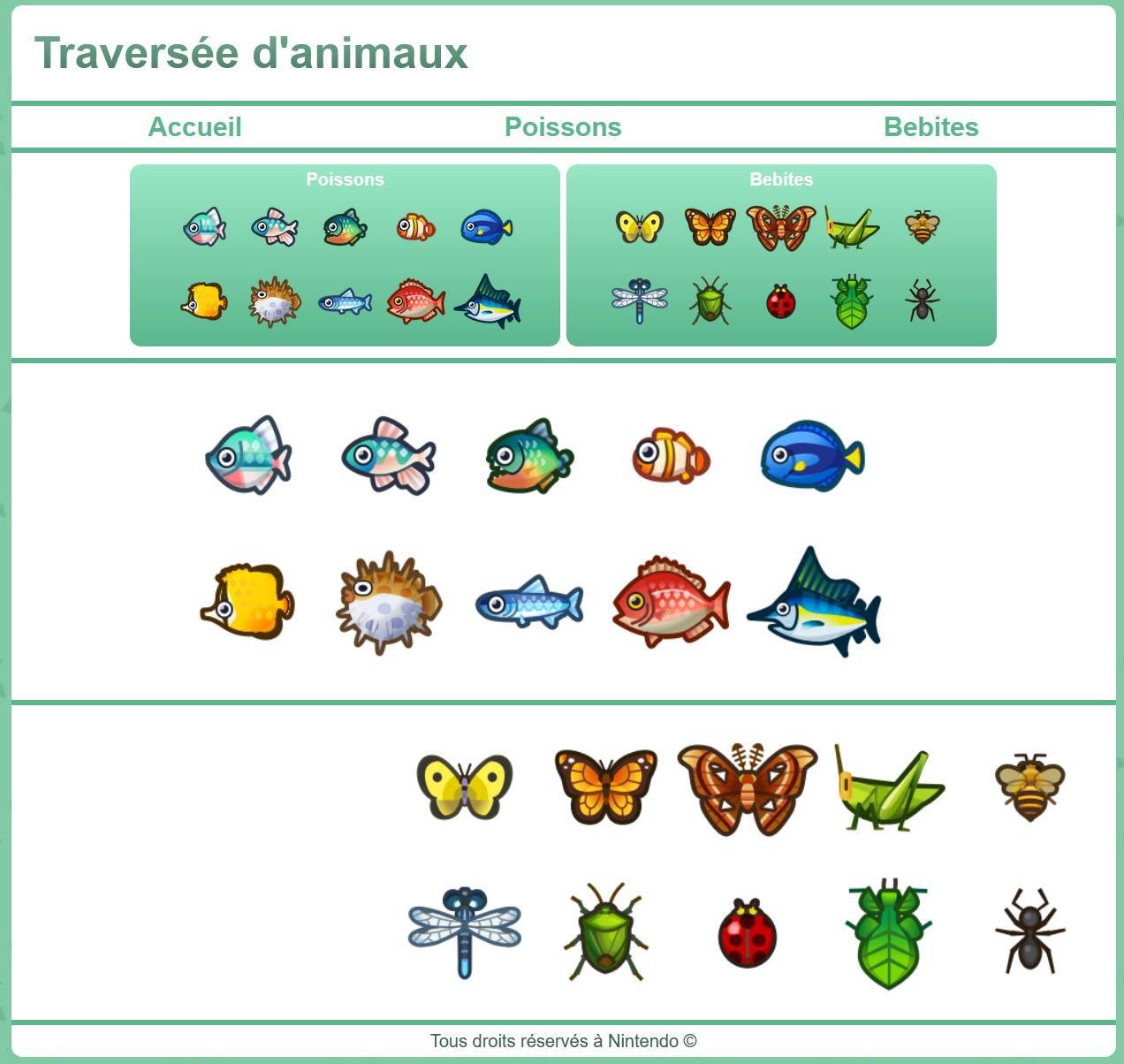
**Laboratoire #3 : Partie 1**

Plusieurs composants

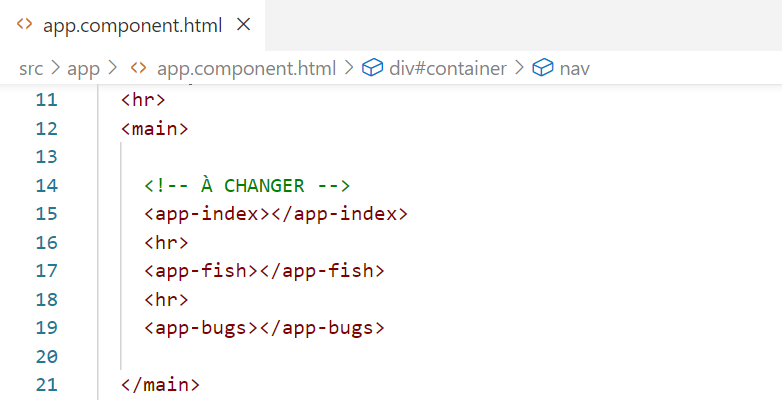
**Étape 1 - Ouvrir un projet Angular sans bugs, mais avec insectes 📦🐞**

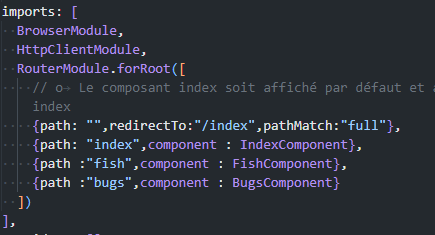
* Réinstallez les dépendances du projet **Angular✨** fourni à l’aide de PowerShell.
* Ouvrez le projet avec Visual Studio Code et exécutez l’application Web.

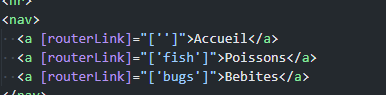
****

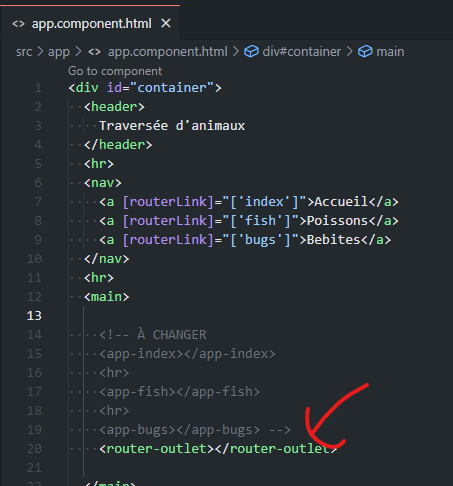
**Étape 2 - Composants déroutés** 💥🚗

* 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 les composants **index**, **fish** et **bugs**, on voit aussi leur contenu en ouvrant l’application.



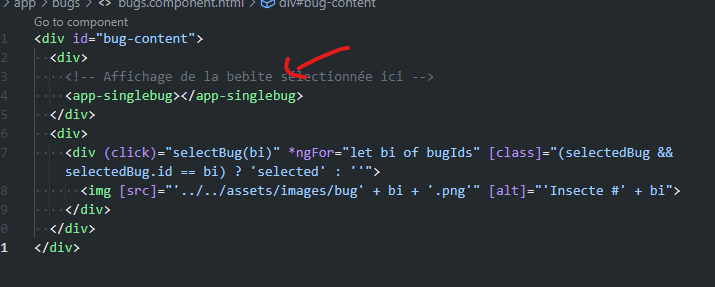
* Cela dit, nous ne voulons pas voir **index**, **fish** et **bugs** en même temps. On veut les séparer en trois pages grâce au routage.
* **9-13**  Créez des **règles de routage** et faites les modifications nécessaires au template HTML du composant « **App** » pour que ...
  + Le composant **index** soit affiché par défaut et avec la route **localhost:4200/index**
  + Le composant **fish** soit affiché avec la route **localhost:4200/fish**
  + Le composant **bugs** soit affiché avec la route **localhost:4200/bugs**
  + De plus, assurez-vous que les trois boutons du **menu de navigation** fonctionnent et permettent de changer de route.

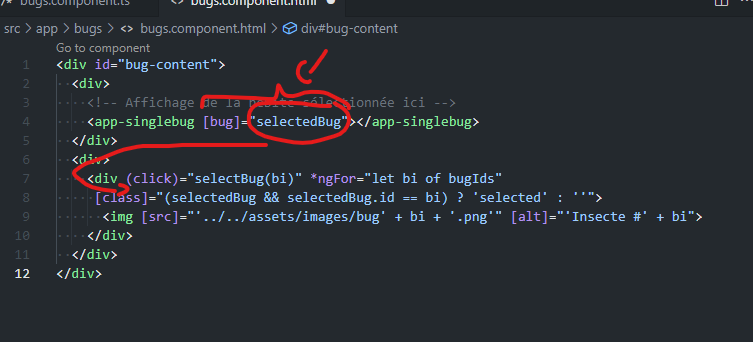


* + - Ce n’est pas tout : dans le composant **index**, deux <div> doivent devenir cliquables et mener vers les composants **fish** et **bugs**. (Les deux gros rectangles verts permettent de naviguer vers les pages Poissons et Bebites comme le menu de navigation)  
      
  + Finalement, **un seul** **composant** parmi les trois devrait être affiché à la fois et nous devrions pouvoir alterner librement entre les trois via le **menu de navigation**.  
    
  + 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 - Pas besoin d’un exterminateur** 🐞



* Nous aimerions que des informations apparaissent à gauche de la liste d’insecte lorsqu’on sélectionne un insecte. On va créer un petit composant pour faire cet affichage :
* **3-8**  Créez un nouveau composant nommé « **singlebug** ».   
  Pas obligé de conserver le fichier. spec*.ts*.
  + Ce composant ne sera pas accessible via le menu de navigation. Intégrez-le dans le template HTML du composant **bugs** grâce à son <sélecteur></sélecteur>, dans la colonne de gauche, qui est actuellement vide.  
    
  + Dans la **classe** du composant **singlebug**, créez une **variable de classe** de type @Input nommée bug, de type Creature. (Elle a le droit d’être undefined)
    - Cette variable va nous permettre d’envoyer une information à partir du composant **bugs** vers le composant **singlebug**.



* + - Dans le template HTML du composant **bugs**, dans le <sélecteur></sélecteur> servant à intégrer le composant **singlebug**, envoyez la valeur de la variable selectedBug à @l’input nommé bug que vous avez créé dans le composant **singlebug**.
  + Utilisez le code HTML ci-dessous pour le **template HTML** de votre composant **singlebug**.

<div \*ngIf="bug">

<img [src]="bug.icon" [alt]="'Insecte #' + bug.id">

<div>Bebite #{{bug.id}}</div>

<div>Nom : {{bug.name}}</div>

<div>Habitat : {{bug.location}}</div>

<div>Rareté : {{bug.rarity}}</div>

<div>Prix : {{bug.price}}</div>

</div>

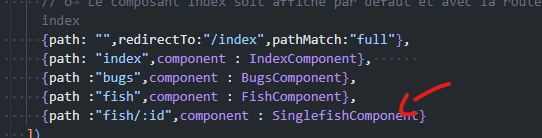
<div \*ngIf="!bug">

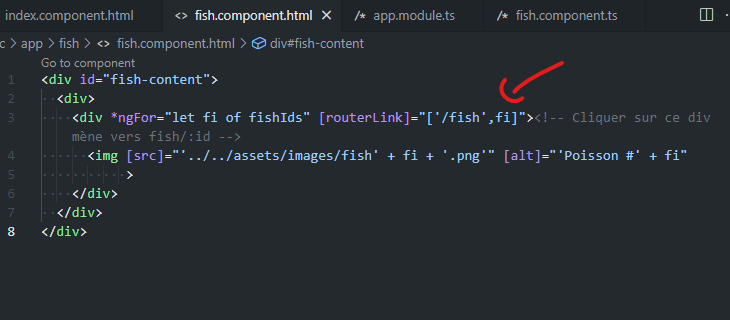
<p>Sélectionnez une bebite...</p>

</div>

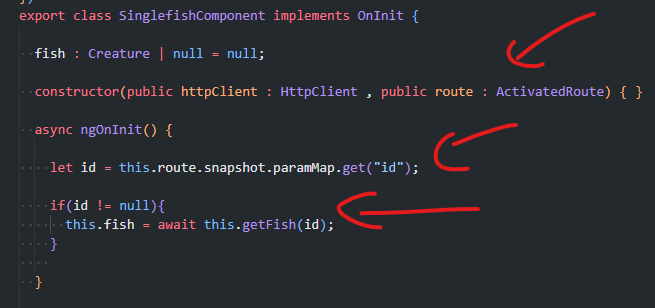
* + Voilà ! Vous devriez être capable de sélectionner un insecte et de voir ses informations. Quand un insecte est sélectionné, une requête HTTP est faite, des données sont obtenues (et transférées au composant singlebug) puis affichées !

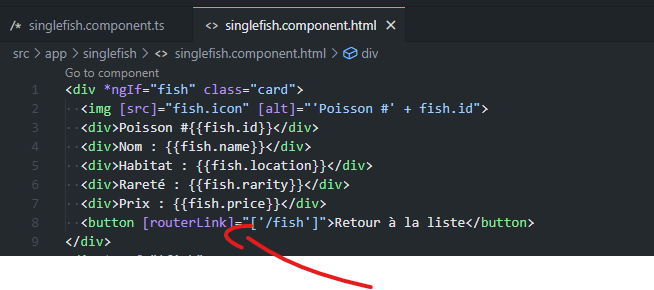
**Étape 4 - Où est Nemo ?** 🐠

* Pour la page des poissons, nous allons faire des ajouts similaires, mais cette fois-ci, cliquer sur un poisson va nous amener dans une autre page qui contiendra seulement les informations du poisson sélectionné.
* Le composant **singlefish** existe déjà et permet d’afficher les informations d’un poisson.
* **14-18**  Créez une règle de routage qui mène vers le composant **singleFish** lorsque la route correspond à **fish/:id**.
  + Notez bien les deux points ( : ) avant **id**. Ça veut dire qu’**id** est un paramètre de route.
  + Modifiez le template HTML du composant **fish** pour que cliquer sur un poisson mène vers la route **fish/:id**. Attention ! La valeur donnée au paramètre :id doit être l’id du poisson cliqué. Vérifiez la boucle \*ngFor dans le HTML pour trouver quoi mettre.



* + Présentement, cliquer sur un poisson devrait nous mener vers le composant **singleFish**, mais la page semble rester bloquée sur « Chargement des données... ». C’est parce que la variable nommée fish dans le composant **singleFish** est null. Nous allons y remédier.
  + Nous avons transmis un nombre avec la route fish/:id, on va maintenant récupérer ce nombre dans le composant **singleFish** et l’utiliser :
    - Dans la fonction ngOnInit, on peut voir **let id = null**. À la place de simplement mettre **= null**, pouvez-vous récupérer la valeur du paramètre id passé dans la route ?
    - Après cette modification, un poisson et ses informations devraient être affichés dans le composant **singlefish** lorsqu’on sélectionne un poisson dans le composant **fish**.



* + Un instant ce n’est pas terminé ! Rendez fonctionnel le bouton dans le bas du composant **singlefish** pour qu’il nous ramène à la liste des poissons dans le composant **fish**.
  + 

**Étape 5 - Qui vit dans un ananas dans la mer ? 🍍🧽**

* Dans le **template HTML** du **composant** **singlefish**, nous allons faire quelques ajouts pour rendre l’affichage plus robuste aux **paramètres de route** **incorrectes**.
  + Il y a seulement 80 poissons dans Animal Crossing. Ça signifie qu’un id supérieur à 80 fera échouer la requête à l’API. Modifiez le if(...) dans la fonction ngOnInit pour gérer ce problème.
    - Cela dit, id est un string. (Un paramètre de route est forcément un string) Glissez-le dans la fonction **parseInt(maVariable)** pour le convertir en number)
  + Essayez les routes *localhost:4200/fish/proute*, *localhost:4200/fish/-1* et *localhost:4200/fish/81* pour tester qu’aucune erreur n’est générée dans la console.

**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 🛠🧰**

* **19-22**  Si on jette un coup d’œil aux composants **bugs** et **index**, on remarque qu’il y a une liste absolument identique qui est présente dans les deux classes... (bugIds) 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 supprimées 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.
* **19-22**  Si on jette un coup d’œil aux composants **index** et **fish**, on remarque qu’il y a une liste absolument identique qui est présente dans les deux classes... (fishIds). Quoi ? Une impression de déjà-vu ?
  + Faites la même chose que pour les ids des insectes : transférez la liste dans le service **data** et retirez-la des composants.

**Étape 3 - La grande migration** 🦆

* **26-30**  Lorsqu’on clique sur une bebite ou un poisson, une requête est lancée à un API pour obtenir des données du jeu Animal Crossing. Vous pouvez retrouver ces deux requêtes dans les composants **singlefish** et **bugs**.
  + Créez un service nommé http dans lequel nous déplacerons les deux requêtes HTTP.
    - Dans le cas de **singlefish**, la fonction getFish() doit disparaître du composant.
    - Dans le cas de **bugs**, la fonction getBug() doit disparaître du composant.

**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 poisson / insecte, c’est signe que vous avez bien tout transféré les données et les requêtes HTTP dans deux nouveaux services.
  + 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.