PROJET FINAL

L'ATLAS DU MONDE

Pondération et échéance

- ☐ Ce travail peut être fait par groupe de 2.
- ☐ Ce travail compte pour 12% de la note finale du bulletin, mais plus important encore, il vous prépare, avec les laboratoires effectués depuis le début de la session, à l'examen final.

- ☐ Date de remise de la solution : Au plus tard le **lundi 8 décembre à 23h59**

  Une pénalité de 10 % par jour de retard, incluant les jours de congé, sera imposée, et ce, jusqu'à concurrence de 5 jours. (Après ce délai, la note attribuée est zéro.)

Vous serez évalués sur le produit final. **Mieux vaut remettre une application incomplète, mais fonctionnelle qu'une qui semble complète, mais qui ne produit pas les bons résultats.**

Pour débuter

Dans le cadre de ce projet final, vous serez appelé à programmer une application Web du début à la fin. Concrètement, vous produirez une application interactive qui présente la liste des pays du monde accompagnée de diverses informations, telles que leur population, leur capitale et leur drapeau national.

Pour réaliser ce projet, vous utiliserez React pour le développement de l'interface utilisateur en intégrant des techniques explorées en classe comme la gestion des états, les composants, et les appels API. La partie serveur sera construite à l'aide d'Express et servira à manipuler et fournir les données nécessaires à votre application.

L'objectif de ce projet est de vous permettre d'améliorer votre familiarisation avec React et la communication entre une application client et un serveur de base. De plus, vous aurez la chance de développer votre autonomie dans un contexte réaliste de résolution de problème.

1. Première partie : serveur Express (25%)

Création du projet

Dans votre répertoire de travail, créez un répertoire ***projet*** avec 2 sous-répertoires ***serveur*** et ***client***. Comme lors du laboratoire 10, ces dossiers seront à la racine de la structure de votre projet.

- Dans votre répertoire serveur, initialisez l'application en lançant la commande

```
$ npm init -y
```

- Installez le module ***express*** et le module ***cors*** avec la commande

```
$ npm install express cors
```

Votre application ***serveur*** est maintenant prête à utiliser ***express*** et ***cors***.

Dans votre répertoire serveur, créez maintenant un fichier qui contiendra éventuellement la logique de votre serveur et nommez le ***app.js***.

Finalement, modifiez votre fichier ***package.json*** pour pouvoir lancer le serveur Web en utilisant ***nodemon***.

```
  "scripts": {
    "start": "npx nodemon app.js"
  },
```

En modifiant votre fichier de la sorte, vous pourrez maintenant démarrer votre application serveur avec la commande

```
$ npm start
```

*Note : **Nodemon** est un outil qui surveille les fichiers d'un projet Node.js et redémarre automatiquement le serveur dès qu'il détecte des modifications (lors de la sauvegarde d'un fichier), facilitant ainsi le développement en temps réel. Vous n'aurez donc pas besoin de redémarrer votre serveur chaque fois que vous y apportez une modification pour voir le changement apporté.

Préparation des données

Un fichier .json faisant office de base de données et nécessaire à la réalisation de ce projet vous a été fourni dans le même dossier que cet énoncé. Ce fichier contient les informations sur les différents pays du monde sous forme d'un objet JSON. Copiez ce fichier dans le dossier de votre application serveur. Prenez bien le temps d'analyser la structure de l'objet JSON à l'intérieur de ce fichier puisque c'est lui que vous devrez manipuler et envoyer à votre client.

Programmation de la logique de base

Dans votre fichier **app.js**, importez d'abord les modules nécessaires (**express** et **cors**), initialisez l'application puis assignez le numéro de port **3005** à une constante. Ces étapes sont expliquées plus en détail dans le laboratoire 10 que vous avez déjà complété.

Ajoutez la ligne suivante à votre serveur pour qu'il puisse consommer les données au format JSON.

```
app.use(express.json());
```

Ajoutez la ligne suivante à votre fichier ***app.js*** pour permettre le ***cross-origin***.

```
app.use(cors());
```

Par la suite, utilisez la méthode **.listen()** de votre application pour définir une fonction d'écoute du serveur sur le port 3005.

Contraintes

- ☐ Le serveur doit permettre le ***cross-origin***
- ☐ L'appel de la méthode **.listen()** doit afficher un message de confirmation dans la console.
- ☐ Ce message doit contenir le numéro du port sur lequel le serveur est à l'écoute.

Programmation des routes

Votre serveur comprendra deux routes pour répondre aux requêtes du client.

La première route, « `/pays` », utilise la méthode GET. Cette route doit simplement retourner au client l'objet contenu dans le fichier **pays.json**.

La seconde route, « `/langues` », utilise aussi la méthode GET. Cette route recevra un paramètre (*request.query*) du client contenant le nom d'un continent.

Les paramètres de requête sont accessibles sur votre application serveur en utilisant l'objet `request.query` lors de la réception d'une requête. Du côté client, vous passerez un paramètre en l'entrant directement dans l'URL de la requête. Voici un exemple de syntaxe à respecter pour la requête du côté client :

```
fetch("http://localhost:3005/langues?continent=amérique+du+nord");
```

Le serveur pourra dans ce cas accéder à la chaine de caractère « amérique du nord » en utilisant `req.query.continent`. Notez qu'on évite les espaces en les replaçant par des « + ». Vous devrez donc garder cela en tête lors de la programmation de votre application client.

À la réception de la requête du client, votre serveur doit donc analyser la requête pour déterminer le continent ciblé en comparant le paramètre reçu aux continents de la liste des pays. N'oubliez pas que les continents de la liste des pays contiennent aussi des espaces et que vous devrez donc modifier la chaine de caractère lorsque vous ferez la comparaison. Finalement, vous devez retourner un objet JSON contenant une liste de toutes les langues officielles des pays de ce continent. La structure exacte de l'objet JSON est à votre discrétion, mais vous devrez parcourir l'objet du fichier **pays.json** pour construire la liste à retourner.

Contraintes

- ☐ Les deux routes retournent un objet JSON
- ☐ La liste des langues officielles est construite à partir du fichier **pays.json** en comparant les paramètres de la requête au continent des pays.
- ☐ La liste retournée au client ne contient pas de doublon (par exemple, ne pas mettre « Anglais » deux fois même s'il apparait dans deux pays différents)
- ☐ La liste contient seulement les langues officielles des pays du continent choisi par le client

2. Deuxième partie : client React (75%)

Création du projet

Dans le dossier ***client***, lancez la commande pour initialiser une application React avec Node. Lorsque l'installation est terminée, vous pouvez lancer l'application puis commencer la programmation de la logique de votre client.

Structure générale du client

Pour vous donner une idée de l'architecture de votre client, votre composant principal devrait avoir une structure similaire lorsque vous aurez terminé :

```
function App() {
  return (
    <div className="App" style={{display: 'flex', flexDirection: 'column', minHeight: '100vh'}}>
      <BrowserRouter>
        <Header />
        <Routes>
          <Route exact path="/" element={<Navigate to="/pays" />} />
          <Route path="/pays" element={<Pays />} />
          <Route path="/langues" element={<Langues />} />
        </Routes>
        <Footer />
      </BrowserRouter>
    </div>
  );
}
```

L'application comportera deux sections de contenu principales : Pays et Langues. L'utilisateur pourra naviguer d'une section à l'autre à l'aide d'un menu de navigation situé dans le Header.

*Note : Puisque vous développerez vos composants un à la suite de l'autre, il est tout à fait normal de ne pas avoir la structure indiquée dès le début de votre projet. Une tactique acceptable pour le développement est d'afficher le composant sur lequel vous travaillez directement dans le composant **App**. Par exemple, comme vous développerez le composant **ListePays** en premier, vous pourriez d'abord le placer directement dans votre composant **App** pour voir le résultat à l'écran.

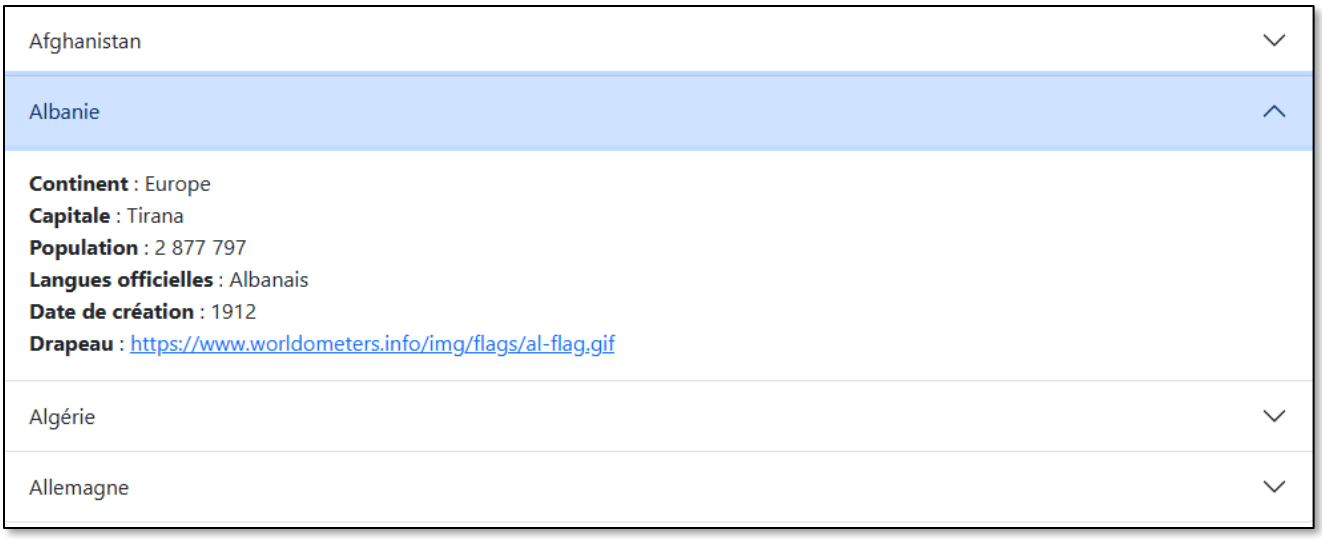
Composant ListePays

Dans un premier temps, vous devez programmer un composant qui se nomme **ListePays**. Ce composant sera responsable de l'affichage de la liste des pays.

Contraintes

- ☐ Ce composant doit recevoir en *props* un tableau d'objets de pays (qui auront la même structure que les pays retournés par notre serveur). Ce *props* sera passé par le composant parent, **Pays**, qui est décrit plus tard dans l'énoncé.
- ☐ Ce composant doit retourner un élément ***Accordion*** de React-Bootstrap (voir le lien vers la documentation plus bas dans l'énoncé).
- ☐ L'élément ***Accordion*** doit contenir un élément ***Accordion.Item*** pour chaque pays de la liste passé en *props*.
- ☐ Le titre de chaque item doit être le nom du pays et le contenu déroulé de chaque item doit être les informations du pays.
- ☐ L'affichage de la population doit utiliser des séparateurs de milliers (par exemple, 3000000 devient 3 000 000). *Indice : une méthode JavaScript existante vous donnera ce résultat en convertissant un nombre en chaine de caractère à l'aide du *locale*.
- ☐ S'il y a plus d'une langue officielle dans le pays, toutes les langues doivent être séparées par une virgule et un espace (par exemple : « Français, Anglais »).
- ☐ Le lien vers le drapeau du pays doit être fonctionnel.
- ☐ Il doit y avoir un retour à la ligne entre chaque information d'un pays.

Exemple du UI pour le composant ListePays :



Documentation pour ***Bootstrap Accordion*** :
https://react-bootstrap.netlify.app/docs/components/accordion/

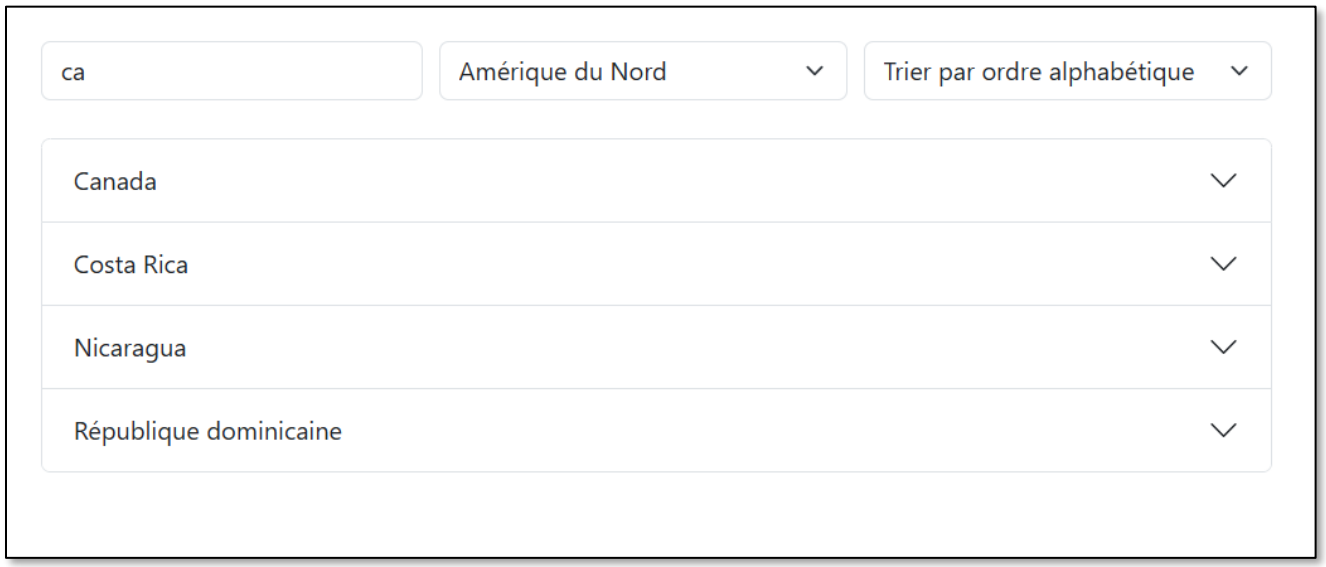
Composant Pays

Ce composant est responsable de l'affichage du contenu de la page Pays. Il est enfant du composant **App**. Il contiendra d'une part les champs nécessaires pour que l'utilisateur puisse chercher un pays par nom et filtrer les pays par continent, et d'autre part le composant **ListePays** pour afficher les pays. De plus, ce composant sera responsable de réaliser la requête serveur pour obtenir la liste JSON des pays.

Dès le chargement de votre composant **Pays**, celui-ci devra effectuer une requête fetch() vers le serveur pour recevoir la liste des pays. Cette liste doit ensuite être conservée dans un état.

Au-dessus de l'affichage de **ListePays**, un filtre devra être affiché pour favoriser la recherche de pays. De plus, deux filtres facultatifs pourront être implémentés en bonus (jusqu'à 15% de plus, voir la description plus bas dans l'énoncé). Toutes les opérations de filtrage et de triage sur la liste originale sont réalisées du côté client, c'est-à-dire que vous ne devriez pas faire d'autre appel serveur dans ce composant après avoir récupéré la liste de pays.

Exemple du UI pour les filtres du composant **Pays** :



Filtre par chaine de caractère

Tout d'abord, cette partie de votre application doit contenir une balise d'entrée de texte (vous pouvez utiliser le composant ***Form.Control*** de Bootstrap). À chaque fois qu'un utilisateur change la valeur du texte, seuls les pays contenant la chaine de caractères entrée doivent être affichés.

Exemple du comportement voulu :

- L'utilisateur tape la lettre « C » dans la zone de texte. Seuls les pays contenant la lettre « C » sont affichés à l'écran.
- L'utilisateur ajoute la lettre « AR » après la lettre « C ». Seuls « Madagascar » et « Nicaragua » sont affichés à l'écran.

*Indice : Il pourrait être utile de créer d'autres états, notamment pour conserver la version filtrée de la liste des pays à des fins d'affichage, puis de passer cette liste au composant **ListePays**.

Contraintes

- ☐ Le filtre doit être appliqué dès qu'il est changé (à chaque fois que l'utilisateur tape/efface un caractère).
- ☐ Le filtre ne doit pas être sensible à la case (il ne devrait pas y avoir de différence entre les lettres majuscules et minuscules).

Filtre par continent

Ce filtre devra être représenté par une liste déroulante située à droite du filtre par chaine de caractères. Bootstrap propose un composant de liste déroulante que vous pouvez utiliser. Les options disponibles dans cette liste déroulante sont :

- Tous les continents *(option par défaut)*
- Afrique
- Amérique du Nord
- Amérique du Sud
- Asie
- Europe
- Océanie

Lorsqu'un utilisateur sélectionne un continent différent, le filtre est aussitôt appliqué et seuls les pays sur le continent correspondant sont affichés.

Options de tri

Le troisième et dernier filtre disponible est une option pour trier les pays dans la liste affichée. Cette fonctionnalité sera aussi réalisée avec une liste déroulante comme le filtre par continent. Les options disponibles dans cette liste déroulante sont :

- Trier par ordre alphabétique *(option par défaut)*
- Trier par population

Attention, la liste que vous recevez du serveur n'est pas parfaitement triée par ordre alphabétique. Pour ce qui est du tri par population, il doit être en ordre décroissant (du pays ayant la plus grande population au pays ayant la plus petite). Le tri doit être réalisé automatiquement au chargement de la page, puis chaque fois que l'utilisateur choisira une option différente dans le menu déroulant.

Important! Tous les filtres doivent fonctionner ensemble. Par exemple, si je change le continent pour Amérique du Nord et que j'écris une chaine de caractère dans le filtre par caractère, les deux filtres (et l'option de tri) doivent tous s'appliquer en même temps!

Composant Header

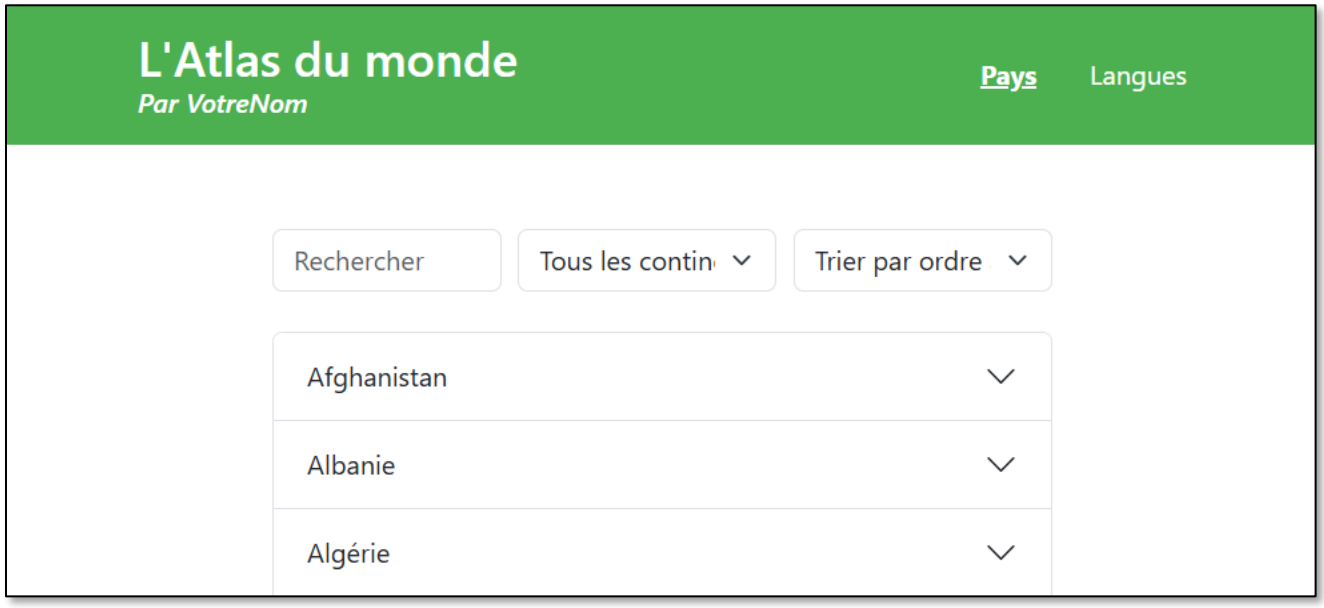
Le composant **Header** contient l'entête de votre application Web. Il est enfant du composant **App**. Il s'agit principalement d'un composant d'affichage qui contient peu de logique. Il vous servira toutefois de menu pour naviguer entre les deux pages de votre application.

Dans un premier temps, installez la librairie react-router-dom avec la commande :

```
npm install react-router-dom
```

Ensuite, consultez la documentation en ligne concernant react-router pour comprendre le fonctionnement général. Vous devrez importer et utiliser les composants **BrowserRouter**, **Routes** et **Route** dans votre composant App. Vous devrez également importer et utiliser le composant **Link** dans votre composant **Header** pour implémenter vos liens de navigation. Tous ces composants proviennent de la librairie que vous venez d'installer.

Exemple de UI du composant **Header** :



Vous êtes libre de donner le style que vous voulez à votre entête, tant qu'il contient répond aux contraintes ci-dessous.

Contraintes

- ☐ L'entête doit contenir le titre « L'Atlas du monde ».
- ☐ L'entête doit contenir votre ou vos noms.
- ☐ L'entête doit contenir deux liens de navigation. Le lien Pays affiche le composant **Pays**, alors que le lien Langues affiche le composant **Langues**.
- ☐ Lorsque votre application est lancée ou lorsque l'on navigue vers le lien « / », l'utilisateur doit être redirigé vers le lien « /pays ».

Composant Langues

Ce composant est responsable de l'affichage du contenu de la page Langues. Il est enfant du composant **App**. Il contiendra un menu déroulant avec la liste des continents ainsi qu'un message indiquant toutes les langues officielles parlées sur ce continent. De plus, ce composant sera responsable de réaliser la requête serveur pour obtenir la liste JSON des langues.

Comme pour le filtre par continent, vous pouvez utiliser le composant Bootstrap Form.select pour implémenter ce menu déroulant. Le menu déroulant doit contenir les options suivantes :

- Choisissez un continent *(option par défaut)*
- Afrique
- Amérique du Nord
- Amérique du Sud
- Asie
- Europe
- Océanie

Lorsqu'un utilisateur arrive sur la page ou lorsque l'option « Choisissez un continent » est sélectionnée, le message suivant doit être affiché sous le menu déroulant : « `Choisissez un continent dans la liste déroulante pour voir toutes les langues officielles qu\'on y parle!` ».

Lorsqu'un utilisateur choisit un continent valide dans la liste, une requête doit être envoyée à la route `/langues` de votre serveur. N'oubliez pas que tel que mentionné, l'URL de cette requête doit aussi contenir un paramètre (*query param*) qui définit le nom du continent choisi du côté client. Ce nom ne devrait pas contenir d'espace (on les remplace généralement par des « + » dans les *query params*). Par exemple, si un utilisateur choisit l'Amérique du Sud, vous pourriez générer et utiliser l'URL :

`http://localhost:3005/langues?continent=amérique+du+sud`

Vous pouvez facilement générer automatiquement les paramètres de requête en utilisant le code suivant (où la variable `continent` contient la chaine de caractère de votre continent):

```
const params = new URLSearchParams();
params.append("continent", continent.toLowerCase());
const url = `http://localhost:3005/langues?${params.toString()}`;
```

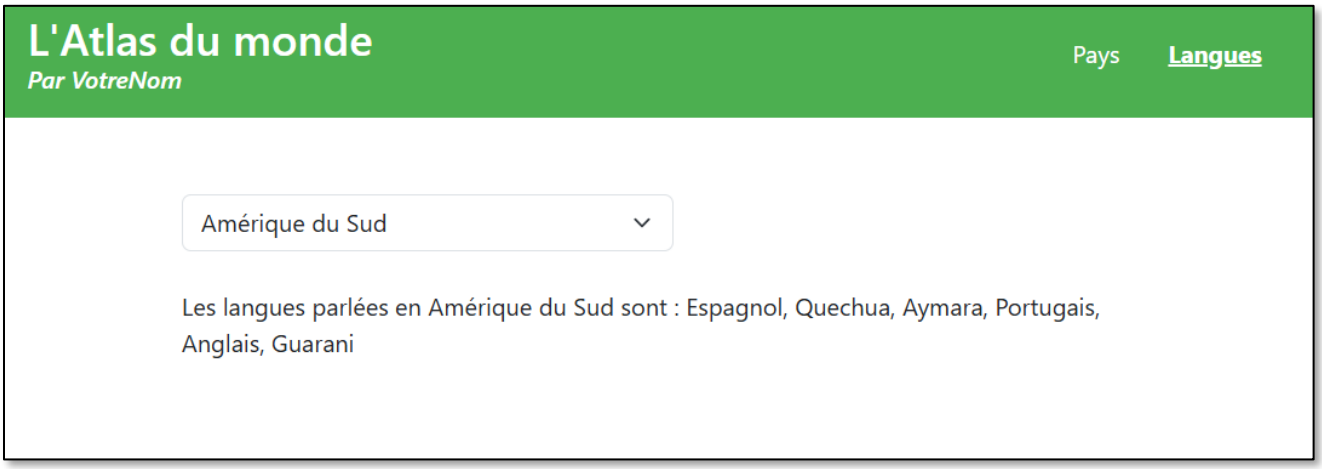
Par la suite, vous n'aurez qu'à réaliser votre requête avec fetch() en passant cette URL en paramètre.

Lorsque vous recevez du serveur le tableau contenant la liste des langues, vous devez formater une chaine de caractère que vous présenterez à l'utilisateur. La chaine devrait avoir la forme suivante :

« `Les langues parlées en {le continent choisi} sont : Langue 1, Langue 2, Langue 3, Langue 4…` »

Faites attention qu'une virgule ne soit pas affichée à la toute fin de la chaine indiquant les langues.

Exemple de UI pour le composant **Langues** :



Contraintes

- ☐ Une requête au serveur doit être réalisée chaque fois que l'utilisateur sélectionne un continent.
- ☐ Le format des chaines de caractère demandé est bien respecté
- ☐ Les langues affichées sont les bonnes.

Composant Footer

Le composant **Footer** contient le pied de page de votre application Web. Il est enfant du composant **App**. Il s'agit principalement d'un composant d'affichage qui ne contient aucune logique.

Votre pied de page devrait contenir au minimum le message « Réalisé par {VosNoms} dans le cadre du cours 420-W33-GG – Automne 2025 ». De plus, il devrait toujours être au moins au bas de l'écran. Pour y arriver, vous pouvez ajouter le code suivant à votre sélecteur .App dans votre fichier de style App.css et vous assurer que vous utilisez bien ce sélecteur à la racine de votre composant App :

```
display: flex;
flex-direction: column;
min-height: 100vh;
```

Exemple de UI du composant **Footer** :

Contraintes

- ☐ Le pied de page contient le message requis
- ☐ Le pied de page est situé à la fin du contenu principal si le contenu dépasse la hauteur de l'écran, sinon au bas de l'écran complètement.

Une fois ce dernier composant terminé, prenez bien le temps de réviser et de tester toutes les fonctionnalités demandées en vous référant au besoin à l'annexe contenant un récapitulatif des requis.

Remise :

Dans la boîte de dépôt Omnivox (section **travaux** du cours), remettez-moi un **dossier compressé** contenant tous vos fichiers de projet (dossiers serveur et client) **sauf les dossiers node_modules de votre client et de votre serveur**.

ANNEXE

Récapitulatif des contraintes à respecter

Serveur

Contraintes générales

- ☐ Le serveur doit permettre le ***cross-origin***
- ☐ L'appel de la méthode **.listen()** doit afficher un message de confirmation dans la console.
- ☐ Ce message doit contenir le numéro du port sur lequel le serveur est à l'écoute.

Contraintes de la route /pays

- ☐ La route se nomme /pays.
- ☐ La route utilise la méthode http GET.
- ☐ La route retourne un objet JSON contenant les pays.

Contraintes de la route /langues

- ☐ La route se nomme /langues.
- ☐ La route utilise la méthode http GET.
- ☐ La liste des langues officielles est construite à partir du fichier **pays.json** en comparant les paramètres de la requête au continent des pays.
- ☐ La liste retournée au client ne contient pas de doublon (par exemple, ne pas mettre « Anglais » deux fois même s'il apparait dans deux pays différents)
- ☐ La liste contient seulement les langues officielles des pays du continent choisi par le client

Client

Contraintes générales

- ☐ Votre client contient deux « pages », Pays et Langues.
- ☐ La navigation est implémentée à l'aide de la librairie react-router-dom.
- ☐ L'entête et le pied de page sont visibles dans les deux pages.

Contraintes du composant ListePays

- ☐ Ce composant doit recevoir en *props* un tableau d'objets de pays. Ce *props* est passé par le composant parent, **Pays**.
- ☐ Ce composant doit retourner un élément ***Accordion*** de React-Bootstrap.
- ☐ L'élément ***Accordion*** doit contenir un élément ***Accordion.Item*** pour chaque pays de la liste passé en *props*.
- ☐ Le titre de chaque item doit être le nom du pays et le contenu déroulé de chaque item doit être les informations du pays.
- ☐ L'affichage de la population doit utiliser des séparateurs de milliers (par exemple, 3000000 devient 3 000 000). *Indice : une méthode JavaScript existante vous donnera ce résultat en convertissant un nombre en chaine de caractère à l'aide du *locale*.
- ☐ S'il y a plus d'une langue officielle dans le pays, toutes les langues doivent être séparées par une virgule et un espace (par exemple : « Français, Anglais »).
- ☐ La liste des langues ne doit pas avoir une virgule à la fin
- ☐ Le lien vers le drapeau du pays doit être fonctionnel.
- ☐ Il doit y avoir un retour à la ligne entre chaque information d'un pays.

Contraintes du composant Pays – Général

- ☐ Le composant **Pays** est parent du composant **ListePays**.
- ☐ Le composant **Pays** comprend aussi les différents filtres qui affectent l'affichage.
- ☐ Les filtres sont affichés au-dessus du composant **ListePays**.
- ☐ La logique des filtres est appliquée côté client (un seul appel au serveur lorsque la page Pays est chargée).

Contraintes du composant Pays – Filtre par nom

- ☐ Un champ de texte permet à l'utilisateur d'entrer des caractères
- ☐ Le filtre doit être appliqué dès qu'il est changé (à chaque fois que l'utilisateur tape/efface un caractère).
- ☐ Le filtre ne doit pas être sensible à la case (il ne devrait pas y avoir de différence entre les lettres majuscules et minuscules).
- ☐ Seul le/les pays qui contiennent la chaine de caractère doivent être affichés.

Contraintes du composant Pays – Filtre par continent

- ☐ L'utilisateur peut choisir un continent à partir d'une liste déroulante.
- ☐ Lorsqu'un utilisateur sélectionne un continent différent, le filtre est aussitôt appliqué et seuls les pays sur le continent correspondant sont affichés.
- ☐ Les autres filtres/tri sont bien appliqué en même temps.

Contraintes du composant Pays – Tri par ordre alphabétique ou population

- ☐ L'utilisateur peut choisir un mode de tri à partir d'une liste déroulante.
- ☐ Par défaut, le tri par ordre alphabétique est appliqué au chargement de la page.
- ☐ Le tri par population est décroissant (pays avec la plus grande population jusqu'à pays avec la plus petite population).
- ☐ Les autres filtres/tri sont bien appliqué en même temps.

Contraintes du composant Langues

- ☐ Une requête au serveur doit être réalisée chaque fois que l'utilisateur sélectionne un continent.
- ☐ Le format des chaines de caractère demandé est bien respecté
- ☐ Les langues affichées sont les bonnes.

Contraintes du composant Header

- ☐ L'entête est toujours situé au-dessus du contenu principal de votre page Web.
- ☐ L'entête doit contenir le titre « L'Atlas du monde ».
- ☐ L'entête doit contenir votre ou vos noms.
- ☐ L'entête doit contenir deux liens de navigation. Le lien Pays affiche le composant **Pays**, alors que le lien Langues affiche le composant **Langues**.
- ☐ Lorsque votre application est lancée ou lorsque l'on navigue vers le lien « / », l'utilisateur doit être redirigé vers le lien « /pays ».

Contraintes du composant Footer

- ☐ Le pied de page contient le message requis
- ☐ Le pied de page est situé à la fin du contenu principal si le contenu dépasse la hauteur de l'écran, sinon au bas de l'écran complètement.