**Appliquez la logique styled components**

Dans styled components, la principale chose à comprendre est que tout est composant. Pour voir cela, créons dès maintenant notre premier styled component (styled composant). npm i styled-components

Dans Card/index.jsx, créons donc le style pour le label.

On a donc:

import styled from 'styled-components'

*const* CardLabel = styled.span``

Et on a réutilisé CardLabel directement dans le JSX :

<CardLabel>{label}</CardLabel>

Mais qu'est-ce que c'est que cette syntaxe bizarre ? 😬

Pas de panique ! Ici, styled-components  utilise des templates literals, que vous pouvez voir [dans la documentation Mozilla](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals). Vous pouvez écrire votre CSS directement à l'intérieur. Ce qui nous donne:

*const* CardLabel = styled.span`

color: #5843e4;

font-size: 22px;

font-weight: bold;

`

Profitons-en pour ajouter du style à notre image. Cette fois-ci, toujours dans le même fichier Card/index.jsx, on a :

*const* CardImage = styled.img`

height: 80px;

width: 80px;

border-radius: 50%;

`

… qu'on utilise dans le code :

<CardImage src={picture} alt="freelance" />

Bravo à vous ! Vous avez généré des éléments **span** et **img** auxquels vous avez appliqué du style avec **styled-components**! 🎉 Vous vous en doutez, ça ne s'arrête pas à **span** et **img:** vous pouvez ainsi générer tous les éléments existants du DOM… mais pas que.

Bravo à vous ! Vous avez généré des éléments span et img auxquels vous avez appliqué du style avec styled-components ! 🎉 Vous vous en doutez, ça ne s'arrête pas à span et img : vous pouvez ainsi générer tous les éléments existants du DOM… mais pas que.

**EN Résumé:**

**Styled components** est une bibliothèque de CSS in JS qui permet de créer du style en appliquant une logique de composants.

* Les template strings utilisées écrites entre ``permettent d'utiliser des variables et de passer des props.
* Il est possible de créer un style global avec createGlobalStyle.

Bravo à vous, maintenant que vous avez intégré du style, l'application que vous codez pour Shiny Agency commence vraiment à prendre forme ! ✨

Dans le prochain chapitre, vous allez tester vos connaissances avec le quiz de cette partie. Alors, bonne chance à vous! 🍀

**Créer une Single Page Application à l'architecture robuste**

Description

Vous êtes maintenant arrivé à la fin de la première partie du cours. C'est le moment de tester vos connaissances. 💪

**Question 1**

**Est-ce que React nous oblige à choisir un type d'organisation de fichier particulier ?**

React laisse les développeurs et les développeuses libres pour organiser les fichiers.

Les réponses 1, 2 et 3 sont bien des manières d'organiser ses fichiers pour une application React, mais ne nous oblige en aucun cas à choisir un type d'organisation particulier. La bonne réponse était la réponse 4.

**Question 2**

**Quel outil permet de repérer facilement l'oubli de la déclaration d'une variable, directement dans son éditeur de code ?**

ESLint + l’extention ESLint est outil qui permet de repérer facilement l’oubli de la déclaration d’une variable, directement dans son éditeur de code.

Create React App est une sorte de boîte à outils qui permet d'installer très rapidement une base de code React avec un ensemble d'outils préconfigurés. Si CRA installe bien des outils, il ne permet pas à proprement parler de faire ce qui est demandé ici. Prettier et son extension permettent de mettre en forme le code, mais pas de signaler une erreur telle que l'oubli de la déclaration d'une variable. La dernière réponse quant à elle n'est même pas une extension existante. La bonne réponse était donc la 1.

**Question 3**

**Quel élément de React Router est le plus fréquemment utilisé pour rediriger un utilisateur vers une autre page de son application lorsqu'il clique dessus ?**

Un composant Link

Si une div qui permet de rediriger un utilisateur avec un redirect en captant la propriété onClick  peut-être considérée comme un élément React Router avec Redirect  , cette méthode n'est pas du tout optimale d'un point de vue accessibilité. Si la balise <a>  permet bien la navigation , elle n'intègre même pas React Router. Le composant <Router  , quant à lui, ne permet pas d'interaction spécifique avec l'utilisateur, mais regroupe l'ensemble des routes possibles. La bonne réponse était donc la réponse 3.

**Question 4**

**Quels sont les deux snippets de code qui, combinés, permettent d'utiliser des paramètres passés dans l'URL ?**

Dans le fichier MonComposant :

import { useParams } from 'react-router-dom'

*function* MonComposant() {

*const* { monParam } = useParams()

return <div>{monParam}</div>

}

Dans le router :

<Route path="/maRoute/:monParam">

<MonComposant />

</Route>

Dans un premier temps, il faut préciser directement dans le router que le paramètre va être utilisé. Pour cela, la bonne syntaxe est celle de la réponse 4. Les paramètres sont ensuite récupérés avec le hook useParams et non passés uniquement dans les props. Les bonnes réponses étaient donc la 2 et la 4.

**Question 5**

**Parmi ces propositions, quels outils permettent d'ajouter une logique de types à votre application ?**

* TypeScript
* PropTypes
* Flow

sont des outils qui permettent d’ajouter une logique de types à votre application.

Dans le cours, nous avons vu comment utiliser les PropTypes pour vérifier le type des props de notre application. Mais il ne s'agit pas de la seule méthode pour exiger des types dans une application. Pour cela, TypeScript et Flow sont des méthodes particulièrement reconnues. Les bonnes réponses étaient donc les 1, 2 et 4.

**Question 6**

**Je code une application pour un site de e-commerce. Pour afficher le prix, je crée un composant  <Price />  . Price a besoin d'une prop amount  qui est un nombre (donc un number  ). Cette prop est indispensable au fonctionnement de mon composant. Comment peut-on exiger de toujours avoir un number  à l'aide des propTypes ?**

Card.propTypes = {

amount: PropTypes.number.isRequired,

}

Afin de déclarer des PropTypes, on utilise le nom de notre composant et on fait .propTypes: les réponses 3 et 4 ne sont pas des options valides. Puis on déclare la prop que notre composant exige, et on déclare son type avec PropTypes (que l'on importe), on fait .number  et on ajoute .isRequired  pour les props qui sont requises. La bonne réponse était donc la réponse 1 ! 🎉

**Question 7**

**J'ai créé un styled component Button. Je veux faire en sorte qu'il soit possible d'avoir un bouton rouge dans certains cas, et un bouton bleu dans d'autres cas. Pour cela, je veux utiliser une prop variation  : si la valeur de variation  est primary  , alors le bouton est bleu, dans tous les autres cas, il est rouge. Quelle est la meilleure manière d'écrire cela ?**

*const* Button = styled.button`

background-color: ${(props) => props.variation === 'primary' ? 'blue' : 'red' };

En soi, la réponse 1 fonctionne. Elle permet bien de changer la couleur du bouton en fonction de variation . Mais ici, notre component se base sur l'élément HTML button: pas besoin du $ devant le nom de la prop.

La réponse 3 ne prend pas en compte le fait que l'énoncé demande spécifiquement de changer le comportement de Button  en fonction d'une prop qui lui est passée. Ici, variation  est une variable globale, donc ça ne fonctionne pas. La réponse 4 ne correspond à aucune syntaxe valide, c'est une sorte de mélange entre l'inline style et styled components.

La bonne réponse était donc la 2. 🎉

**Question 8**

**Je travaille toujours sur le même bouton de la question précédente.**

**Cette fois-ci, je veux que mon bouton grossisse très légèrement au survol de la souris. Comment traduire cela dans mon code ?**

*const* Button = styled.button`

&:hover {

transform: scale(1.1);

}

`

La première option n'est pas valide : il s'agit d'un mélange étrange de JavaScript et de CSS avec onMouseOver. L'option 3 quant à elle correspond à du style appliqué sur une classe hover: ce n'est pas du tout ce qu'on veut.

La quatrième option utilise bien la notation &:  qui permet d'utiliser un pseudosélecteur. Mais mouseOver n'en est pas un. La bonne réponse était donc la réponse 2 ! ☀️

**Exploitez vos connaissances de useState et useEffect pour effectuer des calls API**

**Rafraîchissez vos connaissances de useState et useEffect**

Le state local est présent à l’intérieur d’un composant : ce composant peut être re-render autant de fois que l'on veut, mais les données seront préservées. Pour cela on utilise  useState  , un hook qui permet d'ajouter un state local dans un composant fonction.

useEffect est également un hook, qui permet d'exécuter des actions après le render de nos composants, en choisissant à quel moment et à quelle fréquence cette action doit être exécutée, avec le tableau de dépendances.

Vous vous en doutez sûrement, nous allons les utiliser pour faire des calls API :

* useEffect nous permettra de déclencher le fetch.
* useState permettra de stocker le retour de l'API dans le  state.

**Récupérez les données d'une API**

Reprenez les bases des calls API

Les données sont au cœur d'une application. Qu'il s'agisse de données locales ou bien qu'elles soient récupérées depuis une API, elles viennent **alimenter nos composants** et **nourrir les interactions** avec les utilisateurs.

programmation d’application : c'est un moyen de communication entre deux logiciels. Concrètement, pour nous, c'est ce qui nous permet de **récupérer des données**. Si vous voulez en savoir plus sur les API, je vous conseille l'excellent cours [Adoptez les API REST pour vos projets web](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web).

**Question:** Mais pourquoi on n'a pas mis les données directement dans le front ? On l'a bien fait dans le premier cours et ça fonctionne !

Eh bien... oui, on aurait pu. Ici, on dispose de toutes les données. Mais dans les faits, ce ne sera pas toujours le cas, loin de là. Vous pouvez par exemple avoir besoin que votre contenu soit administré par une personne qui ne sait pas coder. Dans ce cas, vous pourriez [utiliser un CMS](https://openclassrooms.com/fr/courses/5922626-optimisez-le-referencement-de-votre-site-seo-en-ameliorant-ses-performances-techniques/6055246-accelerez-un-site-cree-avec-un-cms) (comme [WordPress](https://wordpress.com/fr/), Ghost, etc.), et récupérer le contenu.

Ou bien tout simplement, vous pouvez créer une application complexe qui requiert un système d'authentification, qui sauvegarde des données utilisateurs, etc. Dans ce cas, une application frontend ne suffit pas et doit être complémentaire avec l'application backend.

Mettons tout ça en application dès maintenant!

Pour notre projet pour l'agence Shiny, nous allons utiliser l'API [que vous trouverez ici](https://github.com/OpenClassrooms-Student-Center/7150606-API-React-intermediaire). Vous trouverez toutes les instructions nécessaires pour la faire tourner dans le README.md. Je vous laisse un petit moment pour cloner le repo et lancer l'API en local.

... C'est bon, vous avez bien l'API qui tourne en local ? On va aller récupérer le contenu de nos questions sur l'API sur la route [http://localhost:8000](http://localhost:8000/) avec [la méthode   HYPERLINK "https://developer.mozilla.org/fr/docs/Web/API/Fetch\_API/Using\_Fetch"fetch HYPERLINK "https://developer.mozilla.org/fr/docs/Web/API/Fetch\_API/Using\_Fetch"()](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch).

fetch()  est la méthode native pour faire des calls API. Nous aurions très bien pu utiliser des outils [tels que  HYPERLINK "https://github.com/axios/axios"axios HYPERLINK "https://github.com/axios/axios"](https://github.com/axios/axios), mais ici la méthode native a été privilégiée, pour vous éviter une nouvelle installation d'outil externe.

***Développez le questionnaire avec les données :***

Pour nous atteler à l'utilisation de l'API, nous allons développer la page /survey. Souvenez-vous, dans le chapitre précédent, nous avions créé des liens pour naviguer entre les questions, et rediriger l'utilisateur sur /results quand il atteignait la dixième question. Eh bien, ici, nous allons continuer à développer cette page afin de récupérer les données depuis l'API. J’ai ajouté un peu de style supplémentaire pour y voir un peu plus clair (que vous pouvez récupérer sur le repo GitHub de ce cours, [sur la branche P2C1-exercice](https://github.com/OpenClassrooms-Student-Center/7150606-React-intermediaire/tree/P2C1-exercice)).

L’API nous renvoie l’ensemble des questions sur l’endpoint [***http://localhost:8000/survey***](http://localhost:8000/survey).

Hé, mais comment on sait ça ?

Bon, c'est facile pour moi parce que j'ai aussi écrit l'API que nous utilisons. Mais vous pouvez tout simplement utiliser la documentation de l'API. [Ici, elle e HYPERLINK "https://github.com/OpenClassrooms-Student-Center/7150606-API-React-intermediaire"st accessible dans le fichier HYPERLINK "https://github.com/OpenClassrooms-Student-Center/7150606-API-React-intermediaire"README.md](https://github.com/OpenClassrooms-Student-Center/7150606-API-React-intermediaire).

On peut donc l’appeler, comme nous l’avons dit, dans notre useEffect pour récupérer les questions. Si vous regardez la documentation, vous verrez que la route correspondant aux questions ***(***[***http://localhost:8000/survey***](http://localhost:8000/survey)***)*** est une route GET  , et qu'elle ne requiert pas de paramètre : on pourra récupérer les données en faisant ***fetch('http://localhost:8000/survey')*** .

Ici, on a donc uniquement besoin d’appeler l’API à la première initialisation de notre composant, et on précise un tableau de dépendances vide dans notre fichier :

useEffect(() => {

fetch(`http://localhost:8000/survey`)

.then((response) => response.json()

.then(({ surveyData }) => *console*.log(surveyData))

.catch((error) => *console*.log(error))

)

}, [])

Ici, nous avons utilisé des Promises. Une autre syntaxe aurait été possible avec des async  / await. Mais attention, il y a une petite subtilité avec useEffect. C'est ce que vous verrez dans le screencast à la fin de ce chapitre.

Bon, ce n’est pas tout d’afficher le retour de notre API dans la console : on veut que ce soit visible dans notre application !

Pour cela, nous allons utiliser le state. À l’aide de useState, on crée donc :

***const [questions, setQuestions] = useState({}) ;***

questions va nous permettre de stocker l’objet qui a été retourné par l’API. À partir de là, on peut exploiter questions assez simplement en appelant:

setQuestions(surveyData).

Ici, vous avez pu voir dans votre console que surveyData est un objet ayant pour clé des nombres. C’est très pratique pour s’assurer que les questions sont toujours ordonnées, et on peut tout simplement accéder à une question avec :

***surveyData[questionNumber]***

De la même manière, pour savoir s’il faut mettre un lien vers le numéro de question suivant, ou bien un lien vers les résultats, vous pouvez tout simplement **vérifier ce que donne l’affirmation**:

***surveyData[questionNumberInt + 1] ?***

Ce qui nous donne le code suivant :

**function Survey() {**

**const { questionNumber } = useParams()**

**const questionNumberInt = parseInt(questionNumber)**

**const prevQuestionNumber = questionNumberInt === 1 ? 1 : questionNumberInt - 1**

**const nextQuestionNumber = questionNumberInt + 1**

**const [surveyData, setSurveyData] = useState({})**

**useEffect(() => {**

**setDataLoading(true)**

**fetch(`http://localhost:8000/survey`)**

**.then((response) => response.json()**

**.then(({ surveyData }) => console.log(surveyData))**

**.catch((error) => console.log(error))**

**)**

**}, [])**

**return (**

**<SurveyContainer>**

**<QuestionTitle>Question {questionNumber}</QuestionTitle>**

**<QuestionContent>{surveyData[questionNumber]} </QuestionContent>**

**<LinkWrapper>**

**<Link to={`/survey/${prevQuestionNumber}`}>Précédent</Link>**

**{surveyData[questionNumberInt + 1] ? (**

**<Link to={`/survey/${nextQuestionNumber}`}>Suivant</Link>**

**) : (**

**<Link to="/results">Résultats</Link>**

**)}**

**</LinkWrapper>**

**</SurveyContainer>**

**)**

**}**

**export default Survey ;**

**Créez un state loading**

Une pratique très répandue consiste à mettre un petit loader pour signifier que les données vont bientôt s’afficher. On pourrait mettre un simple texte “Chargement...”, mais bon, on sait manier le CSS : autant s’amuser avec, non ?

Je vous propose de créer un simple Loader en CSS, directement dans le fichier utils/Atoms.jsx. Pour cela, on a également besoin d’importer keyframes depuis la bibliothèque styled-components. Ce qui nous donne:

**import colors from './colors'**

**import styled, { keyframes } from 'styled-components'**

**const rotate = keyframes`**

**from {**

**transform: rotate(0deg);**

**}**

**to {**

**transform: rotate(360deg);**

**}**

**`**

**export const Loader = styled.div`**

**padding: 10px;**

**border: 6px solid ${colors.primary};**

**border-bottom-color: transparent;**

**border-radius: 22px;**

**animation: ${rotate} 1s infinite linear;**

**height: 0;**

**width: 0;**

**`**

Ici, ce CSS n’est pas l’objet du cours, donc pas de panique si vous ne comprenez pas tout : sachez juste qu’il s’agit d’une utilisation un peu détournée pour avoir un Loader en CSS pur. On va maintenant **utiliser le state pour afficher notre**Loader. Pour cela, on crée une variable isDataLoading avec useState:

*const* [isDataLoading, setDataLoading] = useState(false);

Dans le useEffect, on vient **modifier notre booléen**:

**useEffect(() => {**

**setDataLoading(true)**

**fetch(`http://localhost:8000/survey`)**

**.then((response) => response.json())**

**.then(({ surveyData }) => {**

**setSurveyData(surveyData)**

**setDataLoading(false)**

**}) }, [])**

.. ce qui nous permet ainsi de conditionner le rendu de notre composant : le Loader  s’affiche tant que les données chargent, et une fois qu’on les a bien, le contenu de la question s’affiche à la place du Loader.

<SurveyContainer>

<QuestionTitle>Question {questionNumber}</QuestionTitle>

{isDataLoading ? (

<Loader />

) : (

<QuestionContent>{surveyData[questionNumber]}</QuestionContent>

)}

...

</SurveyContainer>

Yaaaaaaaay ! 🎉 On a bien notre contenu qui s’affiche comme on le souhaitait !!

Pour cela, vous allez devoir :

* Récupérer les profils des freelances sur l’endpoint /freelances de l’API. Vous pouvez utiliser la syntaxe **async / await**  ou les **.then**  .
* Utiliser le Loader  lorsque le contenu des profils freelances est en train de charger.
* Afficher les données dans la page.
* Afficher une erreur s'il y a eu un problème.

**En résumé**

* Les calls API peuvent s’effectuer simplement avec les hooks **useEffect** et **useState** :
* **useEffect** permet de déclencher l’appel API ;
* et **useState**  permet de stocker les données qui sont retournées.
* Il est tout à fait possible d’utiliser des promesses ou des **async / await**  pour effectuer des calls asynchrones en React.
* En termes d’UI, il est possible de créer un state **loading** pour afficher une animation de chargement pendant que les données sont en train de charger.

*Je vous avais dit dans le premier cours que les hooks***useState***et***useEffect***étaient particulièrement utiles. Je ne vous avais pas menti : en plus de pouvoir créer des interactions locales, ils permettent même d'interagir avec des services extérieurs ! Si c’est pas la belle vie. 😎*

*Nous allons maintenant nous plonger un peu plus loin dans le monde des hooks avec***useContext***, et en profiter pour découvrir le Contexte, qui va nous permettre de partager simplement nos données entre nos composants. Alors c’est partiii ! 🚀*

**Découvrez le Contexte dans React**

*Dans la famille des hooks, je veux maintenant celui qui nous permet d’utiliser simplement le Contexte de React !*

*Mais d’ailleurs, React Context, c’est quoi ? 🤔*

*Contexte est un moyen de partager simplement les props entre les composants. Contexte est natif à React et ne nécessite pas d'installer quoi que ce soit de plus.*

*Si vous avez suivi le cours précédent, vous avez vu comment*[*passer de simples HYPERLINK "https://openclassrooms.com/fr/courses/7008001-debutez-avec-react"props HYPERLINK "https://openclassrooms.com/fr/courses/7008001-debutez-avec-react" entre les composants parents et enfants*](https://openclassrooms.com/fr/courses/7008001-debutez-avec-react)*, et comment utiliser les props pour faire remonter le state,*[*au chapitre “Partagez votre state entre différents composants"*](https://openclassrooms.com/fr/courses/7008001-debutez-avec-react/7136837-partagez-votre-state-entre-differents-composants)*. Mais est-ce que vous imaginez ce que ça pourrait donner dans une application complexe, où pour passer une prop à un composant enfant, vous devez le faire passer par des dizaines de composants parents qui n’ont eux-mêmes pas besoin de cette prop ?*

Parent (props.data) Enfant (props.data) Enfant (props.data) Enfant (props.data) Enfant…