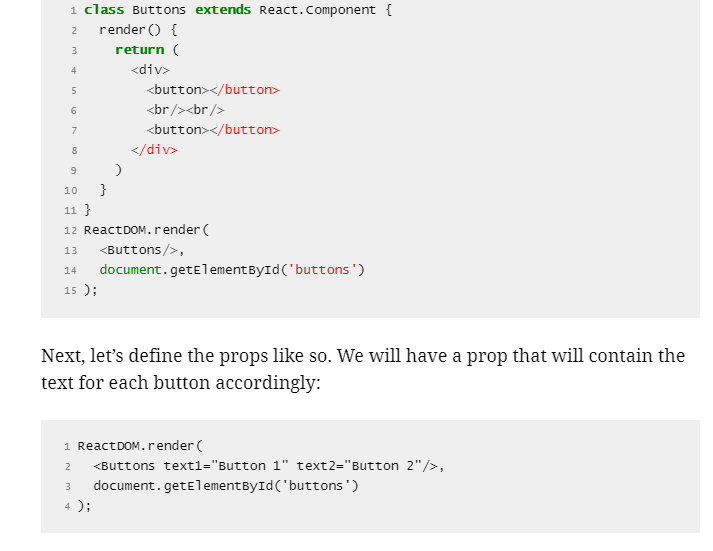
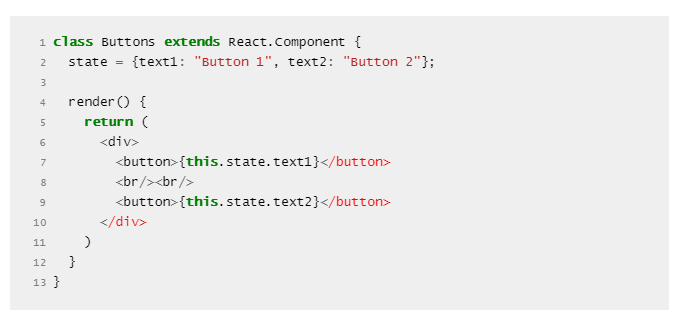
1. **Props**
2. **State**

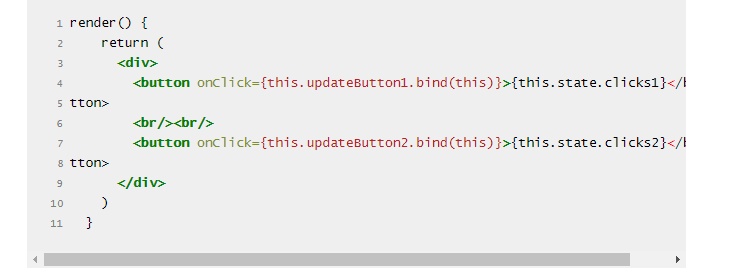
**state is not part of the village design where I’m going to come here and add to component inception and say a state is a collection of villages.**

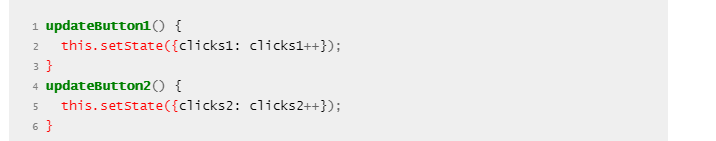
**State is another way encapsulate data in a React component.Unlike props, state is used for data that is updated.**

**For example, think of a weather app.**

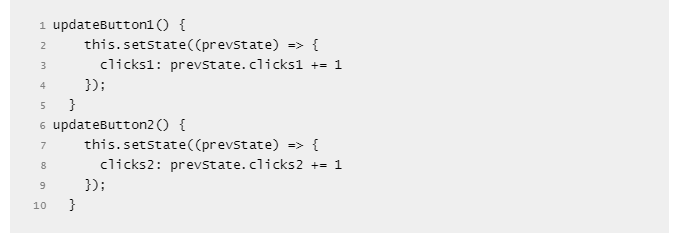
****

1. **Event**





Or we use setState :



En résumé

* Il existe deux types de composants en React, les composants de classes et de fonctions. Il est recommandé d’utiliser les composants de fonctions.
* Un composant React peut prendre en entrée une ou plusieurs *props*, qui sont l’équivalent des attributs en HTML.
* Un composant React retourne une portion de DOM virtuel.
* Nous décrivons notre DOM virtuel avec la syntaxe JSX, car c’est la méthode la plus recommandée.
* Les Hooks permettent de palier aux limites aux composants de fonction, en leur ajoutant des fonctionnalités supplémentaires.
* Le Hook d’état permet de définir un *state* à nos composants.
* Le Hook d’effet permet d’intervenir sur le cycle de vie de nos composants.

Plein écran

Accéder à la session précédente

38. Conclusion

Accéder à la session suivante

40. Afficher les données

En résumé

* La syntaxe {} de JSX permet d’interpréter du code JavaScript dans vos templates, ce qui permet notamment d’afficher les données de nos composants dans les templates.
* On utilise une condition avec l’opérateur logique « && » pour conditionner l’affichage d’un template.
* On utilise l’opérateur ternaire afficher une portion ou une autre d’un template.
* La méthode JavaScript *map* permet d’afficher un tableau de données dans un template.
* Lorsqu’on affiche une liste dans du code JSX, il est important d’appliquer la propriété *key*, sous peine de lever une erreur.
* On peut gérer les interactions d’un utilisateur avec les événements React. Il faut avoir en tête que la syntaxe des événements React n’est jamais identique à celle des événements du DOM. Ainsi, *onclick* devient *onClick* en React.
* On peut récupérer l’événement natif du DOM dans un gestionnaire d’événement, à condition de le passer en dernier paramètre.

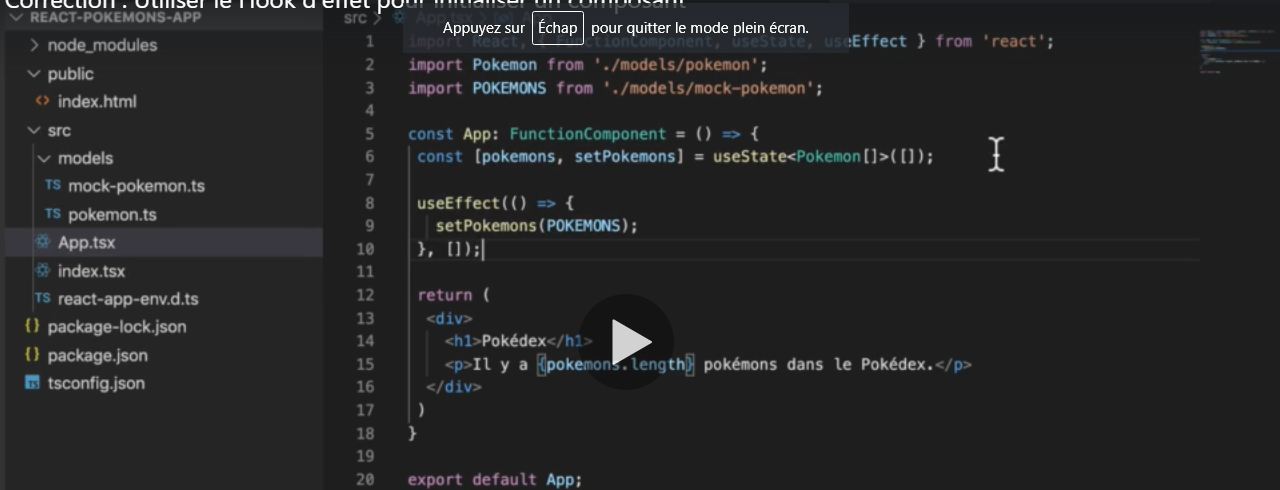
Plein écran

Accéder à la session précédente

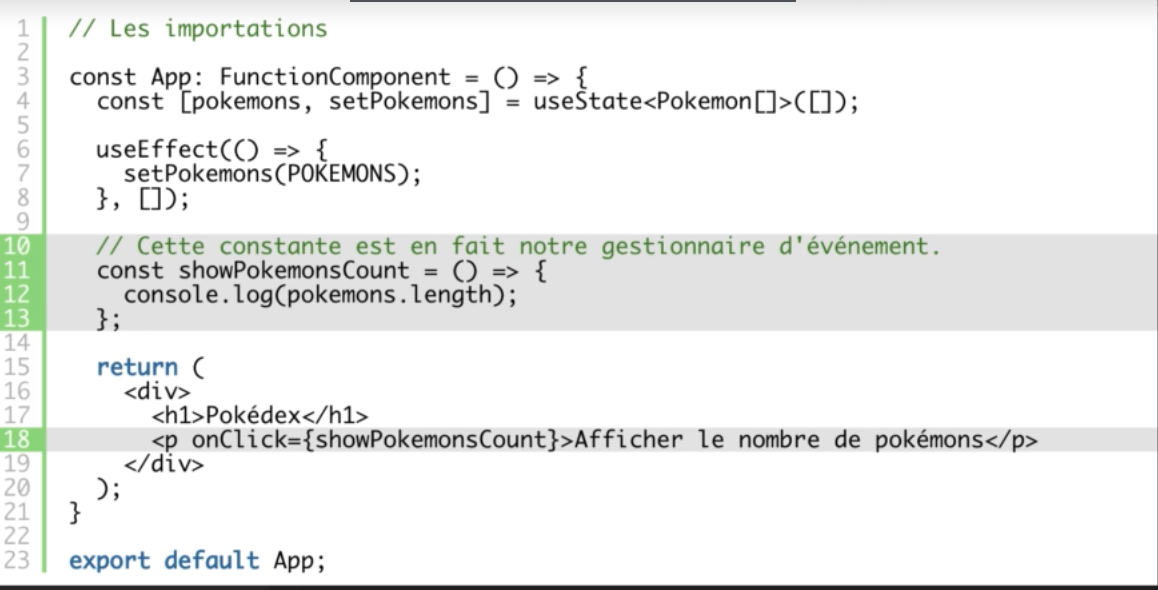
46. Conclusion

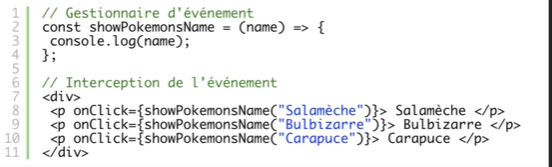
Accéder à la session suivante

48. L'importance des props

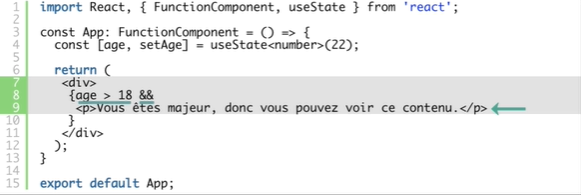


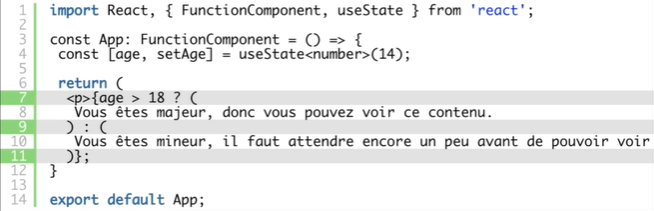
Events

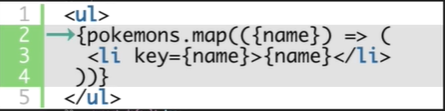


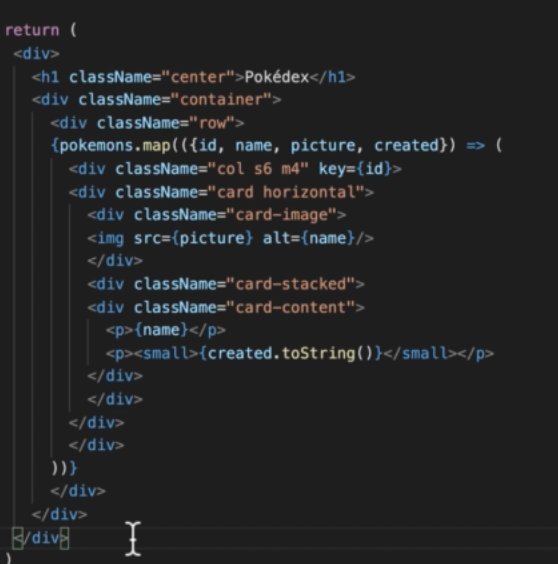


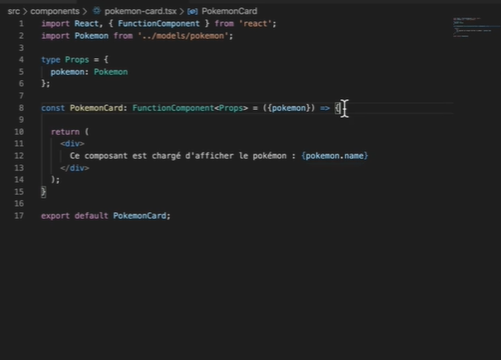
Affichage avec condition :

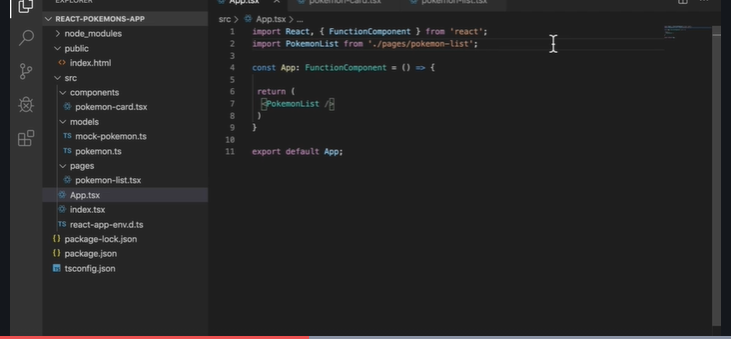


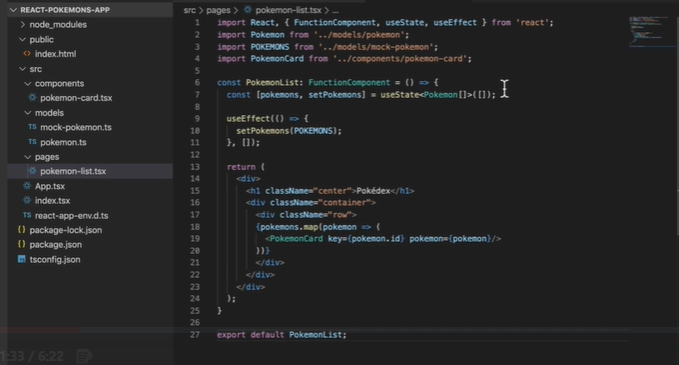


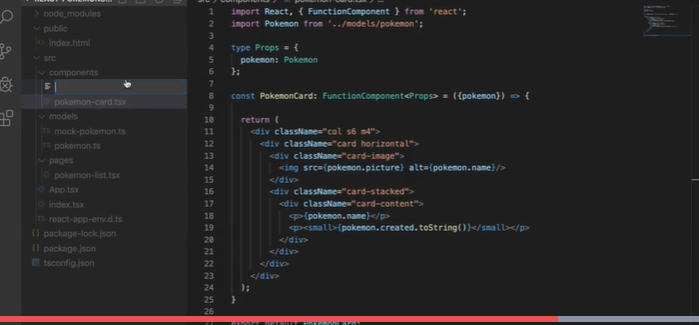


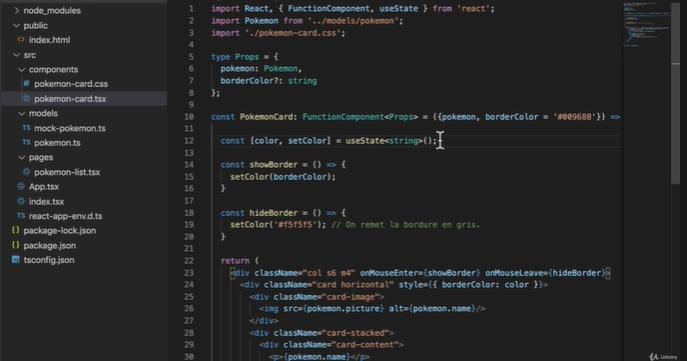












* Les props sont le moyen le plus simple de faire communiquer deux composants entre eux.
* On passe des props depuis un composant parent vers un composant fils.
* Les props permettent de mieux découper notre application en composants plus petits et plus autonomes.
* On peut ajouter un type à nos props, afin de s’assurer que la nature des données reçues par le composant fils.
* On peut définir des props comme facultatives. Il faut alors associer une valeur par défaut à ces props.
* Les props permettent beaucoup de souplesse quant au fonctionnement de nos composants.
* Les propriétés calculées sont simplement des fonctions qui permettent d’effectuer un traitement sur les données d’un composant avant de les afficher à l’utilisateur.
* Il est recommandé de factoriser la logique commune à plusieurs composants dans une fonction à part.
* Si la logique à factoriser implique d’utiliser les Hooks d’un composant, useState et useEffect par exemple, alors il faut utiliser un Hook personnalisé.
* Un Hook personnalisé est une fonction JavaScript, dont le nom commence par "use" , et qui peut appeler d’autre Hooks.
* Un Hook personnalisé peut ensuite être utilisé dans plusieurs composants
* React utilise la librarie react-router-dom pour mettre en place un système de navigation.
* React simule la navigation de l’utilisateur auprès du navigateur, sans que nous n’ayons rien à faire.
* On construit un système de navigation en associant une url et un composant.
* Le routeur de React interprète les routes du haut vers le bas. Il faut donc être prudent quant à l’ordre de déclaration de nos routes.
* Il est possible de faire passer des paramètres depuis nos urls vers un composant, et de typer ces paramètres avec TypeScript.
* La balise <Switch> permet d’injecter le template d’un composant en fonction de l’url demandée par l’utilisateur.
* La balise <Route> permet de définir une route au sein de notre application.
* Pour gérer les erreurs 404 dans notre application, il faut déclarer en dernier un élément Route qui ne prend pas de chemin. Ainsi, toutes les urls seront interceptés par cet élément.
* Il y a deux manières différentes de développer des formulaires avec React: avec les composants contrôlés, ou avec les composants non-contrôlés.
* React recommande fortement d’utiliser les composants contrôlés car ils sont beaucoup moins limités.
* On utilise l’état d’un composant pour sauvegarder la structure et l’état de notre formulaire.
* À chaque modification du formulaire de la part de l’utilisateur, il faut prévoir un traitement spécifique. Autrement dit, il est nécessaire de définir un gestionnaire d’événement pour chacun de nos champs.
* On doit nous même appliquer des règles de validation sur un formulaire. L’utilisation des expressions régulières est une façon de faire pratique, mais libre à vous d’utiliser les traitements que vous souhaitez.
* Si un champ n’est pas valide, il faut toujours prévoir un indicateur visuel à afficher à l’utilisateur. Ce sera beaucoup plus agréable pour lui de remplir votre formulaire en cas d’erreur.
* Il faut toujours effectuer une validation côté serveur en complément de la validation côté client, si vous avez prévu de stocker des données depuis votre application.
* Nous avons utiliser l’API Fetch pour effectuer des requêtes HTTP sur le réseau. Il existe bien sûr d’autres clients HTTP.
* Une requête HTTP est constitué d’une url à appelée, et éventuellement d’une en-tête et d’un corps.
* Il est possible de mettre en place une API Rest de démonstration au sein de votre application. Cela vous permettra d’interagir avec un jeu de données configuré à l’avance.
* Il est rapidement indispensable de maîtriser les quatre opérations de base nécessaire pour interagir avec votre API Rest : ajout, récupération, modification et suppression.
* Les quatre opérations de base pour interagir avec une API Rest sont appelées les opérations CRUD.
* Les API Rest mettent parfois à notre disposition des urls plus avancées que juste les requêtes classiques.
* La souplesse permise par React à travers les composants, le state et les services, permet de développer toutes les fonctionnalités imaginables.
* Les API Rest ne renvoient jamais un résultat instantanément, la requête envoyé au serveur prend toujours un certains temps avant de retourner une réponse.
* Il faut donc prévoir que notre application n’aura pas toujours des données à disposition immédiatement, et anticiper le temps d’attente des requêtes