



Bienvenue

ReactJS - Développement d'applications Web

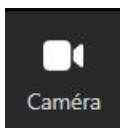
Votre formateur :
Quentin BIDOLET





Avant de démarrer

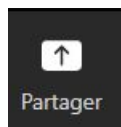
- Règles de fonctionnement



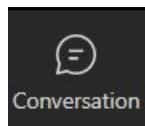
Si possible, pour faciliter les échanges :
Activez votre webcam



Pour le confort de tous
Désactivez votre micro



Pour vous accompagner durant les ateliers
Partagez votre écran



Pour échanger, partager...
Utilisez l'outil de conversation (Tchat)



**Avant de
commencer**

Quentin BIDOLET

- Plus de 5 ans d'expérience en développement web
- Un profil fullstack, je fais aussi bien du back que du front
- Développement représente 70 % de mon activité
- La formation représente 30 % de mon activité



Avant de commencer

- Participants :
- Les bases du HTML, CSS, JS
- Droits admin obligatoires sur les postes
- Quelles sont vos attentes par rapport à cette formation ?
- Quelles sont les technos et langages sur lesquels vous travaillez habituellement ?





- Accès à Slack
- Merci de bien vouloir me rejoindre sur Slack via le lien que je vais vous fournir.





**Avant de
commencer**

- Durée de la formation : 3 jours
- Organisation
 - ▶ Horaires - 9h à 17h-17h30
 - ▶ Pauses : 15 min en matinée et après midi (vers 10h30 et 15h30)
 - ▶ Déjeuner : 12h30 à 13h45





**Avant de
commencer**

Le processus pédagogique

- Pour chaque notion abordée :
 - ▶ Ce que nous voulons faire, quel est l'objectif.
 - ▶ La partie théorique.
 - ▶ La pratique en codant.
 - ▶ Le code sur Slack pour ceux qui veulent faire un copier-coller.



ib
cegos

Emargement



Plan de la formation

J 1 - Matin

Introduction et rappels ES6

Atelier

Le framework React.js

Atelier

J 2 - Matin

Les Hooks

Atelier

Les événements

Atelier

J 3 - Matin

Le routing et la navigation

Atelier

Introduction à Redux et architecture flux

Atelier

J 1 – Après midi

Le JSX et les composants

Les props

Le State et les lifecycles

Atelier

J 2 – Après midi

Rendu conditionnel et liste

Atelier

Les formulaires

Atelier

J 3 – Après midi

Atelier

Les tests

Atelier



1 - Introduction et rappels ES6





Installez NodeJS

- <https://nodejs.org/>
- Version LTS (tout le monde la même version).
- Ouvrez un terminal et exécutez `node -v` et `npm -v` pour vérifier que Node.js et npm ont été correctement installés.



Le standard Ecmascript

- Standardise le JavaScript
- ES5, ES6, ES7, ES8, ES9, ES10, ES11
- ES, ES2015, ES2016, ES2017, ES2018, ES2019, ES2020
- Chaque version apporte son lot de nouveautés
- Tous les navigateurs implémentent ES5
- Les navigateurs les plus récents implémentent ES6
- Utiliser différentes versions de ES dans une application implique qu'il va falloir compiler le Js en ES5 ou ES6



Ecmascript 2015 (ES6)

- Les classes JavaScript
- Les fonctions fléchées (Arrow functions)
- Les modèles de chaîne de caractères (Template literals)
- La déstructuration (Destructuring)
- Les paramètres par défaut (Default parameters)
- Les modules JavaScript (import/export)
- Nous allons utiliser ES6 pendant cette formation.

Installez VsCode et les plugins

- [Visual Studio Code](#)
- [Raccourcis](#)
- [Eslint](#)
- [Prettier](#)
- [Babel JavaScript](#)
- [Path Intellisense](#)
- [Auto Import - ES6, TS, JSX, TSX](#)



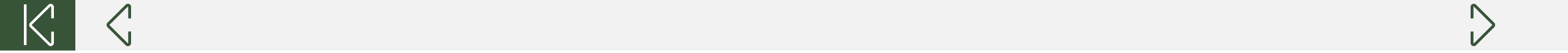
React developer tools

- Une **extension de navigateur** pour Chrome et Firefox qui permet d'inspecter et de déboguer les composants React.
- **Inspection des Composants** : L'outil permet aux développeurs de sélectionner et d'inspecter chaque composant React dans l'arborescence du DOM.
- **Visualisation des Props et État** : Il montre les propriétés et l'état de chaque composant.
- **Performance et Optimisation** : L'outil offre des fonctionnalités pour analyser la performance des composants.



npm

- **npm** : Le gestionnaire de paquets officiel pour Node.js. Il permet **d'installer**, de **partager** et de **gérer les dépendances** dans les projets JavaScript.
- **package.json** : Ce fichier contient des **métadonnées** sur le projet, y compris les **dépendances**, les **scripts**, la **version** etc.
- **Installation de Paquets** :
npm install <nom_du_paquet>.
- **Gestion des Versions** : Utilise le versionnage sémantique (SemVer) pour gérer les versions de paquets de manière à éviter les conflits.
- **npm Scripts** : Permet d'automatiser des tâches telles que le démarrage de l'application ou les tests via des commandes définies dans package.json.



Let, variables locales et constantes



Variables locales : Les variables déclarées à l'intérieur d'une fonction ou d'un bloc de code. Elles ne sont accessibles que dans ce contexte spécifique.

let : Permet de déclarer des variables locales avec une portée de bloc. Contrairement à var, qui a une portée de fonction, let limite la variable au bloc, à l'instruction ou à l'expression où elle est utilisée.

const : Utilisé pour déclarer des constantes, c'est-à-dire des variables dont la valeur ne peut pas être réaffectée après leur initialisation. const a une portée de bloc, similaire à let.



Commit

Commit **variables.js**





Typage et types natifs

- JavaScript est un langage à **typage dynamique**. Les types de variables ne sont pas déclarés explicitement et peuvent changer durant l'exécution du programme.
- **JavaScript a un typage faible**, par exemple, on peut additionner un nombre et une chaîne de caractères sans erreur explicite, ce qui mènera à une conversion implicite des types.
- **Types natifs :**
 - ▶ **Number** : Pour les nombres entiers et flottants.
 - ▶ **String** : Pour les chaînes de caractères.
 - ▶ **Boolean** : Pour les valeurs vraies ou fausses.
 - ▶ **Undefined** : Pour les variables non initialisées.
 - ▶ **Null** : Représente l'absence délibérée de valeur.
 - ▶ **Object** : Pour les structures de données complexes comme les objets, les tableaux et les fonctions.
 - ▶ **Symbol** (introduit en ES6) : Pour créer des identifiants uniques.



Commit

Commit **typage.js**





Paramètres optionnels

- Un **paramètre optionnel** est un argument qui n'a pas besoin d'être fourni lors de l'appel de la fonction. Si un paramètre optionnel n'est pas fourni, il prend une valeur par défaut.
- Depuis **ES6** (ECMAScript 2015), il est possible d'assigner des valeurs par défaut aux paramètres optionnels directement dans la signature de la fonction.

Commit

Commit **parameters.js**





Classes

- Les classes sont introduites en ES6 pour simplifier la création d'objets et la gestion de l'héritage, en remplaçant l'ancienne approche basée sur les fonctions.
- Une classe se déclare avec le mot-clé **class** suivi d'un nom. Elle peut inclure un **constructeur**, des **méthodes**, des **getters**, des **setters** et des **propriétés statiques**.
- **Héritage** : Les classes supportent l'héritage via le mot-clé **extends**.



Commit

Commit **classes.js**





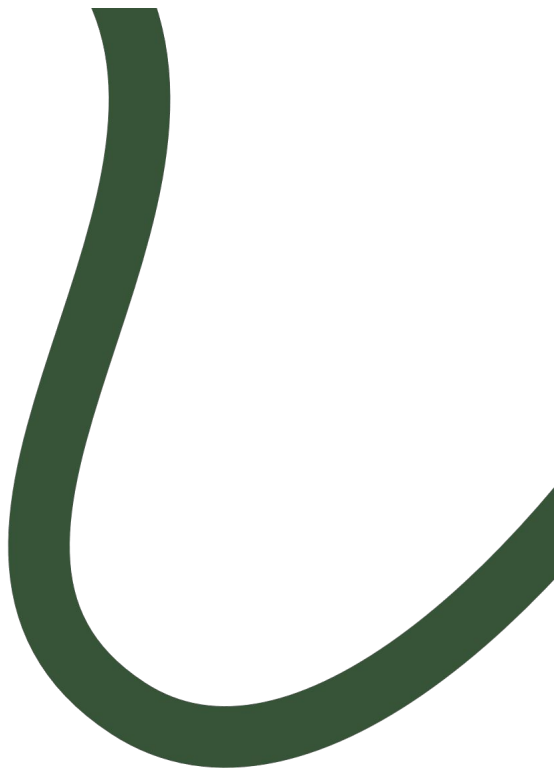
Interfaces

- En JavaScript pur, le concept d'interfaces tel qu'il existe dans les langages de programmation typés statiquement comme Java ou C# n'existe pas.
- React JS, étant basé sur JavaScript, suit la même règle : il n'y a pas de concept d'interface intégré.
- Cependant, vous pouvez simuler un comportement similaire aux interfaces en utilisant **PropTypes**.
- Avec **PropTypes**, vous déclarez les types attendus pour les props, et React vérifie ces types en mode développement, générant des avertissements en cas d'incompatibilité



Commit

Commit **interfaces.js**



Gestion des modules

- JavaScript ES6 introduit un système de modules natif, permettant d'importer et d'exporter des fonctionnalités entre différents fichiers.
- **Export** : Vous pouvez exporter des fonctions, des classes, des objets ou des primitives depuis un fichier JavaScript.
- **Import** : Importez ces fonctionnalités dans d'autres fichiers.
- Modules Nommés (**as**) et Par Défaut (**default**) : JavaScript permet des exportations "nommées" et "par défaut". Les exportations nommées sont utiles pour exporter plusieurs valeurs, tandis que l'exportation par défaut est souvent utilisée pour les composants principaux ou les bibliothèques.



Commit

Commit **modules.js**

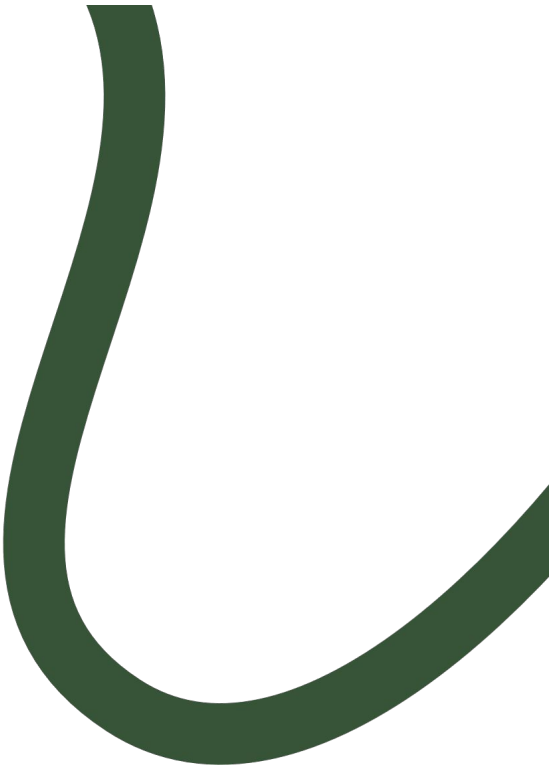


Arrow functions

- Les fonctions fléchées (ou "arrow functions") sont une fonctionnalité introduite dans ES6 pour simplifier l'écriture de fonctions en JavaScript.
- **Forme Basique** : Les fonctions fléchées offrent une syntaxe plus courte. Par exemple, **`(a, b) => a + b`**
- **Sans Paramètres** : **`() => { ... }`**
- **Un Seul Paramètre** : Si la fonction a un seul paramètre, les parenthèses sont optionnelles : **`a => { ... }`**
- **Retour Immédiat** : Pour les fonctions qui retournent immédiatement une valeur, les accolades et le mot-clé return sont optionnels : **`(a, b) => a + b`**

Commit

Commit **arrow-functions.js**





2 - Le framework React.js





Positionnement de ReactJS

- Bibliothèque JavaScript développée par Facebook
- **Avantages** : Grande communauté, flexibilité, performances élevées, utilisation de JSX, compatibilité avec les applications mobiles grâce à React Native.
- **Inconvénients** : Nécessite des bibliothèques tierces pour certaines fonctionnalités, courbe d'apprentissage moyenne.
- **Idéal pour** : Projets de toutes tailles, équipes ayant une préférence pour la flexibilité et la modularité, développement d'applications web et mobiles.
- **Utilisation dans l'industrie** : Facebook, Instagram, Airbnb, Netflix, WhatsApp



Philosophie « composant »

- Les composants sont les **éléments de base** qui permettent de **construire** et de **structurer** une application.
- C'est une **unité indépendante et réutilisable** qui représente une partie de l'interface utilisateur.
- Ils divisent l'interface utilisateur en parties distinctes et modulaires.
- Ils acceptent des entrées quelconque (appelées « **props** ») et renvoient des éléments React décrivant ce qui doit apparaître à l'écran.



Principes clés des composants



- **Réutilisabilité** : Conçus pour être réutilisables, ce qui rend les applications plus faciles à maintenir et à faire évoluer.
- **Propriétés (Props)** : Les props sont des données immuables qui sont passées d'un composant parent à un composant enfant.
- **État (State)** : Permet de gérer leur propre état interne (state), peut être modifié au fil du temps et est utilisé pour stocker des informations sur l'état actuel du composant.
- **Composition** : Peuvent être imbriqués les uns dans les autres pour créer des structures complexes. Cette composition permet de créer des composants plus petits et plus simples qui sont ensuite combinés pour construire des interfaces utilisateur plus élaborées.



Les workflows de développement

- from scratch (customisé)
- intégration à une application web existante
- utilisation d'un outil de création d'une application React (**create-react-app**)





Développement From Scratch

- **Initialisation** : Commencez par initialiser un nouveau projet avec Node.js. Créez un dossier de projet et initialisez-le avec `npm init`.
- **Configuration de Webpack et Babel** : Configurez Webpack pour le bundling et Babel pour la transpilation du code JSX et ES6+. Installez les dépendances nécessaires pour React, Babel, et Webpack.
- **Structure du Projet** : Créez une structure de dossier personnalisée pour les composants, les services, les assets, etc.
- **Environnement de Développement** : Mettez en place un serveur de développement avec hot reloading (par exemple, avec Webpack Dev Server).
- **Ajout de React** : Installez React et ReactDOM. Commencez à construire vos composants.



Intégration à une Application Web Existant

- **Installation de React** : Ajoutez React à votre projet existant via npm ou un CDN.
- **Intégration Progressive** : Commencez par intégrer React dans une petite partie de votre application, comme un widget ou un composant spécifique.
- **Interopérabilité** : Assurez-vous que React coexiste correctement avec les autres parties de votre application, en particulier si d'autres bibliothèques ou frameworks sont en jeu.
- **Refactoring Progressif** : Continuez à refactoriser progressivement les parties de l'application en composants React, selon les besoins.



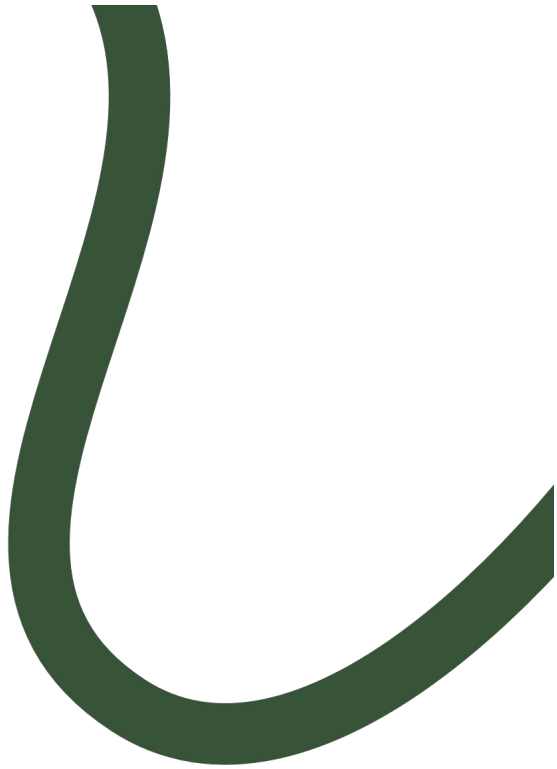
create-react-app

- Un outil officiel pour créer **rapidement** un projet React simple.
- Il inclut un **environnement de développement**, une **configuration Webpack**, et **des scripts** pour construire et déployer votre application.
- **Initialisation avec Create-React-App** : Utilisez **npx create-react-app my-app** pour créer un nouveau projet React.
- Cette commande **installe** et **configure React, Babel**, et **Webpack** automatiquement.
- **Structure et Développement** : create-react-app génère une structure de projet de base.



Commit

Commit **create-react-app.js**





DOM (Document Object Model)

- Le **DOM** est une représentation **en arbre** de la structure d'une page web, il est chargé dans le navigateur.
- Le DOM transforme le code HTML d'une page en une **structure d'objets**, où chaque élément HTML est représenté par un objet (ou "**nœud**") dans l'arbre.
- Lorsqu'une modification est apportée à l'interface utilisateur, **le navigateur met à jour** le DOM pour refléter ces changements.



DOM (Document Object Model)

- Le DOM est une **API** (Application Programming Interface).
- Une API est un **ensemble de règles, de protocoles et d'outils** qui permettent à différents logiciels ou composants de communiquer et d'**interagir entre eux**.
- Le DOM permet aux développeurs de **manipuler** et d'**interagir** avec les éléments HTML en utilisant le JavaScript.
- Cependant, la manipulation du DOM est **coûteuse** en termes de performances, en particulier lorsque de nombreux éléments sont modifiés simultanément.



Virtual DOM avec ReactJS

- Le Virtual DOM est une **représentation légère** et en mémoire du DOM réel sous la forme d'un arbre d'objets JavaScript.
- React utilise le Virtual DOM pour déterminer **quelles parties du DOM réel doivent être mises à jour** lorsque l'état d'un composant change.



Fonctionnement du Virtual DOM

- **Création du Virtual DOM:** Lorsqu'un composant React est rendu le Virtual DOM est créé.
- **Calcul des différences (Diffing):** Lorsqu'un changement d'état se produit dans un composant, React crée un nouveau Virtual DOM pour représenter la nouvelle interface utilisateur. Ensuite, React le compare avec l'ancien pour déterminer les différences entre les deux (ce processus est appelé "diffing").
- **Reconciliation:** Une fois les différences identifiées, React met à jour uniquement les parties du DOM réel qui ont été modifiées. Ce processus est appelé "reconciliation" et permet d'éviter des mises à jour inutiles du DOM.



Avantages du Virtual DOM

- **Performances améliorées** : En ne mettant à jour que les parties du DOM qui ont été modifiées, React minimise le coût des manipulations du DOM, ce qui entraîne de meilleures performances.
- **Simplicité de programmation**: Les développeurs peuvent se concentrer sur la logique de l'application sans se préoccuper des optimisations de performances liées à la manipulation du DOM.
- **Compatibilité entre navigateurs**: Le Virtual DOM gère les différences entre les navigateurs, ce qui simplifie le développement d'applications compatibles avec plusieurs navigateurs.



3 - Le JSX et les composants





Définition d'un élément React

- **Éléments DOM** : Représentent les balises HTML standards, comme `<div>`, `<button>`, `<input>`, etc.
- **Éléments de Composant** : Représentent des composants React définis par l'utilisateur. Ces composants peuvent être des classes ou des fonctions.

Une nouvelle syntaxe : Le JSX

- Une **extension de syntaxe** pour JavaScript qui permet d'écrire des **éléments HTML directement dans le code** JavaScript.
- JSX est utilisé dans React pour décrire l'apparence et la structure des composants.

```
const element = <h1>Bonjour le monde !</h1>;
```



Le plugin de Babel pour le JSX

- **Rôle de Babel** : Babel est un **transpileur** JavaScript qui convertit le code ES6+ (y compris le JSX) en une version de JavaScript compatible avec les navigateurs actuels.
- **JSX en JavaScript** : Le plugin de Babel pour JSX transforme le JSX en appels de fonction JavaScript. Ces appels créent des objets représentant l'interface utilisateur.

Les règles du JSX

- **Fermeture des Balises** : Toutes les balises JSX doivent être fermées. Les balises HTML qui sont normalement auto-fermantes (comme `` et `<input>`) doivent aussi être explicitement fermées en JSX (``, `<input />`).
- **Expressions dans les Accolades** : Pour intégrer des expressions JavaScript dans le JSX, utilisez des accolades `{}`. Cela peut être utilisé pour insérer des variables, des résultats de fonctions, des conditions logiques, etc.
- **Élément Parent Unique** : Une expression JSX doit retourner un seul élément parent. Pour regrouper plusieurs éléments sans ajouter de nœud supplémentaire au DOM, vous pouvez utiliser des fragments `<>...</>`.
- **Syntaxe de Commentaire** : Les commentaires en JSX sont écrits avec la syntaxe `{/* ... */}`.

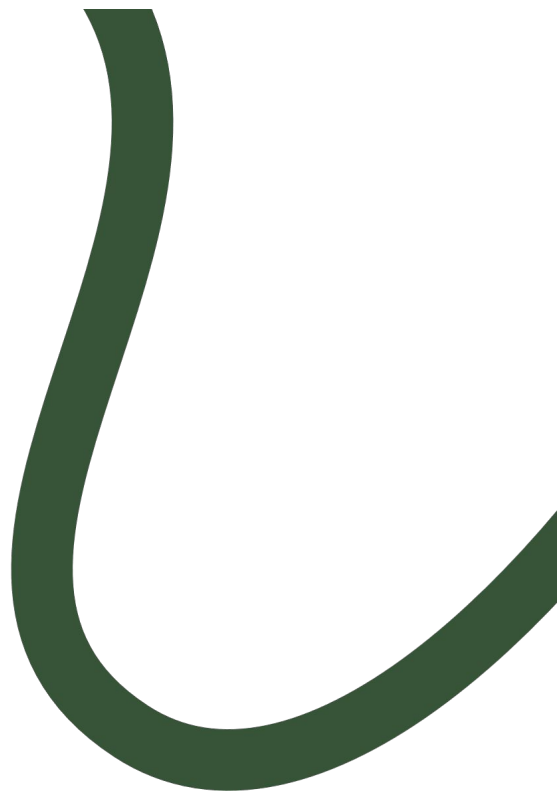


Les attributs JSX

- **Attributs HTML Similaires** : Beaucoup d'attributs JSX sont similaires à ceux que vous trouveriez en HTML, tels que `id`, `className` (équivalent de `class` en HTML), `href` pour les liens, `src` pour les images, etc.
- **Gestionnaires d'Événements** : Les attributs d'événements en JSX commencent souvent par `on`, tels que `onClick`, `onChange`, `onSubmit`, etc. Ils sont utilisés pour définir les réactions aux événements utilisateur.
- **camelCase pour les Attributs** : Les noms d'attributs en JSX suivent la convention `camelCase`, par exemple `onClick` au lieu de `onclick` et `className` au lieu de `class`.
- **Objets pour les Styles en Ligne** : Les styles en ligne en JSX sont définis en utilisant un objet JavaScript. Les noms de propriétés CSS suivent également la convention `camelCase`.



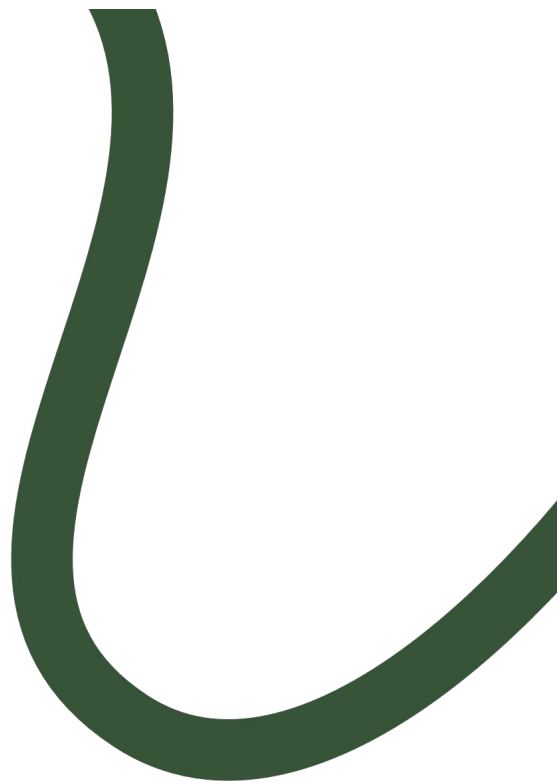
Principes clés des composants



- **Réutilisabilité** : Conçus pour être réutilisables, ce qui rend les applications plus faciles à maintenir et à faire évoluer.
- **Propriétés (Props)** : Les props sont des données immuables qui sont passées d'un composant parent à un composant enfant.
- **État (State)** : Permet de gérer leur propre état interne (state), peut être modifié au fil du temps et est utilisé pour stocker des informations sur l'état actuel du composant.
- **Composition** : Peuvent être imbriqués les uns dans les autres pour créer des structures complexes. Cette composition permet de créer des composants plus petits et plus simples qui sont ensuite combinés pour construire des interfaces utilisateur plus élaborées.



Les composants en mode classe



- Les composants de classe étendent **React.Component**
- **Gestion de l'État** : Les composants de classe peuvent avoir un état local géré à travers **this.state**
- L'état peut être initialisé dans le constructeur ou directement dans la classe.
- **Mise à jour de l'État** : Utilisez **this.setState** pour mettre à jour l'état du composant, ce qui déclenche également un re-render.
- **Binding des Méthodes** : Dans les composants de classe, les méthodes qui traitent des événements ou qui accèdent à **this** doivent être liées au contexte de la classe. Cela peut être fait dans le constructeur ou en utilisant les propriétés de classe avec des fonctions fléchées.



Commit

Commit **composant-classe.js**





Les composants fonctionnels

- **Fonctions Simples** : Les composants fonctionnels sont de simples fonctions JavaScript qui retournent des éléments React (JSX). Ils acceptent un objet de props en argument et retournent du JSX.
- **useState** : Le Hook **useState** permet d'ajouter un état local à un composant fonctionnel.
- **useEffect** : Le Hook **useEffect** remplace plusieurs méthodes de cycle de vie des composants de classe. Il est utilisé pour exécuter des effets secondaires dans le composant.



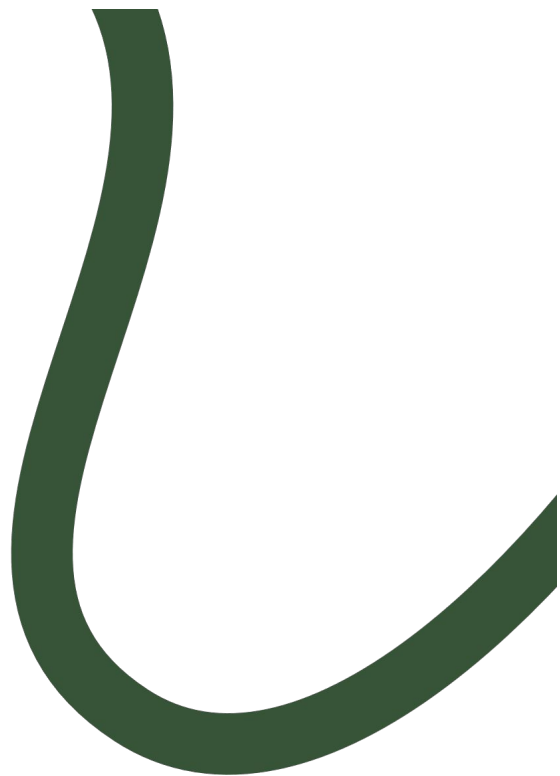
Commit

Commit **composant-fonction.js**





Imbrication de composants



- L'imbrication de composants est une pratique courante en React.
- Elle consiste à insérer des composants à l'intérieur d'autres composants, formant ainsi une hiérarchie ou une arborescence de composants.
- **Réutilisabilité** : Permet de réutiliser des composants, réduisant ainsi la redondance dans le code.
- **Modularité** : Aide à créer une architecture d'application plus modulaire et organisée.
- **Maintenabilité** : Facilite la maintenance du code, car les modifications dans un composant peuvent être isolées de celles des autres.
- **Clarté et Hiérarchie** : Offre une structure claire pour l'interface utilisateur, reflétant visuellement la hiérarchie de l'interface.
- **Composition et Héritage** : Préférez utiliser la composition (imbrication de composants) plutôt que l'héritage pour partager la fonctionnalité entre les composants.



Commit

Commit **imbrication.js**





4 - Les props





Définition

- Les props sont des données que vous passez d'un composant parent à un composant enfant.
- Elles sont utilisées pour **transmettre des informations**.
- Cela permet à ce composant de se comporter de **manière dynamique** ou de s'afficher en fonction des données reçues.
- **Immuables** : Une fois qu'une prop est passée à un composant, elle ne peut pas être modifiée par ce composant.



Envoyer des props

- On peut passer **des données** du composant parent au composant enfant en utilisant les props.
- Vous pouvez **passer des fonctions** via les props pour permettre aux composants enfants de transmettre des informations au composant parent.



Accéder au props

- Dans les composants de classe, les props sont accessibles via **this.props**
- Accessibles via l'**objet props** passé en argument pour les composants fonctionnels.

La props children

- C'est une prop spéciale qui contient tout contenu JSX passé entre les balises ouvrantes et fermantes d'un composant.

```
<MonComposant>  
  <p>Ce paragraphe est passé comme children.</p>  
</MonComposant>
```

Commit

Commit **props.js**





5 - Le State et les lifecycles





Définition

- Le State d'un composant React est un objet qui **contient des données** qui peuvent changer au fil du temps.
- **L'état est local** et encapsulé dans le composant, ce qui signifie qu'il n'est accessible que dans le composant où il est déclaré.



Initialiser le state

- **Composant de classe** : Initialiser l'état dans le constructeur. Utilisez **this.state** pour définir l'état initial.
- **Composant fonctionnel** : Utilisez le Hook **useState** pour initialiser l'état. **useState** retourne un tableau contenant la valeur actuelle de l'état et une fonction pour le mettre à jour.



Commit

Commit **initialiser-state.js**



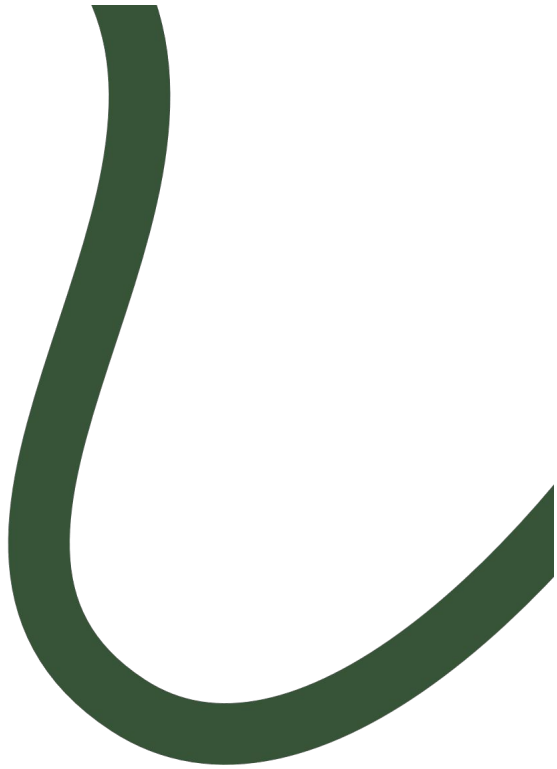


La méthode setState

- **setState** est utilisée pour mettre à jour une partie ou la totalité de l'état d'un composant.
- Les appels à setState sont **asynchrones**.
- React peut regrouper plusieurs appels à setState pour optimiser les performances en effectuant **un seul re-render**.
- **Ne modifiez pas l'état directement** car cela ne déclenchera pas un re-render, utilisez **setState**

Commit

Commit **setState.js**





Cycle de vie

- **Montage (Mounting)** : Phase où le composant **est créé** et inséré dans le DOM
- **Mise à jour (Updating)** : La mise à jour est la phase où le composant est **réexécuté** en raison de changements dans les props, l'état, ou lorsque `forceUpdate()` est appelé.
- **Démontage (Unmounting)** : Le démontage est la phase où le composant est **retiré** du DOM.



Montage du composant

- **componentDidMount** est une méthode appelée immédiatement après que le composant a été inséré dans l'arbre du DOM.
- C'est l'endroit idéal pour effectuer des opérations qui nécessitent que le composant soit présent dans le DOM
- Requêtes de données, initialisation de bibliothèques tierces etc.



Mise à jour du composant

- **componentDidUpdate** est une méthode invoquée immédiatement après la mise à jour d'un composant.
- Cette mise à jour peut être due à des changements dans les props, l'état ou le contexte du composant.
- **componentDidUpdate** est utilisée pour effectuer des opérations qui doivent se produire après qu'une mise à jour a eu lieu et que le DOM a été potentiellement modifié.
- Elle reçoit les anciennes props (**prevProps**) et l'ancien état (**prevState**) comme arguments, vous permettant de comparer l'état et les props actuels avec les précédents.



Démontage du composant

- **componentWillUnmount** est une méthode appelée juste avant que le composant soit retiré et détruit de l'arbre du DOM.
- Cette méthode permet tout nettoyage nécessaire pour éviter les fuites de mémoire et d'autres problèmes de performance.



Best practices

- **Minimisez l'État** : N'utilisez l'état que pour les données qui doivent changer au cours du cycle de vie du composant.
- Traitez l'état comme **immuable**.
- **N'altérez jamais directement this.state**. Utilisez `this.setState` pour des mises à jour prévisibles.
- Lorsque plusieurs valeurs d'état doivent être mises à jour en réponse à un seul événement, **regroupez ces mises à jour dans un seul appel `setState`** pour optimiser les performances.



Commit

Commit **cycle-vie.js**





6 - Les Hooks





Définition

- Les **hooks** sont des fonctions introduites dans React 16.8 qui permettent d'utiliser l'état (state) et d'autres fonctionnalités de React dans les composants fonctionnels, sans avoir besoin de créer des composants basés sur des classes.
- Avant l'introduction des hooks, les composants fonctionnels étaient principalement utilisés pour les composants présentationnels sans état et sans logique de cycle de vie.
- Les hooks ont simplifié la manière d'utiliser et de gérer l'état dans les composants fonctionnels.



Utilisation des hooks

- **useState** : permet d'ajouter un état local à un composant fonctionnel. Il renvoie un tableau contenant la valeur de l'état et une fonction pour mettre à jour cet état.
- **useEffect** : permet la récupération de données, les abonnements ou les mises à jour manuelles du DOM. Il **remplace** les méthodes de cycle de vie **componentDidMount**, **componentDidUpdate** et **componentWillUnmount** des composants basés sur des classes.



Le hook d'effet et la liste de dépendance

- **useEffect** prend une fonction comme premier argument. Cette fonction est exécutée après chaque rendu complet du composant (y compris le premier rendu).
- Le deuxième argument de **useEffect** est une **liste de dépendances**.



Les modes du hook d'effet

- `useEffect` exécute l'effet **uniquement** si les **valeurs dans la liste de dépendances ont changé** depuis le dernier rendu.
- Si la liste de dépendances **est vide** (`[]`), l'effet ne s'exécutera qu'une seule fois, après le premier rendu, **similaire à `componentDidMount`** dans les composants de classe.
- Si **aucun tableau de dépendances** n'est fourni, l'effet s'exécutera **après chaque rendu**.



Les règles des hooks

- Utilisez les Hooks **uniquement dans les composants fonctionnels**.
- Ne placez pas les Hooks à l'intérieur de **boucles**, de **conditions** (if), ou de **fonctions imbriquées**. Ils doivent toujours être utilisés au niveau racine de votre composant.
- Les Hooks doivent être appelés dans le même ordre à chaque rendu.
- Ne créez pas de fonctions qui ressemblent à des Hooks (c'est-à-dire qui commencent par "use") mais qui ne suivent pas les règles des Hooks.



Commit

Commit **ClassCounter.js** et
FunctionCounter.js





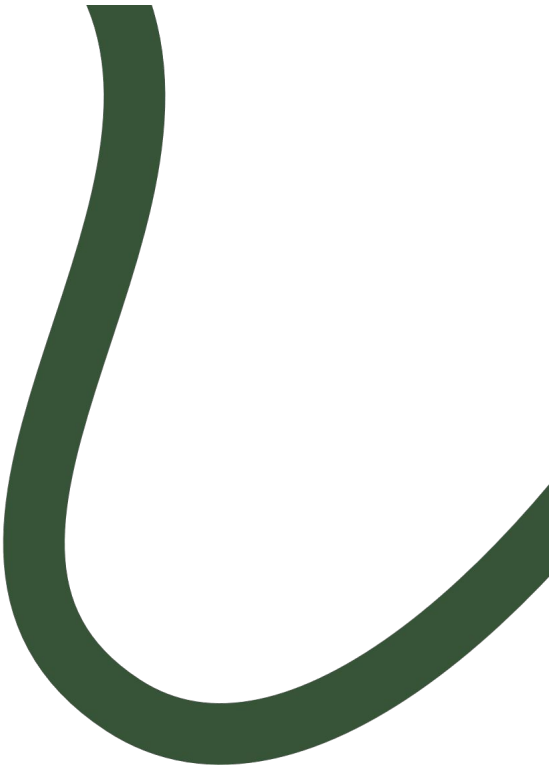
Les custom hooks

- Construire vos propres Hooks vous permet d'extraire la logique des composants dans des fonctions réutilisables.
- Les Custom Hooks doivent **commencer par use** (par exemple, **useDataFetcher**), ce qui indique qu'ils peuvent appeler d'autres Hooks.
- Un Custom Hook **encapsule la logique** d'état, d'effets secondaires, ou toute autre logique réutilisable que vous voudriez partager entre plusieurs composants.

—

Commit

Commit **custom-hook.js**





7 - Les événements





Syntaxe des événements dans le JSX



- **Syntaxe CamelCase** : Contrairement au HTML où les noms d'événements sont en minuscules, JSX utilise la camelCase pour les noms d'événements. Par exemple, onclick et onsubmit en HTML deviennent onClick et onSubmit en JSX.
- En JSX, les gestionnaires d'événements sont passés **en tant que fonctions**, et non en tant que chaînes de caractères.



Techniques de liaison du contexte d'exécution au handler



- **L'autobinding** est la liaison automatique du contexte d'une méthode de classe à l'instance du composant.
- Dans les versions plus anciennes de React, l'autobinding était géré automatiquement pour les méthodes définies dans la classe `React.createClass`. Cependant, avec l'introduction des classes ES6, l'autobinding n'est plus automatique.



Techniques de liaison du contexte d'exécution au handler

- **Lier manuellement** : Vous pouvez lier manuellement le contexte des méthodes de gestion d'événement dans le constructeur de la classe en utilisant la méthode **bind()**.
- **Autobinding avec des méthodes fléchées** : Elles héritent automatiquement du contexte (this). Vous pouvez définir vos méthodes de gestion d'événements comme des propriétés de classe en utilisant des fonctions fléchées.



Objet d'événement

- React encapsule les événements natifs du navigateur dans des instances de **SyntheticEvent**, qui ont la même interface que les événements natifs mais fonctionnent de manière identique sur tous les navigateurs.
- React normalise l'événement pour qu'il ait le même comportement sur différents navigateurs.
- Il contient toutes les propriétés et méthodes que vous attendez d'un événement natif, comme **preventDefault()**, **stopPropagation()**, **target**, **currentTarget** etc.
- Pour des raisons de performance, React "met en pool" les objets d'événement. Cela signifie que l'objet d'événement est réutilisé après que l'événement a été traité. Si vous voulez utiliser l'événement de manière asynchrone, vous devez appeler `event.persist()` pour sortir l'événement du pool et le garder dans la mémoire.



Passage de paramètres supplémentaires au handler

- La manière la plus courante est d'utiliser une fonction fléchée dans l'attribut d'événement.
- La fonction fléchée appelle le gestionnaire et lui passe l'objet d'événement ainsi que d'autres paramètres.
- Dans les composants de classe, une autre approche consiste à utiliser bind pour préfixer des arguments au gestionnaire d'événements.



Envoyer un handler en props

- Envoyer un handler en tant que props est une technique courante pour communiquer entre les composants.
- Cela est utile pour remonter des informations du composant enfant vers le parent (**Lifting State Up**)
- Le pattern Lifting State Up est une approche recommandée dans les applications React pour gérer l'état partagé entre plusieurs composants.
- L'idée est de déplacer l'état partagé vers un ancêtre commun dans l'arborescence des composants, de manière à ce que l'état puisse être facilement partagé et synchronisé entre les composants qui en dépendent.
- Ce pattern permet de maintenir une source unique de vérité pour l'état partagé et de faciliter la synchronisation entre les composants.
- Cela facilite la synchronisation et la communication entre les composants, tout en évitant les problèmes de propagation des données à travers l'arborescence des composants.

Commit

Commit **evenement.js**





8 - Rendu conditionnel et liste



Contenu conditionnel et raccourcis

- Le **contenu conditionnel** en React est réalisée en utilisant des expressions JavaScript pour **inclure ou exclure** certains éléments de l'interface utilisateur.
- L'**opérateur ternaire (?)** est une manière concise de faire une instructions if-else, très utilisé.
- L'opérateur **&&** est utile lorsque vous voulez rendre un élément seulement si une certaine condition est vraie, pour **une condition sans else**.



Commit

Commit **conditionnel.js**



Listes et raccourcis

- Le rendu de listes est souvent géré en utilisant l'expression `.map()` de JavaScript pour transformer un tableau de données en un tableau d'éléments React.

```
function Liste({ items }) {  
  return (  
    <ul>  
      {items.map((item, index) => (  
        <li key={index}>{item}</li>  
      ))}  
    </ul>  
  );  
}
```



Les clés (key) et le DOM Virtuel

- Les clés (key) aident React à **identifier** quels éléments ont changé, ont été ajoutés ou supprimés. Elles doivent être **uniques** parmi les frères/soeurs dans la liste.
- Idéalement, utilisez des **identifiants uniques** provenant **de vos données** comme clés plutôt que l'index du tableau.
- Lorsque les enfants d'une liste changent, React utilise les clés pour **déterminer si un élément peut être réutilisé.**
- Sans clés uniques, React ne peut pas différencier correctement les éléments individuels dans une liste, **ce qui peut entraîner des re-rendus inutiles et affecter les performances.**

Commit

Commit **key.js**





Les fragments

- Les **fragments** sont une fonctionnalité de React qui permet de **regrouper plusieurs éléments JSX** sans avoir à les envelopper dans un élément DOM supplémentaire, comme un `<div>`.
- Les fragments sont utiles lorsque vous souhaitez **retourner plusieurs éléments de même niveau dans un composant**, mais que vous ne voulez pas ajouter de nœuds DOM inutiles à votre arborescence.

Commit

Commit **fragment.js**





9 - Les formulaires





Source de vérité

- Le concept de **source unique de vérité** signifie que les données du formulaire, comme l'état d'un input, sont contrôlées par le composant React lui-même, plutôt que par le DOM.
- Ce principe est mis en œuvre à travers des composants contrôlés.



Composant contrôlé

- Un **composant contrôlé** est un composant dont l'**état de l'input est contrôlé par React**.
- Par exemple, un champ de texte dans un formulaire est un composant contrôlé si sa valeur est liée à l'état du composant React.
- **Consistance des Données** : La valeur de l'input est toujours synchronisée avec les données de l'état du composant, garantissant la consistance des données.
- **Validation et Formatage Faciles** : Vous pouvez facilement valider ou formater la valeur de l'input à chaque modification.
- **Manipulation Facile des Soumissions** : La gestion des données à la soumission du formulaire est simplifiée, car toutes les données sont déjà stockées dans l'état du composant.



L'attribut de valeur universel des champs



- L'attribut **value** joue est utilisé pour **lier la valeur des champs de formulaire** (comme les inputs, les textareas et les select) **à l'état du composant**.
- Cela permet une **gestion cohérente et prévisible des données** du formulaire.



Soumission du formulaire

- Vous gérez la soumission du formulaire **via une fonction** JavaScript.
- Vous passez cette fonction à l'attribut **onSubmit** du composant **<form>**
- Utilisez **event.preventDefault()** dans votre gestionnaire **handleSubmit** pour empêcher le comportement par défaut du formulaire (qui est de recharger la page).

Commit

Commit **formulaires-controles.js**



Composants non contrôlés

- Un **composant non contrôlé** est un élément de formulaire dont la **valeur est gérée directement par le DOM**, et non par l'état local d'un composant React.
- Pour accéder à la valeur d'un composant non contrôlé, vous devez utiliser une **référence (ref)** pour **lire directement la valeur** de l'élément de formulaire.



Les refs

- **Les refs** sont un moyen d'accéder directement aux nœuds du DOM (Document Object Model) ou aux éléments React dans le code de vos composants.
- Elles sont indispensables pour gérer les formulaires non-contrôlés.
- Vous pouvez en créer avec **React.createRef** dans le constructeur et les attacher aux éléments React dans la méthode render.
- Avec le Hook **useRef** dans les composants fonctionnels.

Commit

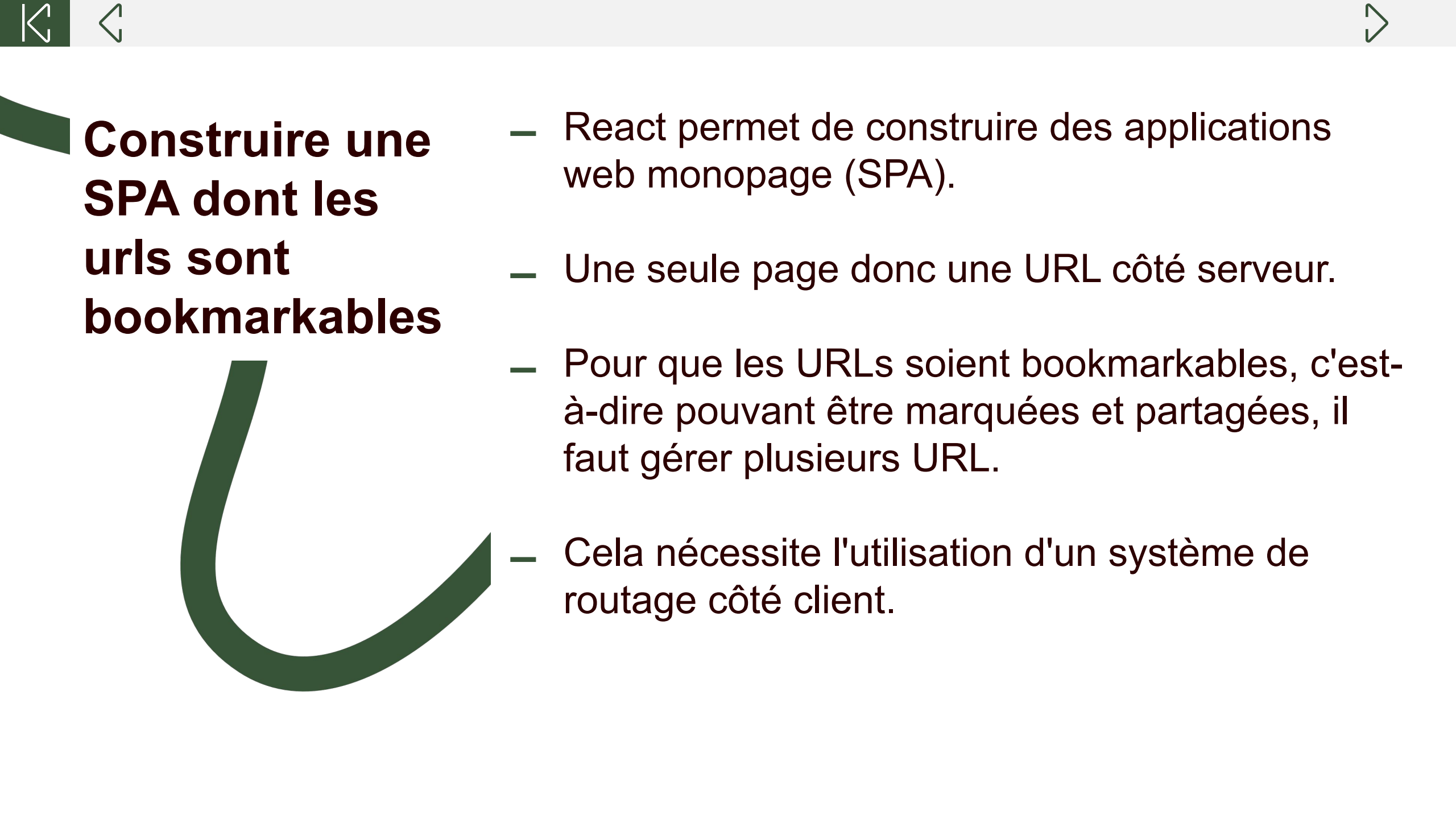
Commit **formulaires-non-controles.js**





10 - Le routing et la navigation





Construire une SPA dont les urls sont bookmarkables

- React permet de construire des applications web monopage (SPA).
- Une seule page donc une URL côté serveur.
- Pour que les URLs soient bookmarkables, c'est-à-dire pouvant être marquées et partagées, il faut gérer plusieurs URL.
- Cela nécessite l'utilisation d'un système de routage côté client.



La librairie react-router-dom (version 5)

- **react-router-dom** est une bibliothèque populaire pour ajouter des fonctionnalités de routage côté client.
- react-router-dom permet **un routage déclaratif** dans votre application React.
- Vous pouvez définir des routes en utilisant des composants React, ce qui rend votre code plus lisible et facile à comprendre



Le router

- Le router est un composant de bas niveau qui **gère la synchronisation de l'interface utilisateur** de votre application **avec l'URL**.
- Il est rarement utilisé directement, car il existe des variantes plus spécifiques du Router qui sont généralement préférées pour des cas d'utilisation courants.
- Le **BrowserRouter** est un wrapper autour de Router qui utilise l'API de l'historique HTML5 pour garder l'interface utilisateur de votre application synchronisée avec l'URL.
- Le **HashRouter** est une alternative qui utilise l'hash de l'URL (la partie de l'URL commençant par #) pour garder l'interface utilisateur et l'URL synchronisées.
- Il est utile si vous servez votre application depuis un serveur statique ou un environnement où **vous n'avez pas le contrôle sur la gestion des routes serveur** (comme GitHub Pages).



Les liens

- Le composant **Link** est utilisé pour créer des liens de navigation dans votre application.
- Le Link utilise l'attribut **to**, qui spécifie le chemin (path) vers lequel naviguer. Ce chemin peut être une chaîne ou un objet.

```
<Link to="/contact">Contact</Link>
```



Les routes

- Utilisez le composant **Route** pour définir une correspondance entre le chemin (path) et le composant à rendre.

```
<Route path="/about">  
  <About />  
</Route>
```

Le switch

- **Switch** est utilisé pour regrouper plusieurs Route. Il rend le premier Route enfant dont le chemin correspond à l'URL actuel.

```
<Router>
  <Switch>
    <Route path="/">
      <Home />
    </Route>
  </Switch>
</Router>
```


Les paramètres d'url

- Vous pouvez définir des routes qui acceptent des paramètres. Par exemple, une route pour un profil utilisateur pourrait inclure l'identifiant de l'utilisateur dans l'URL.

```
<Route path="/users/:userId">  
  <User />  
</Route>
```



Les navigations imbriquées

- Vous pouvez imbriquer les Route à l'intérieur de composants pour créer des **sous-routes**.

Commit

Commit **react-router.js**





11 - Introduction à Redux et architecture flux





Immutabilité des variables partagées

- L'immutabilité se réfère à la création de variables dont la valeur ne peut pas être modifiée une fois qu'elles ont été assignées.
- React s'appuie sur l'immutabilité pour garantir que les mises à jour de l'état (state) et des props sont prévisibles.
- Lorsque vous modifiez l'état d'un composant en utilisant `setState`, vous ne modifiez pas directement l'état actuel, mais plutôt vous créez une nouvelle version de l'état qui est ensuite fusionnée avec l'état actuel.
-



Les composants d'ordre supérieur

- Les **composants d'ordre supérieur** sont un modèle avancé dans React pour réutiliser la logique des composants.
- Un HOC est une fonction qui prend un **composant en paramètre** et **renvoie un nouveau composant** avec des fonctionnalités supplémentaires ou modifiées.
- Les HOC sont utiles pour **partager la logique commune** entre plusieurs composants **sans dupliquer le code**.



Problème de la gestion d'état

- **État local vs état global** : Difficile de décider si un état doit être local à un composant ou partagé globalement dans l'application.
- **Prop Drilling** : Passer des props sur plusieurs niveaux de composants rends le code difficile à maintenir.
- **Synchronisation de l'état avec des sources externes** : Garder l'état synchronisé avec des sources externes comme des bases de données, des API peut être difficile.
- **Complexité et performances** : Les applications React peuvent être ralenti à cause d'une mauvaise gestion des mis à jours d'état.



L'architecture flux

- L'**architecture Flux** est un pattern de conception d'application pour améliorer la gestion de l'état dans les applications React.
- Flux suit un modèle de flux de données unidirectionnel, ce qui signifie que les données dans l'application suivent un chemin unique et prévisible.
- Composants Principaux de Flux :
 - ▶ **Actions** : Les actions sont des paquets d'informations qui envoient des données du point d'interaction de l'utilisateur, comme un clic sur un bouton, au dispatcher.
 - ▶ **Dispatcher** : Le dispatcher reçoit des actions et les diffuse aux différents stores.
 - ▶ **Stores** : Les stores contiennent l'état et la logique d'une application. Chaque store est responsable d'un domaine spécifique de l'état de l'application.
 - ▶ **Vues** (Composants React) : Les vues réagissent aux changements dans les stores et mettent à jour l'interface utilisateur en conséquence.



Comment Flux fonctionne

- Lorsqu'un **utilisateur interagit** avec l'application (par exemple, en cliquant sur un bouton), **une action est déclenchée**.
- Cette action est **envoyée au dispatcher**, qui la **transmet aux stores** appropriés.
- **Les stores mettent à jour leur état** en fonction des données de l'action, puis émettent un événement indiquant qu'un changement a eu lieu.
- **Les vues écoutent ces changements** dans les stores. Lorsqu'elles détectent un changement, elles se mettent à jour pour refléter le nouvel état.



Redux : définition et installation

- **Redux** est une bibliothèque JavaScript pour la gestion de l'état des applications.
- Elle est fréquemment utilisée avec React, mais peut être utilisée avec n'importe quelle autre bibliothèque ou framework JavaScript.
- Redux se base sur les principes de l'architecture Flux, avec un flux de données unidirectionnel et une gestion d'état centralisée.

```
npm install redux react-redux
```



Les actions

- Ce sont des objets JavaScript qui décrivent ce qui s'est passé dans l'application (par exemple, une action pour ajouter un élément à une liste).
- Chaque action a un type et, généralement, des données supplémentaires (appelées payload).



Les reducers

- Ce sont des **fonctions pures** qui prennent l'état actuel et une action en paramètres puis retournent un nouvel état.
- Les reducers sont responsables de la **mise à jour de l'état en réponse aux actions**.
- **Ils ne modifient pas l'état actuel**, mais produisent un nouvel état basé sur l'action reçue.



Le store

- C'est l'objet central qui détient **l'état global de l'application**.
- Il n'y a qu'**un seul store** dans une application Redux.
- Il est responsable de la gestion de l'état et de sa mise à jour en réponse aux actions.



Utilisation avec React

- On utilise la bibliothèque **react-redux**, qui fournit des **composants** et des **hooks** pour interagir avec le store Redux.
- Les principaux éléments de react-redux sont :
 - ▶ **Provider** : Un composant qui enveloppe l'application et permet à tous les composants enfants d'accéder au store Redux.
 - ▶ **le HOC connect** : Une fonction de niveau supérieur qui relie un composant React au store Redux en mappant l'état et les actions sur les props du composant.
 - ▶ **useSelector** et **useDispatch** : Des hooks qui permettent respectivement d'accéder à l'état du store Redux et d'envoyer des actions.

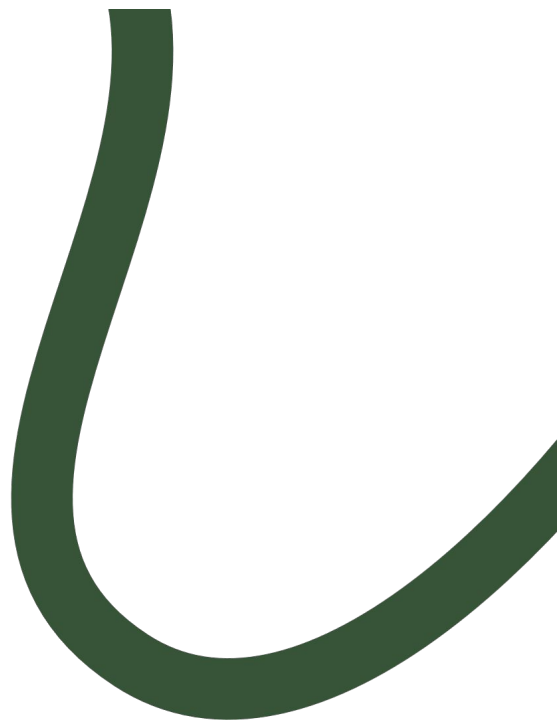
Commit

Commit **redux.js**





La méthode `mapStateToProps`



- La méthode **`mapStateToProps`** sert de pont entre l'état du store Redux et les props d'un composant React.
- **`mapStateToProps`** est une fonction qui prend l'état global du store Redux comme argument et retourne un objet.
- Les clés de cet objet **deviennent des props** pour le composant React auquel vous connectez le store.
- Elle permet de **sélectionner les parties** de l'état Redux **que le composant a besoin de consommer**.
- Cela signifie que le composant ne sera pas affecté par les changements d'état dans Redux qui ne le concernent pas.

Commit

Commit `mapStateToProps.js`





La méthode **mapDispatchToProps** **Props**

- **mapDispatchToProps** permet de fournir à vos composants React des fonctions qu'ils peuvent appeler pour dispatcher des actions Redux.
- Cela abstrait la logique de dispatching et rend les composants plus réutilisables et moins dépendants de Redux.
- **mapDispatchToProps** est utilisée avec la fonction **connect**.
- Elle prend en argument la fonction dispatch de Redux et retourne un objet contenant une ou plusieurs méthodes qui dispatchent des actions.

Commit

Commit `mapDispatchToProps.js`





12 - Les tests





Introduction au framework Jest

- **Jest** est une bibliothèque de test JavaScript open-source développée par Facebook.
- Jest permet de faire des **tests unitaires**, et bien plus, comme la génération de "snapshots" pour les tests de rendu, le "mocking" (simulation) de fonctions et de modules et le support des tests asynchrones.



La React testing library

- "The more your tests resemble the way your software is used, the more confidence they can give you" (Plus vos tests ressemblent à la manière dont votre logiciel est utilisé, plus ils peuvent vous donner confiance).
- Les tests se concentrent sur le **comportement et l'interaction** du composant plutôt que sur les détails de l'implémentation.
- Il s'intègre bien avec Jest, les bibliothèques sont complémentaires.



ReactJS - Développement d'applications Web



Votre formateur :
Quentin BIDOLET

