

# React JS

Laurence  
X  NATIS



# Projet

## Application Web & Web Mobile :

### Quizz

# Application Web Mobile Quizz personnalité





The home page features a dark background with a white icon of a balance scale at the top. Below the icon, the text "Quel type de développeur(se) êtes-vous ?" is displayed in white. Underneath, there is a text input field with the placeholder "Entrez votre email :" and the email address "michael@xonatis.com" entered. At the bottom, there is a green button with the text "Répondre au quizz ?" and a small white question mark icon.

Page  
Home



The first question page has a dark background with a white question mark icon at the top. Below the icon, the text "Aimez-vous faire le design d'un site internet ?" is displayed in white. At the bottom, there are five white buttons labeled "1", "2", "3", "4", and "5" for selecting an answer.

Page  
Question 1



The second question page has a dark background with a white question mark icon at the top. Below the icon, the text "Aimez-vous le langage Javascript ?" is displayed in white. At the bottom, there are five white buttons labeled "1", "2", "3", "4", and "5" for selecting an answer.

Page  
Question 2

A dark-themed mobile app screen for a question. At the top center is a large white question mark icon. Below it, the text "Aimez-vous React.js, Vue.js ou Angular ?" is displayed. At the bottom, there are five white square buttons with black numbers 1, 2, 3, 4, and 5 arranged horizontally.

A large question mark icon is centered at the top.

Aimez-vous React.js, Vue.js ou Angular ?

Below the question, there are five numbered buttons (1, 2, 3, 4, 5) for selection.

Page  
Question 3

A dark-themed mobile app screen showing the result. At the top center is a white icon of a laptop with code symbols. Below it, the word "Résultat" is displayed in a large font. Underneath, the text "Votre type de développement est :" is shown. Below this text are two light green rectangular boxes: the first contains the email "michael@xonatis.com" and the second contains the word "Frontend". At the bottom center is a blue button with a white left-pointing arrow and the text "Retour".

A laptop icon with code symbols is centered at the top.

Résultat

Votre type de développement est :

michael@xonatis.com

Frontend

Retour

Page  
Résultat

Whhaat ??

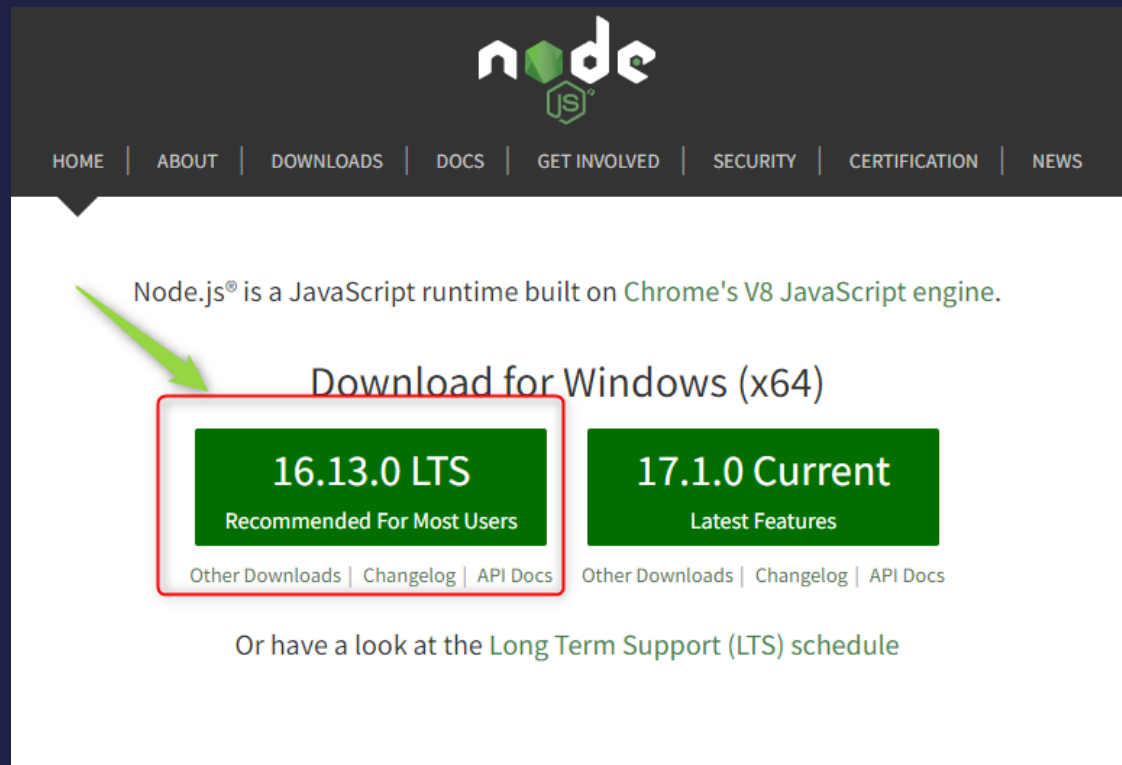
Page  
Not Found

# 1. Création du Projet

# Installer Nodejs

Il faut installer node JS (plateforme logicielle et environnement d'exécution JavaScript) sur l'ordinateur.

<https://nodejs.org/en/>

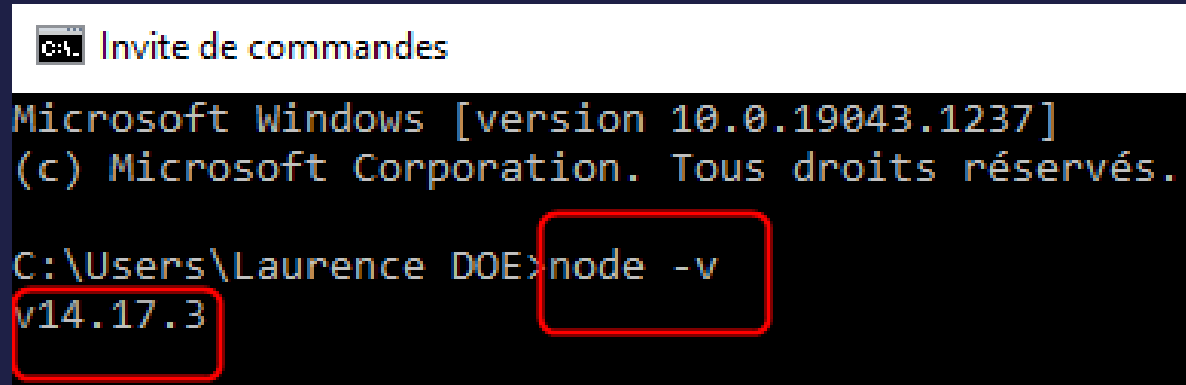


👉 **Remarque :** Choisir la version recommandée par le site



# Installer Nodejs

Vérifier ensuite l'installation effective dans le terminal (depuis le bureau via cmd)



```
Invite de commandes

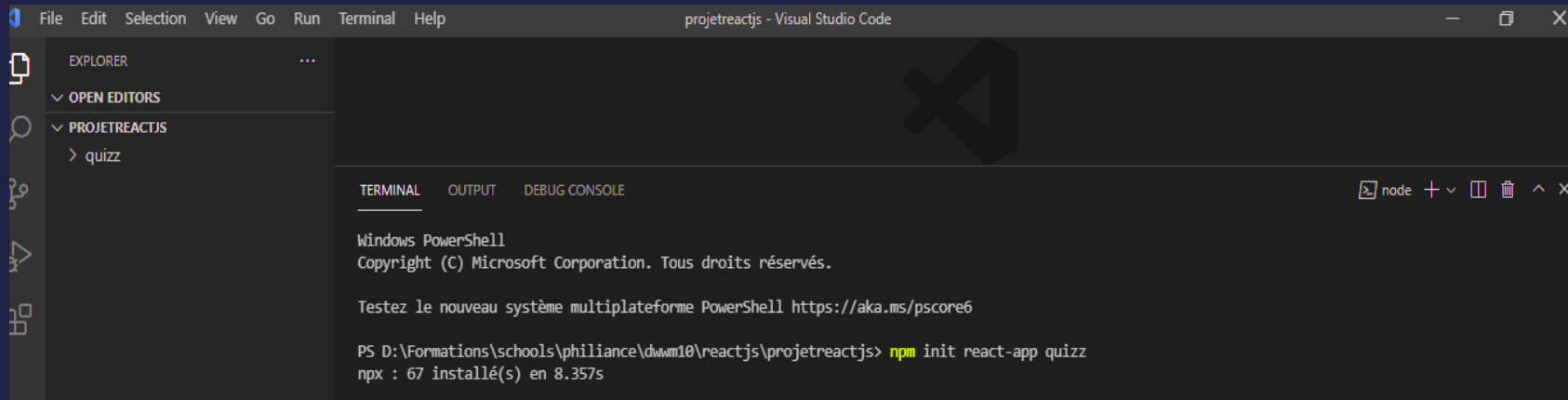
Microsoft Windows [version 10.0.19043.1237]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\Laurence DOE>node -v
v14.17.3
```

👉 **Remarque :** Node permet notamment l'installation de paquets via son gestionnaire **npm** et va contribuer à construire l'application déployée (lors du lancement via la commande **npm run start**).

Suite à cette commande, **webpack** (outil servant à générer des fichiers compilés) transforme le tout en quelque chose de lisible et compréhensible pour le navigateur.

# Créer un projet avec **npm**



The screenshot shows the Visual Studio Code interface with the 'Terminal' panel open. The terminal output is as follows:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS D:\Formations\schools\philiance\dwmm10\reactjs\projetreactjs> npm init react-app quiz
npx : 67 installé(s) en 8.357s
```

```
Desktop\test-react> npm init react-app quiz
```

👉 **Remarque :** Penser à bien se positionner à la racine du dossier projet.  
Et noter que la création peut prendre plusieurs minutes.

# Créer un projet avec **npx**

```
npx create-react-app mon-app  
cd mon-app  
npm start
```

👉 **Remarque :** En cas d'erreur (voir exemple ci-joint)

```
PS C:\Users\Laurence DOE> npx create-react-app mon-app  
Error: EPERM: operation not permitted, mkdir 'C:\Users\Laurence'  
Commande non trouvée : create-react-app
```

Installer la commande « **create-react-app** » puis relancer de nouveau la création de l'application

```
PS C:\Users\Laurence DOE> npm install create-react-app  
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.  
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\Laurence DOE\package.json'  
npm notice created a lockfile as package-lock.json. You should commit this file.  
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\Laurence DOE\package.json'  
npm WARN Laurence DOE No description  
npm WARN Laurence DOE No repository field.  
npm WARN Laurence DOE No README data  
npm WARN Laurence DOE No license field.  
  
+ create-react-app@4.0.3  
added 67 packages from 25 contributors and audited 67 packages in 11.311s  
  
4 packages are looking for funding  
run `npm fund` for details  
  
found 5 high severity vulnerabilities  
run `npm audit fix` to fix them, or `npm audit` for details
```

## Différence entre **npm** et **npm**

Bien que leurs noms soient proches et que certaines commandes puissent être utilisées avec les deux outils, ce sont deux outils qui n'ont pas le même but, et la même commande lancée avec l'un ou l'autre des outils ne s'utilise pas de la même façon.

**npm** (Node Package Manager) est avant tout un référentiel en ligne pour la publication de projets open-source Node.js mais c'est aussi un **outil CLI** (Command-line interface ou interface en ligne de commande) qui aide à installer des packages et à gérer leurs versions ainsi que leurs dépendances. npm par lui-même n'exécute aucun package. Si l'on souhaite exécuter un package à l'aide de npm, on doit spécifier ce package dans le fichier **package.json**.

Si l'on utilise la commande "`npm mon-paquet`", cela n'exécutera le paquet que s'il a été installé globalement.

<https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

## Différence entre **npm** et **npx**

**npx** (Node Package eXecute) est un outil spécialement conçu pour l'exécution des paquets. C'est aussi un outil CLI dont le but est de faciliter l'installation et la gestion des dépendances hébergées dans le registre npm.

Si l'on souhaite exécuter un package installé localement, il suffit de taper:

👉 `npx your-package`

Un autre avantage majeur est la possibilité d'exécuter un package qui n'a pas été installé auparavant. Parfois, on veut simplement utiliser certains outils CLI, mais on ne souhaite pas les installer globalement juste pour les tester. Cela signifie que l'on peut économiser de l'espace disque et ne les exécuter que lorsqu'on en a besoin.

**Conclusion** : npx fournit un moyen clair et simple pour exécuter des packages, des commandes, des modules et aide à éviter les versions, les problèmes de dépendance ainsi que l'installation de packages inutiles que l'on veut juste essayer.

<https://docs.npmjs.com/cli/v7/commands/npx>

## 2. Lancement de l'application

# Lancement du projet

Voici quelques commandes (en bleu) suggérées par React :

```
found 24 vulnerabilities (8 moderate, 15 high, 1 critical)
  run `npm audit fix` to fix them, or `npm audit` for details

Created git commit.

Success! Created mon-app at C:\Users\Laurence DOE\mon-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd mon-app
  npm start

Happy hacking!
```

On peut également la commande suivante :

```
TERMINAL  OUTPUT  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/powershell

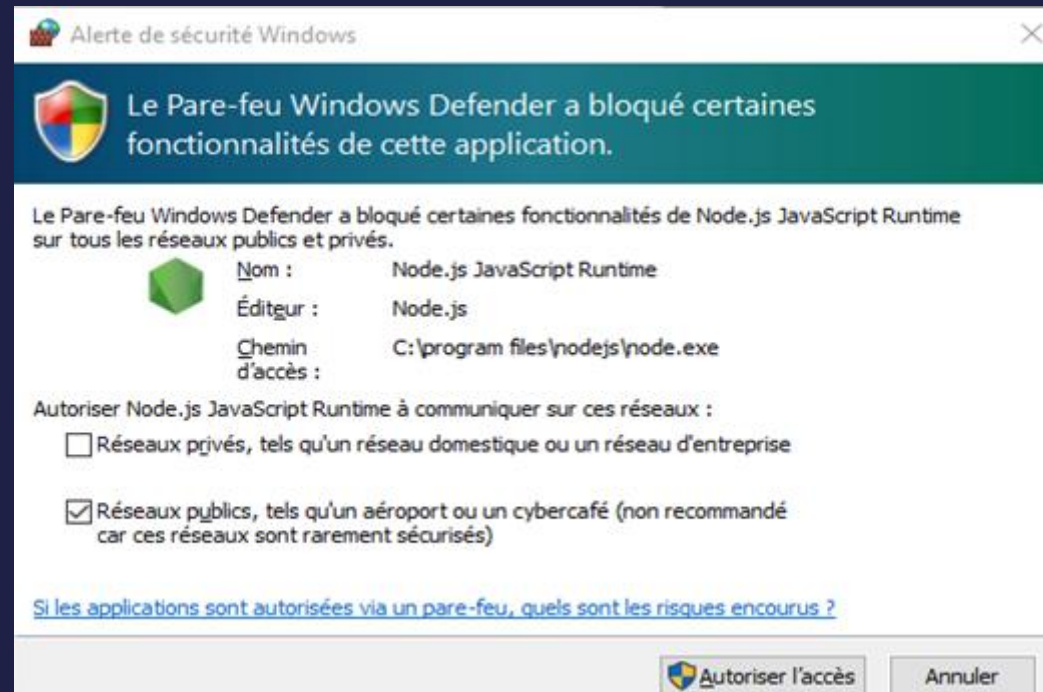
PS D:\Formations\schools\philliance\dawam10\reactjs\projetreactjs\quizz> npm run start
```

# Lancement du projet

Au lancement, une invite apparaît pour autoriser le server node.js :

```
Starting the development server...
```

```
[]
```





# Lancement du projet

Chargement de la page :



Edit src/App.js and save to reload.

[Learn React](#)

Compiled successfully!

You can now view **mon-app** in the browser.

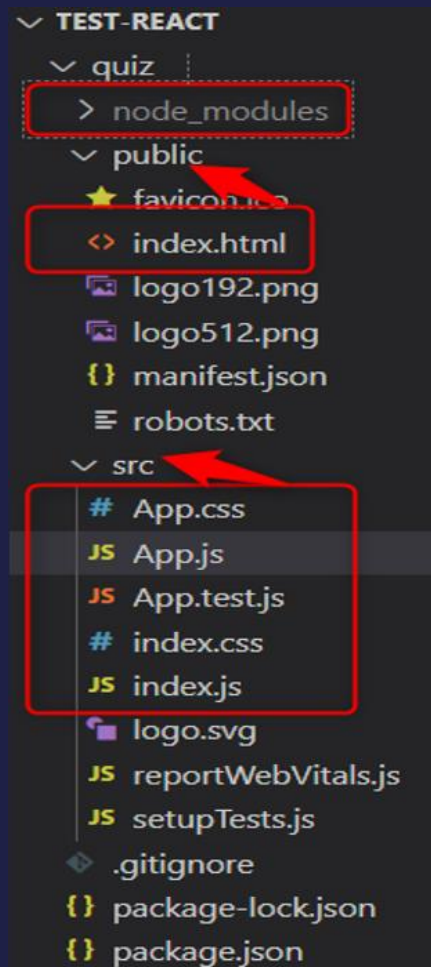
**Local:** http://localhost:3000

**On Your Network:** http://192.168.1.23:3000

Note that the development build is not optimized.  
To create a production build, use `npm run build`.

# Structure des fichiers

Les fichiers suivants sont installés par défaut :



**node\_modules** : Dans l'ensemble, on a généralement pas besoin d'ouvrir ce dossier, il contient toutes les librairies javascript dont on aura besoin.

**public** : Le dossier public contient le fichier index.html qui est le point départ avec la div qui a pour class **root** et qui permet de signifier à React l'endroit où il doit venir s'ancrer.

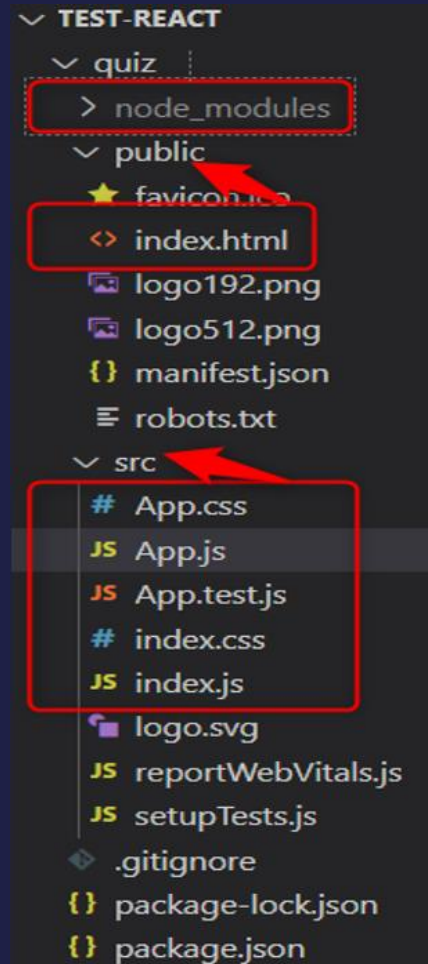
**src** : Le dossier src est l'endroit où toute la magie de l'application va opérer, on y retrouve le fichier index.js qui est le fichier qui charge le premier composant (App) qui est ensuite greffé à la div avec la class **root** qui se trouve dans le fichier **index.html** au niveau du dossier public.

**gitignore** : Le fichier .gitignore a pour rôle de signifier à git les fichiers qui ne doivent pas être envoyés (push) sur le répertoire (repository) distant notamment le dossier node\_modules.

**package.json** : Le fichier package.json est certainement le fichier qui contient plusieurs informations sur votre application React notamment les scripts, les dépendances.

# Structure des fichiers

Les fichiers suivants sont installés par défaut :



**index.html** : Ce fichier contrôle toute l'application (via la div avec id "**root**"). Il faut considérer que toute l'application se trouve à ce niveau (cela est utile notamment pour l'ajout d'image ou média qui se trouveront également dans le dossier public donc au même niveau).

**Attention !** il faut utiliser les classes conventionnelles (class) dans ce fichier.

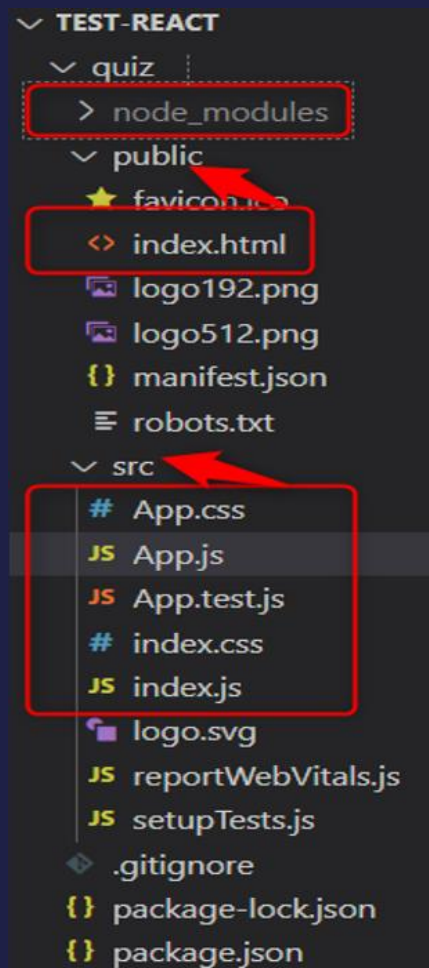
**App.css** : Contient le css par défaut de la page **App.js**

**App.js** : Page affichée par défaut lors du 1er lancement du projet (avec le logo React). On peut en modifier le contenu et afficher nos données mais on peut également de manière plus simple créer de nouveaux composants, de nouvelles pages. Idéalement un fichier à extension js pour chaque page.

Sur la page **App.js**, on peut afficher des balises html avec des classes (ici **className**) et des attributs entre accolades {xxx} que l'on appelle "**props**". On utilise aussi la syntaxe JSX pour déclarer des variables, spécifier des attributs.

# Structure des fichiers

Les fichiers suivants sont installés par défaut :



Si l'on crée d'autres composants, par convention il est d'usage de conserver le fichier `App.js` qui va servir de router (pour définir les routes et les redirections).

Avant de l'utiliser à cet effet, il faudra installer Router en utilisant la commande :

`npm install react-router-dom`

`index.css` : Contient le css par défaut de la page `index.js`

`index.js` : Contient le code javascript qui permet le chargement du premier composant (`App`)

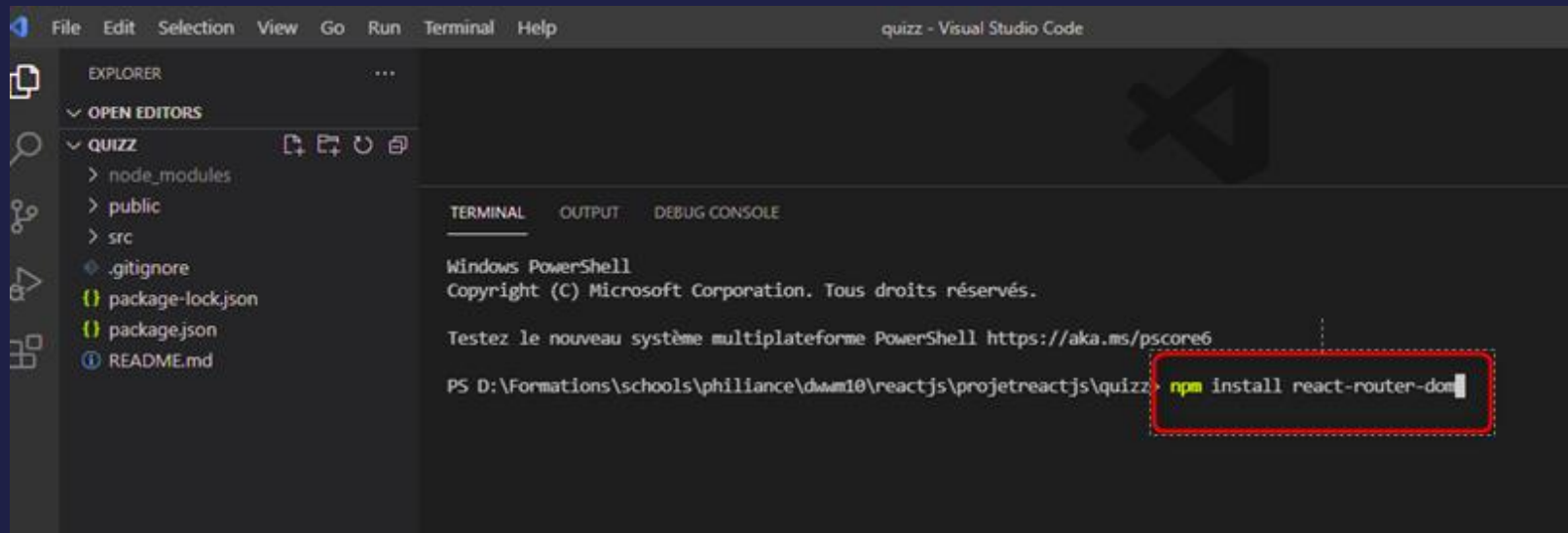
`app.test.js` : C'est la page dédiée aux tests unitaires (fonctionnement des composantes, qualité du code). Fichier à supprimer si l'on en a pas le besoin.

## 3. Routing

# Installation du **router**

Le **Router** est une bibliothèque de routage (routing) standard dans React qui permet d'afficher différentes pages à l'utilisateur. Cela signifie qu'il donne la possibilité de naviguer entre les différentes parties d'une application en entrant une URL ou en cliquant sur un élément.

Par défaut, React vient sans routage. Pour l'activer, il faut ajouter une bibliothèque nommée **React-router**.



The screenshot shows the Visual Studio Code interface with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'QUIZZ' with folders 'node\_modules', 'public', and 'src', and files '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The terminal is open to a Windows PowerShell session. The prompt is 'PS D:\Formations\schools\philliance\dwm10\reactjs\projetreactjs\quizz>'. The command 'npm install react-router-dom' is being typed and is highlighted with a red dashed box.

```
File Edit Selection View Go Run Terminal Help quizz - Visual Studio Code

EXPLORER
  OPEN EDITORS
  QUIZZ
    node_modules
    public
    src
    .gitignore
    package-lock.json
    package.json
    README.md

TERMINAL OUTPUT DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS D:\Formations\schools\philliance\dwm10\reactjs\projetreactjs\quizz> npm install react-router-dom
```

## Installation du **router**

Le **React Router** fournit deux composants tels que `<BrowserRouter>` & `<HashRouter>`. Ces deux composants sont différents dans le type d'URL qu'ils créent et synchronisent.

`<BrowserRouter>` est utilisé plus couramment, il utilise le History API incluse dans HTML5 pour surveiller l'historique du routeur tandis que `<HashRouter>` utilise le hash de l'URL (`window.location.hash`) pour tout mémoriser.

```
// <BrowserRouter>  
http://example.com/about  
  
// <HashRouter>  
http://example.com/#/about
```

## Installation du **router**

Le composant `<Route>` définit une relation (mapping) entre une URL et un Component. Cela signifie que lorsque l'utilisateur visite une URL sur le navigateur, un Component correspondant doit être rendu sur l'interface.

```
<BrowserRouter>
  <Route exact path="/" component={Home}/>
  <Route path="/about" component={About}/>
  <Route path="/topics" component={Topics}/>
</BrowserRouter>

<HashRouter>
  <Route exact path="/" component={Home}/>
  <Route path="/about" component={About}/>
  <Route path="/topics" component={Topics}/>
</HashRouter>
```

L'attribut `exact` est utilisé dans le `<Route>` afin de dire que ce `<Route>` ne fonctionne que si la URL sur le navigateur correspond absolument à la valeur de son attribut `path`.



## 4. ReactDOM & Render

## ReactDOM

**React-dom** : Module qui fournit des méthodes spécifiques au DOM que l'on peut utiliser au niveau racine de l'appli et comme échappatoire pour travailler hors du modèle React si l'on a besoin.

Contrairement aux éléments DOM d'un navigateur, les éléments React sont de simples objets. React DOM se charge de mettre à jour le DOM afin qu'il corresponde aux éléments React. Les **éléments** représentent la **base des composants**.

*Exemple : Une balise `<div>` dans un fichier HTML. On parle de nœud DOM « racine » car tout ce qu'il contient sera géré par React DOM.*

*Pour faire le rendu d'un élément React dans un nœud DOM racine, on passe les deux à la méthode ReactDOM.render() :*

```
<div id="root"></div>
```

```
const element = <h1>Bonjour, monde</h1>;  
ReactDOM.render(element, document.getElementById('root'));
```

## Render()

**Render()** : Affiche un élément React au sein du nœud DOM spécifié par container et renvoie une référence sur le composant (ou renvoie null pour les fonctions composants).

Si l'élément React était déjà affiché dans container, cette méthode effectuera plutôt une mise à jour et ne **modifiera le DOM** que **là où c'est strictement nécessaire** pour refléter l'élément React à jour.

Si la fonction de rappel optionnelle est fournie, elle sera exécutée après que le composant est affiché ou mis à jour.

**ReactDOM.render()** ne modifie pas le nœud conteneur lui-même (seulement ses enfants). Il peut arriver qu'insérer un composant dans un nœud DOM existant n'en modifie pas les enfants.

## Export default

`default export` (exportation par défaut) est la convention si l'on souhaite exporter un seul objet (variable, fonction, classe) du fichier (module). Il ne peut y avoir qu'une seule exportation par défaut par fichier, mais pas limitée à une seule exportation .

Dans React, Vue et de nombreux autres frameworks, l'exportation est principalement utilisée pour **exporter des composants réutilisables** afin de **créer des applications modulaires**.

## useHistory

**useHistory** Dans le framework React, l'objet "history" permet d'interagir avec l'historique du navigateur. Le framework fonctionnant intégralement en JavaScript, il n'y a pas de changement de page comme avec un site en HTML classique. C'est grâce à cet objet que l'on peut envoyer une URL dans l'historique du navigateur pour revenir en arrière. Depuis la version 4 du framework, son utilisation est limitée à certains composants. Il est en revanche possible d'y accéder en dehors en créant un nouvel objet "history" dans votre module.

## state

### Que fait `setState` ?

`setState()` planifie la mise à jour de l'objet `state` du composant. Quand l'état local change, le composant répond en se rafraîchissant.

### Quelle est la différence entre `state` et `props` ?

`props` (diminutif de « propriétés ») et `state` sont tous les deux des objets JavaScript bruts. Même s'ils contiennent tous les deux des informations qui influencent le résultat produit, ils présentent une différence majeure : `props` est passé au composant (à la manière des arguments d'une fonction) tandis que `state` est géré dans le composant (comme le sont les variables déclarées à l'intérieur d'une fonction).

La data stockée dans le `state` sera accessible via l'invocation `this.state`, par exemple dans notre exemple ci-dessus, on utilisera `this.state.monTexte` pour afficher notre texte dans un `<div>`. En utilisant la notation “`{ }`”

Ex : `<div>{this.state.monTexte}</div>`

## state

De la même manière, on va pouvoir afficher dynamiquement des éléments dans notre composant grâce à une combinaison de l'utilisation du JSX et du state.

```
Ex : {  
  // Si l'attribut "afficherMonElement" est égal à true  
  // alors on affiche notre div  
  (this.state.afficherMonElement === true) &&  
  <div>Voici mon élément !</div>  
}
```

### Modification du state

Lorsqu'un attribut du state est modifié, un re-rendering va être déclenché dans notre composant, qui sera mis à jour avec la nouvelle valeur du state. C'est ce qui va nous permettre d'afficher ou non certaines parties de notre composant, par exemple.

Attention: le state devra uniquement être modifié via la méthode **this.setState** à laquelle on passera un objet avec l'attribut à modifier et la nouvelle valeur à appliquer:

## props

Conceptuellement, les composants sont comme des fonctions JavaScript. Ils acceptent des entrées quelconques (appelées « props ») et renvoient des éléments React décrivant ce qui doit apparaître à l'écran.

"**props**" est l'abréviation de "**properties**", mais il est un concept dans le ReactJS. A la base, props est un objet. Il stocke les valeurs des attributs (**attribute**) d'une étiquette (**Tag**).

Lorsque React rencontre un élément représentant un composant défini par l'utilisateur, il transmet les attributs JSX et les enfants à ce composant sous la forme d'un objet unique. Cet objet est appelé « **props** ».



Les **REST API** permet à plusieurs systèmes de **communiquer entre eux** de manière banalisée (au **format JSON** majoritairement).