

Reactjs

Michael
X  NATIS



Compétence demandée :
Développer un projet simple
en Reactjs

1. L'architecture du front
 2. Construction d'un projet Reactjs
-

1. La page
2. Le routing
3. Le state
4. Les properties
5. Le context

Facebook

Über

Instagram

Codecademy

Yahoo Mail!

New York Times

Paypal

Netflix

Etc.

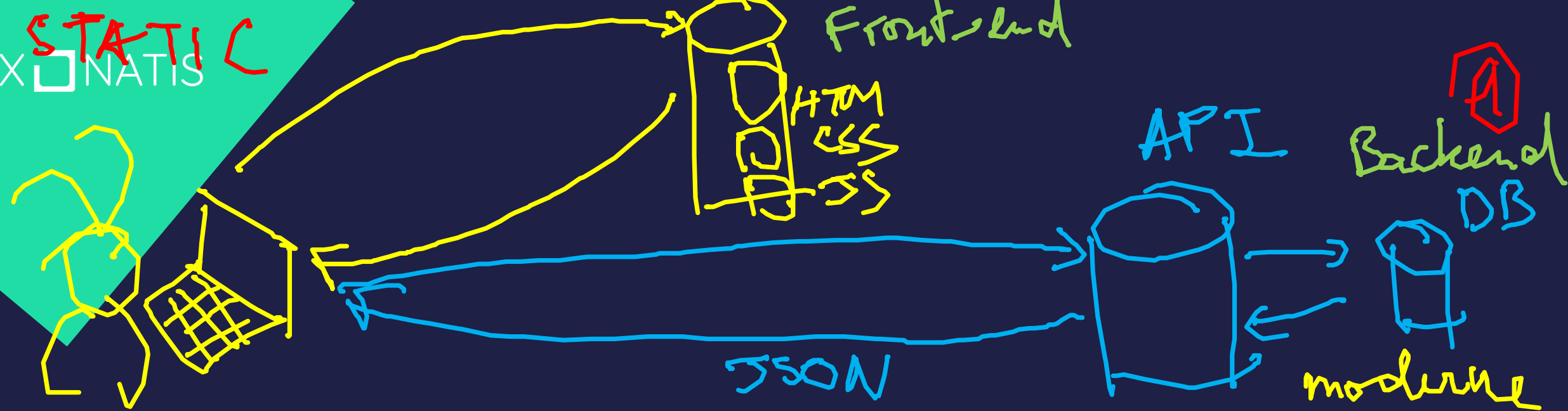
Le **but** des librairies et frameworks JS modernes

Le **but** des librairies et frameworks JS modernes

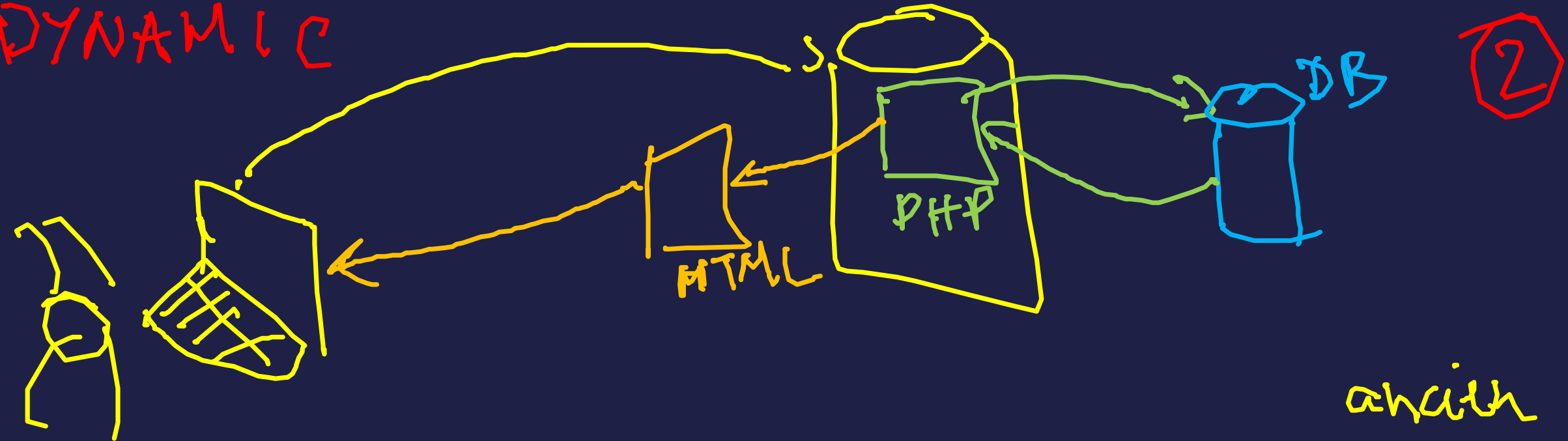
Produire une page statique
(static page)

STATIC

X N A T I S



DYNAMIC



Une page statique : une page qui est renvoyée au client telle qu'elle a été stockée sur le serveur

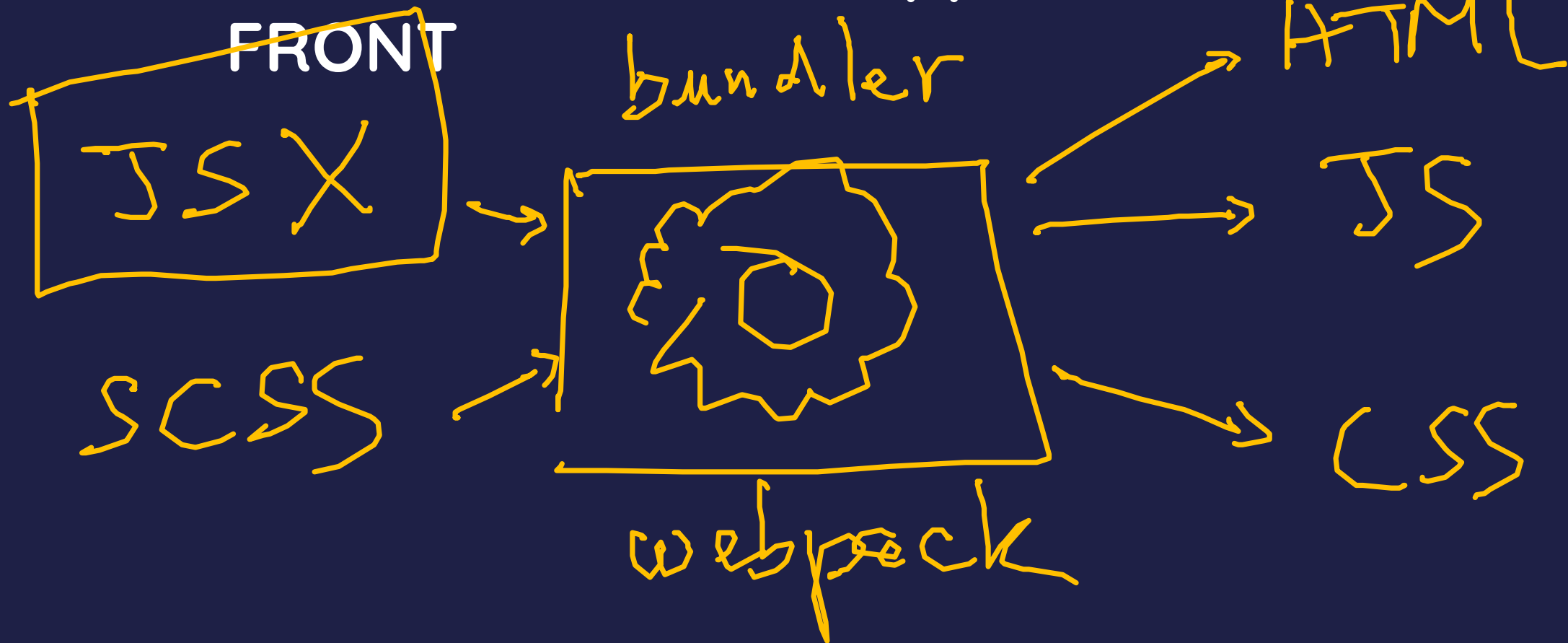
Une page statique : une page qui est **renvoyée au client telle qu'elle a été stockée sur le serveur**



- Avoir un **changement instantané**
- Pouvoir faire des **traitements en parallèle**
- Améliorer le UX

Processus de développement

FRONT



1. ARCHITECTURE FRONT

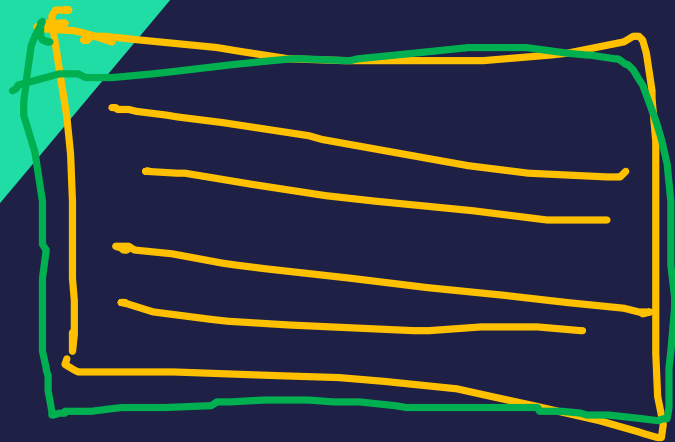
Du vrai front et du vrai back ?

- Project séparé
- Hébergé séparément
- Des développeur(se)s dédié(e)s

Un architecture de fichiers
séparés

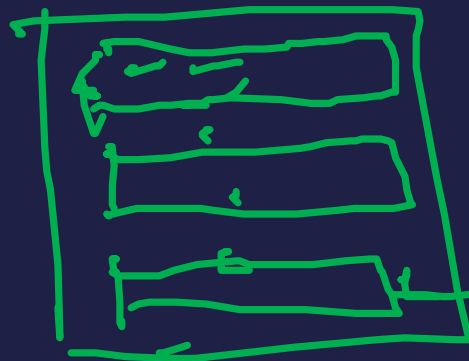
Un architecture de **transport** **séparé**

Un architecture applicative en composants



<Pavier />

Pavier



<Item />

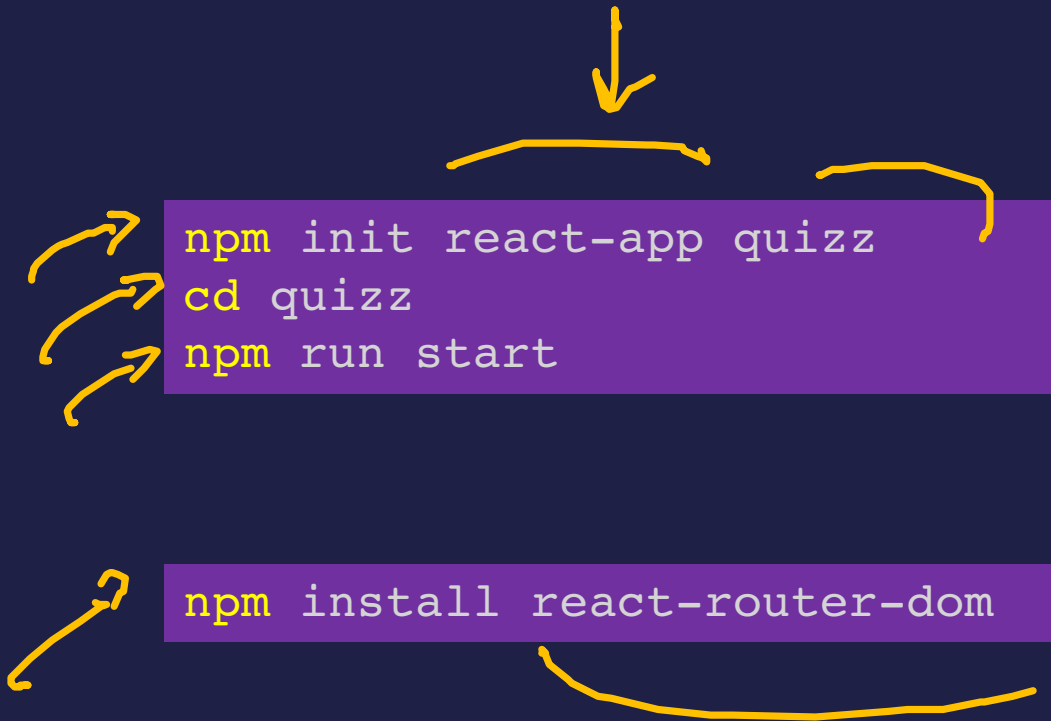
Un architecture de fichiers séparés

Un architecture de transport séparé

Un architecture applicative en
composants



2. LA CONSTRUCTION D'UN PROJET



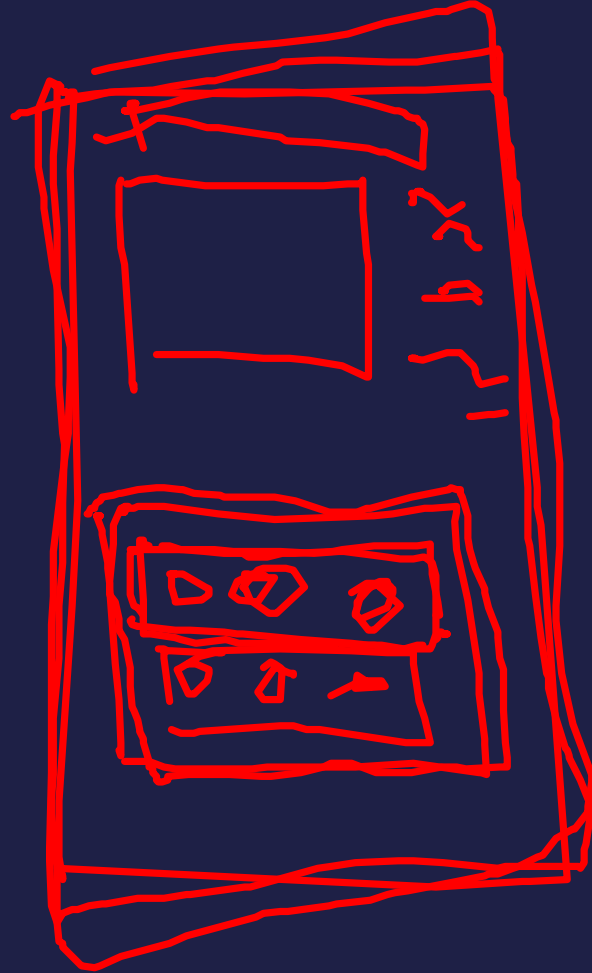
```
npm init react-app quizz  
cd quizz  
npm run start
```

A diagram showing a sequence of terminal commands. A yellow arrow points down to the first command. A yellow bracket connects the first and second commands. A yellow arrow points from the first command to the second. A yellow arrow points from the second command to the third. A yellow arrow points from the third command to the first command.



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

A diagram showing a terminal command. A yellow arrow points from the first command block to this command. A yellow bracket connects this command to the first command block.



1. LA PAGE

Une page est un
composant fait de
composants



```
function App() {  
  return (  
    <h1>It works!</h1>  
  );  
}  
  
export default App;
```

```
class App {  
  render() {  
    return (  
      <h1>It works!</h1>  
    );  
  }  
}  
  
export default App;
```



JSX = JS + X
XML

Les **functional components** sont des fonctions



Les **class components** sont des classes

2. LE ROUTING

ROUTE ?

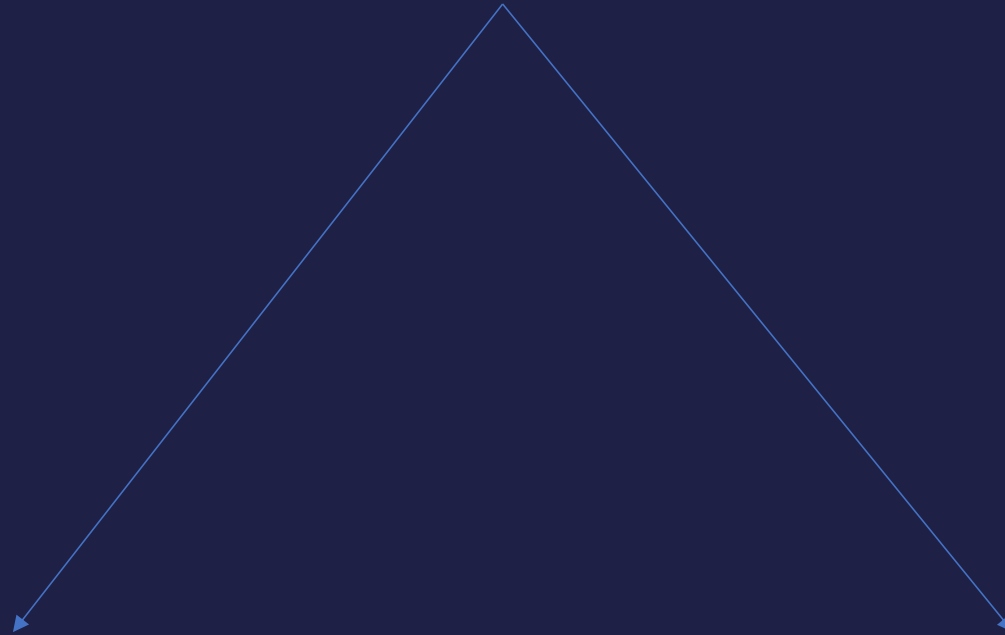
<https://localhost/blog/12/edit>

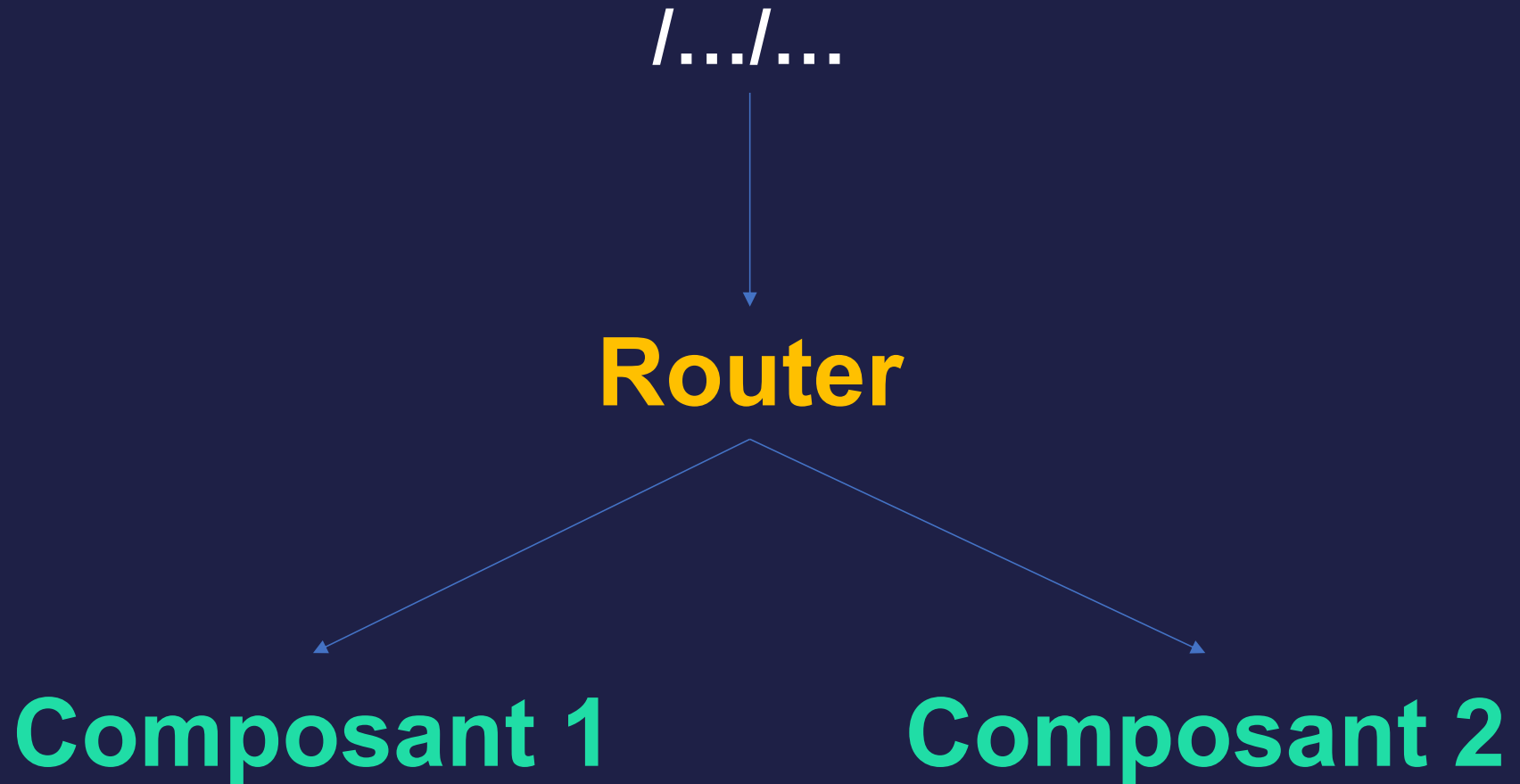
<https://localhost/blog/12/edit>

/.../...

Composant 1

Composant 2





```
import { HashRouter, Route, Switch, Redirect } from 'react-router-dom';  
// ... import Resultat, Page3, NotFound etc.
```

```
<HashRouter>  
  <main>  
    <Switch>  
      <Redirect from="/" to="/commencez" exact />  
      <Route path="/resultat" component={Resultat} exact />  
      <Route path="/question-3" component={Page3} exact />  
      <Route path="/question-2" component={Page2} exact />  
      <Route path="/question-1" component={Page1} exact />  
      <Route path="/commencez" component={Home} exact />  
      <Route component={NotFound} />  
    </Switch>  
  </main>  
</HashRouter>
```


La doc est magnifique

<https://fr.reactjs.org/>

3. LE STATE

State  **Vue**



Input

API

Tout !

State

Vue



Input

API

Tout !

Single
Source
Of
Truth

State

Vue



```
import { useState } from 'react';

function App() {

  const [minuterie, setMinuterie] = useState(3);

  return (
    <div>
      <div>{ minuterie }</div>
      <button onClick={() => setMinuterie(6)}>Augmenter</button>
    </div>
  );
}

export default App;
```


Les **formulaire**s

« Controlled components »

```
import { useState } from 'react';

function Home() {

  const [email, setEmail] = useState('');

  return (
    <input type="email" value={email} onChange={e => setEmail(e.target.value)} />
  );
}

export default Home;
```


Pour les **formulaire**s,
N'oubliez pas de
changer le state !

Pour les **formulaire**s,
N'oubliez pas de
changer le state !



4. PROPERTIES



```
function App() {  
  return (  
    <MonJolieBouton nomDuBouton="Acheter maintenant"></MonJolieBouton>  
  );  
}  
  
export default App;
```

```
function MonJolieBouton(props) {  
  return (  
    <button className="css-jolie">{props.nomDuBouton}</button>  
  );  
}  
  
export default MonJolieBouton;
```



```
function App() {  
  return (  
    <MonJolieBouton nomDuBouton="Acheter maintenant"></MonJolieBouton>  
  );  
}  
  
export default App;
```

```
function MonJolieBouton(props) {  
  return (  
    <button className="css-jolie">{props.nomDuBouton}</button>  
  );  
}  
  
export default MonJolieBouton;
```


5. LE CONTEXT

Context

Partagé par tous !
(pas besoin d'être parent)

```
import React from 'react';

const ContextGlobal = React.createContext({
  email: null,
  age: null
});

export default ContextGlobal;
```

```
import { useContext } from 'react';
import ContextGlobal from '../common/ContextGlobal';

function Home() {
  const context = useContext(ContextGlobal);

  const handleClick = () => {
    context.email = email;
  };

  return (
    <button onClick={handleClick}>Enregistrer</button>
  );
}

export default Home;
```

```
import { useContext } from 'react';
import ContextGlobal from './context-global'

function App() {
  const context = useContext(ContextGlobal);
  return (
    <div>
      Bonjour {context.email}
    </div>
  );
}

export default App;
```


8. LES REST API

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

Le **CRUD** sont les opérations de base de manipulation de données :

- **C** : Create
- **R** : Read/Retrieve
- **U** : Update
- **D** : Delete

```
list() {  
  return fetch('**URL DE L'API**', {  
    method: 'GET',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    }  
  }).then((response) => response.json());  
}
```

```
create(data) {  
  return fetch('**URL DE L'API**', {  
    method: 'POST',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    },  
    body: JSON.stringify(data),  
  }).then((response) => response.json());  
}
```

```
update(data) {  
  return fetch('**URL DE L'API**', {  
    method: 'PUT',  
    headers: {  
      'Accept': 'application/json',  
      'Content-Type': 'application/json',  
    },  
    body: JSON.stringify(data),  
  }).then((response) => response.json());  
}
```

```
delete(id) {  
    return fetch('**URL DE L'API**', {  
        method: 'DELETE',  
        headers: {  
            'Accept': 'application/json',  
            'Content-Type': 'application/json',  
        }  
    }).then((response) => response.json());  
}
```