

NEXT .JS




NextJS

Animé par Mazen Gharbi

Qu'est-ce donc

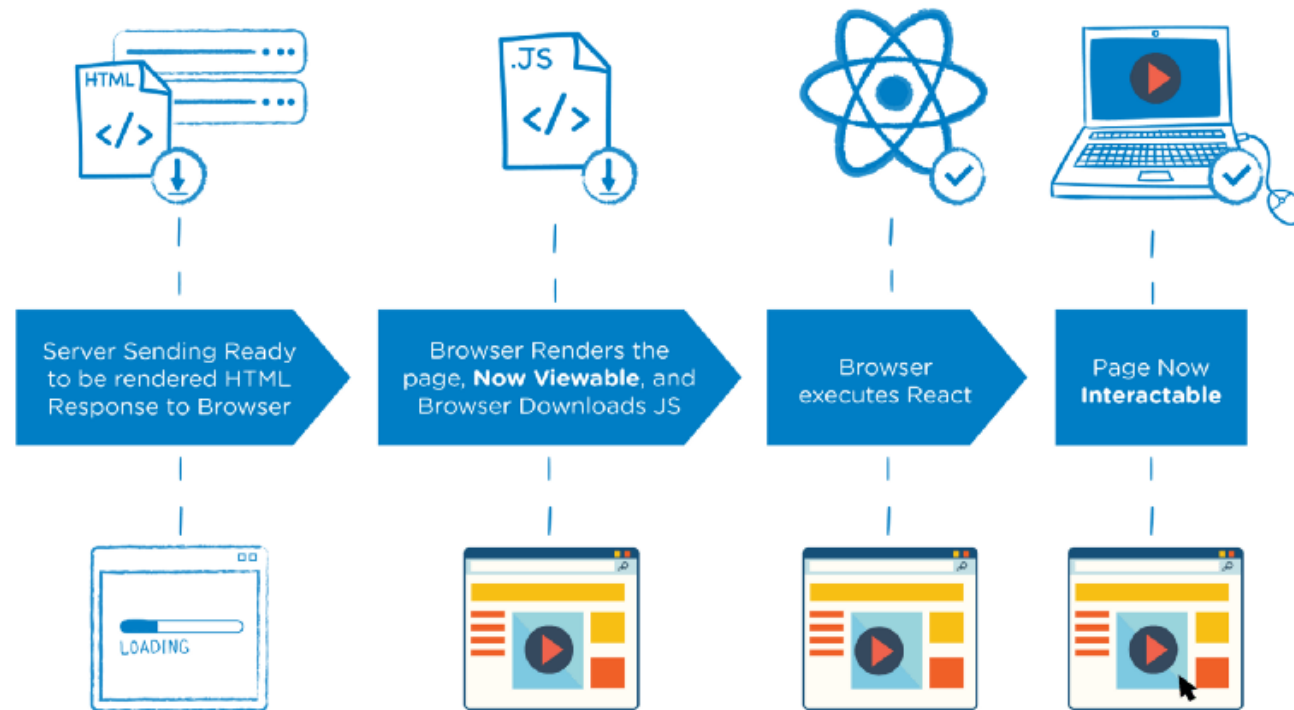
- ▷ Outil développé par [Zeit](#)
- ▷ Framework permettant d'effectuer le rendu des applications web React par les serveurs
 - › [Server Side Rendering](#)
- ▷ Bâti sur [React](#), Webpack et Babel
- ▷ Objectif : Concevoir les applications [client-serveur](#)

Server Side Rendering

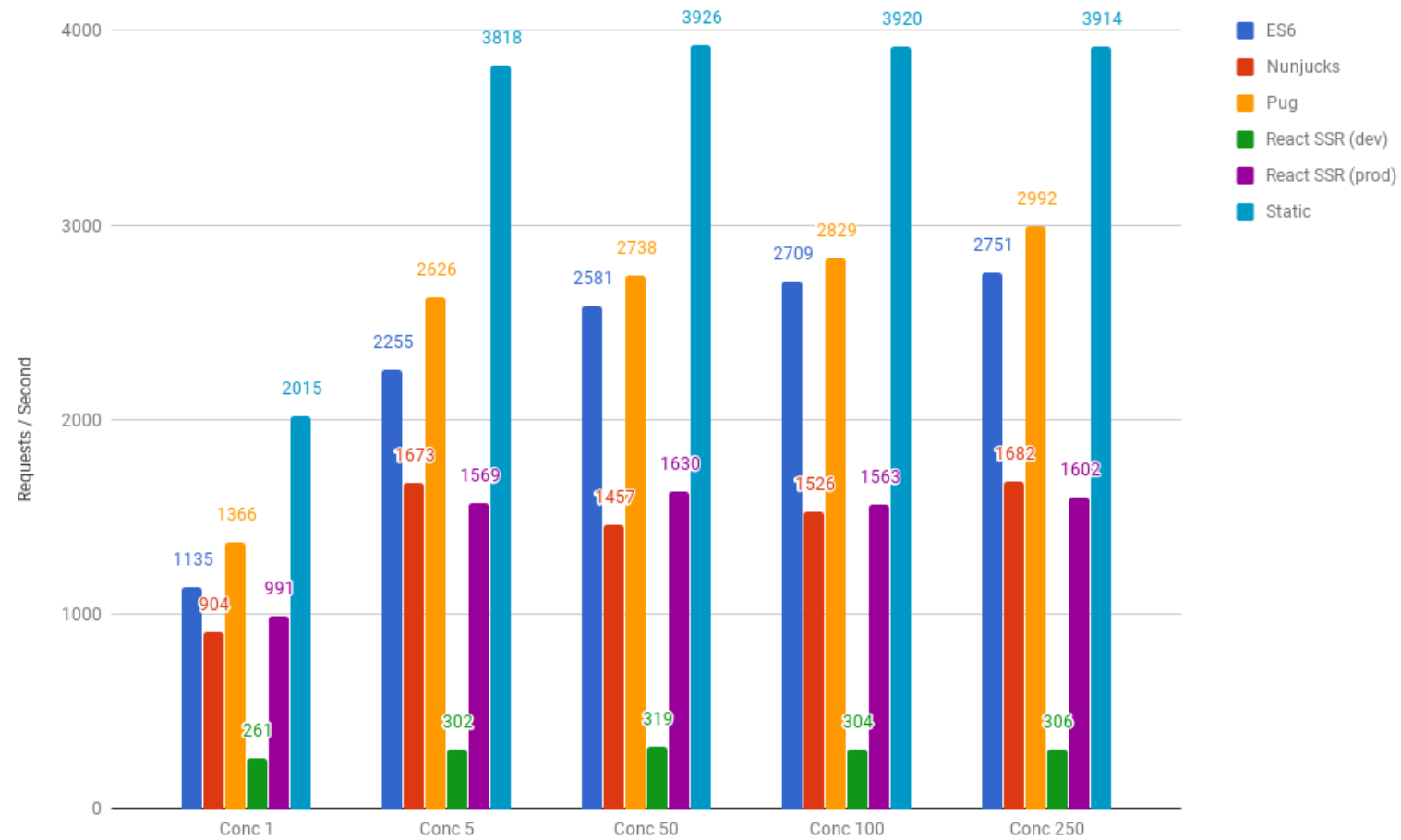
- ▷ Une application Angular, React ou Vue doit préparer son contexte avant d'afficher la page
 - › **Bootstrap time**
- ▷ On va chercher à optimiser ce temps de chargement initial :
 - › Pour optimiser le SEO 
 - › Améliorer l'expérience utilisateur !
- ▷ 2 choses possibles avec le SSR !

Server Side Rendering

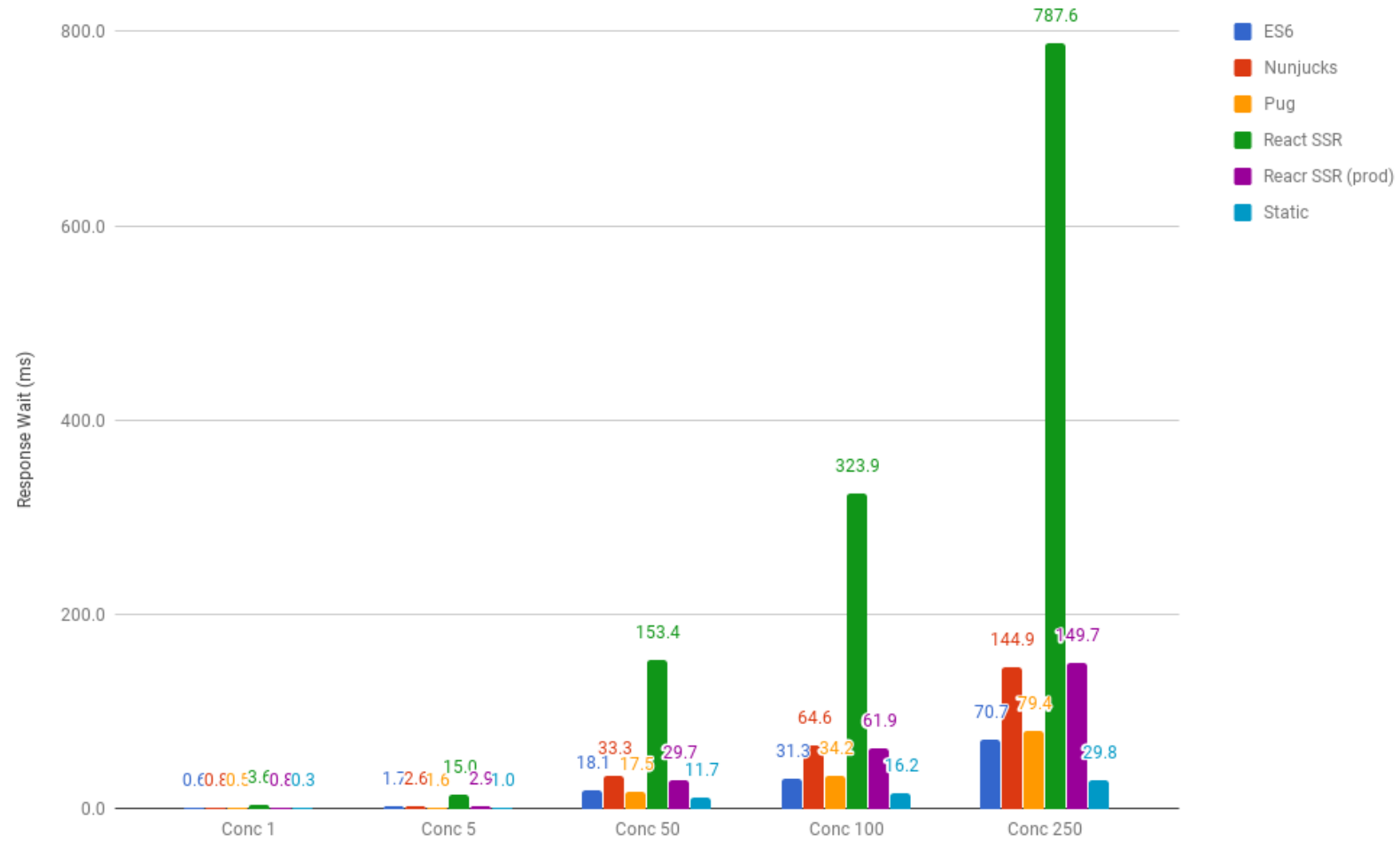
SSR



Affichage bien plus rapide



Contrepartie



Mise en place

- Next est un framework, il importe donc une configuration bien spécifique

```
> npx create-next-app
```

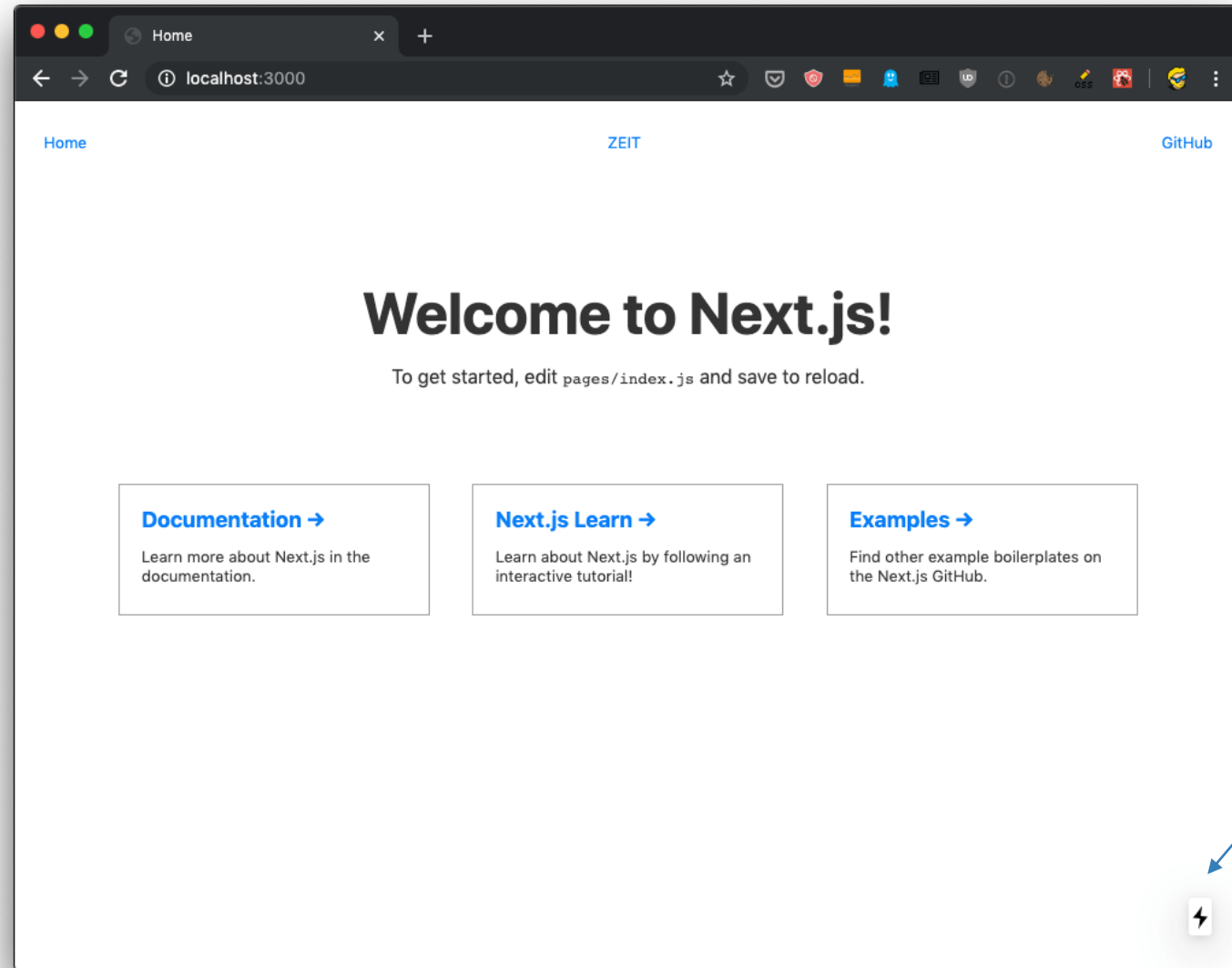
```
npx: installed 1 in 4.581s  
? What is your project named? » my-app
```



```
> npm run dev
```

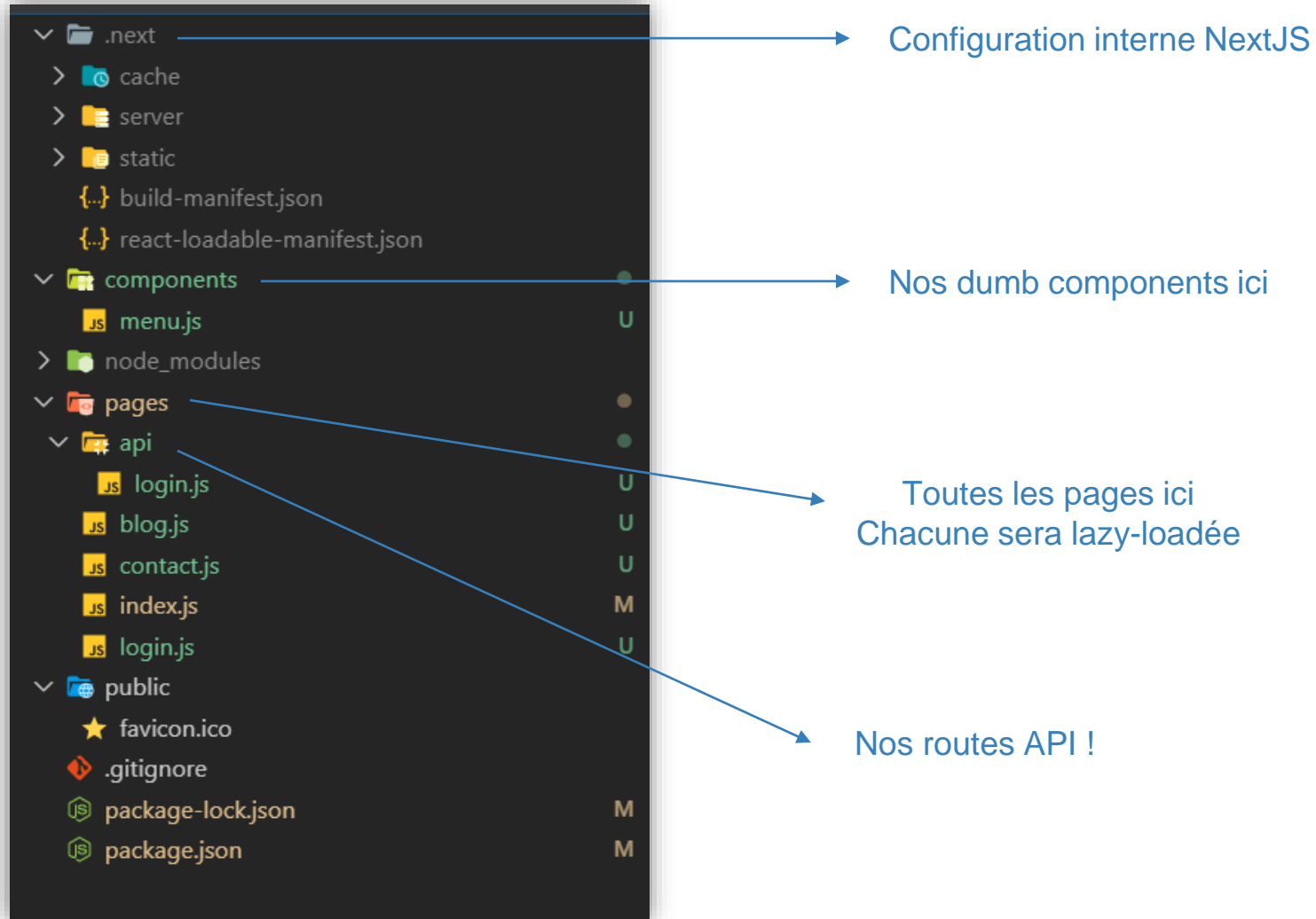
- On entre le nom de notre application et le projet est créé

Résultat



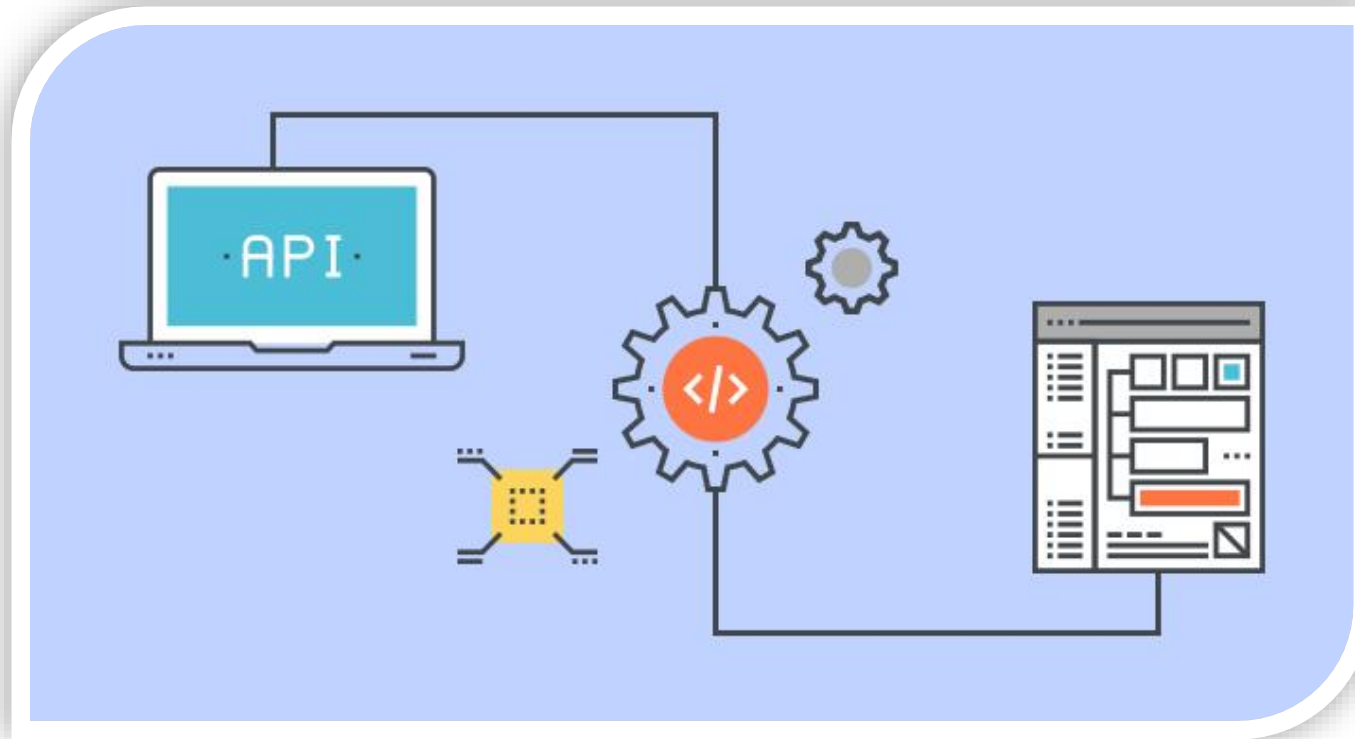
Indique que le SSR
est actif !

Architecture



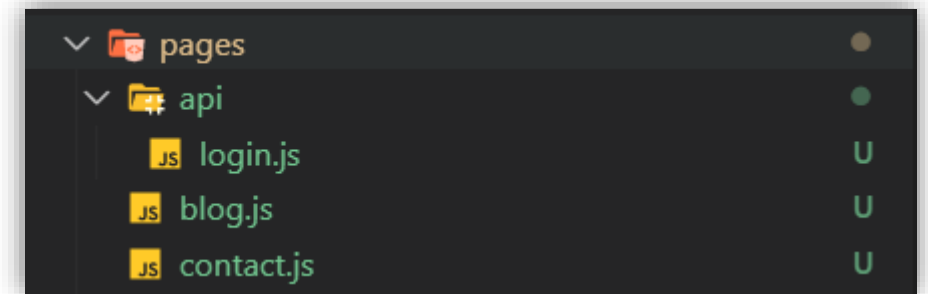
Front & Back

► Next nous permet de gérer des routes API avec lesquelles notre front pourra communiquer

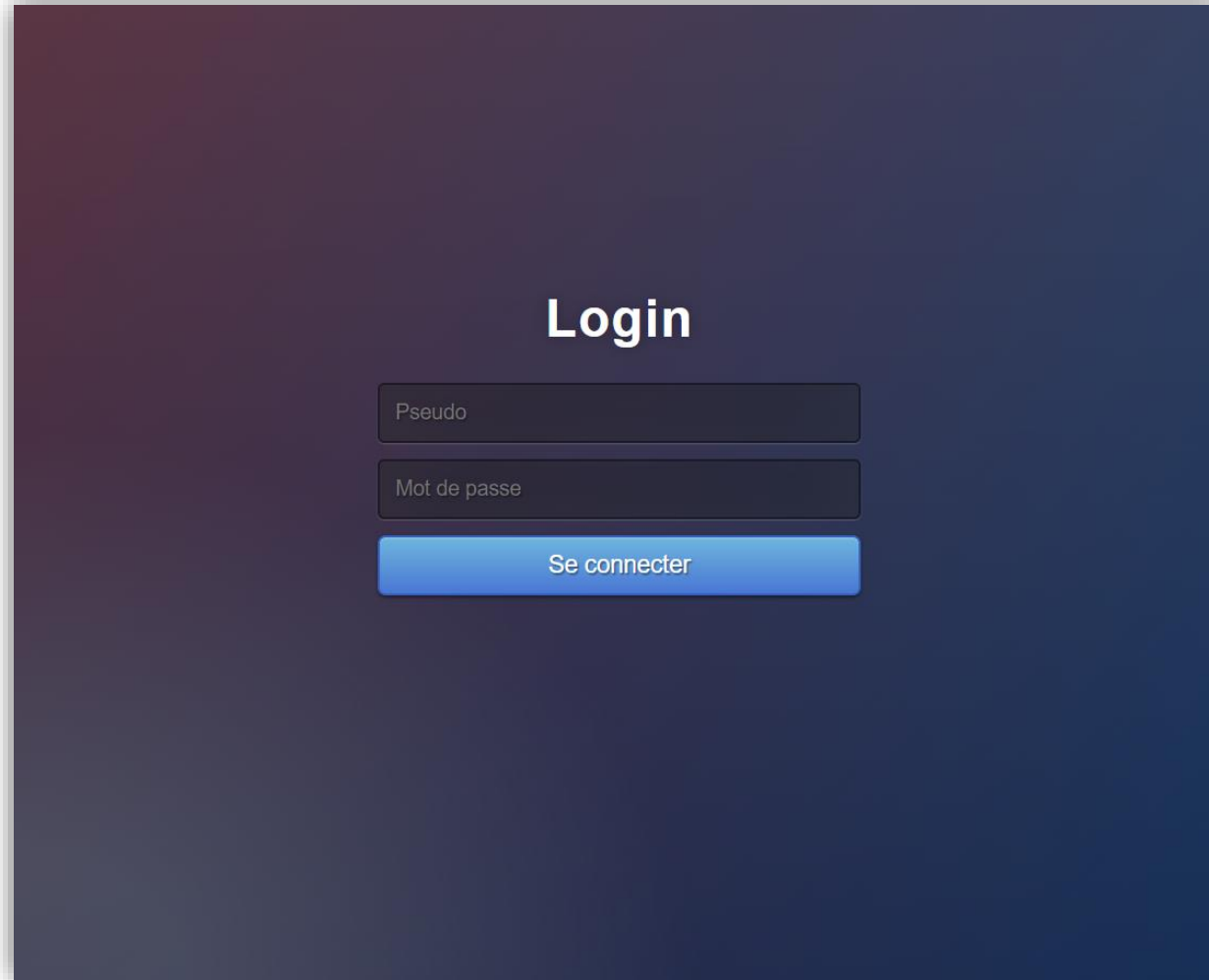


API

- ▶ On va mettre en place une route côté back-end qui permettra à notre application React de nous logger ;
- ▶ NextJS fournit une **solution simple** pour construire notre API
 - › Il faut commencer par créer un dossier **api/** dans le dossier **pages/**
- ▶ Chaque fichier dans **./pages/api** est mappé sur **/api/***. Par exemple, **./pages/api/login.js** est mappé sur la route **/api/login**.



Mise en place de la page de login

A login form mockup with a dark blue gradient background. The form is centered and contains the title 'Login', two input fields for 'Pseudo' and 'Mot de passe', and a 'Se connecter' button.

Login

Pseudo

Mot de passe

Se connecter

Mise en place de la page de login

pages/login.js

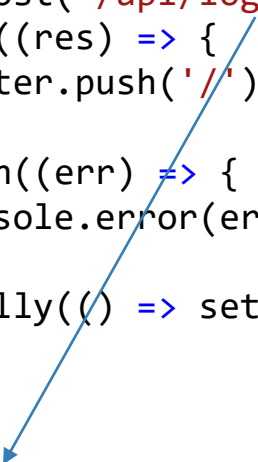
```
let [form, setForm] = useState({ username: '', password: '' });
let [loading, setLoading] = useState(false);
...
<form method="post">
  <input
    type="text" value={form.username}
    onChange={(ev) => setForm({ ...form, username: ev.target.value })}
    name="username"
    placeholder="Pseudo"
    required="required" />
  <input
    type="password" value={form.password}
    onChange={(ev) => setForm({ ...form, password: ev.target.value })}
    name="password"
    placeholder="Mot de passe"
    required="required" />
  {
    loading ||
    <button type="submit" onClick={onLogin} disabled={!form.username || !form.password}>
      Se connecter
    </button>
  }
</form>
```

Requête au serveur

pages/login.js

```
function onLogin() {
  setLoading(true);

  axios.post('/api/login', { username: form.username, password: form.password })
    .then((res) => {
      Router.push('/')
    })
    .catch((err) => {
      console.error(err);
    })
    .finally(() => setLoading(false));
}
```



```
function Login(req, res) {
  let { username, password } = req.body;
  if (username === 'admin' && password === 'password') {
    res.status(200).json({ log: true });
  } else {
    res.status(401).json({ error: 'Mauvais identifiants' });
  }
}
```

pages/api/login.js

Routing avec NextJS

- ▷ Contrairement au React natif, il n'y a aucun fichier permettant de déclarer les routes de notre application
- ▷ Chacune des routes est **représentée par un fichier**
- ▷ Pour une meilleure maintenabilité & performance, chaque route est **lazy-loadé**, donc le chargement de la page n'est fait qu'au moment de sa requête

Naviguer entre nos routes

- ▷ Comme pour le routing basique, un simple Link :

```
<Link href="/login">  
  <a>Login</a>  
</Link> -
```

- ▷ Mais l'import est différent :

```
import Link from "next/link";
```

- ▷ On peut également changer de route à partir du modèle :

```
import Router from 'next/router';  
Router.push('/');
```


Gestion des paramètres de route

- ▷ Ici malheureusement, c'est bien moins simple
- ▷ Il va être nécessaire de configurer le serveur ExpressJS pour rediriger l'url avec un ID vers cette même route avec un query

```
server.get('/blog/:id', (req, res) => {  
  return app.render(req, res, '/blog', { id: req.params.id })  
})
```

/server.js

- ▷ Et au moment de l'envoi :

/pages/login.js

```
<input type="number" placeholder="Id du blog" value={id} onChange={(ev) => setId(ev.target.value)} />  
<Link href={"/blog?id=" + id} as={"/blog/" + id}>  
  <button>On y va</button>  
</Link>
```

Donnée envoyé en tant que paramètre GET

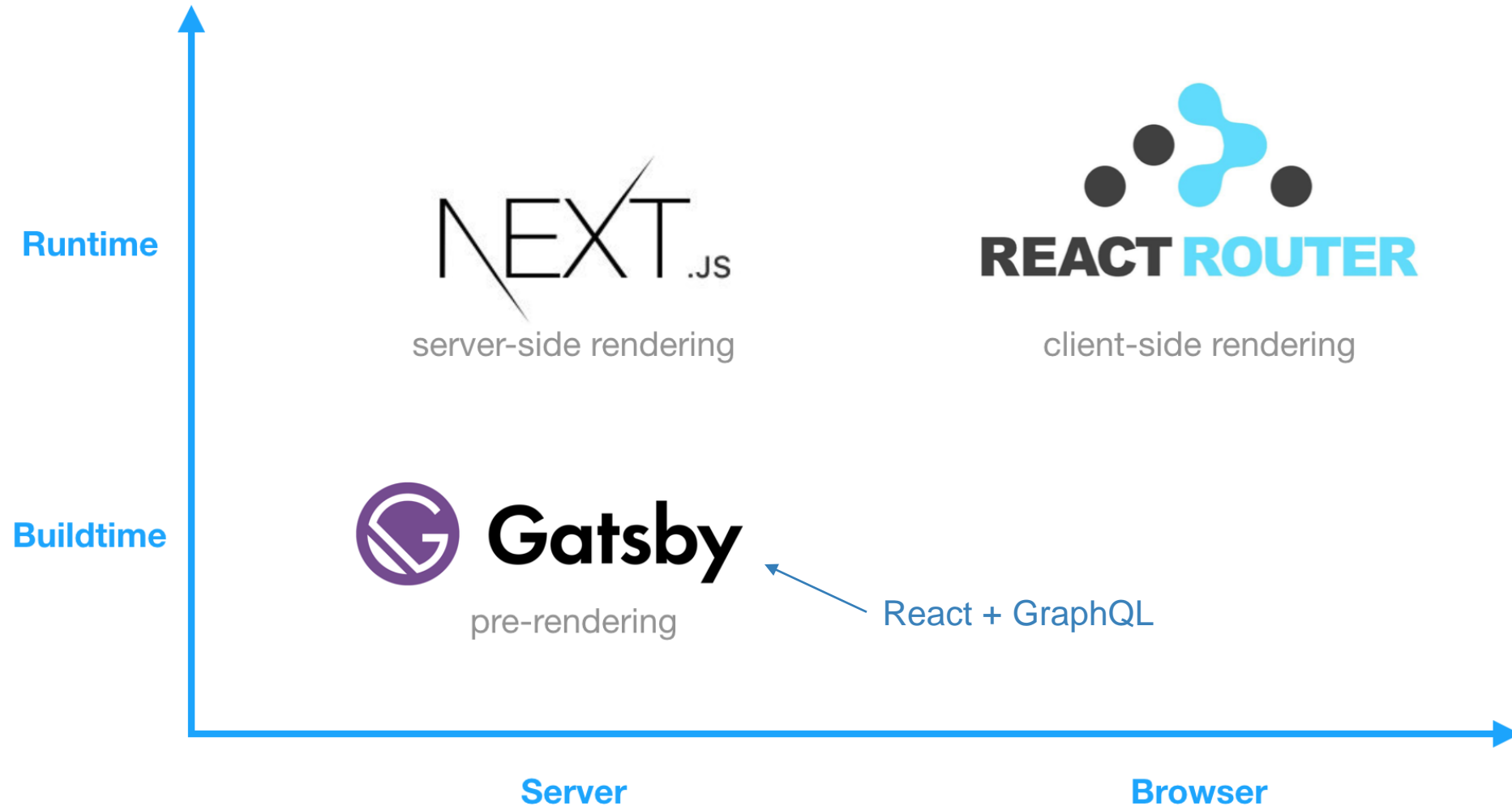
L'alias permet d'afficher l'URL sans le param GET

Pré-chargement des pages

- ▷ Lorsque vous appliquez la balise `<Link>` dans votre page, NextJS préchargera **automatiquement** le contenu de cette page en background
 - › Ce qui signifie qu'il téléchargera cette page pendant que vous naviguez
- ▷ Néanmoins, il existe certaines pages où vous savez pertinemment qu'elles ne seront que très rarement visitées, il est possible de désactiver le « prefetch » pour celles-ci :

```
<Link href="/ma-page" prefetch={false}>  
  <a>Ma page</a>  
</Link>
```

Pour finir



TP

- ▷ Créez une application NextJS avec 2 pages différentes :
 - › Page **Login** ;
 - › Page **Home**.
- ▷ Pour la connexion, créez une route API « **/login** » côté backend
- ▷ Créez également une route API « **/movies** » permettant de récupérer un tableau de film
- ▷ Affichez ce tableau de films dans la page **Home**

Questions ?