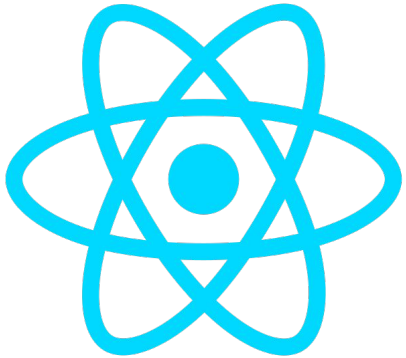
Applications web universelles avec React et nodejs

Rendus côté serveur d'application monopage



Les grandes avancées du web 1.0

- 1980 : ENQUIRE Document nodes
- 1990 : HTML : Hypertext edges
- 1994 : cookie
- 1994 : PHP
- 1994 : Yahoo!
- 1995 : javascript
- 1999: web 2.0

Pages
Network (www)
Server side sessions
Page Templates
Page Indexation
Rich Internet Application

Les deux stratégies majeures

Rendu à partir de modèles, côté serveur (gros .html, petit .js)

- persistence des données utilisateurs : cookies, sessions
- accès rapide à une base de données locale
- pages rendues mis en cache
- balises SEO / url
- les données utilisateurs injectées dans les modèles ne sont pas mises en cache et montent mal en échelle
- les scripts d'interface sont complexes isolés, même s'ils dépendent du template et vice-versa
- mauvaise scalabilité

Les deux stratégies majeures

Navigation / rendus côté client (petit .html, gros .js)

- temps de premier rendu lent / reste de la navigation très rapide
- très peu de bande passante html
- mise en cache des assets js
- très bonne scalabilité
- API's, microservices, etc pour servir le contenu et interagir avec l'utilisateur
- pas de balises SEO

Isomorphisme client/serveur

Challenge : Navigation ⇔ Requête url

avoir un même HTML/DOM que la page soit générée côté serveur ou côté client

- différences de langage
- différences d'api
- différence d'environnement
- différence de version
- ...

Application universelle

Challenge: maintenir une seule base de code et pouvoir l'exécuter côté client et côté serveur

- différences de langage:

Le Javascript est contrainte du browser, on choisit un serveur nodejs

- différence d'environnement:

React est naturellement 'agnostique' vis à vis de l'environnement d'exécution, instancier une app peut se faire sans être dans un navigateur.

react-dom monte le DOM, react-dom/server retourne du HTML.

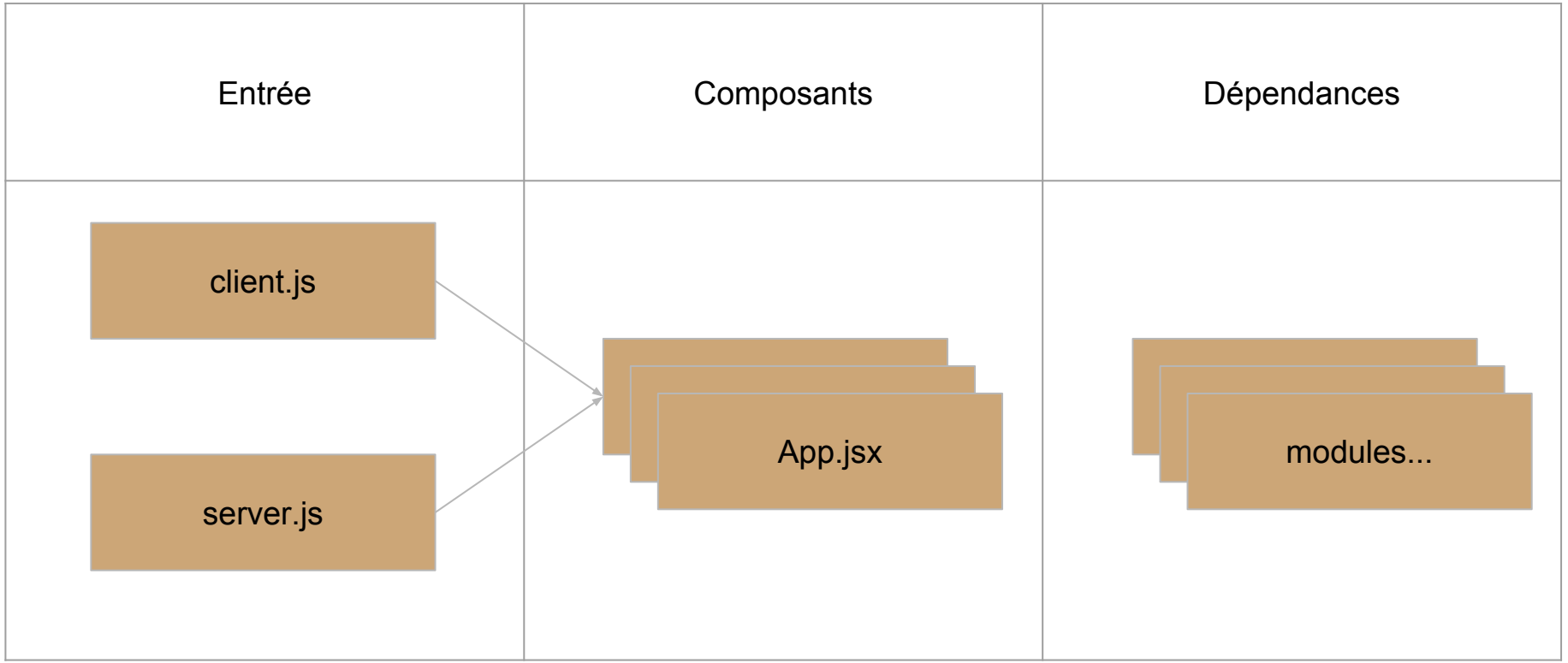
Ecrire du code universel

Même si react est agnostique, notre code va surement faire appel à l'API du navigateur ou celle de node... qui divergent.

- différences d'api

Challenge: Maintenir un code universel qui s'exécute toujours sans erreur côté serveur ET côté client

Échapper du code côté serveur / côté client



Point d’entrée du serveur express

Challenge : React n’a pas de callback “j’ai fini de monter une app”, comment gérer certains composants / pages qui font appels à des ressources récupérées de façon asynchrone…

Solutions :

- parser l’url et télécharger les props avant d’instancier l’application
- implémenter des espions à injecter dans l’application pour faire remonter les états résolus

A noter : certaines fonctionnalités comme setCookie(), ou le SEO, peuvent facilement être contrôlés par les composants react avec cette dernière méthode…

server.js

```
import express from 'express';
import { match, RouterContext } from 'react-router';
import { renderToString } from 'react-dom/server';

import routes from './app/routes';

const app = express();
```

```
app.get('*', (req, res) => {
   match({
        routes,
        location: req.url
   }, (err, redirectLocation, renderProps) => {
        if (err) {
            res.status(500).send();
        } else if (redirectLocation) {
            res.redirect(302, redirectLocation.pathname);
        } else if (!renderProps) {
            res.status(404).send();
        } else {
            const reactApp = renderToString(
                <RouterContext { ...renderProps } />
            );
            res.send(
`<!DOCTYPE html>
   <body>
        <div id="react-app">
            <div>${reactApp}</div>
        </div>
   </body>
`
            );
        }
   });
});
```

Point d’entrée du client

De nombreuses librairies externes vont devoir être configurées avant de lancer react, comme Redux par exemple. On peut aussi vouloir réhydrater un état à partir du localstorage, ou récupérer des variables d’environnement émises par le serveur et injectées dans le HTML

client.js

```
import { render } from 'react-dom';
import { Router, browserHistory as history } from 'react-router';
import routes from './app/routes';
```

```
const router = {
    history,
    routes
};

render(<Router {...router} />, document.getElementById('react-app'));
```

Les composants React : cycles de vie

Un composant react stateful peut implémenter des méthodes cycles de vie, comme

- componentWillMount
- componentDidMount
- componentWillUpdate
- componentDidUpdate
- …

Mais côté serveur, seules les méthodes ‘will’ mount et update seront invoquées.
On peut mettre sans risque du code dépendant d’un browser dans ces méthodes.

détecter l’environnement

Pour le reste du code, petites fonctions ou gros exports, on peut tester l’environnement avec :

```
if (typeof window === "undefined") {
        ...
} else {
        ...
}
```

Les modules compatibles avec les app universelles

react helmet permet de changer le titre d'une page côté client avec l'api du navigateur, et de créer un objet 'head' côté serveur, que l'on n'a plus qu'a injecter dans la head du HTML.

```
        const reactApp = renderToString(
            <RouterContext { ...renderProps } />
        );
        const head = Helmet.renderStatic();
        res.send(
`<!DOCTYPE html>
  <body>
      <head>
          ${head.title.toString()}
          ${head.meta.toString()}
          ${head.link.toString()}
      </head>
      <div id="react-app">
          <div>${reactApp}</div>
      </div>
  </body>
`
        );
```

```
const Product = ({product}) => {
 const url = `${getEnv('basePath')}/product/${product.id}`;
 return (
   <div className="product-container">
     <Helmet>
       <title>{product.title}</title>
       <meta property="og:type" content="article" />
       <meta property="og:title" content={product.title} />
       <meta property="og:image" content={product.image} />
       <meta property="og:url" content={url} />
       <meta property="og:description" content={product.description} />
       <link rel="canonical" href={url} />
     </Helmet>
     <ProductImage image={product.image} />
     <ProductDescription description={product.description} />
     <ProductAvailability remaining={product.remaining}
total={product.total} />
   </div>
 );
};
```