Conegui els marcs React i JavaScript

React és una biblioteca JavaScript per crear interfícies d'usuari. Intenta fer que les aplicacions web complexes tinguin menys temps de construir, siguin més fàcils de mantenir i siguin més performants . És un dels marcs web JavaScript més populars que s'utilitzen avui en dia i sovint ha liderat el camp en el rendiment i l'adopció de nous patrons de disseny.

React també compta amb una enorme i activa comunitat mundial i el suport de Facebook , cosa que la converteix en una opció fiable per als desenvolupadors que es preguntin on invertir el seu temps.

En aquest curs coneixerem els conceptes bàsics per escriure una aplicació a React i crearem una aplicació senzilla que permeti als usuaris crear i gestionar una petita llista de productes en una botiga.

Conceptes clau

Vegem algunes de les idees clau de React. Escoltarà aquests termes molt a Internet i en debats sobre marcs javascript i aplicacions de pàgina única en general, així que val la pena conèixer què volen dir. 🤔

```
### SPAs (sol·licituds d'una sola pàgina)

Quan parlem de "aplicacions" de Javascript, normalment parlem de SPAs. Les
sigles fan referència als llocs web que gestionen la navegació a la pàgina,
sense carregar pàgines noves mitjançant HTTP. Un lloc web de SPA normalment
es carrega una vegada i, després, gestiona tots els enllaços i enviaments
de formularis amb Javascript mitjançant sol·licituds AJAX en segon pla.
Javascript també gestiona els canvis a la pàgina i fins i tot les
actualitzacions d'adreça URL.
```

El DOM virtual

El DOM virtual és un objecte Javascript que descriu l'estat actual de la pàgina web de la vostra aplicació de la manera més senzilla possible. Per descomptat, hi ha un altre objecte Javascript que ja descriu l'estat de la pàgina web: el Model d'objectes de document o DOM .

```
Per què tenen dos DOM?
```

El DOM real és la representació de la pàgina web del navegador i té molts usos. També és un objecte molt gran i és bastant costós d'interactuar. El DOM virtual, en canvi, és la representació de la pàgina web de la vostra aplicació i és molt més petit i és menys costós d'actualitzar.

El DOM virtual permet al marc fer un seguiment de la seva pròpia idea de com hauria de ser la pàgina i comparar-ho amb l'estat anterior de l'aplicació, sense incórrer en la sobrecàrrega de modificar i llegir constantment l'objecte DOM real del navegador. Això va resultar fantàstic per al rendiment i algunes versions d'aquest enfocament han estat adoptades per molts altres marcs de JavaScript.

Components

A React i a la majoria d'altres marcs Javascript actuals, en lloc d'escriure una pàgina web com una pàgina llarga d'elements HTML niats, definim components que encapsulen blocs de funcionalitat i disseny. Poden ser tan petits o tan grans com vulgueu, però, com passa amb les funcions de Javascript, el codi ben organitzat produeix components relativament fàcils de llegir i entendre.

Components web

Els navegadors avancen cap a l'adopció d'un concepte diferent però relacionat anomenat Components web . Es tracta d'elements de pàgina reutilitzables que encapsulen fragments de funcionalitat, com ara una entrada de cerca que ofereix visualitzacions prèvies dels resultats de la cerca o un botó que mostra un suggeriment al cursor.

Els components web us permeten crear funcions complexes en un element HTML personalitzat que podeu compartir i reutilitzar a qualsevol pàgina web sense preocupar-vos de si és compatible amb altres biblioteques de components o JavaScript de la pàgina.

Sabeu que els components web són una tecnologia web nativa i que són diferents dels components React. Complementen els components React i són interoperables amb ells .

Modularitat

React no és un marc JavaScript complet. La seva feina és gestionar els elements de la pàgina web i assegurar-se que reflecteixen correctament les seves dades. El programador decideix com gestionar la comunicació amb el servidor. 🖳🔄📱

El fet que React sigui modular d'aquesta manera significa que es pot utilitzar en molts contextos diferents per resoldre problemes de rendiment de la representació i organització del codi. De fet, la comunitat React ha desenvolupat diverses solucions com Flux i Redux per completar l'arquitectura de la vostra aplicació.

Aquesta modularitat també significa que els desenvolupadors poden triar moltes altres opcions per desenvolupar una aplicació frontend, més enllà de "farem servir React". Hi ha diferents arquitectures, biblioteques i cadenes d'eines que configuren l'entorn d'una aplicació React, i moltes d'aquestes configuracions tenen els seus propis recursos i comunitat. És un ecosistema molt innovador, però hi ha molt a aprofitar.

Com a resultat, al nostre curs ens centrarem en React com a biblioteca d'interfície d'usuari autònoma i utilitzarem una eina de creació , create-react-app , que us abstracte de la majoria de les opcions de la cadena d'eines. Tingueu en compte que React sovint es combina amb una biblioteca com Flux o Redux a la pràctica, i que qualsevol de les dues biblioteques farà un excel·lent pas per conèixer l'ecosistema React quan hagueu acabat aquest curs.

HTML a Javascript i JSX

HTML es va dissenyar originalment com una eina per crear documents principalment estàtics que els usuaris podien navegar per la xarxa mundial. Els primers dies, Javascript era útil per millorar aquestes pàgines amb elements interactius senzills com menús desplegables. Javascript tractava de manipular un document la definició principal del qual era en HTML.

Tot i així, el Javascript de les pàgines web d'avui s'ha tornat cada vegada més complex, fins al punt que sovint no estem segurs de si volem mirar l'HTML o el Javascript per entendre el document. 😩 Aquest tipus de Javascript està ple de fragments de codi que identifiquen els llocs del document HTML mitjançant selectors i, sovint, creen elements nous per inserir a l'HTML que no hi eren al document origen original o modificant el document original d'altres maneres .

```
Està bé fins a un punt, però pot sortir de les mans, donant lloc a un codi
que és difícil de llegir, encara més difícil de raonar i, per tant, difícil
de mantenir.
```

Els frameworks Frontend resolen aquest problema donant la volta a la relació i definint HTML dins de Javascript. Eviten la "sopa de selecció" 🍜 organitzant la lògica de renderització en una forma coherent i reproduïble que sigui fàcil de llegir i raonar.

Cada framework té una manera lleugerament diferent de gestionar HTML a Javascript. Aquests enfocaments s'anomenen **llenguatges de plantilla**. La majoria dels marcs separen Javascript de les plantilles que defineixen HTML en fitxers separats, però React utilitza un llenguatge de plantilla anomenat **JSX** , que literalment combina HTML i Javascript en un sol fitxer. Això significa que sovint, a React, escriviu codi a JSX, en lloc de Javascript estricte. No us preocupeu, però, és només una forma augmentada de Javascript i tot el que heu après sobre Javascript encara s'aplicarà aquí. 👌

Marcs Javascript i renderització retardada

La representació del client que fan les aplicacions d'una sola pàgina significa la renderització retardada i el lliurament de més JavaScript al client. Aquesta és una limitació que tenen tots els marcs JavaScript, tot i que és ideal per mitigar-la gràcies a una interacció més ràpida després de carregar la pàgina.

Atès que aquest tipus d'aplicacions web no serveixen directament com a contingut HTML principal, hi ha un retard després de carregar la pàgina HTML inicial, abans que Javascript renderitzi l'aplicació. Especialment en dispositius mòbils, això pot suposar un llarg retard, ja que el framework sovint té una gran biblioteca per descarregar a més del codi de l'aplicació que el programador escriu ell mateix.

Els marcs treballen cada vegada més per solucionar aquest problema creant motors de representació del servidor a Nodejs que poden renderitzar prèviament l'aplicació com a HTML estàtic, de manera que el navegador tingui HTML per llegir-lo immediatament a la càrrega de la pàgina. Les comunitats React , Ember i Angular 2 estan desenvolupant versions d'aquesta tecnologia.

Ara que teniu alguns antecedents sobre els conceptes clau de React, donem un cop d'ull a la manera de construir la vostra primera aplicació.

Utilitzeu 'create-react-app' per crear la vostra aplicació React

Utilitzarem la línia d'ordres a la vostra màquina local per crear la vostra primera aplicació React. Per fer-ho, haureu d'instal·lar Nodejs i algunes biblioteques, si encara no en teniu.

Les instruccions d'instal·lació poden semblar avorrides, però seguiu-les tal com s'indica i hauríeu d'estar bé.

Nodejs

En primer lloc, els Nodejs, sempre populars.

Aneu a la pàgina d'instal·lació del gestor de paquets de node.js , seleccioneu el vostre sistema operatiu i seguiu les instruccions per descarregar i instal·lar Nodejs a l'ordinador.

Un cop fet això, assegureu-vos que Nodejs s'estigui executant al vostre sistema. Depenent del tipus de màquina que utilitzeu, és possible que hàgiu d'obrir una nova finestra de terminal o tornar a obtenir el vostre shell. Sabreu que heu funcionat si podeu escriure

```
node --help
```

a la pantalla i obteniu una llista d'ordres i les seves descripcions

Instal·lació de Yarn

Yarn és un gestor de paquets per a Nodejs. L'utilitzaríem per instal·lar create-react-app i configurar-nos amb una aplicació React que funcionés.

Aneu a la pàgina d'instal·lació del projecte de filats i seleccioneu la pestanya del vostre sistema operatiu. Seguiu les instruccions aquí per instal·lar Yarn a la vostra màquina.

Instal·lació de create-react-app

Ara que això està fora del camí, podem tornar a React. Amb Yarn instal·lat, executeu des de la vostra línia d'ordres.

```
yarn global add create-react-app
```

Per provar que funciona, executeu

```
create-react-app --help
```

i assegureu-vos que us mostri una llista d'ordres i instruccions per utilitzar-les.

Si no veieu la sortida esperada, alguna cosa ha fallat en algun lloc del camí i potser necessiteu ajuda per configurar el vostre entorn. Si és així, no tingueu por de demanar ajuda.

```
La gestió del vostre entorn local és una tasca complexa plena d'obstacles i
és una habilitat per si sola.
```

Crea una aplicació React!

Un cop hàgiu creat-react-app instal·lat i funcionant, penseu en un nom per a la vostra aplicació i executeu-lo

```
create-react-app my-app-name
```

Això hauria de produir un llarg flux de sortida, començant per un missatge com

```
S'està creant una nova aplicació React a ...
```

i acabant amb un missatge d'èxit i instruccions per publicar la vostra nova aplicació React.

Hauríeu de veure una llista de fitxers de sortida. Si inspeccioneu el vostre sistema de fitxers, ara hauríeu de veure el contingut de la vostra nova aplicació React. A la secció següent veurem aquests fitxers generats.

Ta-ta, ja heu acabat el procés d'instal·lació.

Mireu què s'ha instal·lat i generat

Vegem l'estructura de la vostra nova aplicació React:

```
node_modules/
public/
src/
.gitignore
README.md
package.json
yarn.lock
```

La major part del que ens preocuparà en aquest curs es troba dins del **src/** directori, però dediquem-nos un moment a revisar la resta del contingut del directori arrel.

node_modules / i package.json

Aquests us haurien de ser familiars si heu tingut alguna experiència treballant en un projecte node.js en el passat. En resum, **node_modules/** conté totes les biblioteques i el codi extern que el gestor de paquets carrega en funció de la llista de dependències que definiu package.json .

yarn.lock

De la mateixa manera, "yarn.lock" es refereix a la gestió de paquets. Si **package.json** descriu els requisits de dependència de l'aplicació, es **yarn.lock** descriu exactament què ha d'instal·lar per complir aquests requisits, fins a la versió exacta de cada dependència.

Aquest fitxer és útil per fer un seguiment de la versió exacta de les dependències que utilitzeu a la vostra aplicació. Si utilitzeu npm, un altre gestor de paquets de nodes popular, ni tan sols tindreu aquest fitxer de bloqueig.

```
No canvieu res   yarn.lock.
```

README.md

Aquest fitxer generat conté una àmplia **documentació** sobre l'ús de **create-react-app** . 📝 Hi ha molta informació útil sobre el desenvolupament i el manteniment de la vostra create-react-app aplicació. Tingueu en compte que generalment fa referència al gestor de paquets npm i, ja que estem utilitzant yarn, haureu de modificar les ordres npm adequadament.

```
Podeu trobar molta ajuda al web per traduir ordres de npm a fil, si ho
necessiteu.
```

.gitignore

create-react-app genera un **.gitignore** per a vosaltres, que no dubteu a substituir-los o ampliar-los com convingueu.

públic /

Public/ és l'arrel pública del vostre lloc. Els fitxers aquí són accessibles a qualsevol persona del web. És un bon lloc per guardar fitxers i recursos estàtics que no es gestionen al src/ directori de la vostra aplicació React.

src /

El codi de la vostra sol·licitud. 😎

hauria de ser:

```
App.css
App.js
App.test.js
index.css
index.js
logo.svg
```

Podeu veure que aquí hi ha CSS i SVG a més de JavaScript. Això es deu al fet que les nostres eines de compilació us permeten importar altres recursos directament al vostre codi JavaScript, de la mateixa manera que us permeten expressar HTML directament al vostre codi.

A la secció següent veurem els fitxers JavaScript i com es relacionen tots.

Gestió de recursos no javascript a React

Quan consultem el **src** directori del capítol següent, és possible que observeu **import** s'utilitzen sentències JavaScript per carregar actius que no siguin JavaScript, com ara CSS, imatges i tipus de lletra web.

Un lloc web tradicional carrega recursos com aquests des de fitxers remots basats en URL en **src** atributs d'etiquetes HTML o regles de full d'estil, però els carregadors Webpack (que create-react-app utilitza entre bastidors per crear la vostra aplicació) us permeten carregar-los directament a Javascript des de fitxers al directori 'src'.

```
Webpack ho permet analitzant el contingut dels fitxers CSS, imatge i tipus
de lletra que voleu importar i fent que aquests continguts estiguin
disponibles com a valors emmagatzemats en un mòdul JavaScript.

D'aquesta manera, és fàcil integrar aquests recursos a la compilació
d'aplicacions i gestionar totes les dependències de frontend amb una sola
eina.
```

Muntatge React dins de la pàgina web

Abans d'entrar al codi JavaScript, fem una ullada a on comença el navegador amb la vostra pàgina web.

Aquest és el fitxer HTML estàtic des del qual es publica **./public/index.html** . En mirar aquest fitxer, s'indica que realment és un intèrpret d'ordres d'una pàgina web. Bàsicament, aquí no hi ha res d'interès per a un usuari final. Principalment, hi ha un enllaç a un favicon i un div buit al cos amb un identificador de **root.**

Mirant-ho més de prop, aquesta línia de l' etiqueta conté un URL estrany:

```
        <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
```

Això no funcionaria si es servís a un navegador tal qual, i el comentari generat a continuació explica que tot aquest fitxer és realment un objectiu de l'escript de compilació. Bàsicament, el procés de compilació generarà una nova versió d'aquest fitxer que té l'URL adequat aquí.

De fet, el slug **%PUBLIC_URL%** es modificarà al lloc allà on aparegui en aquest fitxer, cosa que us permet col·locar altres recursos a la URL pública aquí i enllaçar-los dins d'aquest fitxer en format HTML senzill.

L'altra cosa que interessa aquí és la línia:

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

Es defineix l'element en què es renderitzarà l'aplicació React. Tingueu en compte que aquesta pàgina HTML estàtica no ha de quedar en blanc. Podeu afegir el marc de disseny estàtic i els elements que preferiu al voltant de l' **root** element i saber que l'aplicació apareixerà dins d' aquest element quan s'executi JavaScript.

```
Tingueu en compte que aquí no hi ha etiquetes per carregar JavaScript ni
CSS. Aquestes etiquetes també es generaran mitjançant el procés de
compilació i s'inseriran al <body> element aquí.
```

Vegem ara el fitxer JavaScript que s'encarrega de muntar la vostra aplicació dins del **root** element.

```
./src/index.js
```

El nom d'aquest fitxer suggereix la seva relació amb **./public/index.html** . De la mateixa manera **./index.html** que el navegador busca les instruccions inicials per crear la vostra pàgina web, **./index.js** és on React cerca les instruccions inicials per crear la vostra interfície d'usuari.

Com que aquest és també el primer fitxer Javascript que hem analitzat, dediquem un minut a entendre què passa aquí

Mòduls ES6

Si no heu treballat amb JavaScript compilat al servidor, és possible que les declaracions **import** de la part superior del fitxer semblin estranyes. **import** és una expressió del patró del mòdul a JavaScript. Ens permet importar unitats de funcionalitat JavaScript d'altres fitxers JavaScript.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';
```

Path vs. imports

Les dues primeres línies importen la funcionalitat principal de la pròpia biblioteca React. La tercera i la quarta línia importen mòduls des de la nostra pròpia aplicació.

Tingueu en compte que la tercera i la quarta línia anterior utilitzen camins de fitxers, mentre que les dues primeres línies només fan servir noms de mòduls. Això es deu al fet que les línies 1 i 2 invocen mòduls definits a les **./package.json** dependències. De fet, tots els mòduls definits en aquesta llista es poden importar per nom aquí (un cop hàgim executat la instal·lació **yarn**).

Els mòduls que no s'anomenen encara es poden carregar especificant un camí al fitxer on es defineixen aquests mòduls. Els components i altres mòduls definits a la nostra aplicació pertanyen a aquest segon grup, i això és el que s'està important a les línies 3 i 4 anteriors.

Anem a fer una ullada a aquests fitxers en un minut, però primer anem a veure com es munta l'aplicació.

```
Les   **import**  sentències de la part superior del fitxer contenen
cadascun dos elements definits per l'usuari: el nom o la ruta del mòdul a
```

```
importar i el nom de la variable a la qual s'assignarà el mòdul carregat.
Podeu utilitzar els noms de variables que vulgueu aquí, però quan carregueu
mòduls amb nom, és una bona idea fer servir un nom de variable que
coincideixi amb el nom de variable del mòdul al fitxer on està definit.
```

Renderització de la pàgina web

A continuació, tenim el codi que invoca alguns dels mòduls importats per dir a React que renderitzi la interfície d'usuari dins de l' root element de la pàgina web:

```
ReactDOM.render(
    <App />,
    document.getElementById('root')
);
```

En primer lloc, vegem la **render** funció. Pertany al mòdul ReactDOM, que es va importar a la línia 2 anterior. **render()** adopta dos arguments:

```
. un component React per representar

. un node DOM que serveixi de contenidor de renderització.
```

Ja hem analitzat com **root** s'ha definit l' element **./public/index.html** , de manera que el segon argument es proporciona aquí mitjançant la consulta del DOM.

El primer argument és on comença a prendre forma la nostra aplicació. Aquesta és una referència al component "Aplicació", que podem trobar definit a **./src/app.js** .

JSX

src/index.js té una extensió de fitxer **.js** , però en realitat no és JavaScript vàlid, com podeu veure en aquesta línia:

```
    <App />,
```

Això provocaria un error de sintaxi si s'executés en un intèrpret de JavaScript. Aquesta és una expressió del llenguatge de plantilla JSX. Serà substituït per l'intèrpret JSX pel component anomenat dins dels claudàtors. Per cert, també podeu escriure aquí etiquetes HTML "normals".

És possible que observeu que **React** s'importa a la part superior del fitxer però no s'utilitza en cap lloc del codi. En un fitxer Javascript, seria segur suprimir una biblioteca importada no utilitzada com aquesta. Però a JSX, hem d'incloure React perquè les etiquetes JSX es compilaran en trucades a **React.createElement** , i el codi s'estavellaria si l'objecte **React** no fos present a la referència.

Proveu de substituir l'expressió per un HTML normal i vegeu què passa quan torneu a carregar l'aplicació.

```
Proveu de substituir l'expressió   <App />  per un HTML normal i vegeu què
passa quan torneu a carregar l'aplicació.
```

Consulteu el component "App"

és el primer component que cridarà l'aplicació React i serveix de contenidor per representar la resta de components de la vostra aplicació.

Obriu-lo **src/app.js** a l'editor de text.

A la part superior veureu les **import** afirmacions conegudes de **src/index.js** .

Aquest component importa uns quants mòduls, defineix una nova classe **App** i després exporta aquesta classe com a mòdul.

Podeu trigar un minut a comprovar com funciona la línia 1. Utilitza la desestructuració per establir una constant **Component** igual a la propietat **Component** de l'objecte del mòdul React. Aquest és un sucre sintàctic que us permet fer referència directa **Component** en aquest fitxer en lloc de **React.Component** .

La línia 5 defineix la classe de components de l'aplicació. Hereta de la classe base **Component** i aquí només defineix una propietat, la funció **render(**) S'espera que tots els components defineixin una funció anomenada **render()** que retorna els elements que el component ha de representar, i podeu veure que això és el que fa aquest.

Actualment, aquest component té un text de marcador de posició i, principalment, crea elements estàtics que semblen HTML normals, a excepció de l'atribut **src** de la imatge del logotip. Aquest és l'únic exemple aquí d'una expressió de plantilla real, així que anem a veure com funciona.

Dins de les etiquetes de plantilla JSX, els claudàtors funcionen de manera diferent que en altres llocs de JavaScript. Aquí indiquen que el codi dins dels claudàtors és JavaScript i que el valor avaluat de l'expressió JavaScript que s'hi imprimeix s'hauria d'imprimir en lloc del codi entre claudàtors del marcatge generat.

Proveu d'afegir les vostres pròpies expressions de plantilla dins de les etiquetes de la funció de renderització per esbrinar com representar valors dinàmics en un component. Podeu posar qualsevol expressió JavaScript (però no sentències) dins d'un parell de claudàtors.

Comenceu a publicar i desenvolupar la vostra aplicació React

Aneu a una línia d'ordres i executeu des del directori de l'aplicació:

```
    yarn start
```

El navegador s'hauria d'obrir i hauríeu de veure l'aplicació executant-se a la pàgina. Proveu de fer modificacions a **App.js** , desant-les i vegeu com apareixen.

```
Tingueu en compte que no cal actualitzar la pàgina web per veure els
canvis. Això es deu al fet que, en mode de desenvolupament, l'aplicació es
torna a representar quan deseu canvis al codi.
```

Altres scripts

A més **yarn start** , **create-react-app** us proporciona alguns scripts més. Estan disponibles a la línia d'ordres mitjançant **yarn** perquè s'especifiquen a la scripts secció del **package.json** fitxer. Podeu llegir-ne al README que es va generar amb la vostra aplicació.

yarn test

create-react-app també us proporciona un corredor de proves complet, a través de Jest .

El corredor de proves observa els canvis a la vostra aplicació i intenta executar les proves relacionades.

L'aplicació generada ja us proporciona un exemple de "prova de fum" a **App.test.js** . Si hi heu fet canvis **App.js** , el corredor repetirà aquesta prova.

El nom d'aquest fitxer implica que conté proves del component **App** . Una "prova de fum" sol representar el component en qüestió, per veure que no genera cap error. Hi ha moltes maneres de localitzar i anomenar proves i moltes estratègies de prova. Us recomano que en llegiu més informació als documents create-react-app .

yarn run build

Estem treballant a la nostra aplicació React en mode de desenvolupament. Tot i així, no hauríeu d'intentar executar l'aplicació tal com està en un servidor públic, perquè els fitxers no estan empaquetats per obtenir un rendiment eficient. Definitivament, tampoc no voleu publicar la vostra aplicació des del servidor de desenvolupament que fem servir quan executem **yarn start** .

En lloc d'això, **yarn run build** empaqueta la vostra aplicació per a un desplegament de producció. En aquest curs no entrarem en els detalls sobre l'optimització i el desplegament de la vostra aplicació per a la producció, però executem ara l'escriptura de compilació per veure què fa.

Podeu veure que us indica les mides de fitxers dels vostres recursos creats després de comprimir-los. La vostra aplicació actual es troba ara a **./build/static/js/** . Mireu el fitxer "principal" que hi ha per veure l'aspecte del vostre codi combinat i reduït. Podeu veure que aquest fitxer té aproximadament 153 KB. Això es deu al fet que estem veient el fitxer minificat abans de comprimir-lo.

Resum

Per saber si tenim instal·lat node

```
        node -v
```

Per saber si tenim instal·lat npm

```
npm -v
```

web de npm

```
https://www.npmjs.com/get-npm
```

Instal·lar nodejs windows

Saber les comandes disponibles de node

```
node --help
node -h
```

Web de Node

```
https://nodejs.org/es/download/
```

Instal·lar yarn a windows

```
 npx install --global yarn
```

o

```
 npm install --global yarn
```

Saber la versio de yarn

```
yarn --version
```

Instal·lar create-react-app

```
yarn global add create-react-app
```

si hi problemes provar

```
npm install -g create-react-app
```

Veura que fa

```
create-react-app --help
```

Crear una app

```
create-react-app my-app-name
```

si hi ha problemes

```
npx create-react-app hello-world
```

Arrencar servidor prova

```
yarn start
```

boostrap a react

```
npm install --save bootstrap
```

or

```
yarn add bootstrap
```

Jquery a bootstrap

```
npm install --save jquery
```

Instal·lar popper

```
npm install --save popper.js
```

escriure els imports a src/index.js