React JS

# Cuprins

[**Cuprins**](#_syc6mnmq3ba7) **2**

[**1. Introducere**](#_h6vqsy16oa9b) **3**

[**2. React și restul lumii**](#_buh943f4fna6) **3**

[2.1. Inainte de orice framework a fost Javascript Vanilla (Virtual DOM)](#_s9vgeduvhasq) 3

[2.2 . Angular si Vue (Asemănări și performanță)](#_9aoo6fs9kfuy) 3

[2.3. Prin ce se evidențiază React (Sintaxă, SEO, native)](#_lhyy3sfyrace) 4

[**3. Hello world**](#_tp4qz9ssyr1n) **4**

[3.1. Instalare React](#_t3ez2m64wq1w) 4

[3.2. Structura de fisiere](#_bt18be6xmcen) 5

[3.3 Demo - filtru acordeon peste o lista](#_byzk8luk8oz) 6

[3.3.1. Librării și tehnologii](#_qe1igo8vh8cv) 6

[3.3.2 Index.js](#_b7nffprvu3uz) 7

[3.3.3. Rădăcina aplicației](#_ew2rgescx549) 7

[3.3.4. Context și acțiuni](#_xrpu0cnaydk7) 8

[3.3.5. Conectarea contextului și utilizarea React Hooks](#_ksa5aog7cb7l) 10

[**4. Concluzii**](#_iuteb7miskya) **10**

[**5. Bibliografie**](#_rd6kb934862p) **12**

# 1. Introducere

React este o librarie javascript care permite realizarea interfețelor de utilizator. A fost expusă pentru prima dată în 2011 în cadrul Facebook, iar creatorul acesteia, Jordan Walke (software engineer) s-a inspirat din XHP, un framework pentru PHP. În perioada următoare React s-a bucurat de o deosebită popularitate mai ales după lansarea React Native (2015) – aplicații native de Android/iOS, dar și după lansarea oficială a React 16.0 care a venit cu numeroase îmbunătățiri de performanță și stabilitate (React Fiber). Ultimul trend este reprezentat de React Hooks (februarie 2019) care deschide calea programării funcționale în React.

React este o librărie, iar spre deosebire de un framework, acesta nu este o soluție completă – de cele mai multe ori este nevoie de mai multe librării pentru a construi o aplicație. Acest lucru oferă librăriei flexibilitate și nu constrânge programatorul de a scrie codul într-o singură manieră.

React este declarativ – acest lucru înseamnă că se descrie interfața, iar atenția cade asupra a ce se dorește și nu asupra modului în care se realizează lucrurile. Astfel, React optimizează funcționalitățile aplicației prin așa numitul virtual DOM.

# 2. React și restul lumii

## 2.1. Inainte de orice framework a fost Javascript Vanilla (Virtual DOM)

Înainte de React (Javascript Vanilla, jQuery), cand era nevoie să se interacționeze cu API-ul DOM (Document Object Model), tendința era de a traversa arbore DOM și de a actualiza nodurile necesare. Acest lucru era costisitor mai ales pentru că procesarea se realiza într-un singur fir de execuție, același fir în care se tratează și interacțiunea utilizatorului cu browser-ul web, acționarea butoanelor sau a câmpurilor text.

Aceasta problemă a fost rezolvată printr-un concept revoluționar: **Virtual DOM**. De fiecare data cand un nod din arborele DOM trebuie actualizat, React creaza un arbore virtual pe care îl reține în memorie. Cand un alt nod trebuie actualizat, se compară arborele curent cu cel din memorie și se desenează doar acel subarbore care a fost modificat.

## 2.2 . Angular si Vue (Asemănări și performanță)

Popularitatea și performanța React poate fi comparată cu Angular și Vue, două framework-uri JS care au fost adoptate pe scară largă. Toate cele trei tehnologii sunt bazate pe componente, practic, orice element construit poate fi folosit ulterior. De asemenea, o altă asemănare ar fi că toate excelează în ceea ce privește manipularea DOM - de exemplu în **Fig 1 - stanga** se poate observa ca Angular este ușor mai rapid decat celelalte în majoritatea situațiilor. Cat despre alocarea de memorie, se poate observa ca React și Vue sunt mult mai eficiente (**Fig 1 - centru**), iar in cazul timpilor de pronire Vue este mult mai rapid (**Fig 1 - dreapta**).

|  |
| --- |
|  |

Fig. 1: Durata în ms pentru diferite manipulari DOM (stanga), metrici despre alocarea de memorie (centru) și metrici despre startup (dreapta) - sursa <https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/>

## 2.3. Prin ce se evidențiază React (Sintaxă, SEO, native)

Dacă vorbim despre popularitate, marketing și performanță e greu de sesizat prin ce React se face remarcat. Un prim lucru de remarcat la React este că se poate învăța relativ ușor față de celelalte tehnologii de pe piață. Avand o sintaxă foarte apropiata de Javascript Vanilla, React necesita învățarea doar a unui set restrâns de metode. De asemenea, React reprezintă o bună alegere mai ales cand se dorește SEO (Search Engine Optimisation) deoarece către browser se trimit fișiere staticie care sunt ușor de indexat și procesat de motoarele de căutare. Cu toate acestea, un plus uriaș pe care îl aduce React este posibilitatea de a realiza aplicații mobil native prin intermediul React Native - „*Learn once, write anywhere.*”.

# 3. Hello world

## 3.1. Instalare React

Configurarea mediului de lucru în React se poate obține relativ simplu și sunt necesare doar cateva componente pentru acesta, astfel:

* **Node.js** este un mediu de executare al aplicațiilor Javascript în afara browserului
* **NPM** sau orice alt manager de pachete Javascript disponibil
* **Create React App** - preconfigurarea mediului React și instalarea acestuia (exista și o alternativa la aceasta metoda, dar necesita mai mult timp)

După instalarea tuturor componentelor, se pot rula comenzile următoare, iar o aplicație simplă se va deschide în browser la adresa *localhost:8080.*

|  |
| --- |
| $ npx create-react-app my-app  $ cd my-app  $ npm start |

## 3.2. Structura de fisiere

|  |
| --- |
|  |

Fig. 2: Structura de fișiere a proiectului my-app - editorul VSCode. Aici se pot observa principalele structuri din componenta unui proiect React

După configurarea și instalarea primei aplicații React, cele mai importante structuri de fișier sunt:

* **Node\_modules** - directorul rădăcină al pachetelor npm care permit redarea mediului React, dar și a diferitelor librării
* **Public** - directorul în care se afla soluția statică a aplicației React (fisierele compilate)
* **Src** - directorul sursa cu toate componentele javascript, stilurile, imaginile și alte resurse statice
* **Package.json** reprezinta fisierul de configurare al pachetelor NPM. Tot aici se găsesc și diverse scripturi.

## 3.3 Demo - filtru acordeon peste o lista

|  |
| --- |
|  |

Fig. 3: Interfața aplicației demo - o lista de Cloud Providers care poate fi filtrata după diversi parametrii (partea dreapta) și componenta de filtrare de tip acordeon (partea stanga)

### 3.3.1. Librării și tehnologii

* **React Hooks** (useState, useContext, useEffect) - sunt necesare pentru fluxul datelor și setărilor în componente
* **React Context** - arbore global cu date necesare componentelor. Aici se salvează și răspunsurile de la API
* **Axios** - librărie care permite efectuarea de cereri HTTPS
* **Material UI** - librarie cu componente functionabile
* **Lodash** - o alternativa la modul de lucru cu obiecte, stringuri și array-uri
* **Styled Components** - alternativa la CSS3 (stil inclus prin Javascript)
* **My-JSON** - api dummy la care se pot face cereri HTTPS

### 3.3.2 Index.js

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import App from './App';  import \* as serviceWorker from './serviceWorker';  ReactDOM.render(<App />, document.getElementById('root'));  serviceWorker.unregister(); |

Fig. 4: Fișierul index care inițializează aplicația React

În acest fișier se importa rădăcina aplicației (**App**) și cu ajutorul **ReactDom** se randeaza în elementul #root. ReactDOM are drept scop procesare și desenarea DOM-ului virtual în browser.

### 3.3.3. Rădăcina aplicației

|  |
| --- |
| import React from 'react'  ...  function App() {  return (  <HistoryContextProvider>  <FiltersContextProvider>  <HistoryList />  </FiltersContextProvider>  </HistoryContextProvider>  )  }  export default App |

Fig. 5: Componenta App.jsx - practic radacina aplicatiei

Orice componenta React este scrisă într-un fișier și împărțită în 3 secțiuni. Prima secțiune cuprinde partea de importuri în care se aduc librăriile și componentele necesare în acea componenta, de exemplu aici se importa React și cateva componente descrise mai târziu. Functia **App** reprezinta componenta în sine, iar aici se declara variabile, metode (după modelul claselor) și se returnează un arbore JSX. La sfarsitul componentei se afla zona de export unde se declara ce anume sa poate accesa din exterior.

**HistoryList** reprezinta componenta functionala a aplicatiei, practic lista și filtrarul de tip acordeon din Fig. 3. **HistoryContextProvider** și **FiltersContextProvider** reprezinta cei doi arbori care cuprind structuri de date necesare în componenta de filtrare și în cea de listare. Se vor prezenta acestea în următoarele secțiuni.

### 3.3.4. Context și acțiuni

|  |
| --- |
| import { generateContext } from '../../helpers'  ...  const initialState = {...}  const reducer = (*state*, *action*) => {  switch (action.type) {  ...  case GET\_HISTORY\_SUCCESS:  return {  ...state,  history: action.payload,  loading: false  }  ...  ...  }  const { Context, ContextProvider } = generateContext(initialState, reducer)  export const HistoryContext = Context  export const HistoryContextProvider = ContextProvider |

Fig. 6: Componenta HistoryContext

Orice context necesita o actualizare și o funcție numită reducer care permite modificarea arborelui cu date în funcție de necesități. Astfel, aici se găsește un **switch case** prin intermediul căruia se poate spune ce anume vrem sa actualizam în arborele contextului. De exemplu, în cazu **GET\_HISTORY\_SUCCESS** se salveaza datele primite în variabila **history** a contextului.

|  |
| --- |
| ...  export const getHistory = async ({ *historyContext*, *filters* }) => {  const query = encodeFilters(filters)  ...  try {  response = await Axios({  url: `${API\_URL}/history?${query}`,  method: 'get'  })  historyContext.dispatch({  type: GET\_HISTORY\_SUCCESS,  payload: response.data  })  } catch (err) {  ...  }  }  export const deleteHistory = async ({ *historyContext*, *id* }) => {...} |

Fig. 6: În acest fișier sunt descrise metodele prin care se manipulează History Context

În Fig. 6 se realizeaza câteva acțiuni care vor fi necesare sub diferite circumstanțe. De exemplu, **get History** realizeaza un apel la API-ul pentru a obține lista istoricului filtrata după anumiți parametrii, iar dacă apelul a avut succes se returnează datele obținute care vor fi ulterior adaugate în variabila **history** a contextului

### 3.3.5. Conectarea contextului și utilizarea React Hooks

|  |
| --- |
| ...  export default function Filters({ *data*, *fetchFunction* }) {  const [expanded, setExpanded] = useState(initialState)  const filtersContext = useContext(FiltersContext)  useEffect(() => {  getFilters({ filtersContext, data })  }, [data])  const handleExpansion = *panel* => (*event*, *isExpanded*) => {  setExpanded(  isExpanded ? { ...expanded, [panel]: true } : { ...expanded, [panel]: false }  )  }  const {  loading,  filters: { providerName, cloudAccount }  } = filtersContext  return (...)  } |

Fig. 7: În acest fișier sunt descrise metodele prin care se manipulează History Context

Pentru a putea utiliza contextul într-o componentă este necesar sa apela hook-ul useContext. După acest lucru, datele vor fi accesibile din interiorul componentei. Tot aici putem utiliza useState pentru a initializa variabile locale pe care le putem muta ulterior. De asemenea, în componența se pot definit și diverse metode care vor fi apelate în urma evenimentelor de la nivel de interfață. De exemplu aici s-a utilizat ateState pentru a crea o stare care reține ce filtru este expandat, s-a utilizat useContext pentru a obține filtrele disponibile din api

# 4. Concluzii

Drept dospozitii finale se poate spune ca React reprezinta o unealta care poate fi valorificată în anumite circumstanțe și de anumiți actori. Se poate observa că abordarea pe care o propune nu este universal valabilă și ca sunt competitori care vin cu propuneri mai convenabile în aceleași scenarii. Tot aici trebuie menționat ca React este o abordare eleganta care se bucura de popularitatea Javascript și de o comunitate care privește spre inovare, stabilitate și performanța. React este ușor de învățat, dar necesita cunostinte bine fixate de javascript. Spre deosebire de competitorii sai, React nu impune o singura abordare cu toate ca în acest articol s-a propus modelul cu context extern.

Cat despre partea practica a articolului, se poate spune ca hook-urile și contextul sunt doua caracteristici care au propulsat React, sunt ușor de utilizat și datele raman bine structurate cu o singura sursa de adevăr, fără a polua inutil componentele. Codul de acest demo cu puține adaptări poate deveni cu usurinta o aplicație de mobile, asta face din React o soluție ușor de aplicat și în mediul mobile.

# 5. Bibliografie

1. The Complete Introduction to React - <https://jscomplete.com/learn/complete-intro-react>
2. ReactJS Documentation - <https://reactjs.org/docs/getting-started.html>
3. Learn React JS in 5 minutes — A tutorial for beginners - <https://medium.com/free-code-camp/learn-react-js-in-5-minutes-526472d292f4>
4. What is React - <https://www.w3schools.com/whatis/whatis_react.asp>
5. A declarative, efficient, and flexible JavaScript library for building user interfaces - <https://github.com/facebook/react>
6. Getting Started With React Hooks (2019) - <https://www.valentinog.com/blog/hooks/>
7. Replacing Redux with the new React context API - <https://medium.com/free-code-camp/replacing-redux-with-the-new-react-context-api-8f5d01a00e8c>
8. A Guide For Building A React Redux CRUD App - <https://medium.com/@rajaraodv/a-guide-for-building-a-react-redux-crud-app-7fe0b8943d0f?source=search_post---------1>
9. Create React App - <https://create-react-app.dev/docs/getting-started>
10. React (web framework) - <https://en.m.wikipedia.org/wiki/React_(web_framework)>
11. Angular vs. React vs. Vue: A performance comparison - <https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/>
12. Top 6 Reasons Why We Love React - <https://medium.com/serverless-with-syncano/top-6-reasons-why-we-love-react-d70d3a289e4b>