Projecte Redux JS

M06 UF2

Curs 2021-22

Raúl Heredia i Ana Menéndez



Índex

[Què és Redux? 3](#_Toc94977165)

[Avantatges del store de Redux 4](#_Toc94977166)

[Com funciona Redux? 4](#_Toc94977167)

[Accions en Redux 4](#_Toc94977168)

[Reducers en Redux 5](#_Toc94977169)

[Store en Redux 5](#_Toc94977170)

[Exemple bàsic de funcionament. 6](#_Toc94977171)

[El nostre projecte 7](#_Toc94977172)

[APIS Utilitzades 7](#_Toc94977173)

[Web Storage 7](#_Toc94977174)

[IndexedDB 8](#_Toc94977175)

[Drag And Drop 8](#_Toc94977176)

[FileReader 8](#_Toc94977177)

[Lògica de la aplicació 9](#_Toc94977178)

[Emmagatzematge de dades 10](#_Toc94977179)

[POO amb JS 11](#_Toc94977180)

[Funcions amb JS 12](#_Toc94977181)

[Arrays amb JS 14](#_Toc94977182)

[Métodes d’array utilitzats 14](#_Toc94977183)

# Què és Redux?

Redux és una llibreria JavaScript de codi obert per al maneig de l’estat de les aplicacions. És comunament utilitzada amb altres llibreries com React o Angular per a la construcció d’interfícies d’usuari.

Al 2015 per Dan Abramov i Andrew Clark es van inspirar en una altra llibreria de Facebook, Flux per crear Redux. Abramov es trobava impressionat per la similitud del patró Flux amb la funció de reduir.

*"Estava pensant sobre Flux com una operació reduir... el teu magatzem, com acumulen un estat en resposta a unes accions. Vaig pensar a anar més enllà. Què si el teu magatzem Flux no fos un magatzem sinó una funció reduir?"*

Abramov va contactar amb Clark per col·laborar junts. Gràcies a Clark hi ha les eines que fan possible l'ecosistema Redux, va ajudar a definir un API coherent i implementar la possibilitat d’extensió mitjançant middleware i store enhancers.

Redux és una petita llibreria amb una API simple i limitada que está dissenyada per ser un contenidor predictible de l’estat de l’aplicació.

El propòsit de Redux és fer predictibles els canvis d'estat, imposant certes restriccions sobre com i quan es poden produir les actualitzacions. Redux aconsegueix que la teva gestió d'estat sigui transparent i determinista, cosa que entre altres coses aporta:

* Millor comprensió de l’evolució de l’estat en un moment donat.
* Facilitat per incorporar noves característiques a l’app.
* Nou ventall d’eines de debugging.
* Capacitat de reproduir un bug.
* Millores en el procés de desenvolupament i poden reiniciar l’execució a partir d’un estat concret.

## Avantatges del store de Redux

* Model de dades més consistent i segur
* Gestió senzilla de múltiples fonts de dades:
* Dades accessibles en temps real:

# Com funciona Redux?

La manera com funciona Redux és senzilla. Hi ha una “Store” central que conté tot l'estat de l'aplicació. Cada component pot accedir a l'estat emmagatzemat sense haver d'enviar accessoris d'un component a un altre, això resulta molt util en aplicacions de Angular o React que tenen diferents components.

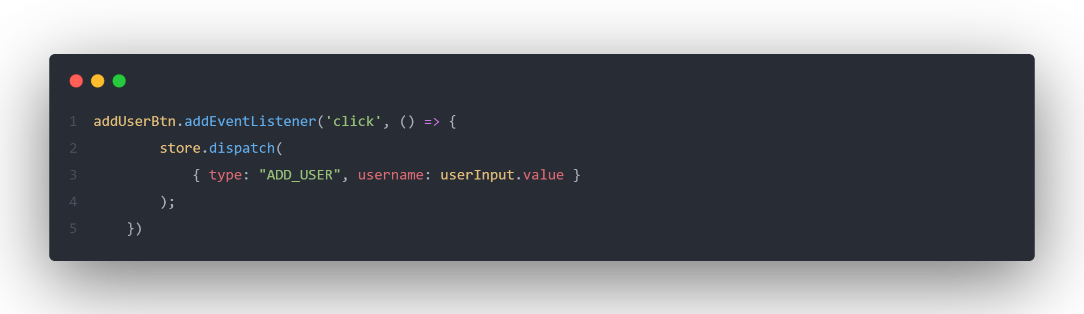
Principalment, Redux es divideix en tres parts: **accions**, **Store** i **reductors**. Ara, veurem que fan cadascuna.

## Accions en Redux

En poques paraules, les accions són esdeveniments. Són l'única manera d'enviar dades de la teva sol·licitud a la nostra Store de Redux. Les dades poden provenir d'interaccions d'usuari, trucades d'API o fins i tot d'enviaments de formularis.

Les accions s'envien mitjançant el mètode store.dispatch(). Les accions són objectes JavaScript simples, i han de tenir una propietat tipus per indicar el tipus d'acció a dur a terme. També han de tenir una càrrega útil que contingui la informació que ha de ser treballada per l'acció. Les accions es creen a través d'un creador d'accions.

**Per exemple:**



En aquest cas, quan cliquem el botó afegir usuari, enviem una acció de tipus “ADD\_USER” amb la propietat “username” i el valor agafat d’un input.

## Reducers en Redux

Els reductors són funcions pures que prenen l'estat actual d'una aplicació, realitzen una acció i retornen un nou estat. Aquests estats s'emmagatzemen com a objectes i especifiquen com canvia l'estat d'una aplicació en resposta a una acció enviada a la store.

Es basa en la funció de reducció en JavaScript, on un sol valor es calcula a partir de múltiples valors després de realitzar una funció de resposta.

**Per exemple:**



En aquest cas, veiem que el reducer està composat per l’estat i la acció, cada vegada que fem un dispatch, la acció arriba al reducer. Com podem veure, si la petició es de tipus “ADD\_USER”, retornarà l’estat més el nom d’usuari que li acabem de passar, en cas de que la acció no sigui de tipus “ADD\_USER”, simplement retornarà l’estat sense fer canvis

Store en Redux

La Store, té l'estat de l'aplicació. És molt recomanable mantenir només una Store en qualsevol aplicació Redux. Amb la Store, podem accedir a l'estat emmagatzemat, actualitzar l'estat i registrar o deixar de registrar listeners.

Els mètodes més utilitzats son Store.subscribe() i Store.getState(). Subscribe es un mètode que es truca cada vegada que canvia l’estat de la nostra aplicació, getState serveix per obtenir l’estat de la nostra aplicació.

**Per exemple:**



Com podem veure, fem un subscribe amb una funció anònima, això es el que farà cada vegada que s’actualitzi l’estat. En aquest cas el que fem es eliminar el contingut d’una llista HTML i el value del input d’usuari, seguidament fem un getState i iterem mitjançant un forEach, per cada usuari que hi hagi a l’estat crearem un element ‘li’, el qual tindra una classe de bootstrap i el text que contindrà serà l’usuari, finalment fem un appendChild del element ‘li’ per afegir-lo a la llista.

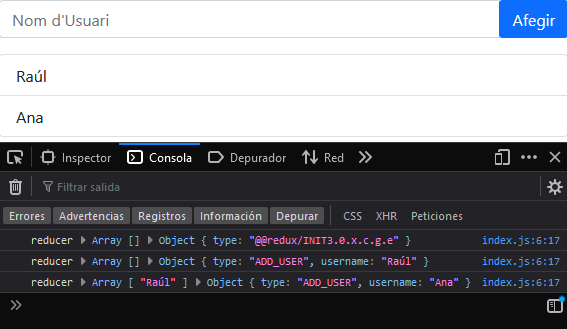
## Exemple bàsic de funcionament.

Texto

Descripción generada automáticamente

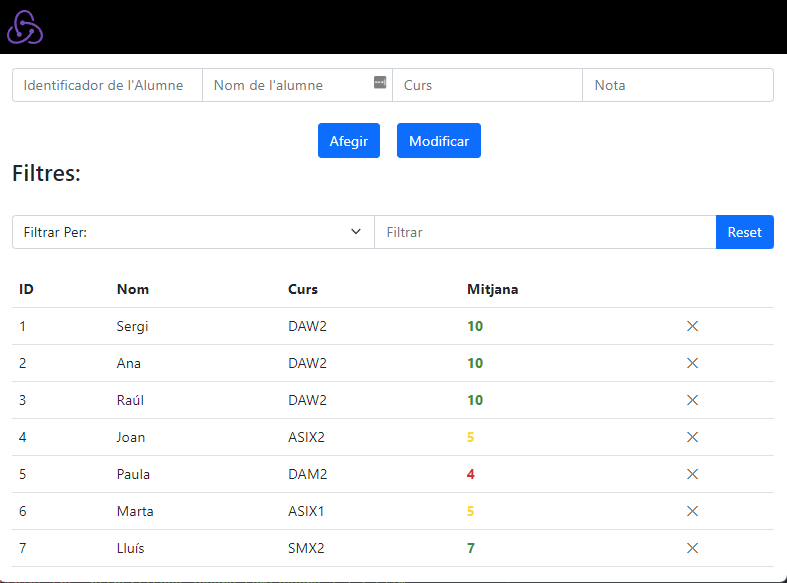
En aquest cas, podem veure un codi sencer, amb el reducer, la Store i la acció que em vist anteriorment.

Cada vegada que introduim un usuari i fem clic al botó afegir, s’envia una acció al reducer mitjançant un dispatch i canvia l’estat, llavors amb el subscribe cada vegada que aquest canvia, actualitza la llista:



# El nostre projecte

El projecte es una aplicació web que permet afegir alumnes, amb un identificador, nom curs i nota. En cas de voler modificar-ne algun, simplement posem el identificador, la dada que volem modificar i fem clic al botó de modificar. Podem filtrar els alumnes per aprovats i suspesos, a més de poder buscar-los per nom o curs, escrivint al input anomenat filtrar.



## APIS Utilitzades

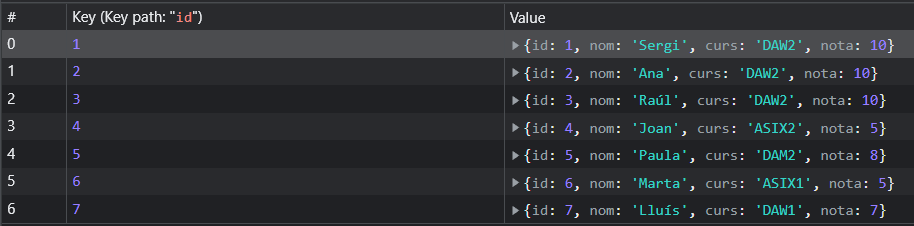
### [Web Storage](https://developer.mozilla.org/es/docs/Web/API/Web_Storage_API)

Hem utilitzat Web Storage per emmagazemar únicament el ID de l’alumne i el seu nom. Quan carrega la pàgina obté les dades de IndexedDB, cada vegada que afegim un usuari, es guarda el ID i el nom, quan eliminem un usuari, s’elimina també, es guarda mitjançant Local storage i no Session Storage. A més la utilitzem per a mostrar les dades per consola mitjançant una funció dinàmica que es veurà més endavant.



### [IndexedDB](https://developer.mozilla.org/es/docs/Web/API/IndexedDB_API)

Hem utilitzat IndexedDB per a guardar els objectes dels alumnes. La key que utilitza es el id d’usuari. Al entrar a la aplicació, IndexedDB envia cada usuari al reducer de la api Redux i d’aquesta forma els carreguem en el estat, que es mostrat en la taula d’alumnes



### [Drag And Drop](https://developer.mozilla.org/es/docs/Web/API/HTML_Drag_and_Drop_API/File_drag_and_drop)

Hem utilitzat la API de drag and drop per a poder arrossegar arxius JSON, amb els quals podem afegir alumnes de forma massiva.

### [FileReader](https://developer.mozilla.org/es/docs/Web/API/FileReader)

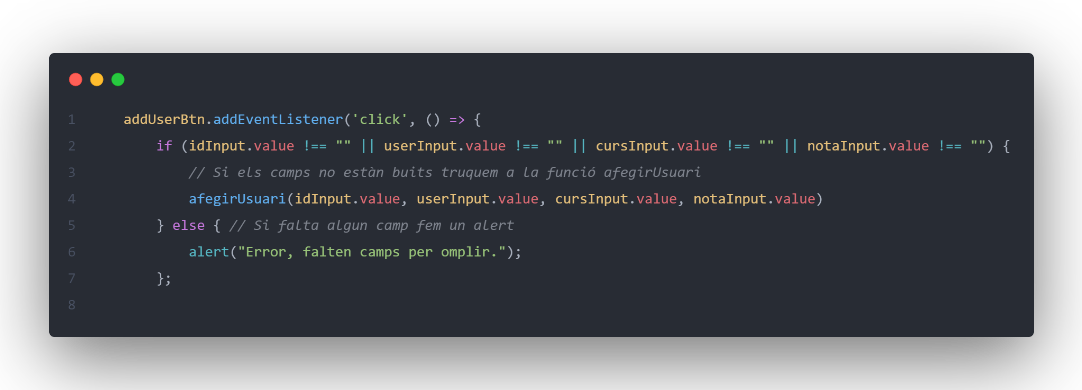
Hem utilitzat la API FileReader per a processar les dades del fitxer.



Aquí podem veure la funció DropJSON, a la cual passem el element en el qual arrossegarem i deixarem l’arxiu i retorna un callback, a l’hora de trucar, fem una funció anònima en la qual iterem cada usuari de l’arxiu i cridem la funció afegirUsuari passant cada propietat de l’usuari.

## Lògica de la aplicació

La lògica en un projecte Redux es sencilla, enviem accions al reductor, el qual canvia l’estat i quan aquest canvia, la Store realitza algunes accions, a mode d’exemple veurem el procés de afegir un alumne:



En el moment de fer clic al botó d’afegir, comprovem que tots els camps han estat omplerts, en el cas de que faltés algun camp, apareixeria una alerta, en cas de que tots els camps hagin estat omplerts, cridem a la funció afegirUsuari pasant els paràmetres necessaris.



Ara, creem la variable objectPos i apliquem el mètode map() a l’estat, per a que ens retorni el índex de l’usuari que acabem d’introduir, en cas de que l’index no sigui -1, alertem a l’usuari de que l’alumne ja existeix i netejem els camps, si l’índex es -1, vol dir que l’alumne no existeix i guardarem l’usuari a IndexedDB i WebStorage, finalment farem un dispatch amb la acció “ADD\_USER” per a que passi al reducer de Redux, el qual té una condició if per a les accions de tipus ADD\_USER, la qual es igual que la que hem vist a l’exemple d’ús, però en lloc d’afegir un nom a l’array, afegeix un objecte. Finalment la Store detecta el canvi d’estat i genera la taula un altre cop.

## Emmagatzematge de dades

S’ha utilitzat IndexedDB per a guardar els alumnes, les dades de IndexedDB son llegides i enviades al reducer de Redux a l’iniciar l’aplicació, amb una ordre “ADD\_USER”, concretament quan s’estableix la connexió amb indexedDB.



S’ha fet així, per a assegurar-nos de que iniciava indexedDB abans de que s’executés l’ordre, ja que si no donava un error ja que deia que la transacció tenia un valor undefined. Com podem veure, quan inicia guarda la Key i el nom al webstorage també.

Quan afegim un alumne, aquest es afegit, quan el modifiquem es modificat i quan l’eliminem, es eliminat de indexedDB, tot això es controla segons les accions enviades al Reducer. En el cas de webstorage, quan afegim un usuari s’afegeix, en cas de modificar un alumne només es modifica si modifiquem el nom i en cas d’eliminar un alumne s’elimina.

Les dades de WebStorage son llegides via consola a l’arrancar la aplicació mitjançant una funció dinàmica:



## POO amb JS

Hem creat una classe emmagatzematge que conté tres mètodes estàtics. El mètode desar, el mètode esborrarAlumne i el mètode modificarAlumne.



També, hem creat una classe anomenada WebStorageEmmagatzematge que hereta de Emmagatzematge, aquesta compta amb un override dels mètodes desar i esborrarAlumne, ja que per a modificar simplement utilitzem el mètode desar i sobreescriu el nom.

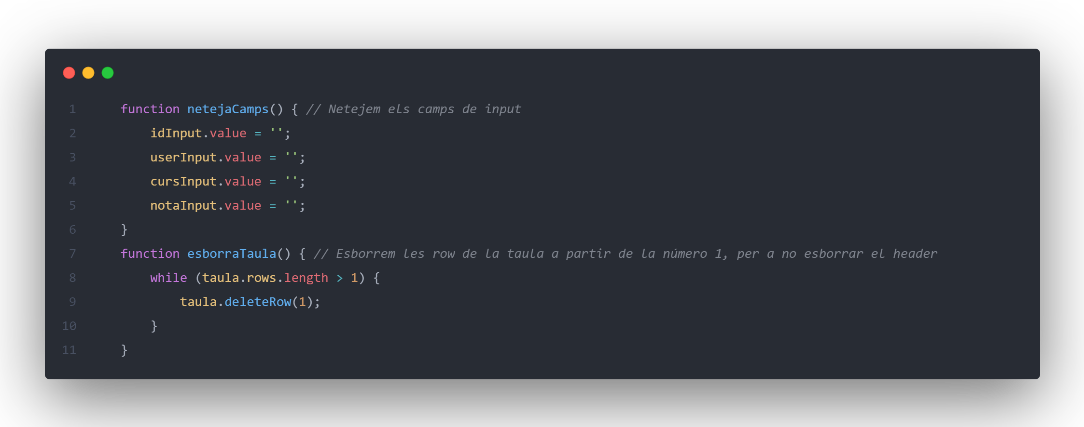
Texto

Descripción generada automáticamente

## Funcions amb JS

Hem utilitzat funcions, arrow functions i una funció dinàmica. Les funcions que necessitàvem executar a diferents parts del codi son funcions normals, la majoria de funcions anònimes s’han utilitzat arrow functions i la funció dinàmica s’ha utilitzat a l’hora de llegir les dades del Web Storage.

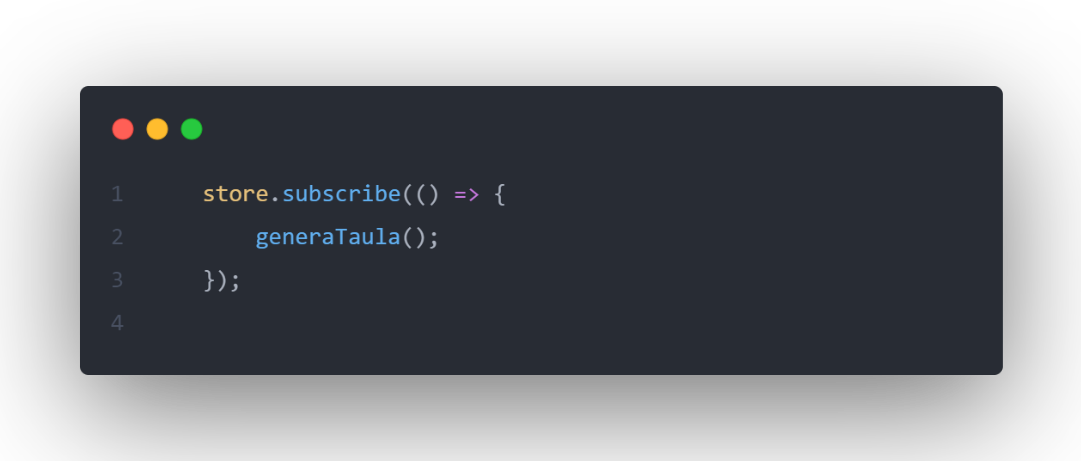
**Exemple funcions:**

****

A mode d’exemple, tenim aquestes dues petites funcions, la funció netejaCamps, principalment es executada després de afegir o modificar un alumne, ja que ens interessa que els camps de introduir dades tornin a estar buits.

La funció esborraTaula, s’encarrega d’esborrar la taula abans de generar una nova quan hi ha un canvi d’estat.

**Exemple Arrow Function:**

****

Un exemple podria ser el Store.subscribe, que te una arrow function, cada vegada que l’estat canvia, crida la funció generaTaula, tot i que podríem trucar directament a generaTaula, no ho hem fet ja que podríem fer moltes mes coses a part de generar la taula.

**Un altre Exemple d’Arrow Function:**

****

En aquest cas tenim un listener que l’unic que fa es que quan cliquem el botó de resetejar filtres, reseteja el value del input de filtrar i selecciona la opció 0 del select list, finalment envia una acció de tipus “RESET\_FILTER”.

**Exemple funció dinàmica:**

****

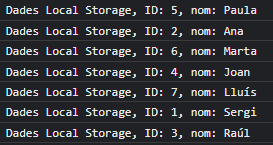
La funció en si, depenent de la variable valor, farà una cosa o una altre, en aquest cas el que fa es sumar retornar la suma de x, y i z, si el valor fos diferent a 1 faria la multiplicació, finalment creem la funció anomenada funcDinàmica amb un new Function, on definim els paràmetres i el que retorna. Tenint en compte que s’executa a l’inici quan ja s’han carregat tots els valors a web storage, fem un for i mostrem per consola “Dades Local Storage ID: ”, ara truquem a la funció dinàmica i passem:

Com a valor X: localStorage.key(key).

Com a valor Y: “, nom: ”.

Com a valor Z: localStorage.getItem(localStorage.key(key)

Finalment, obtenim per consola:



## Arrays amb JS

Nomes hem utilitzat un array d’objectes que es el array de l’estat del Reducer. A part d’algun array temporal utilitzat per a filtrar l’estat, etc.

### Métodes d’array utilitzats

#### map()

Hem utilitzat el mètode map(), per a comprovar si els usuaris existien o no abans d’afegir-los, modificar-los o eliminar-los. Exemple:



Com podem observar a la imatge, en la condició “DELETE\_USER” del reducer, tenim la variable objectPos, la qual utilitzem per a comprovar si existeix l’usuari abans d’eliminar-lo, en cas de que la posició sigui diferent a -1, l’esborrem amb un splice i l’esborrem tant de indexeddb com de webstorage.

#### filter()

El mètode filter, ha estat usat per a poder filtrar les dades de la taula, s’ha hagut de fer fora del reductor, ja que si es filtrava dintre, per molt que es fes creant una nova variable, el filter modificava l’estat original i no es podien recuperar els usuaris que s’havien filtrat (és a dir, si demanava els usuaris suspesos, no podia recuperar els aprovats després). Exemple:

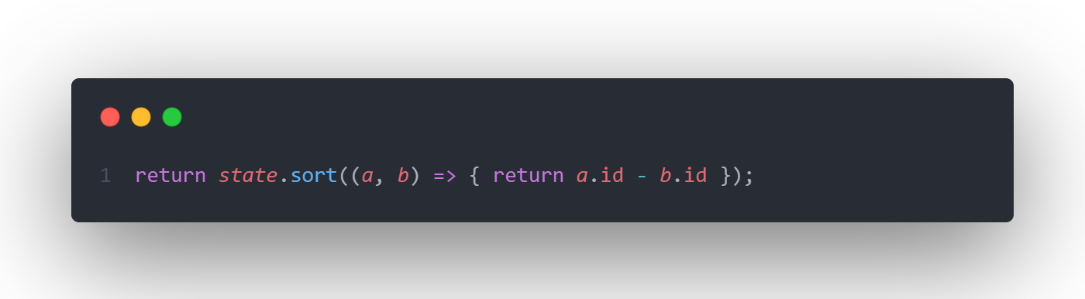


Per a filtrar amb el input text, tenim un listener keyup, cada vegada que introduïm una lletra, creem una variable anomenada filterText, aquesta obté el estat, i el filtrem on el nom comença per el valor que conté inputFiltrar o el curs comença pel valor que te inputFiltrar, un cop filtrat truquem a la funcion generaTaulaFiltre i li passem el array ja filtrat.

Em el cas de voler filtrar per la mitjana, com es un select list tenim un listener change, cada cop canviï el valor, si el valor es “aprovats” filtrarà per els alumnes aprovats, nota >= 5, en cas de ser “suspesos” filtrarà per els alumnes supesos, nota <5. Un cop hagi fet el filtre truca a la funció generaTaulaFiltre passant el array filtrat.

#### sort()

El mètode sort s’utilitza al llarg del reductor per a ordenar el estat per ID, tal i com podem veure a la ultima línia del reductor:



#### splice()

el mètode splice, s’utilitza per a esborrar els usuaris:



Com podem observar, si compleix amb la condició de que existeix (objectPos !== -1), fem un splice del estat passant-li el seu index, al posar un 1, només s’elimina l’usuari seleccionat.

#### forEach()

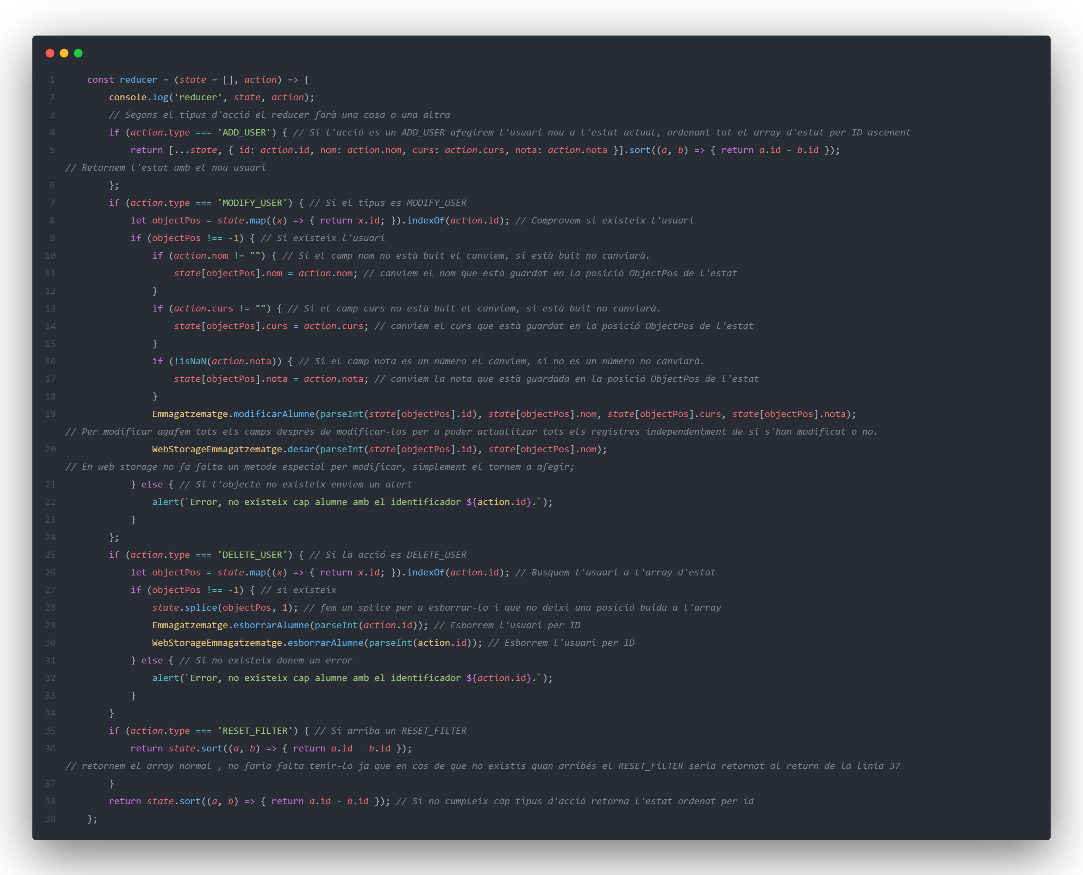
El mètode forEach s’ha utilitzat per a iterar tant l’estat com els arrays filtrats de l’estat:



Podem veure que fem un Store.getState().forEach, d’aquesta forma iterem cada alumne i els afegim a la taula.

#### reduce()

Com ja em esmentat prèviament, tot Redux es basa en la funció reduce de JS, a la cual li arriben accions i retorna un estat segons les condicions d’aquesta acció. El nostre Reducer de Redux es el següent:



Com podem veure segons les accions que arriben, realitza unes accions o unes altres.

#### for in

La estructura for in ha estat usada per iterar els usuaris que rebem del fitxer JSON quan aquest es arrossegat.

