<https://material.io/>

<https://material-ui.com/>

yarn add @material-ui/core

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" />

Plug in

Eslint PARA VALIDAR CODIGO INCORRECTO. NO PERMITE UN CHEQUEO ESTATICO DEL CODIGO.

**KEY DE LOS ARRAY PARA EL RENDERIZADO**

Cualquier componente que se genere dentro de un array necesita una key para el renderizado sea más eficiente. Se puede usar como key el índice del array. Pero si queremos agregar un componente o reorganizar uno no se va a resolver el problema del renderizado (va a ser lo mismo). Por eso es mejor que sea una clave NATURAL, algo que sabemos que no se repita (no depende de la ubicación del componente dentro de array como el índex

**COMUNICACIÓN ENTRE COMPONENTES**

Si queremos la comunicación entre componente (ósea que cuando aplicamos una acción sobre uno este se lo comunique a otro) podemos hacerlo de diferentes formas:

* Burbujeo: un componente a través de un evento le manda a este evento a tu componente padre y a partir de este componente padre se lo manda al correspondiente… o depende donde se ubique el componente va a ser el recorrido que va a realizar. Muy complejo, es engorroso.
* url: al seleccionar un componente se altera la URL, entonces hacemos que el componente sea reactivo frente al cambio de URL. No hay una comunicación directa ente componentes.
* Redux: podemos establecer un estado de la aplicación y así se mostrarían los valores. Después lo vamos a ver bien.

**PARA METER ESTILOS Y RESPONSIVE**

* React-bootstrap: https://react-bootstrap.github.io/
* React-Flexbox-Grid: <https://roylee0704.github.io/react-flexbox-grid/>

yarn add react-flexbox-grid

* Media Query

**CICLO DE VIDA DE RACT**

1. Se ejecuta el constructor de nuestro componente principal (app.js). Estableciendo el STATE con =…. “state =”
2. Render () {} con código JSX.
   1. Componentes nuestros
      1. Constructor que puede ser vacío o no (opcional)
      2. Render (si o si)
   2. Componentes h1, div, a
      1. Constructor que puede ser vacío o no (opcional)
      2. Render (si o si)
3. ---------------------------------SE CARGA LA PANTALLA------------------------------------
4. Puede surgir un evento que provoque una nueva renderizacion del componente. Una de estas es

This.setState => nuevo render

1. Provoca el renderizado de app.js y luego todos los demás.

*“También si se cambia una propiedad se fuerza el renderizado del componente”*

*shoulComponentUpdate() 🡪 función que permite una mejora de la eficiencia*

**FASES:**

* Montaje
* constructor
* getDeriverGetPromp
* render (genera los elementos mostrados en pantalla)
* componentDidMount (es invocado luego que el componente es insertado en el árbol del componente del DOM. Se usa para hacer peticiones.
* Actualización
* Render
* componentDidUpdate: invocado inmediatamente después que ocurre una actualización permite comparar valores de las propiedades antes o después de la actualización
* getDerivedStateFromProps
* ShouldComponentUpdate
* getSnapshotBeforeUpdate
* Desmontaje: retirar del DOM
* componentWillUnmount

[*http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/*](http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)

**LIBRERÍAS**

* Moments (para la transformación de fechas) 🡪 <https://momentjs.com/>
* Material-UI
* FlexboxGrid
* ReactWeatherIcons