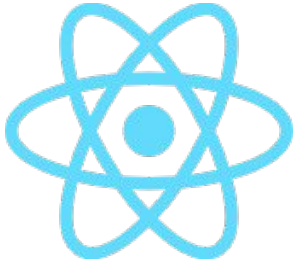


React Avanzado

@cesalberca

César Alberca

cesalberca@gmail.com

@cesalberca

@cesalberca

https://github.com/EscuelaIt/react-avanzado

Hooks

Versión 16.8, se basan en funciones. Cuentan con Hooks predeterminados como:

- useState
- useEffect
- useContext

https://reactjs.org/docs/hooks-intro.html

https://www.youtube.com/watch?v=dpw9EHDh2bM

Clases vs Hooks

```
import * as React from "react";
import { Card, Row, Input, Text } from "./components";
import ThemeContext from "./ThemeContext";

export default class Greeting extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "Harry",
      surname: "Potter",
      width: window.innerWidth
    };
    this.handleNameChange = this.handleNameChange.bind(this)
    this.handleResize = this.handleResize.bind(this)
    this.handleSurnameChange = this.handleSurnameChange.bind(this)
  }
  componentDidMount() {
    window.addEventListener("resize", this.handleResize);
    document.title = this.state.name + ' ' + this.state.surname
  }

  componentDidUpdate() {
    document.title = this.state.name + ' ' + this.state.surname
  }

  componentWillUnmount() {
    window.removeEventListener("resize", this.handleResize);
  }
  handleNameChange(name) {
    this.setState({ name });
  };

  handleSurnameChange(surname) {
    this.setState({ surname });
  };

  handleResize() {
    this.setState({ width: window.innerWidth });
  };

  render() {
    let { name, surname, width } = this.state;
    return (
      <ThemeContext.Consumer>
        {theme => (
          <Card theme={theme}>
            <Row label="Name">
              <Input value={name} onChange={this.handleNameChange} />
            </Row>
            <Row label="Surname">
              <Input value={surname} onChange={this.handleSurnameChange} />
            </Row>
            <Row label="Width">
              <Text>{width}</Text>
            </Row>
          </Card>
        )}
      </ThemeContext.Consumer>
    );
  }
}
```

```
import React, { useState, useContext, useEffect } from "react";
import { Card, Row, Input, Text } from "./components";
import ThemeContext from "./ThemeContext";

export default function Greeting(props) {
  let theme = useContext(ThemeContext);

  let [name, setName] = useState("Harry");
  let [surname, setSurname] = useState("Potter");
  useEffect(() => {
    document.title = name + " " + surname;
  });

  let [width, setWidth] = useState(window.innerWidth);
  useEffect(() => {
    let handleResize = () => setWidth(window.innerWidth);
    window.addEventListener("resize", handleResize);
    return () => {
      window.removeEventListener("resize", handleResize);
    };
  });

  return (
    <Card theme={theme}>
      <Row label="Name">
        <Input value={name} onChange={setName} />
      </Row>
      <Row label="Surname">
        <Input value={surname} onChange={setSurname} />
      </Row>
      <Row label="Width">
        <Text>{width}</Text>
      </Row>
    </Card>
  );
}
```

Reglas de los hooks

- No se pueden ejecutar dentro de condicionales o bucles
- Solamente se pueden invocar desde “funciones de React”

https://reactjs.org/docs/hooks-rules.html

useState

```
const [state, setState] = useState(initialState);
```

```
setState(newState);
```

https://reactjs.org/docs/hooks-reference.html#usestate

useEffect

```
useEffect(() => {
  const subscription = props.source.subscribe();
  return () => subscription.unsubscribe();
}, [a, b, c]);
```

https://reactjs.org/docs/hooks-reference.html#useeffect

useReducer

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

https://reactjs.org/docs/hooks-reference.html#usereducer

¿Cuándo usar useReducer?

- Cuando gestionamos muchos estados
- Cuando el estado que gestionas dependen entre sí
- Para lógica de gestión de estado compleja

https://kentcdodds.com/blog/should-i-usestate-or-usereducer

useEffect + setInterval

```
function App() {
  const [count, setCounter] = useState(0)
  useEffect(() => {
    const id = setInterval(() => {
      setCounter(count + 1)
    }, 1000)

    return () => clearInterval(id)
  }, [])
  return (
    <span>{count}</span>
  )
}
```

```
import React, { useState, useEffect, useRef } from 'react';

function useInterval(callback, delay) {
  const savedCallback = useRef();

  // Remember the latest callback.
  useEffect(() => {
    savedCallback.current = callback;
  }, [callback]);

  // Set up the interval.
  useEffect(() => {
    function tick() {
      savedCallback.current();
    }
    if (delay !== null) {
      let id = setInterval(tick, delay);
      return () => clearInterval(id);
    }
  }, [delay]);
}
```

https://overreacted.io/making-setinterval-declarative-with-react-hooks/

Contexto

El API de contexto nos permite definir un estado global al que cualquier componente a cualquier nivel puede observar. Esto evita el “prop drilling” y puede sustituir a Redux para la creación de estado global.

https://reactjs.org/docs/context.html

Crear el contexto

```
export const MyContext = createContext<number[]>([]);
```

Proveer el contexto

En un componente (normalmente en la App) proveemos el context y es ahí donde definimos ese estado global.

```
<MyContext.Provider value={/* some value */}>
    <SomeComponent />
</MyContext.Provider>
```

Consumir el contexto

Desde el JSX podremos consumir el contexto:

```
<MyContext.Consumer>
  {value => /* render something based on the context value */}
</MyContext.Consumer>
```

Consumir el contexto II

También podremos consumir el contexto programáticamente usando el hook useContext(MyContext).

Actualizar el contexto

Si queremos que algún hijo pueda alterar el valor del contexto podremos pasarle una función que actualice dicho contexto:

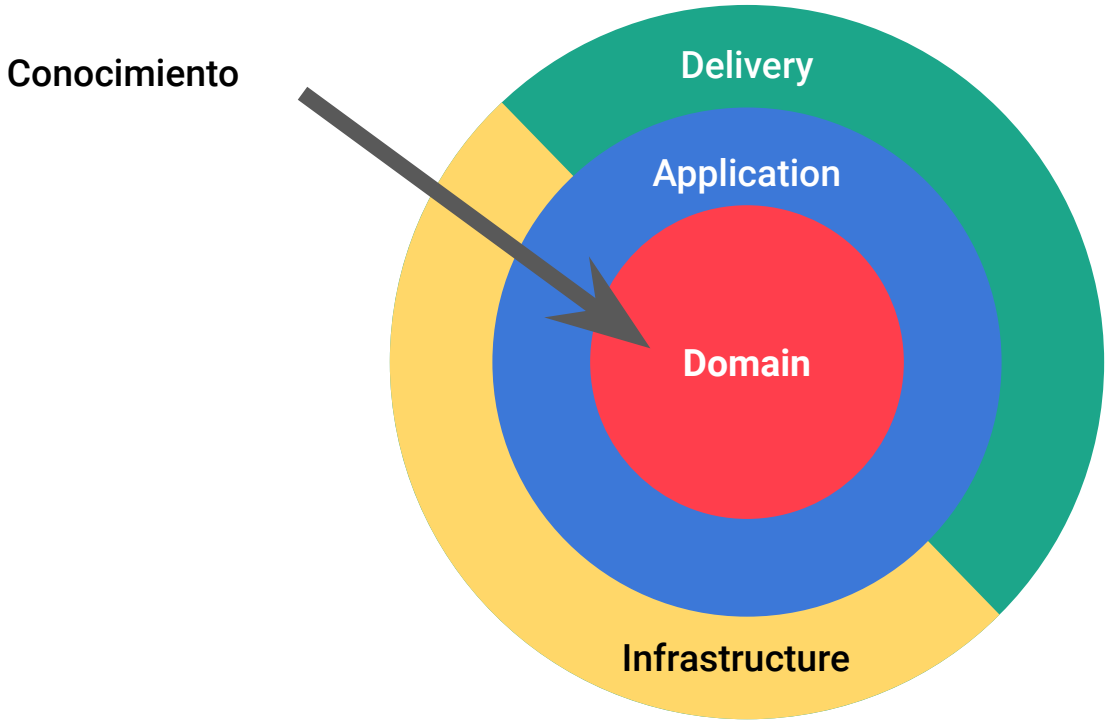
```
const [state, setState] = useState('foo')

<MyContext.Provider value={
        { value: state, setValue: (newValue) => setState(newValue) }
}>
        <SomeComponent />
</MyContext.Provider>
```

Arquitectura

4 reglas del diseño simple

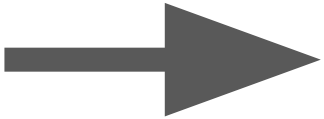
- Los tests pasan
- Revela intención
- No contiene duplicación
- Tiene el tamaño necesario

Separación por capas



Separación por directorios

- domain
- application
- ui
- infrastructure



- features
  - featureA
    - domain
    - application
    - ui
    - infrastructure
  - featureB
    - …
- core

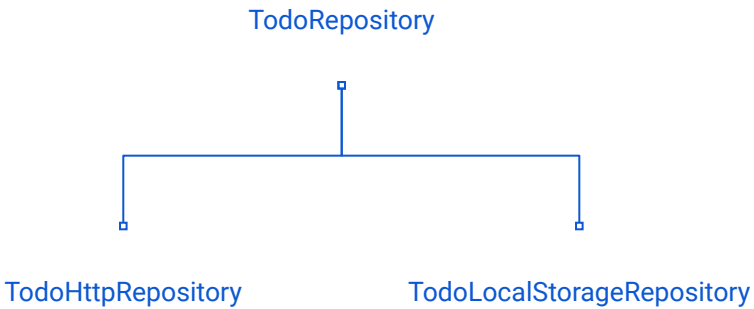
Casos de uso

Un caso de uso representa una acción que un usuario puede acometer sobre el sistema. Los casos de uso se dividen en:

- Queries
- Commands

Repositorios

Los repositorios **interactúan con datos**. Encuentran entidades de dominio, actualizan, borran, etc. Debemos definir una interfaz en el **dominio**, y en **infraestructura** implementamos esa interfaz:



Repositorios II

Los repositorios opcionalmente pueden trabajar con DTOs. Los DTOs son objetos que sirven para transferir información y no tienen comportamiento. Las respuestas de un backend podríamos definirlas como DTOs. Nuestros repositorios pueden usar Mappers para transformar los DTOs a nuestras entidades.

Mappers o Converters

Transforman un objeto a otro. Podemos usarlos en los repositorios para transformar lo que nos devuelve el back a entidades de dominio que nosotros usaremos en nuestro código. Los Mapper están en infraestructura generalmente.

Inyección de dependencias

Un motor de inyección de dependencias se encarga de crear y gestionar las instancias.

https://en.wikipedia.org/wiki/Dependency_injection#:~:text=In%20software%20engineering%2C%20dependency%20injection,it%20depends%20on%2C%20called%20dependencies.&text=The%20intent%20behind%20dependency%20injection,increase%20readability%20and%20code%20reuse.

TSyringe

Biblioteca de inyección de dependencias hecha por Microsoft.

https://github.com/microsoft/tsyringe

Variables de entorno

Las variables de entorno nos permiten almacenar valores que van a cambiar en la construcción del proyecto.

https://create-react-app.dev/docs/adding-custom-environment-variables/

Testing de componentes

Testing library son una serie de utilidades para hacer que los tests sean más robustos y hace que se apliquen buenas prácticas.

https://testing-library.com/

React form validation

https://react-hook-form.com/

i18n

https://react.i18next.com/

CI/CD

La integración continua permite automatizar procesos de despliegue y de la construcción de proyectos de manera que sea determinista.

https://github.com/features/actions

Referencias

https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367

https://overreacted.io/how-are-function-components-different-from-classes/

https://overreacted.io/why-do-hooks-rely-on-call-order/

https://martinfowler.com/bliki/AnemicDomainModel.html

https://refactoring.guru/refactoring/smells

https://www.thesaurus.com/browse/achievements

https://martinfowler.com/bliki/UbiquitousLanguage.html

Referencias II

https://www.amazon.es/Domain-Driven-Design-Quickly-Floyd-Marinescu/dp/1411609255

https://tc39.es/proposal-temporal/docs/

https://martinfowler.com/bliki/GivenWhenThen.html#:~:text=Given%2DWhen%2DThen%20is%20a,%2DDriven%20Development%20(BDD).&text=You%20can%20think%20of%20it,pre%2Dconditions%20to%20the%20test.

https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests/

https://martinfowler.com/bliki/ObjectMother.html

https://www.npmjs.com/package/ts-mockito

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

https://en.wikipedia.org/wiki/Multiple_inheritance#:~:text=The%20%22diamond%20problem%22%20(sometimes,from%20both%20B%20and%20C.&text=It%20is%20called%20the%20%22diamond,inheritance%20diagram%20in%20this%20situation.

Referencias III

https://github.com/archimedes-projects/archimedes-js/blob/main/packages/utils/src/http-client/http-client.ts

https://kentcdodds.com/blog/replace-axios-with-a-simple-custom-fetch-wrapper

https://flukeout.github.io/

https://martinfowler.com/articles/2021-test-shapes.html

https://kentcdodds.com/blog/the-testing-trophy-and-testing-classifications/

https://www.youtube.com/watch?v=WSes_PexXcA

Referencias IV

https://docs.microsoft.com/en-us/windows/wsl/install

https://github.com/nvm-sh/nvm

https://storybook.js.org/

https://github.com/cesalberca/blog

https://github.com/cesalberca/budget

https://github.com/cesalberca/who-am-i

https://www.archimedesfw.io/