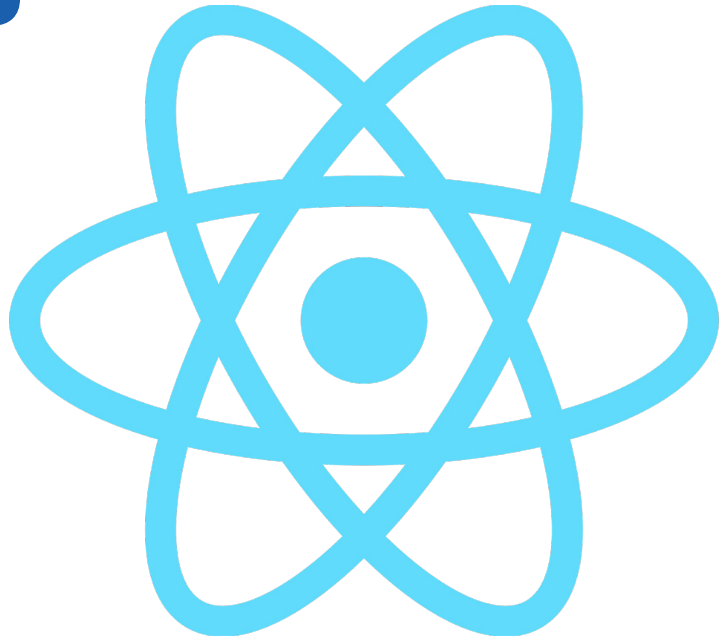
Clase 5

REACT



Temario

1. React
2. React Components and Props
3. Composing React Components and Stateful React Components
4. Events, conditional rendering
5. **Lists and forms**
6. React Router I
7. React Router II
8. Debugging and Testing

Listas y keys

En nuestro ejercicio de tablas hemos renderizado los registros de personas. La manera más común de hacerlo es a través del método **map** de un array, que recorre la lista y lo convertimos en un elemento JSX a renderizar.

El método map devuelve un nuevo array de elementos JSX que podemos renderizar como una variable más (**listStudents**)

Si nos fijamos en el primer ejemplo, nos marca el componente **li** informando que se requiere la definición de una **key.**

Las **keys** en un array ayudan a React a identificar qué items han cambiado, han sido agregados o eliminados. Las keys deben ser dadas a los elementos dentro del array para darle a los elementos una identidad estable.

```
function App() {
  const students = ['Joe', 'María', 'Michel'];
  const listStudents = students.map((student) => {
    return <li>{student}</li>
  });
  return (
    <div className="App">
      {listStudents}
    </div>
  );
}
```

```
function App() {
  const students = ['Joe', 'María', 'Michel'];
  const listStudents = students.map((student, index) => {
    return <li key={index}>{student}</li>
  });
  return (
    <div className="App">
      {listStudents}
    </div>
  );
}
```

Listas y keys: keys

Una **key** es un string que identifica únicamente a un elemento de la lista entre sus hermanos. Habitualmente se usan los IDs de los registros.

Si no tenemos IDs estables para cada registro, como última instancia se puede utilizar como key el índice de los elementos del array.

La práctica recomendada es utilizar campos únicos entre los registros, por ejemplo: ID, DNI, CIF, etc. Datos que no se puedan repetir en la lista.

```
function App() {
  const students = [
    {
      dni: '11111111A',
      name: 'Joe',
      date_of_birth: '1987-01-02'
    },
    {
      dni: '22222222B',
      name: 'María',
      date_of_birth: '1989-01-02'
    }
  ];
  const listStudents = students.map((student, index) => {
    return <li key={student.dni}>{student.name}</li>
  });
  return (
    <div className="App">
      {listStudents}
    </div>
  );
}
```

Listas y keys: keys

Las **keys** sólo tienen sentido en el contexto del array que las envuelve. Si refactorizamos y extraemos el componente ListItem, la key no tiene sentido que se declare en el componente ListItem.

La key tiene que definirse dentro del map, asignarlo al elemento ListItem.

Otra característica a tener en cuenta es que la **key** no se pasa como prop, si necesitamos ese dato dentro del componente hay que pasarlo expresamente en otra prop. Por ejemplo:

```
const listStudents = students.map((student, index) => {
  return (
    <ListItem key={student.dni} dni={student.dni} value={student.name} />
  );
});
return <div className="App">{listStudents}</div>;
```

```
const ListItem = ({value}) => {
  return <li>{value}</li>
}

function App() {
  const students = [
    {
      dni: '11111111A',
      name: 'Joe',
      date_of_birth: '1987-01-02'
    },
    {
      dni: '22222222B',
      name: 'María',
      date_of_birth: '1989-01-02'
    }
  ];
  const listStudents = students.map((student, index) => {
    return <ListItem key={student.dni} value={student.name} />
  });
  return (
    <div className="App">
      {listStudents}
    </div>
  );
}
```

Formularios: Componentes controlados

Los elementos de formularios en HTML conservan naturalmente algún estado interno, su funcionamiento es un poco diferente a otros elementos del DOM en React.

Un formulario por defecto navega a otra página cuando el usuario envía el formulario a través del botón Submit. Sin embargo, en la mayoría de casos, es conveniente tener una función Javascript que tenga acceso a los datos que el usuario introdujo en el formulario y se encargue del envío del formulario. La forma predeterminada para conseguir esto es una técnica llamada "componentes controlados".

Componente controlado es aquel cuyo valor es controlado por React, de manera que el estado de React es la "**única fuente de la verdad**". De esta manera los componentes React que rendericen un formulario también controlan lo que pasa en ese formulario con las subsecuentes entradas del usuario.

En el ejemplo vemos un input que recibe como atributo **value** el campo **value** del estado del componente, y, a su vez, mediante el manejador **handleChange** actualizamos el estado con lo introducido por el usuario, de esta manera el valor del input siempre está dirigido por el estado de React.

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: "" };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({ value: event.target.value });
  }
  handleSubmit(event) {
    alert("A name was submitted: " + this.state.value);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input
            type="text"
            value={this.state.value}
            onChange={this.handleChange}
          />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Formularios: Componentes controlados

Al especificar la prop **value** en un componente controlado se evita que el usuario cambie la entrada a menos que así lo quiera. Si se ha especificado **value** pero el input aún es editable, es porque el valor de la prop **value** es **undefined** o **null.**

Es por este motivo que en un componente controlado es necesario construir un manejador o controlador de eventos para cada elemento que actualice el estado del componente y lo asigne al valor del componente controlado.

```
return (
  <form onSubmit={this.handleSubmit}>
    <label>
      No editable:
      <input type="text" value="hi" />
    </label>
    <label>
      Editable:
      <input type="text" value={null} />
    </label>
    <input type="submit" value="Submit" />
  </form>
);
```

Formularios: Componentes controlados

textarea
La etiqueta textarea define su texto por sus hijos.

```
<textarea>
  Hello there, this is some text in a text area
</textarea>
```

select
La etiqueta select define su lista desplegable con sus hijos, y se selecciona una opción de la lista a través del atributo **selected** del tag **option.**

```
<select>
  <option value="grapefruit">Grapefruit</option>
  <option value="lime">Lime</option>
  <option selected value="coconut">Coconut</option>
  <option value="mango">Mango</option>
</select>
```

En ambos casos React utiliza el atributo **value** para asignar su valor, el texto en el caso del textarea y la opción seleccionada en el caso del select. El objeto **event** que devuelve el evento **onChange** obtiene su valores con el campo **value** (event.target.value)

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      textArea: "Hello there, this is some text in a text area",
      select: "coconut",
    };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    const { name, value } = event.target;
    this.setState({ [name]: value });
  }
  handleSubmit(event) {
    alert("textarea submitted: " + this.state.textArea);
    alert("select submitted: " + this.state.select);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Text area:
          <textarea
            value={this.state.textArea}
            name="textArea"
            onChange={this.handleChange}
          />
        </label>
        <br />
        <label>
          Pick your favorite flavor:
          <select value={this.state.select} name="select"
            onChange={this.handleChange}>
            <option value="grapefruit">Grapefruit</option>
            <option value="lime">Lime</option>
            <option value="coconut">Coconut</option>
            <option value="mango">Mango</option>
          </select>
        </label>
        <br />
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Ejercicios

1. Replicar el código de clase:
    - App my-eleventh-app

ALBAÑILESDIGITALES

Developing your new career

servicio navarro de empleo

das·nano

VeriDas