

**Autor: RAFAEL SOLÍS IÓPEZ**

<https://www.linkedin.com/in/rafael-solis-595294192/>

<https://github.com/RAFASOLIS/PROYECTOS>

## ¿Qué es React?

### Declarativo

React te ayuda a crear interfaces de usuario interactivas de forma sencilla. Diseña vistas simples para cada estado en tu aplicación, y React se encargará de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien.

Las vistas declarativas hacen que tu código sea más predecible, por lo tanto, fácil de depurar.

### Basado en componentes

Crea componentes encapsulados que manejen su propio estado, y conviértelos en interfaces de usuario complejas.

Ya que la lógica de los componentes está escrita en JavaScript y no en plantillas, puedes pasar datos de forma sencilla a través de tu aplicación y mantener el estado fuera del DOM.

Aprende una vez, escríbelo donde sea

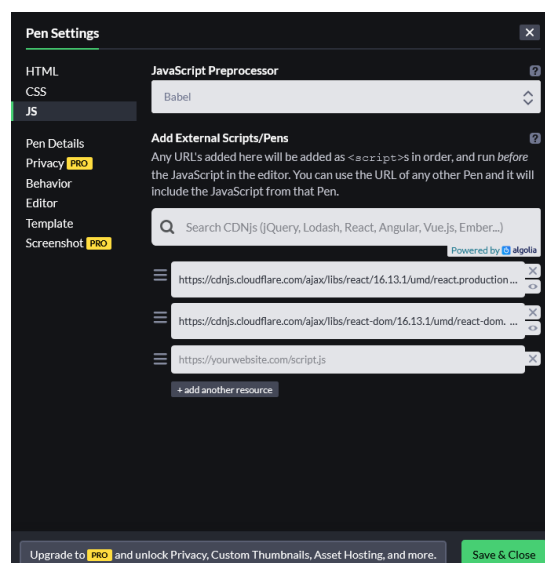
En React no dejamos fuera al resto de tus herramientas tecnológicas, así que podrás desarrollar nuevas características sin necesidad de volver a escribir el código existente.

React puede también renderizar desde el servidor usando Node, así como potencializar aplicaciones móviles usando [React Native](#).

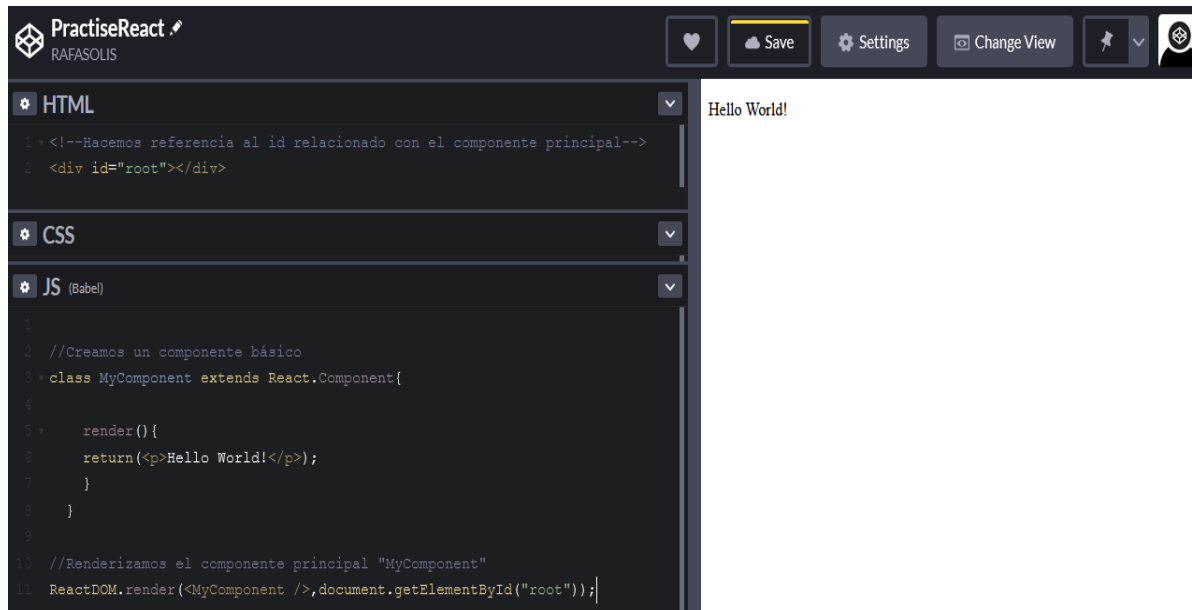
<https://es.reactjs.org/>

### Ejemplo 1. Creación de React básico

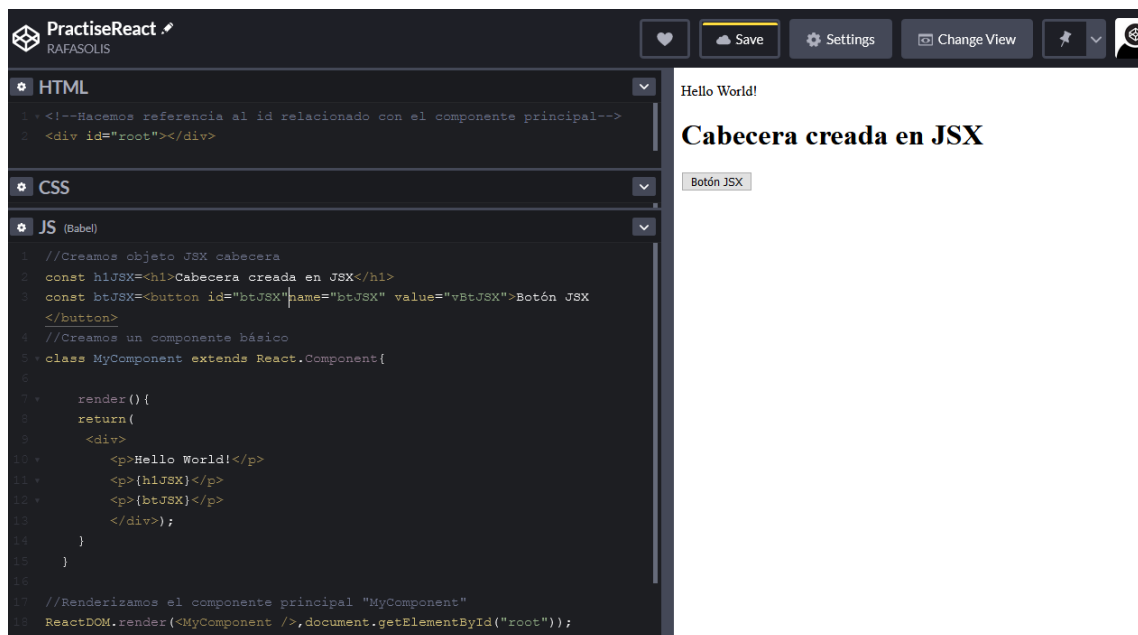
Creación de un proyecto simple en la plataforma Codepen.io. Debemos elegir el preprocesador “Babel” e importar las librerías “React” y “React-dom”.



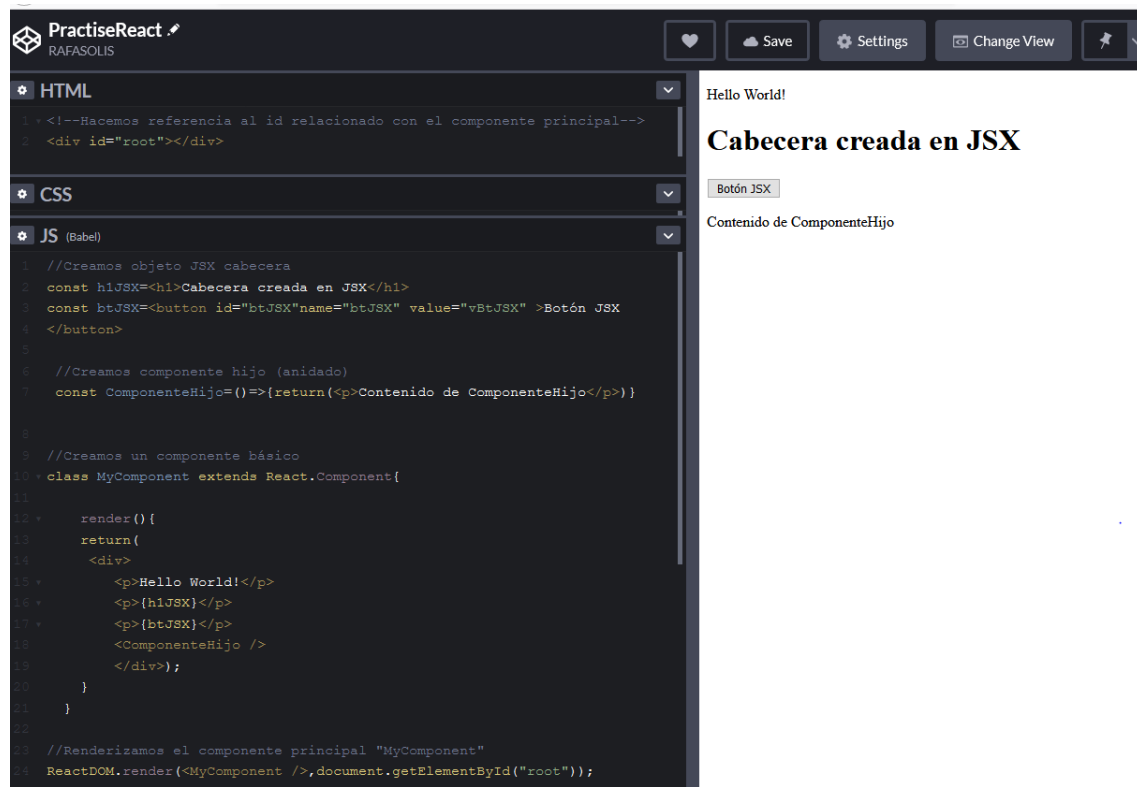
## 14 EJEMPLOS REACT



### Ejemplo 2. Inclusión de objetos JSX



### Ejemplo 3. Creación de un componente hijo anidado en el componente principal



PractiseReact  
RAFA SOLIS

HTML

```
1 <!--Hacemos referencia al id relacionado con el componente principal-->
2 <div id="root"></div>
```

CSS

JS (Babel)

```
1 //Creamos objeto JSX cabecera
2 const h1JSX=<h1>Cabecera creada en JSX</h1>
3 const btJSX=<button id="btJSX" name="btJSX" value="vBtJSX" >Botón JSX
4 </button>
5
6 //Creamos componente hijo (anidado)
7 const ComponenteHijo={()=>{return(<p>Contenido de ComponenteHijo</p>)}}
8
9 //Creamos un componente básico
10 class MyComponent extends React.Component{
11
12   render() {
13     return(
14       <div>
15         <p>Hello World!</p>
16         <p>{h1JSX}</p>
17         <p>{btJSX}</p>
18         <ComponenteHijo />
19       </div>;
20     )
21   }
22 }
23
24 //Renderizamos el componente principal "MyComponent"
25 ReactDOM.render(<MyComponent />,document.getElementById("root"));
```

Hello World!

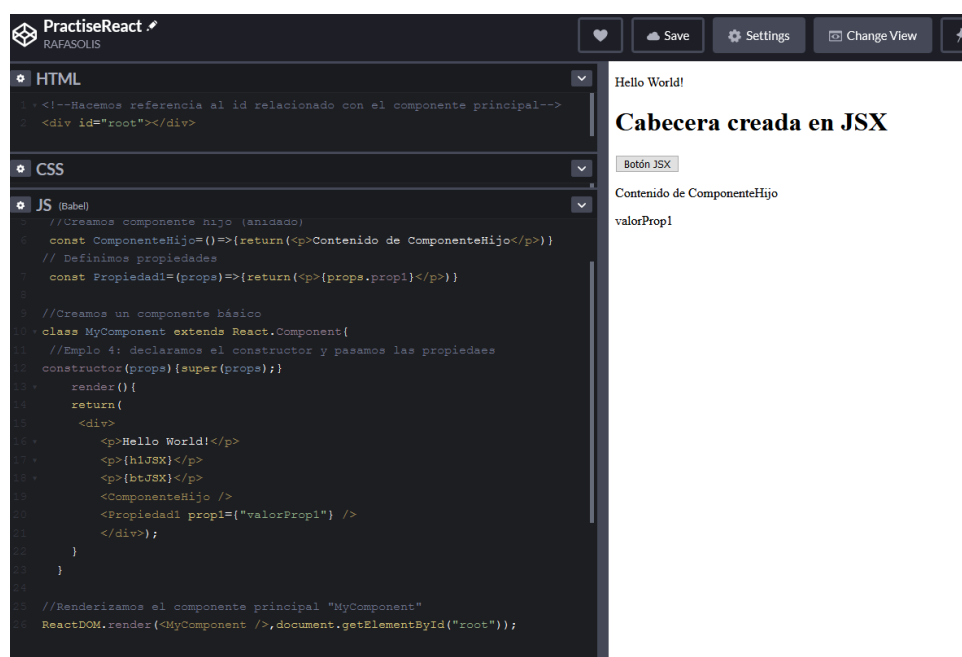
## Cabecera creada en JSX

Botón JSX

Contenido de ComponenteHijo

### Ejemplo 4. Definimos propiedades

Debemos incluir dentro de la clase donde vamos a usar las propiedades el constructor.



PractiseReact  
RAFA SOLIS

HTML

```
1 <!--Hacemos referencia al id relacionado con el componente principal-->
2 <div id="root"></div>
```

CSS

JS (Babel)

```
1 //Creamos componente hijo (anidado)
2 const ComponenteHijo={()=>{return(<p>Contenido de ComponenteHijo</p>)}}
3 // Definimos propiedades
4 const Propiedad1=(props) =>{return(<p>{props.prop1}</p>)}
5
6 //Creamos un componente básico
7 class MyComponent extends React.Component{
8   //Ejemplo 4: declaramos el constructor y pasamos las propiedades
9   constructor(props) {super(props);}
10
11   render() {
12     return(
13       <div>
14         <p>Hello World!</p>
15         <p>{h1JSX}</p>
16         <p>{btJSX}</p>
17         <ComponenteHijo />
18         <Propiedad1 prop1="valorProp1" />
19       </div>;
20     )
21   }
22 }
23
24 //Renderizamos el componente principal "MyComponent"
25 ReactDOM.render(<MyComponent />,document.getElementById("root"));
```

Hello World!

## Cabecera creada en JSX

Botón JSX

Contenido de ComponenteHijo

valorProp1

### Ejemplo 5. definimos propiedades. Array de valores

Para mostrar los valores del array separados debemos incluir `.join("separador")` dentro de la definición de la propiedad de referencia.

The screenshot shows the PractiseReact editor interface. The left pane displays the code for a React component. The right pane shows the rendered output.

**HTML:**

```
<!--Hacemos referencia al id relacionado con el componente principal-->
<div id="root"></div>
```

**CSS:**

**JS (Babel):**

```
//Creamos componente hijo (anidado)
const ComponenteHijo={()=>{return(<p>Contenido de ComponenteHijo</p>)}}
// Definimos propiedades
const Propiedad1=(props)=>{return(<p>{props.prop1}</p>)}
//Definición propiedad con array
const PropiedadArray=(props)=>{return(<p>{props.propArray.join(", ")}
</p>)}

//Creamos un componente básico
class MyComponent extends React.Component{
  //Ejemplo 4: declaramos el constructor y pasamos las propiedades
  constructor(props){super(props);}
  render(){
    return(
      <div>
        <p>Hello World!</p>
        <p>{this.props.prop1}</p>
        <p>{this.props.propArray}</p>
        <ComponenteHijo />
        <Propiedad1 prop1="valorProp1" />
        <PropiedadArray propArray=[1,2,3,4] />
      </div>
    );
  }
}

//Renderizamos el componente principal "MyComponent"
ReactDOM.render(<MyComponent />,document.getElementById("root"));
```

**Rendered Output:**

Hello World!

### Cabecera creada en JSX

Botón JSX

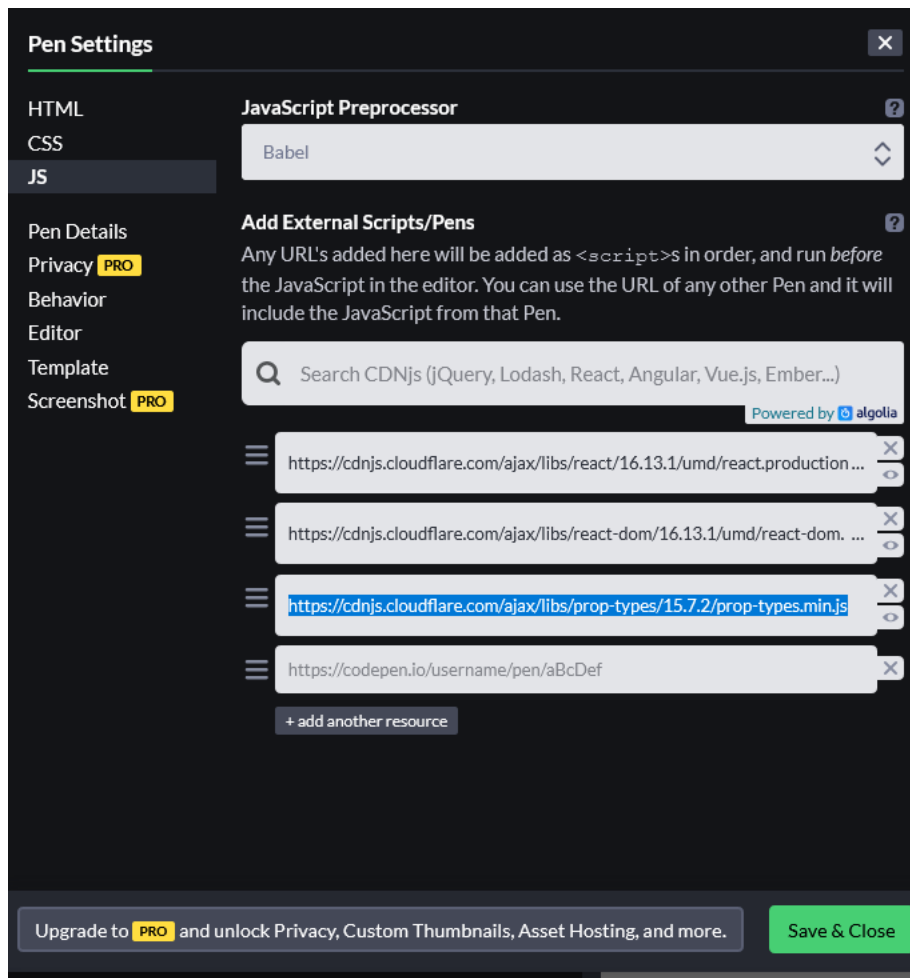
Contenido de ComponenteHijo

valorProp1

1,2,3,4

<https://codepen.io/rafasolis/pen/jOqZjj>

### Ejemplo 6. Uso de propTypes



```

12 //Definimos la función pque pasaremos al atributo Function1 de Prototipos
13 function suma(a,b){return (a+b);}
14 //Definimos propiedades
15 * const Prototipos=(props)=>{return(<p>PropTypes:
16 *   <ul>
17 *     <li> Prop. Number: {props.number}</li>
18 *     <li> Prop. Check: {props.ckeck}</li>
19 *     <li> Prop. Text: {props.text}</li>
20 *     <li> Prop. Func: {props.function1}</li>
21 *   </ul>
22 *   </p>)}
23
24
25 * Prototipos.propTypes={
26   function1:PropTypes.func.isRequired,
27   number:PropTypes.number.isRequired,
28   check:PropTypes.bool.isRequired,
29   text:PropTypes.string.isRequired,
30   object:PropTypes.object.isRequired
31 }
32
33
34 //Creamos un componente básico
35 class MyComponent extends React.Component{
36   //Emplo 4: declaramos el constructor y pasamos las propiedaes
37   constructor(props){super(props);}
38   render(){
39     return(
40       <div>
41         <p>Hello World!</p>
42         <p>{h1JSX}</p>
43         <p>{btJSX}</p>
44         <ComponenteHijo />
45         <Propiedad1 prop1={"valor propiedad 1"} />
46         <PropiedadArray propArray={[1,2,3,4]} />
47         <Prototipos number={10} check={false} text={"texto"} function1={suma(1,2)} />
48       </div>);
49   }
50 }
51
52 //Definimos el valor pro defcto de la propiedad 1
53 MyComponent.defaultProps={prop1:"Valor por defecto Propiedad1"};
54
55 //

```

<https://codepen.io/rafasolis/pen/iOqZjiJ>

<https://www.freecodecamp.org/learn/front-end-libraries/react/use-proptypes-to-define-the-props-you-expect>

### Ejemplo 7. Uso de funciones

Las props son de sólo lectura, independientemente que sean declaradas como funciones, componentes o clases.

Funciones puras: siempre devuelven el mismo resultado antes mismas entradas.

Ej: `function sum(a,b){return a+b;}`

Funciones impuras: el resultado cambia su entrada:

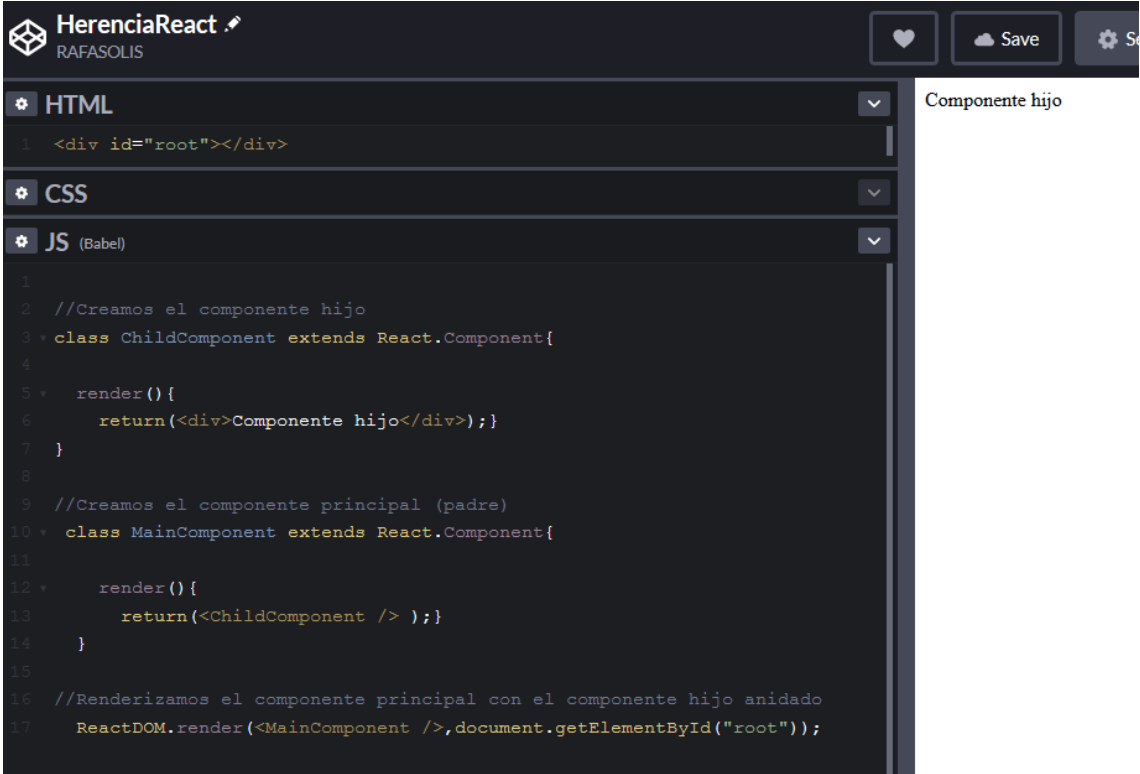
Ej: `function incremento(saldo, ingreso){saldo.total+=ingreso; }`

### Ejemplo 8. Propiedades

Las propiedades pueden pasarse de padres a hijos.

Sólo el padre puede modificarla.

Ej 8.1: creamos los componentes padre (MainComponent) e hijo (ChildComponent). Llamamos al componente hijo desde el padre.



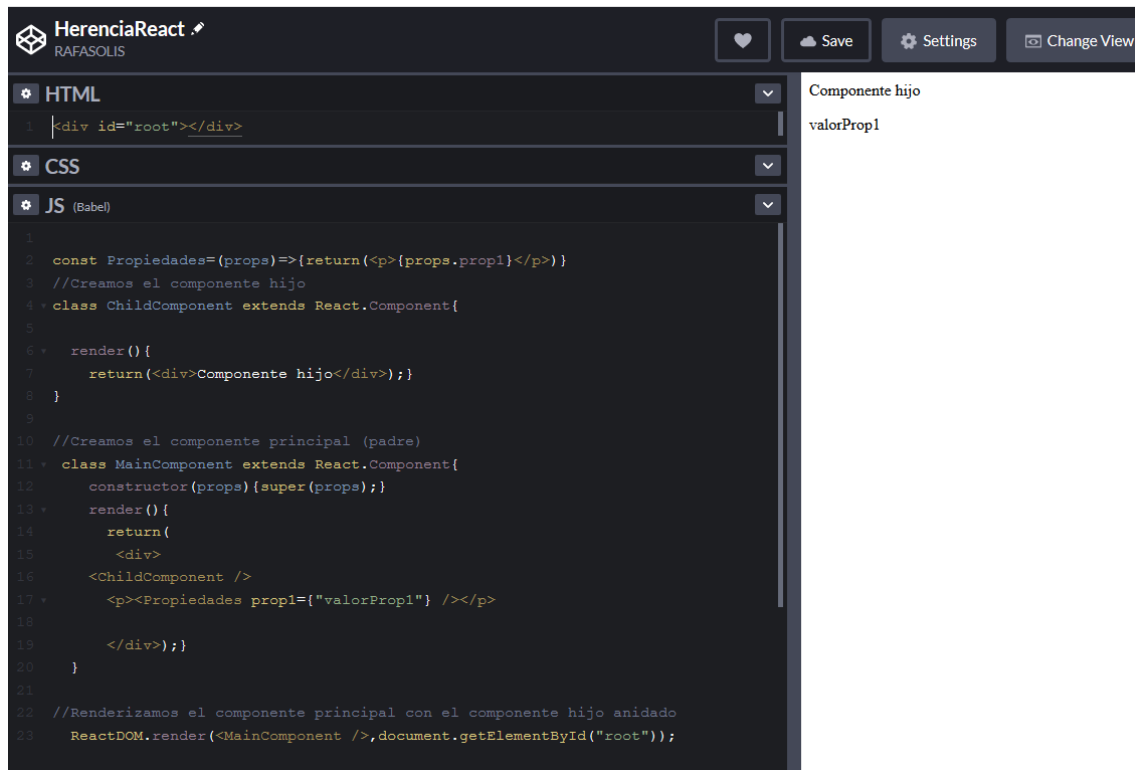
The screenshot shows the HerenciaReact editor interface. The top bar includes the logo, the name 'HerenciaReact', the author 'RAFASOLIS', and buttons for 'Save' and 'Settings'. The left sidebar shows the file explorer with 'HTML', 'CSS', and 'JS (Babel)' files. The main editor area displays the following code:

```
1 <div id="root"></div>
2 //Creamos el componente hijo
3 class ChildComponent extends React.Component{
4
5   render(){
6     return(<div>Componente hijo</div>);}
7   }
8
9 //Creamos el componente principal (padre)
10 class MainComponent extends React.Component{
11
12   render(){
13     return(<ChildComponent /> );}
14   }
15
16 //Renderizamos el componente principal con el componente hijo anidado
17 ReactDOM.render(<MainComponent />,document.getElementById("root"));
```

On the right side, there is a preview area labeled 'Componente hijo' showing a visual representation of the component.

## 14 EJEMPLOS REACT

Ej 8.2: sobre el ejemplo anterior. Definimos un componente con propiedades (Propiedades). Damos valor a la propiedad definida (prop1) y renderizamos a través del componente Propiedades. Para ello debemos crear en el componente principal el constructor para pasarle las propiedades externas de la clase.



```
HerenciaReact
RAFA SOLIS

HTML
1 | <div id="root"></div>

CSS

JS (Babel)
1
2 | const Propiedades=(props)=>{return(<p>{props.prop1}</p>)}
3 | //Creamos el componente hijo
4 | class ChildComponent extends React.Component{
5 |
6 |   render() {
7 |     return(<div>Componente hijo</div>);
8 |   }
9 |
10 | //Creamos el componente principal (padre)
11 | class MainComponent extends React.Component{
12 |   constructor(props) {super(props);}
13 |   render() {
14 |     return(
15 |       <div>
16 |       <ChildComponent />
17 |       <p><Propiedades prop1={"valorProp1"} /></p>
18 |       </div>);
19 |   }
20 | }
21
22 //Renderizamos el componente principal con el componente hijo anidado
23 ReactDOM.render(<MainComponent />,document.getElementById("root"));
```

Componente hijo  
valorProp1

Ej3: sobre el ejemplo anterior. Definimos un componente con propiedades (Propiedades). Damos valor a la propiedad definida (prop1) y renderizamos a través del componente hijo. Para ello debemos crear el constructor, además de en la clase padre en la clase hijo. En la clase padre podremos darle valor a la propiedad y la clase hijo será la que renderize su valor.



The screenshot shows a code editor with the following content:

```

HerenciaReact
RAFASOLIS
Autosave enabled.
Save
Settings

HTML
1 <div id="root"></div>

CSS

JS (Babel)
1
2 const Propiedades=(props)=>{return(<p>{props.prop1}</p>)}
3 //Creamos el componente hijo
4 class ChildComponent extends React.Component{
5   constructor(props){super(props);}
6   render(){
7     return(<div>Componente hijo
8       <p>{this.props.prop1}</p>
9     </div>);}
10 }
11
12 //Creamos el componente principal (padre)
13 class MainComponent extends React.Component{
14   constructor(props){super(props);}
15   render(){
16     return(
17       <div>
18         <ChildComponent prop1={"valorProp1"}/>
19
20       </div>);}
21 }
22
23
24 //Renderizamos el componente principal con el componente hijo anidado
25 ReactDOM.render(<MainComponent />,document.getElementById("root"));
```

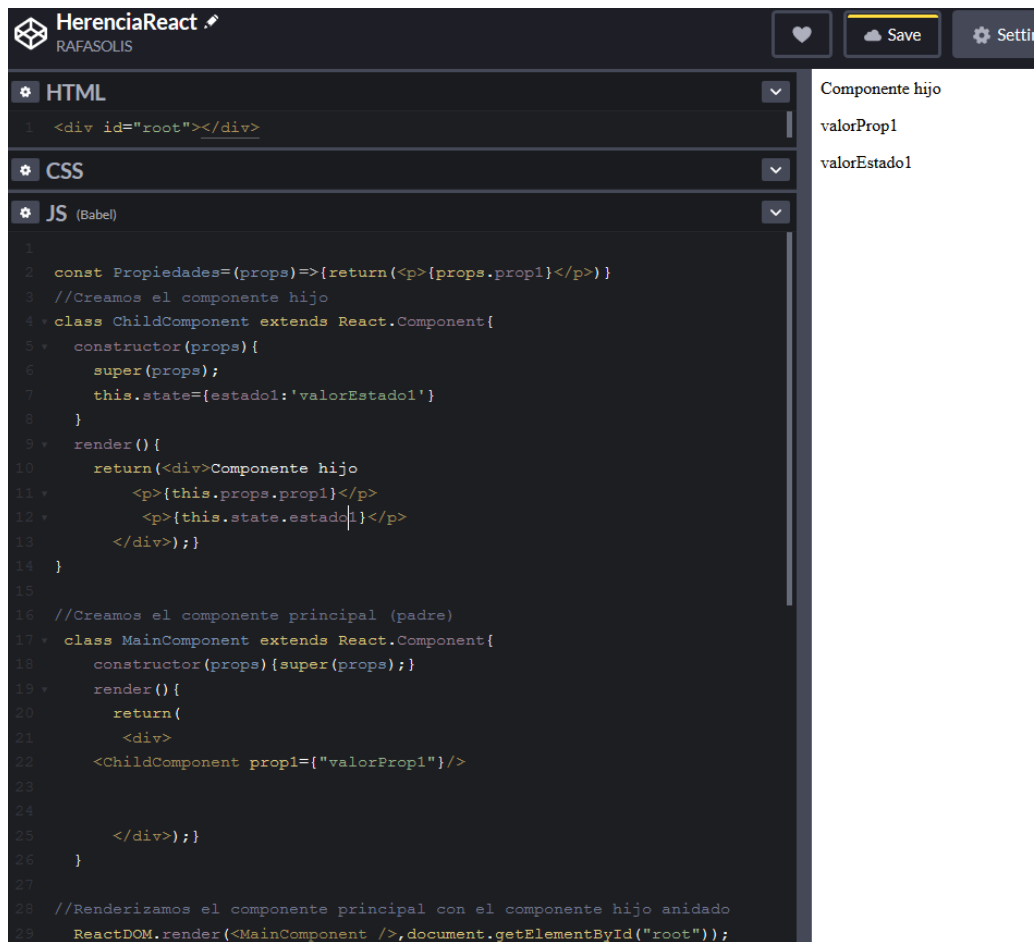
Componente hijo  
valorProp1

<https://codepen.io/rafasolis/pen/xxVWzWa>

### Ejemplo 9. Estados

Ej 9.1: Declaramos un estado en el constructor del componente hijo y lo renderizamos.

## 14 EJEMPLOS REACT



The screenshot shows a code editor with three tabs: HTML, CSS, and JS (Babel). The JS tab is active, displaying the following code:

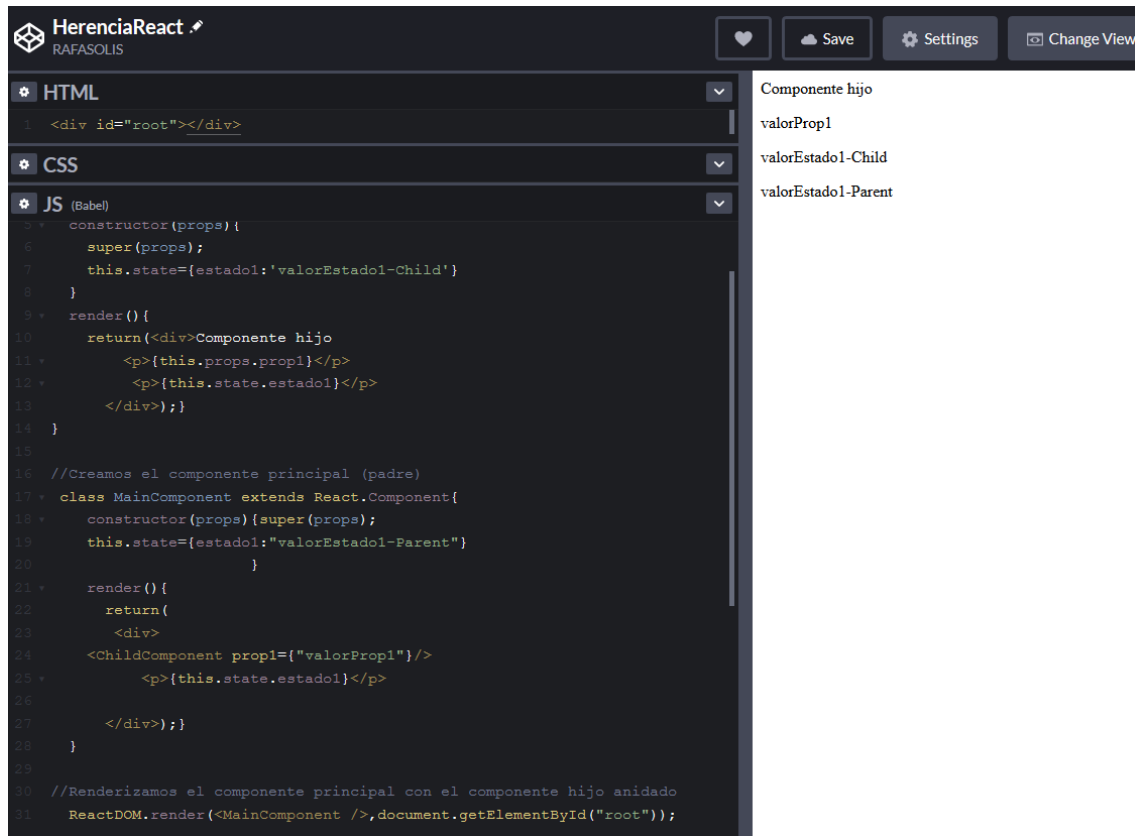
```
1
2 const Propiedades=(props)=>{return(<p>{props.prop1}</p>)}
3 //Creamos el componente hijo
4 class ChildComponent extends React.Component{
5   constructor(props){
6     super(props);
7     this.state={estado1:'valorEstado1'}
8   }
9   render(){
10    return(<div>Componente hijo
11      <p>{this.props.prop1}</p>
12      <p>{this.state.estado1}</p>
13      </div>);}
14 }
15
16 //Creamos el componente principal (padre)
17 class MainComponent extends React.Component{
18   constructor(props){super(props);}
19   render(){
20     return(
21       <div>
22         <ChildComponent prop1={"valorProp1"}/>
23
24       </div>);}
25 }
26
27 //Renderizamos el componente principal con el componente hijo anidado
28 ReactDOM.render(<MainComponent />,document.getElementById("root"));
```

On the right side of the editor, there is a component inspector showing the state of the 'Componente hijo' component:

- Componente hijo
- valorProp1
- valorEstado1

Ej 9.2: Declaramos un segundo estado en el componente padre, cada componente tiene un estado. Lo renderizamos en el componente padre.

## 14 EJEMPLOS REACT



Ej 9.3: Almacenamos el valor del estado (estado1) creado en el padre en una variable (var\_estado1) y la renderizamos.

**HerenciaReact**  
RAFASOLIS

HTML

```
1 <div id="root"></div>
```

CSS

JS (Babel)

```
3 constructor(props) {
4   super(props);
5   this.state={estado1:'valorEstado1-Child'}
6 }
7
8 render() {
9   return(<div>Componente hijo
10     <p>{this.props.prop1}</p>
11     <p>{this.state.estado1}</p>
12     </div>);}
13
14 }
15
16 //Creamos el componente principal (padre)
17 class MainComponent extends React.Component{
18   constructor(props){super(props);
19   this.state={estado1:"valorEstado1-Parent"}
20 }
21
22 render(){
23   const var_estado1=this.state.estado1;
24   return(
25     <div>
26     <ChildComponent prop1={"valorProp1"}/>
27     <p>{this.state.estado1}</p>
28     <p>Variable: {var_estado1}</p>
29     </div>);}
30 }
31
32 //Renderizamos el componente principal con el componente hijo anidado
33 ReactDOM.render(<MainComponent />,document.getElementById("root"));
```

Componente hijo

- valorProp1
- valorEstado1-Child
- valorEstado1-Parent
- Variable: valorEstado1-Parent

<https://codepen.io/rafasolis/pen/xxVWzWa>

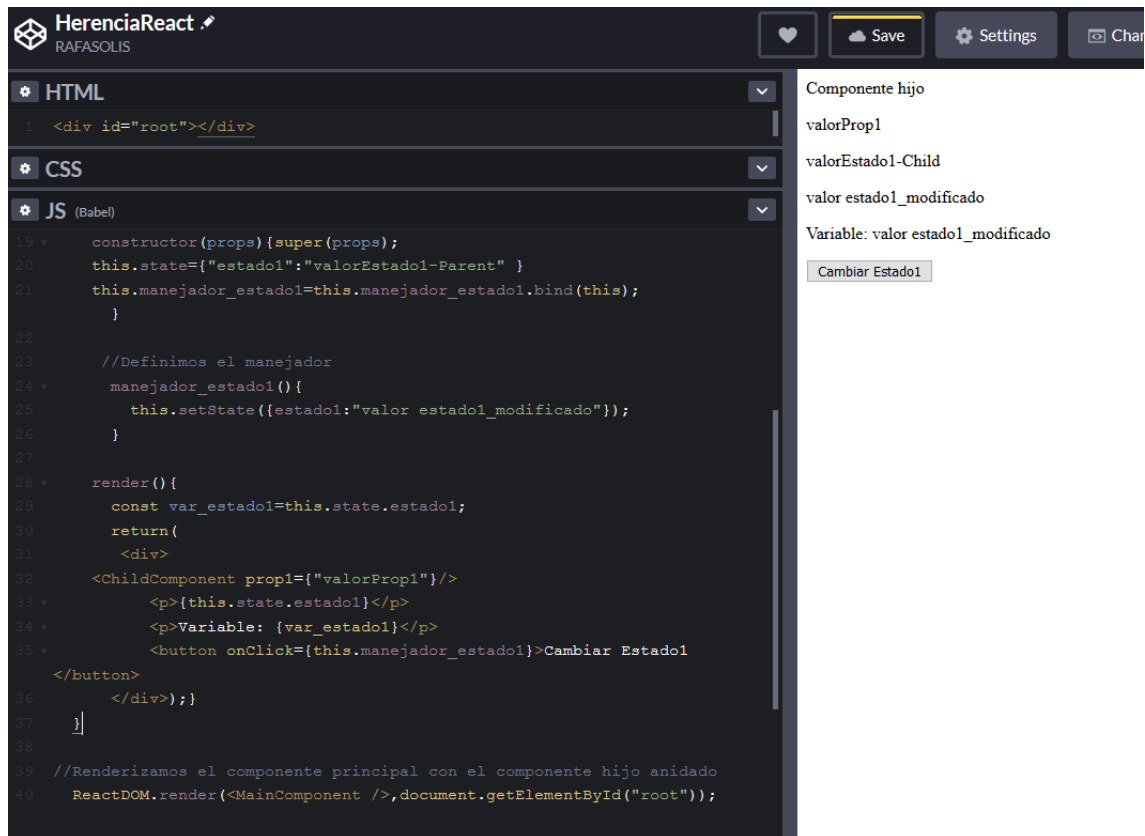
#### Ejemplo 10. Estados y manejadores de estado. Funciones.

Para crear un manejador de estados (función) deben incluirse 3 elementos en la clase donde se va a usar el mismo.

- Dentro del constructor: `this.manejador=this.manejador.bind(this);`
- Dentro de la clase: `manejador(){setState({estado:acción...});}`
- Dentro de render/return: llamamos a la función. Ej: `onClick={this.manejador}`

## 14 EJEMPLOS REACT

Ej 10.1. Al producirse el evento el valor del estado se modifica.

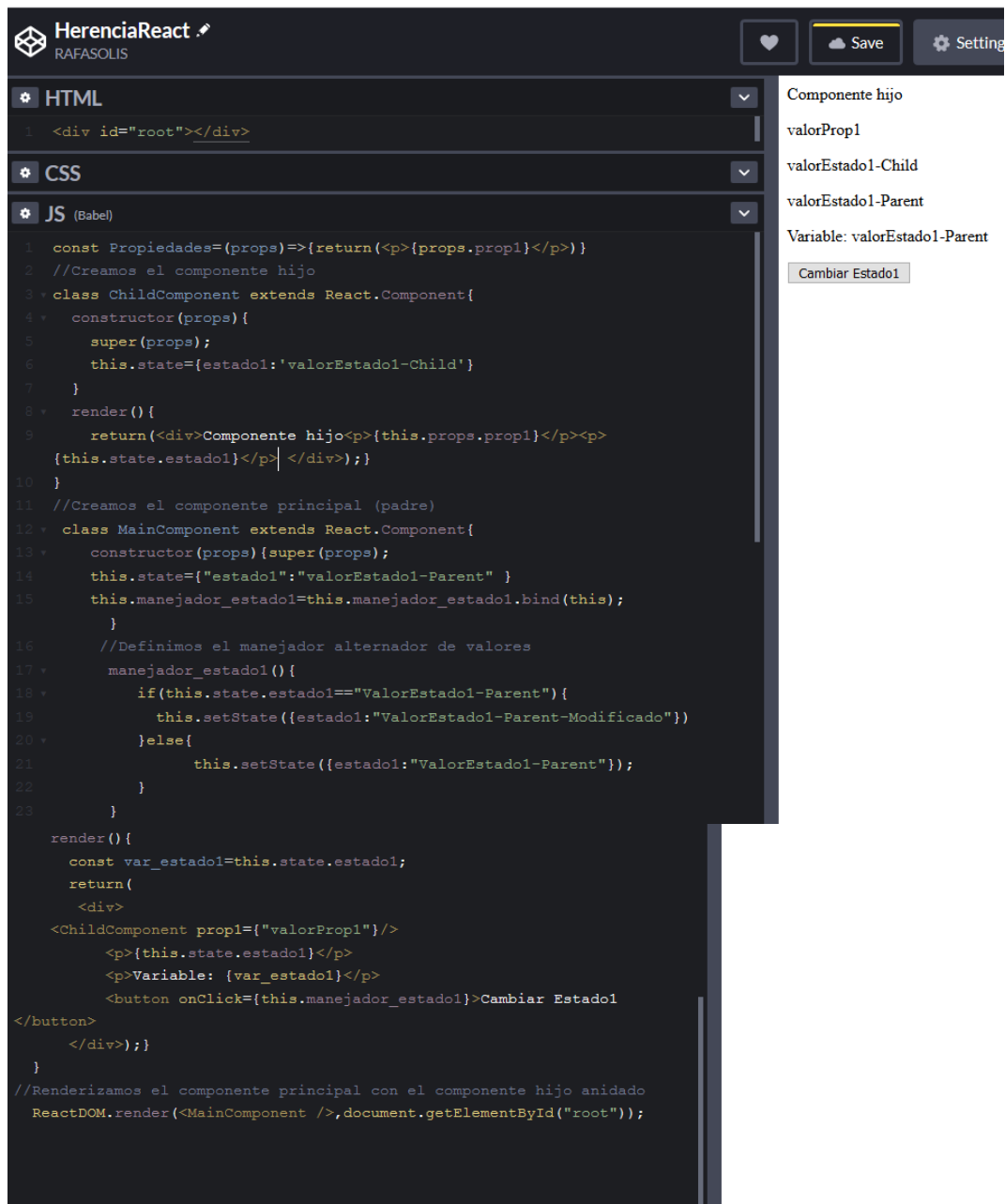


The screenshot shows the HerenciaReact IDE interface. The top bar includes the logo 'HerenciaReact' by 'RAFASOLIS', a heart icon, a 'Save' button, a 'Settings' button, and a 'Chat' button. The main editor is divided into three panes: HTML, CSS, and JS (Babel). The JS pane contains the following code:

```
19 * constructor(props) {super(props);
20   this.state={ "estado1": "valorEstado1-Parent" }
21   this.manejador_estado1=this.manejador_estado1.bind(this);
22 }
23 //Definimos el manejador
24 * manejador_estado1() {
25   this.setState({estado1: "valor_estado1_modificado"});
26 }
27
28 * render() {
29   const var_estado1=this.state.estado1;
30   return(
31     <div>
32       <ChildComponent prop1={"valorProp1"}/>
33       <p>{this.state.estado1}</p>
34       <p>Variable: {var_estado1}</p>
35       <button onClick={this.manejador_estado1}>Cambiar Estado1
36     </button>
37     </div>);
38 }
39 //Renderizamos el componente principal con el componente hijo anidado
40 ReactDOM.render(<MainComponent />,document.getElementById("root"));
```

On the right side, there is a component inspector showing the 'Componente hijo' with props: 'valorProp1', 'valorEstado1-Child', and 'valor\_estado1\_modificado'. Below the props, it shows 'Variable: valor\_estado1\_modificado' and a button labeled 'Cambiar Estado1'.

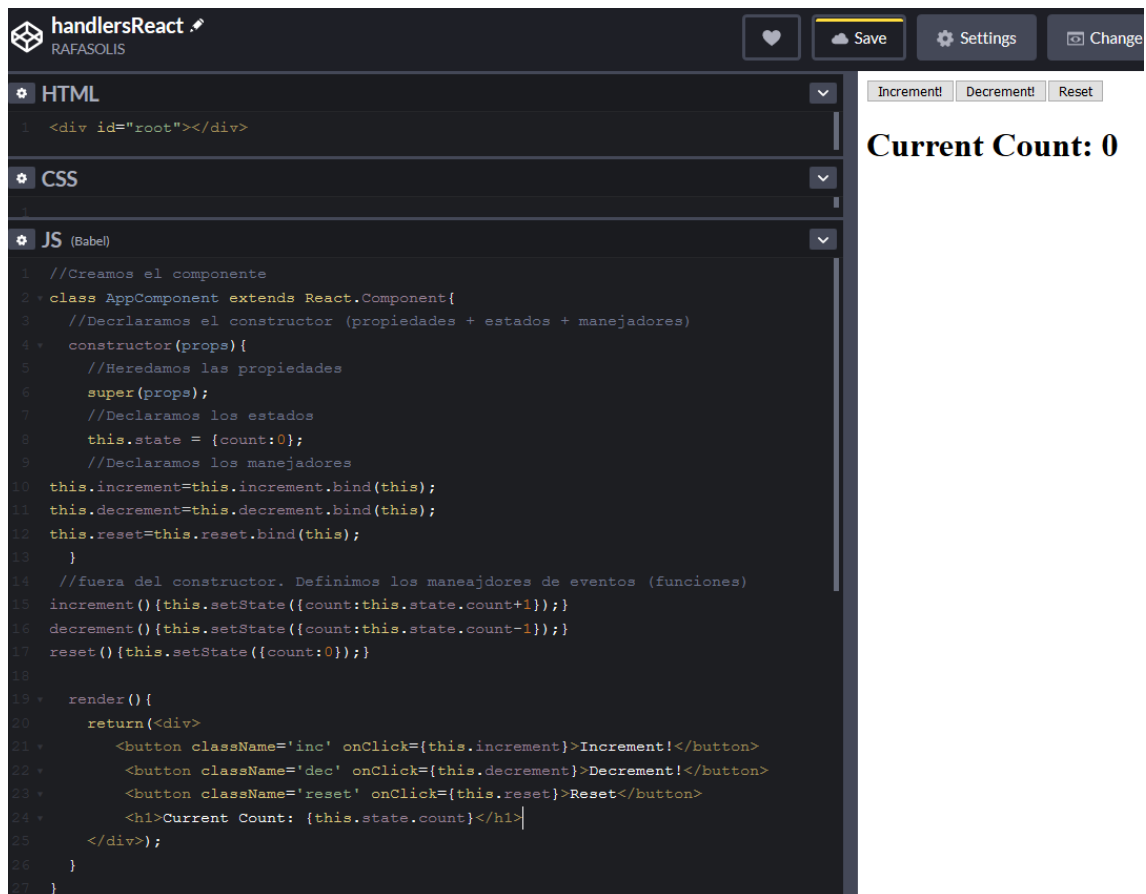
Ej2: Modificamos el manejador para que alterne el valor.



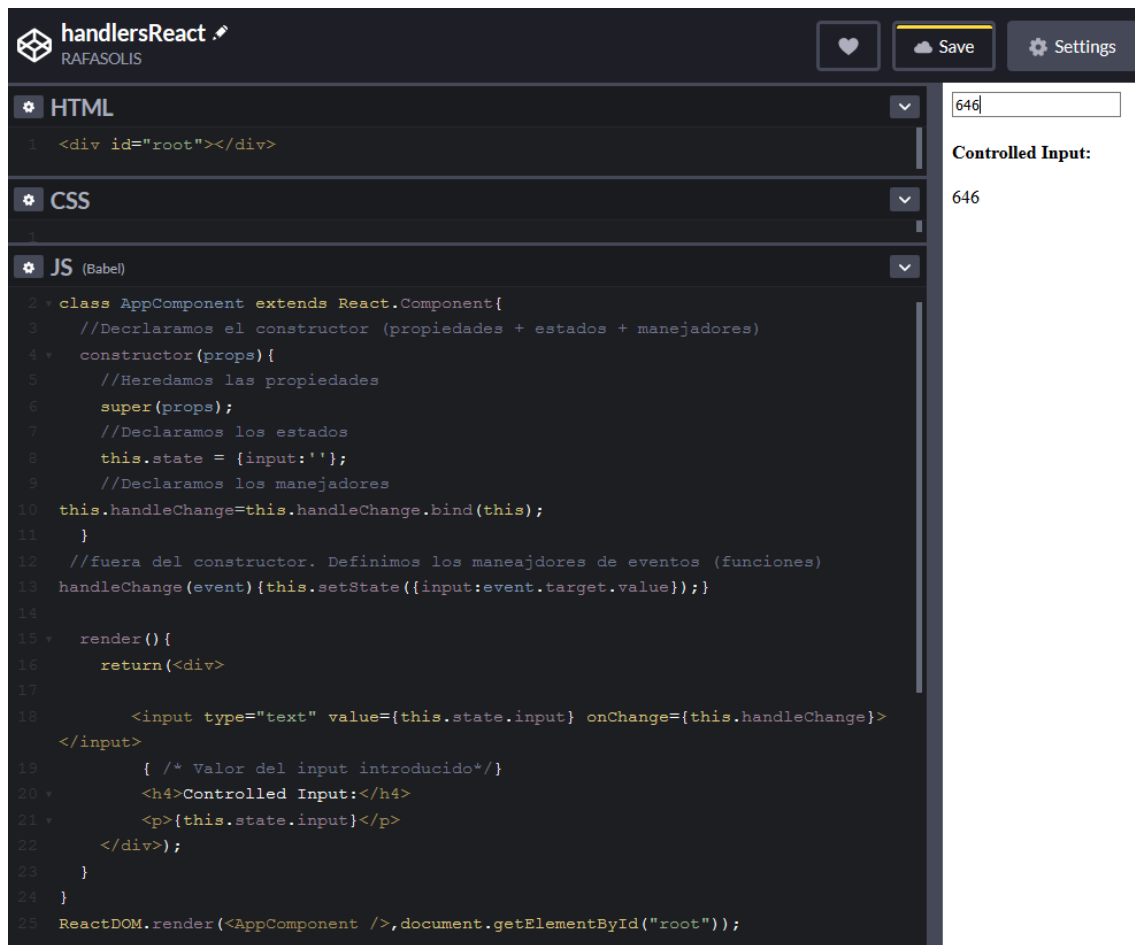
Ej 10.3: Creamos una aplicación con 3 botones y un manejador de eventos asociados a cada uno de ellos. Los tres manejadores modifican un estado declarado dentro del componente creado (count).

Dentro del constructor declaramos tanto los estados como los manejadores. Fuera del mismo definimos cada uno de ellos y e incluimos `this.setState({stado:valor})` para modificar cada uno de los estados.

## 14 EJEMPLOS REACT



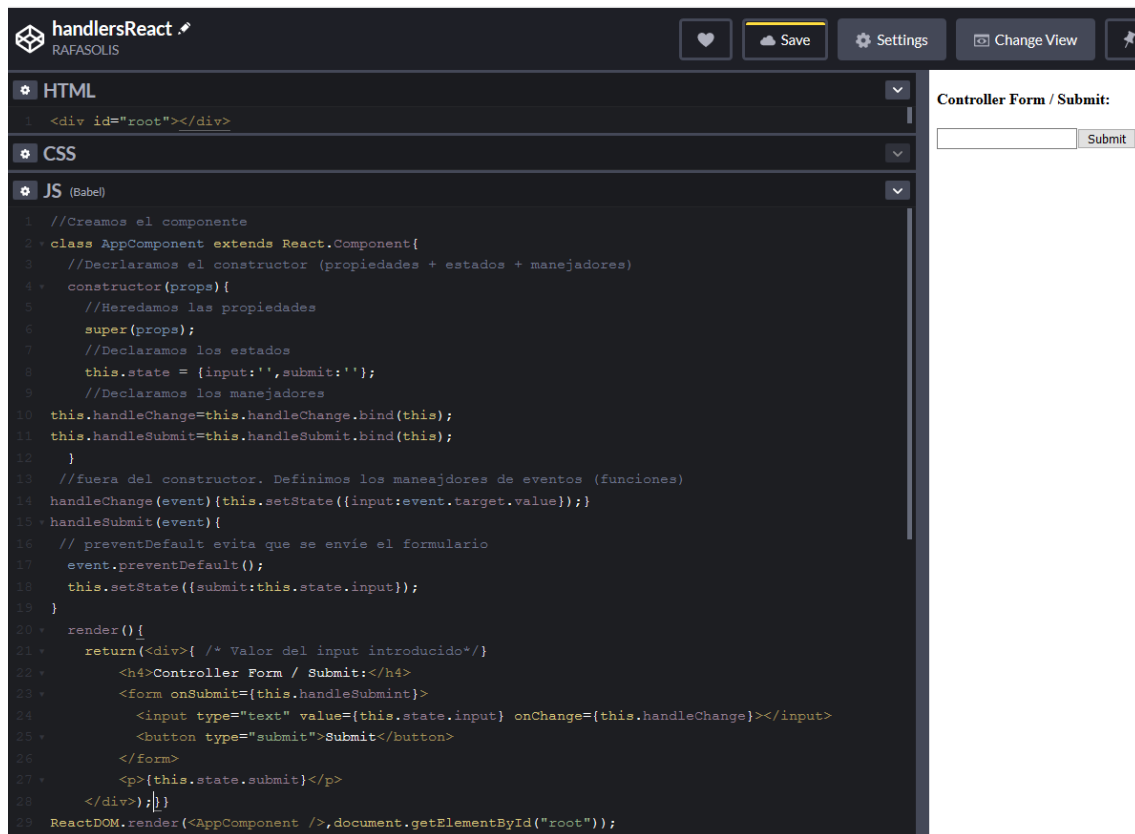
Ej 10.4: Definimos un manejador que controla el evento de input a cambiar.



Ej 10.5: construimos 2 manejadores. Uno recoge 2 eventos a través de sus correspondientes manejadores creados. Un manejador capture valor del elemento input al cambiar su valor y otro el evento de envío de formulario.



## 14 EJEMPLOS REACT



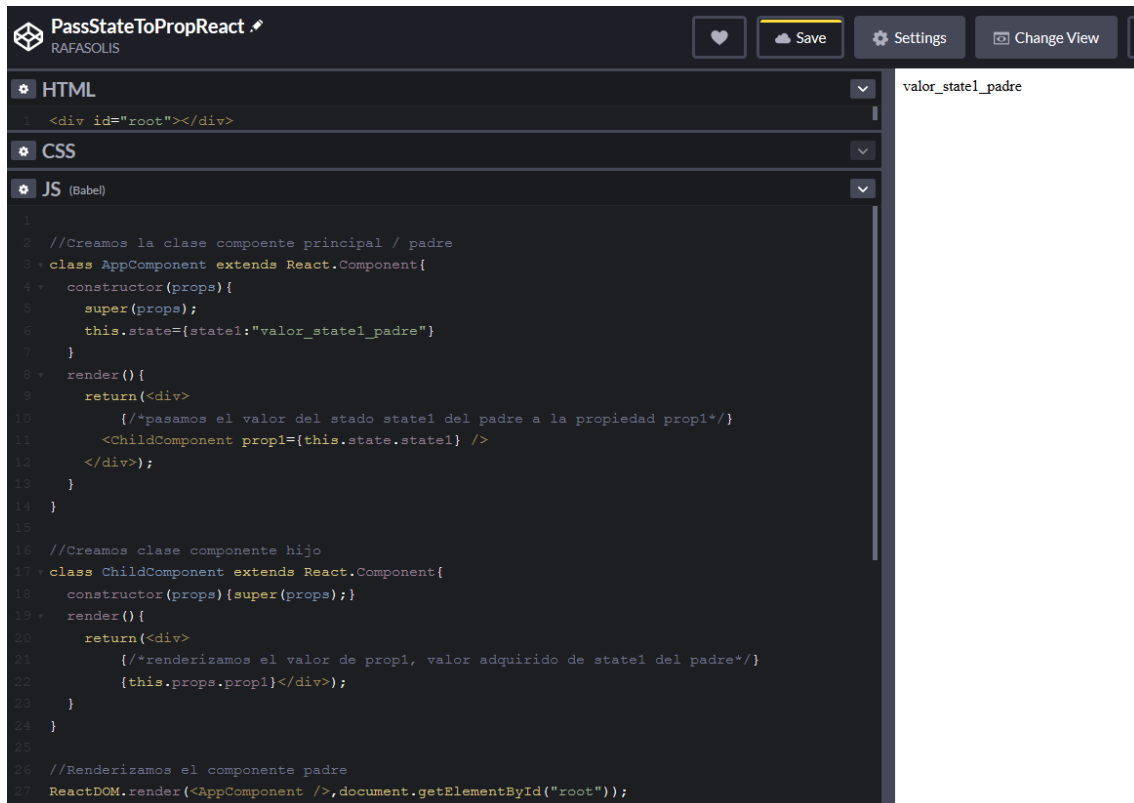
```
1 <div id="root"></div>
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

<https://codepen.io/rafasolis/pen/NWNzvLE>

### Ejemplo 11. Paso de estados del padre al hijo como propiedades

Ej 11.1. Creamos un estado en el padre (state1) y al renderizar el componente hijo pasamos el valor de dicho estado a la propiedad creada prop1.

## 14 EJEMPLOS REACT



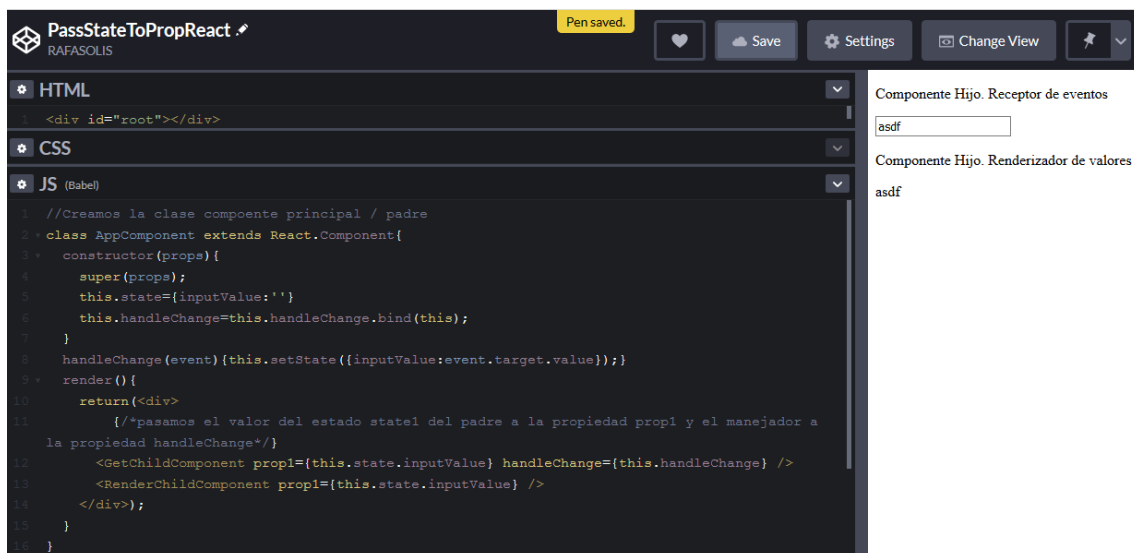
The screenshot shows the PassStateToPropReact editor interface. The top bar includes the logo, name, and buttons for Save, Settings, and Change View. The main editor is divided into three tabs: HTML, CSS, and JS (Babel). The JS tab is active, showing the following code:

```
1 //Creamos la clase componente principal / padre
2 class AppComponent extends React.Component{
3   constructor(props){
4     super(props);
5     this.state={state1:"valor_state1_padre"}
6   }
7   render(){
8     return(<div>
9       /*pasamos el valor del estado state1 del padre a la propiedad prop1*/
10      <ChildComponent prop1={this.state.state1} />
11    </div>);
12   }
13 }
14
15 //Creamos clase componente hijo
16 class ChildComponent extends React.Component{
17   constructor(props){super(props);}
18   render(){
19     return(<div>
20       /*renderizamos el valor de prop1, valor adquirido de state1 del padre*/
21       {this.props.prop1}</div>);
22   }
23 }
24
25 //Renderizamos el componente padre
26 ReactDOM.render(<AppComponent />,document.getElementById("root"));
```

The right sidebar shows the state of the component: `valor_state1_padre`.

Ej 11.2. En el siguiente ejemplo se crean tres componentes:

- Un componente padre (AppComponent) que pasa por propiedades el valor de un estado (inputValue) y un manejador de eventos (handleChange).
- Un componente hijo intermedio (GetChildComponent) que recibe desde el padre los parámetros anteriores a través de las propiedades.
- Un componente hijo final (RenderChildComponent) que renderiza los valores finalmente.



The screenshot shows the PassStateToPropReact editor interface. The top bar includes the logo, name, and buttons for Save, Settings, and Change View. The main editor is divided into three tabs: HTML, CSS, and JS (Babel). The JS tab is active, showing the following code:

```
1 //Creamos la clase componente principal / padre
2 class AppComponent extends React.Component{
3   constructor(props){
4     super(props);
5     this.state={inputValue:''}
6     this.handleChange=this.handleChange.bind(this);
7   }
8   handleChange(event){this.setState({inputValue:event.target.value});}
9   render(){
10    return(<div>
11      /*pasamos el valor del estado state1 del padre a la propiedad prop1 y el manejador a
12      la propiedad handleChange*/
13      <GetChildComponent prop1={this.state.inputValue} handleChange={this.handleChange} />
14      <RenderChildComponent prop1={this.state.inputValue} />
15    </div>);
16   }
17 }
```

The right sidebar shows the state of the component: `Componente Hijo. Receptor de eventos` and `Componente Hijo. Renderizador de valores`, both displaying the value `asdf`.

## 14 EJEMPLOS REACT

```
//Creamos clase componente hijo/ intermedio. Captura los valores del padre al generarse el
evento
class GetChildComponent extends React.Component{
  constructor(props){super(props);}
  render(){
    return(<div>
      <p>Componente Hijo. Receptor de eventos</p>
      /*Capturamos el valor del input ante el evento indicado a través de su manejador
      correspondiente definido*/
      <input type="text" value={this.props.prop1} onChange={this.props.handleChange}/>
    </div>);
  }
}

class RenderChildComponent extends React.Component{
  constructor(props){super(props);}
  render(){return(<div>
    /*Renderiza el valor del estado del padre capturado en el componente intermedio
    GetChildComponent al producirse el evento onChange*/
    <p>Componente Hijo. Renderizador de valores</p>
    <p>{this.props.prop1}</p></div>);}
}

//Renderizamos el componente padre
ReactDOM.render(<AppComponent />,document.getElementById("root"));
```

### Ejemplo 12. Ciclo de vida de los componentes

LifeCycleReact  
RAFASOLIS

Save Settings Change View

HTML

<div id="root"></div>

CSS

JS (Babel)

```
1 const lifestates=[];
2 class AppComponent extends React.Component{
3   constructor(props){super(props);
4     this.state={lifestate:''}
5   /*Definimos varios métodos que conforman el ciclo de vida del componente */
6   }
7   componentWillMount(){this.setState({lifestate:"ComponentWillMount: antes de
montarse en el DOM."});}
8   /*Suele emplearse para cargar librerías de terceros y para usar setInterval,
setTimeout*/
9   componentDidMount(){this.setState({lifestate:"ComponentDidMount: al
actualizar elementos del DOM."});}
10  componentWillUnmount(){this.setState({lifestate:"ComponentWillUnmount: justo
antes de destruir el componente."});}
11  render(){
12    lifestates.push(this.state.lifestate);
13    return(<div>
14      <h1><u>Lifeciclye Methods React</u></h1>
15      <ul>{lifestates.map(item=><li>{item}</li>)}</ul>
16    </div>)
17  }
18 }
19 ReactDOM.render(<AppComponent />, document.getElementById("root"));
```

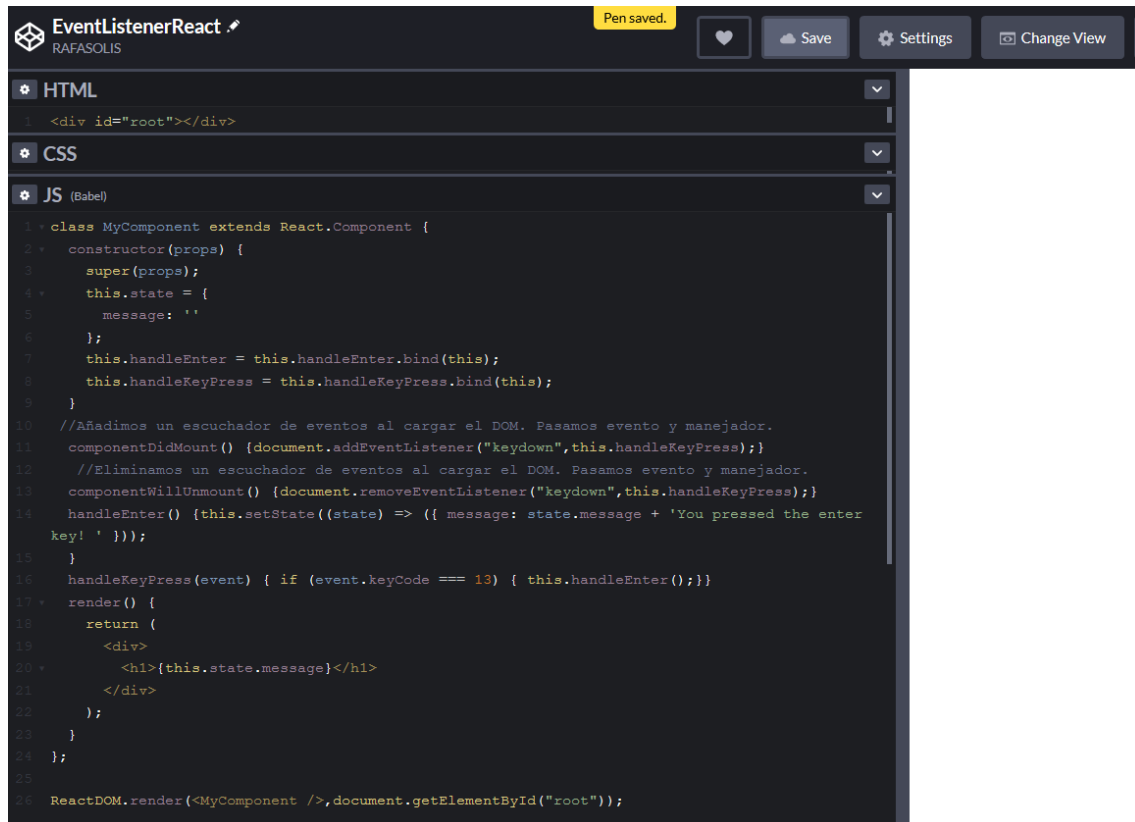
### Lifeciclye Methods React

- ComponentWillMount: antes de montarse en el DOM.
- ComponentDidMount: al actualizar elementos del DOM.

<https://codepen.io/rafasolis/pen/XWdBWRE>

Rafael Solís López

## Ejemplo 13. Incorporación / eliminación de escuchadores de eventos.



The screenshot shows a CodePen editor for a project named 'EventListenerReact' by 'RAFASOLIS'. The editor has tabs for HTML, CSS, and JS (Babel). The JS tab is active, showing a React class component named 'MyComponent'. The component has a state with a 'message' property. It uses 'componentDidMount' to add a 'keydown' event listener and 'componentWillUnmount' to remove it. The 'handleKeyPress' method updates the state when the 'enter' key is pressed. The 'render' method returns a div containing an h1 with the current message. The HTML tab shows a single div with id 'root'. The CSS tab is empty.

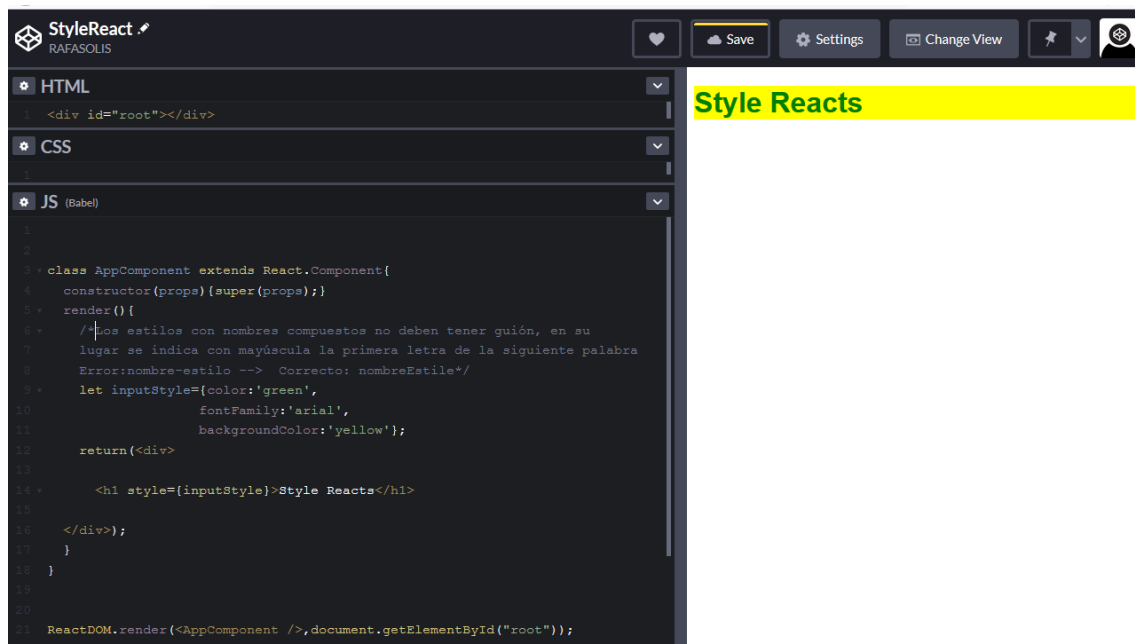
```

1 <div id="root"></div>
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

```

<https://codepen.io/rafasolis/pen/bGpjGXR>

## Ejemplo 14. Incorporación de estilos



The screenshot shows a CodePen editor for a project named 'StyleReact' by 'RAFASOLIS'. The editor has tabs for HTML, CSS, and JS (Babel). The JS tab is active, showing a React class component named 'AppComponent'. The component has a 'render' method that returns a div containing an h1 with inline styles. The styles are defined in a variable 'inputStyle' with values for color, font family, and background color. The HTML tab shows a single div with id 'root'. The CSS tab is empty. A yellow banner with the text 'Style Reacts' is visible on the right side of the editor.

```

1 <div id="root"></div>
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

```

## 14 EJEMPLOS REACT

<https://codepen.io/rafasolis/pen/NWNBPVE>