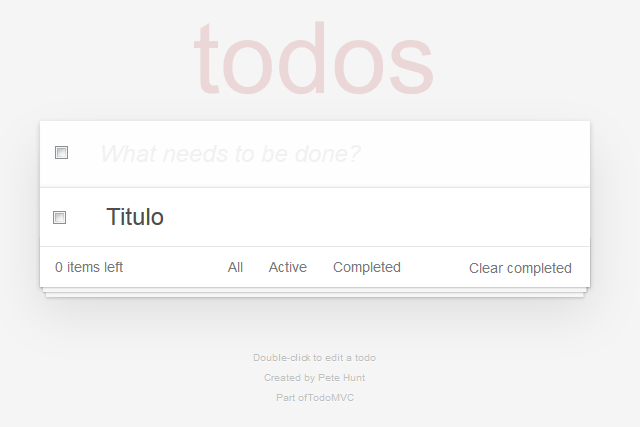
Diseño de la interfaz



Iniciando el proyecto:

Dentro de la carpeta principal de nuestro nuevo proyecto creamos un archivo llamado package.json con es siguiente json:

Archivo: package.json

|  |
| --- |
| {  **"private"**: **true**,  **"dependencies"**: {  **"backbone"**: "^1.1.2",  **"backbone.localstorage"**: "^1.1.7",  **"classnames"**: "^2.1.2",  **"jquery"**: "^2.1.0",  **"react"**: "^0.13.3",  **"todomvc-app-css"**: "^1.0.0",  **"todomvc-common"**: "^1.0.1",  **"underscore"**: "^1.6.0"  }  } |

Sino tenemos instalado nodejs es el momento de instalarlo desde la pagina del proyecto <https://nodejs.org/>

Ahora ya podemos instalar los paquetes que necesitaremos y que están indicados en el package.json, desde la línea de comandos y ubicado en la ruta de tu proyecto ejecuta:

npm install

Cuando el proceso termine te creara una carpeta llamada node\_modules que es donde estaran los paquetes necesarios para iniciar el proyecto.

Aquí esta una breve descripción de los paquetes que acabos de instalar:

|  |  |
| --- | --- |
| Paquetes | Descripcion |
| **backbone** |  |
| **backbone.localstorage** |  |
| **classnames** |  |
| **jquery** |  |
| **react** |  |
| **todomvc-app-css** | Aquí esta la hoja de estilos del proyecto (index.css) |
| **todomvc-common** |  |
| **underscore** |  |

Vamos a crear la base de la interfaz del proyecto, creamos un archivo index.html en la raíz del mismo.

Archivo index.hml:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**React \_ Backbone • TodoMVC**</title>**

**<link** rel="stylesheet" type="text/css" href="node\_modules/todomvc-app-css/index.css"**>**

**</head>**

**<body>**

**<section** id="todoapp"**>**

**<div>**

**<header** id="header"**>**

**<h1>**todos**</h1>**

**<input** ref="newField" id="new-todo" placeholder="What needs to be done?"**/>**

**</header>**

**<section** id="main"**>**

**<input** id="toggle-all" type="checkbox" **/>**

**<ul** id="todo-list"**>**

**<li>**

**<div** class="view"**>**

**<input** class="toggle" type="checkbox" **/>**

**<label>**Tarea 1**</label>**

**<button** class="destroy" **/>**

**</div>**

**<input** ref="editField" class="edit" **/>**

**</li>**

**</ul>**

**</section>**

**<footer** id="footer"**>**

**<span** id="todo-count"**>**

**<strong>**1 **</strong>** item left

**</span>**

**<ul** id="filters"**>**

**<li><a** href="#/"**>**All**</a></li>**

**<span>** **</span>**

**<li><a** href="#/active"**>**Active**</a></li>**

**<span>** **</span>**

**<li><a** href="#/completed"**>** Completed**</a></li>**

**</ul>**

**<button** id="clear-completed"**>** Clear completed **</button>**

**</footer>**

**</div>**

**</section>**

**<footer** id="info"**>**

**<p>**Double-click to edit a todo**</p>**

**<p>**Created by **<a** href="http://github.com/petehunt/"**>**Pete Hunt**</a></p>**

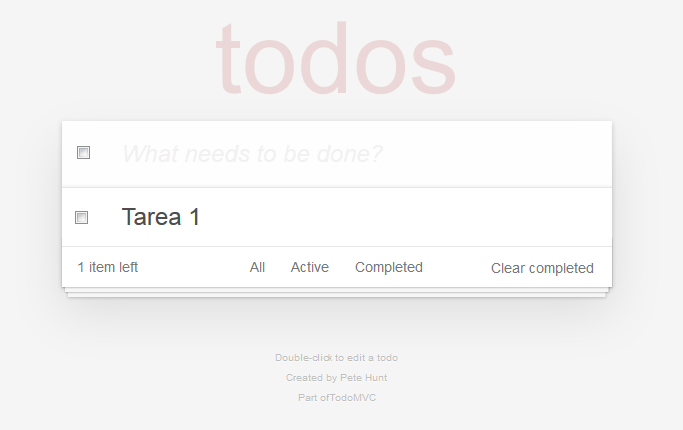
**<p>**Part of**<a** href="http://todomvc.com"**>**TodoMVC**</a></p>**

**</footer>**

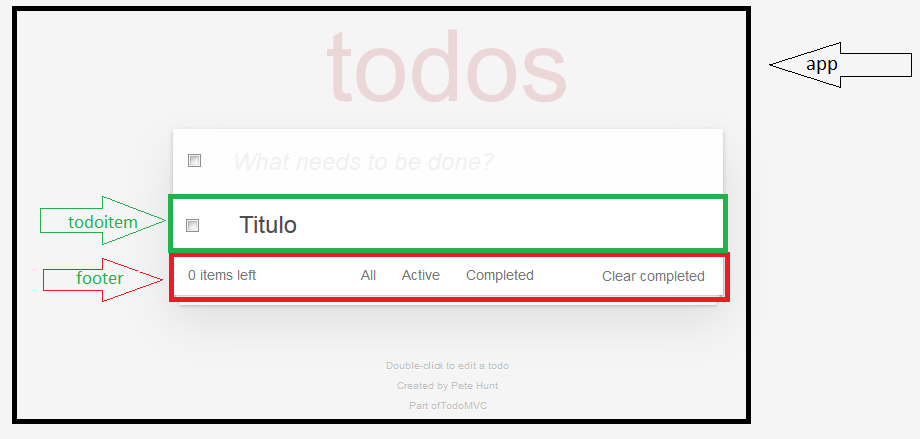
**</body>**

**</html>**

Ya tenemos una plantilla base de como se vera nuestro proyecto y si abres el index.html desde el navegador deberías de ver una pantalla parecida a la siguiente:



La filosofía de reactjs se basa en la división de componentes veamos como podríamos dividir nuestra aplicación



Como puedes ver la aplicación se ha dividido en 3 componentes

Todoitem: Aquí se mostraran las tareas y nos indicara si ya se completo esa tarea.

Footer: Tendra enlaces para el filtrado de tareas y nos indicara cuantas tareas hay sin completar.

App : Este componente tendrá el titulo y aquí se escribirán las nuevas tareas y será el componente padre, aquí estarán todoitem y footer.

Es el momento de ver en que parte del código están nuestros componentes para poder sacarlos de allí ponerlos en un archivo separado.

Solo dividiremos la sección main.



Vamos crear dentro de la raíz del proyecto una carpeta con el nombre js y dentro de esa carpeta creamos los siguiente tres archivos :

|  |  |  |
| --- | --- | --- |
| app.jsx | todoitem.jsx | footer.jsx |
| **var** app = app || {};  (**function** () {  'use strict';  app.TodoApp= React.createClass({  render: **function** () {  **return** (  );  }  });  })(); | **var** app = app || {};  (**function** () {  'use strict';  app.TodoItem = React.createClass({  render: **function** () {  **return** (  );  }  });  })(); | **var** app = app || {};  (**function** () {  'use strict';  app.TodoFooter = React.createClass({  render: **function** () {  **return** (  );  }  });  })(); |

Explicando un poco el código:

Declarando un object literals que es una lista de nombres de propiedades asociadas con su respectivo valor, si la app ya existe utilizamos la instancia ya creada, sino existe usamos una nueva con {}

**var** app = app || {}

Dentro de cada función forzamos a que use el modo estricto que nos forzara a utilizar buenas practicas.

'use strict';

Nuestra app guardara una instancia de cada uno de nuestros 3 componentes

app.TodoApp

app.TodoItem

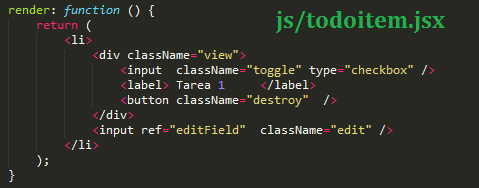
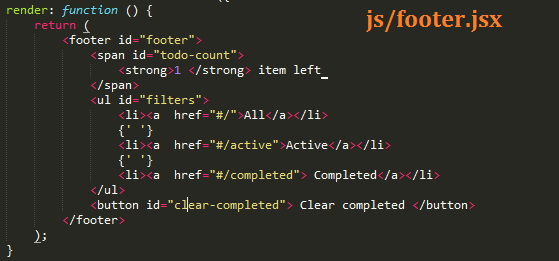
app.TodoFooter

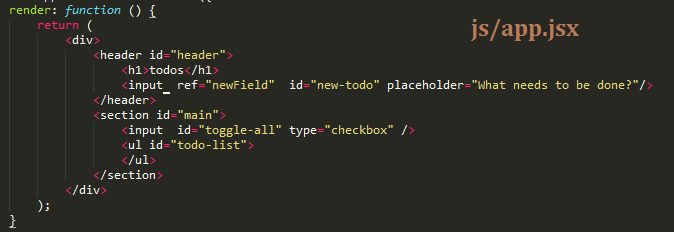
Para crear un component se utliza la funcion **React.createClass**.

Dentro de render estará la descripción de nuestro componentes por medio de JSX el cual es muy parecido a HTML.

El siguiente paso es cortar y pegar cada pedazo de nuestro respectivo html a su respectivo al render de cada componente, comienza en el siguiente orden: todoitem, footer, app.

El único cambio que tenemos que hacer para que sea reconocido como JSX es cambiar la palabra class por className, esto es porque ahora estamos utilizando JSX y no html y class es una palabra reservada.





Si en este momento abrimos nuestro index.html ya no veremos la información de los datos que acabamos de separar, para poder mostrar nuestro componentes en la pagina principal realizaremos los siguientes pasos:

1.- Agregar las referencias a las librerías:

react: para utilizar la librería de react.

JSXTransformer:Transforma nuestro archivos jsx a javascript en el navegador (solo se debe usar en desarrollo, para producción investiga sobre react-tools).

app: Es nuestro componente principal en el vivirán TodoItem y TodoFooter.

Nuestro control principal siempre debe ser el ultimo de la referencia, en este caso es app.jsx

**<link** rel="stylesheet" type="text/css" href="node\_modules/todomvc-app-css/index.css"**>**

**<script** src="node\_modules/react/dist/react.js"**></script>**

**<script** src="node\_modules/react/dist/JSXTransformer.js"**></script>**

**<script** type="text/jsx" src="js/todoitem.jsx"**></script>**

**<script** type="text/jsx" src="js/footer.jsx"**></script>**

**<script** type="text/jsx" src="js/app.jsx"**></script>**

2. Indicamos en el archivo app.jsx cual es el componente que se cargara y en que lugar del index.html se mostrara.



Si abrimos el index solo nos mostrara datos del componente TodoApp.

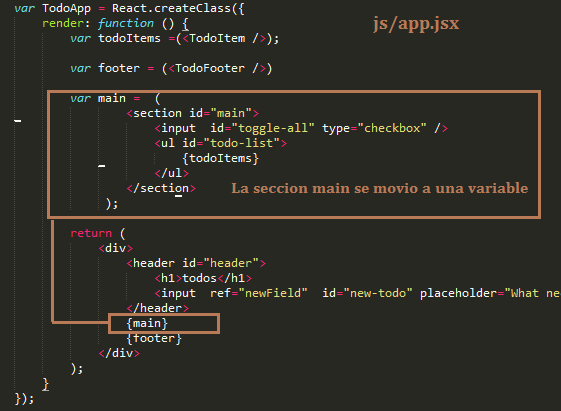
**Nota: Si usas chrome posiblemente marcara errores de CORS, como no es una aplicación de producción podemos correr chrome deshabilitando la seguridad: chrome --args --disable-web-security**

Para poder mostrar los componente TodoItem y TodoFooter necesitamos incluirlos dentro del return del componente padre que es app, el componente se puede declarar de la forma <NombreComponente /> o como en este ejemplo en el que guardamos el componente en una variable, y para que react sepa que es un código javascript ponemos la variable entre crochetes { }



Abrimos nuestro index.html en el navegador y ya tenemos el mismo diseño inicial solo que ahora separado con componentes de react .js

Para terminar con esta primera parte y para tener mejor organizado el codigo vamos a mover el section main a una variable:



Con esto terminamos la primera parte, es algo muy básico en las siguientes entregas se estáran viendo las ventajas de usar react.

El código lo puedes consultar en <https://github.com/rulotr/todomvc_backbone_react/tree/%281%29Dise%C3%B1oInterfaz/>

**PARTE 2 MAS TAREAS**

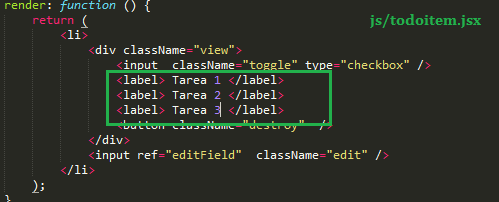
Partiendo del código de la primera parte <https://github.com/rulotr/todomvc_backbone_react/tree/%281%29Dise%C3%B1oInterfaz/>

Vamos a agregar las siguientes funcionalidades a nuestro ToDo:

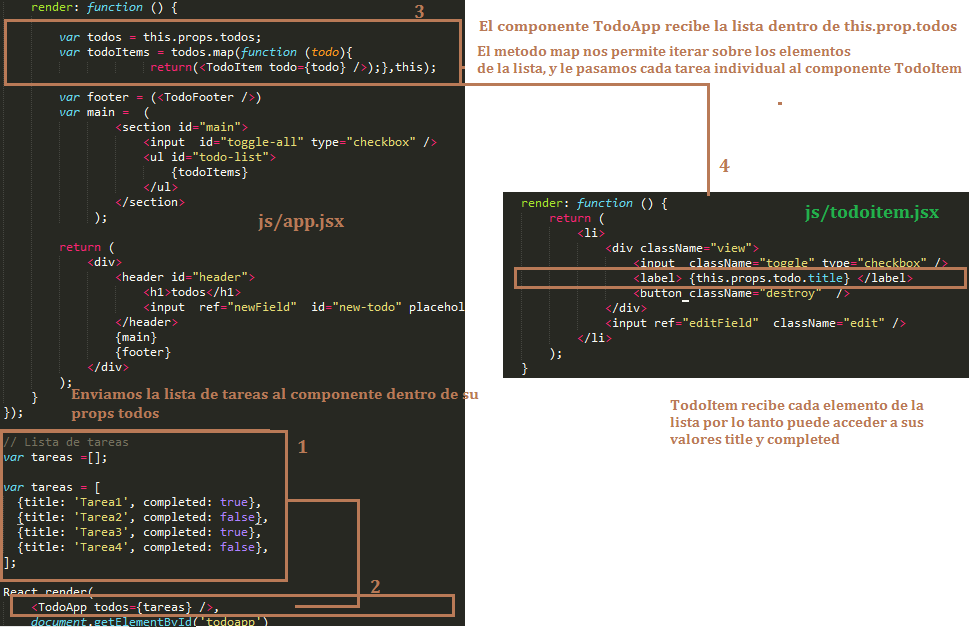
* Mostrar una lista de Tareas.
* Marcar las Tareas realizadas.
* Contar el numero de Tareas pendientes de realizar.

**Mostrar una lista de Tareas.**

En este momento si quisiéramos mostrar mas tareas tendríamos que agregarlas en nuestro componente TodoItem



Una mejor forma es pasarle al componente una lista de tareas y el componente de react obtendra los valores atravez de lo que se llama props, además esta lista también indicara que tarea se completo, estos son los cambios que tenemos que hacer:

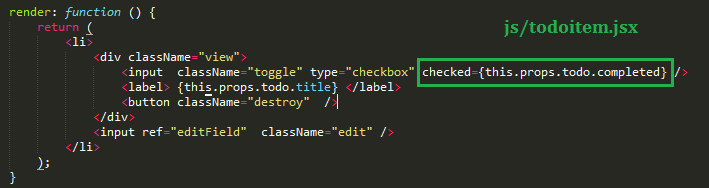


Si abrimos nuestro index.html podremos ver las 4 tareas.

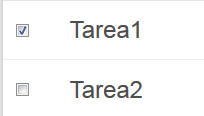


**Marcar tareas realizadas.**

En la pagina ya estamos mostrando las tareas, ahora necesitamos marcar que tareas ya fueron completadas, nuestra lista de javascript tareas contiene esa información en el campo completed, estos son los cambios que se necesitan para marcar las tareas realizadas:

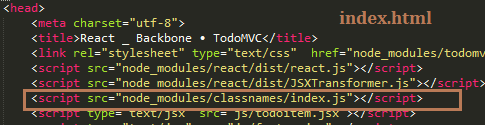


En la pagina ya se marcaran las tareas completadas

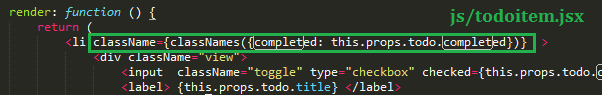


Vamos a usar el css del proyecto para mostrar mejor que tareas se completaron, el css se encuentra en todomvc-app-css/index.css , no vamos a usar directamente el css sino que usaremos una función que también es parte del proyecto esta se encuentra en todomvc-app-css/classnames/index.js , si la función recibe un valor true asignara el css si es false no tendrá css.

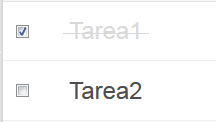
Agregamos la referencia a la función que devolverá el css:



Si la tarea de nuestro componente tiene true en su completed entoces se asignara el estilo del css que marca la tarea.

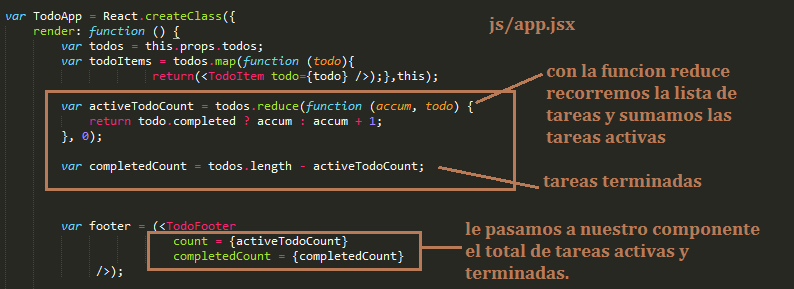


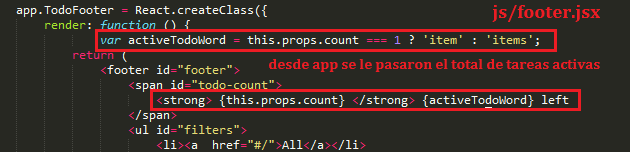
Y nuestro index.html queda asi:



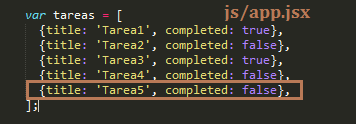
**Contar el numero de Tareas pendientes de realizar.**

Desde el app.jsx le mandamos el total de tareas al componente TodoFooter y este puede acceder a esos valores por medio de sus props.





Vamos a agregar una tarea sin complentar a nuestra lista.



Abrimos nuestra pagina index.html y ya nos indica que hay 3 tareas sin completar.

