MARCO TOMASELLO

FREE
WITH THIS
COURSE

CHAPTER 1

Learn Redux

In this chapter you will learn how to manage a web application state using Redux as standalone tool

SECTION 1

FIRST REDUX APP

SET-UP DEPENDENCIES AND WEBPACK

1. Open terminal and run the following commands:
   - **$ mkdir reduxApp**
   - **$ cd reduxApp**
   - **$ npm init -y**
2. If you never installed Webpack, run the command: **npm install webpack -g** (to install webpack globally)
3. Install the following dependences:
   - **$ npm install --save babel-core@6.21.0 babel-loader@6.2.10 babel-preset-react@6.23.0 babel-preset-es2015@6.18.0 babel-preset-stage-1@6.16.0 redux express**
   - **$ npm install --save-dev webpack**
4. Create a Webpack config
   - **$ touch webpack.config.js**
   - **$ Atom .** (if you are using Atom editor)
   - **open webpack.config.js file**

webpack.config:

```
var path = require('path');

const webpack = require('webpack');

module.exports  = {
  entry: './src/app.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'public')
  },
  watch: true,
  module:{
    loaders: [
      {
        test:/\.js$/,
        exclude:/node_modules/,
        loader: 'babel-loader',
        query: {
          presets: ['react', 'es2015',
'stage-1']
        }
      }
    ]
  }
}
//-----------------------------------------------------------------
```

entry**:** tells Webpack which is the entry point of the application

output**:** tells Webpack where to output the bundle.js file

watch**:** when true, tells to Webpack to create a new bundle.js every time users save a change. This will work if you run the command webpack

loaders**:** tells Webpack what compiler to use

loader**:** tells Webpack to use Babel as compiler

query: tells Webpack to use Babel compiler to transform your ES6/es2015 and ES6/stage-1 javascript version into a javascript version compatible with the browsers. React tells webpack to compile jsx into javascript

```
<!DOCTYPE>
<html>
<head>
  <title>Hello redux</title>
<meta name="viewport"
content="width=device-width,
initial-scale=1.0 maximum-scale=1.0" />
</head>
<body>
  <div id="app"></div>
  <script src="bundle.js"></script>
  <h1>Hello first Redux App</h1>
</body>
</html>
//----------------------------------------
```

INDEX.HTML

6. Create an html file where to render the app:
   - **make a folder named: public**
   - **create a index.html file inside the public folder**
   - **create a file: index.html**

index.html:

SET-UP WEB-SERVER

7. Create a basic Express web-server:
   - **create server.js file inside the main directory (ReduxApp/server.js)**
   - **write the below code and save server.js**

server.js:

```
'use strict'
var express = require('express');
var app = express();
```

```
var path = require('path');

// DEFINES A FOLDER FOR THE STATIC FILES
app.use(express.static('public'));

// DEFINES THE MAIN ENTRY POINT
app.get('/', function(req, res){
  res.sendFile(path.resolve(__dirname,
'public', 'index.html'))
});

app.listen(3000, function(){
  console.log('App web-server listening on
port 3000');
});
//----------------------------------------
```

TEST THE WEB-SERVER

8. Make sure Express runs correctly on port 3000:
   - $ **node server.js**
   - **go to http://localhost:3000**

CREATE A COUNTER APP WITH REDUX

9. Create Redux source file:
   - **make a folder named: src**
   - **make a file app.js inside src folder**
   - **copy the following code and save app.js**

app.js:

```
"use strict"
import {createStore} from 'redux';

//STEP 3 define reducers
const reducer = function(state={}, action){
  switch(action.type){
    case "POST_BOOK":
```

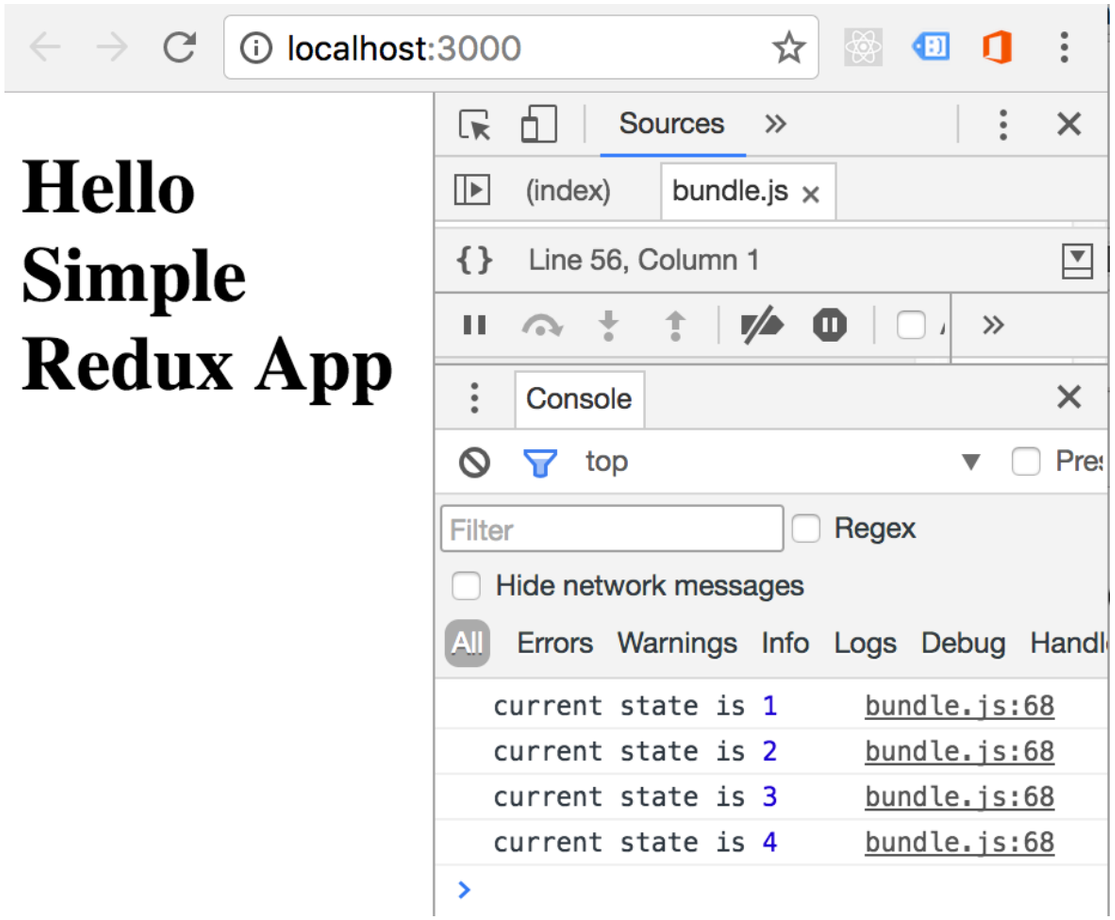
```
    return state = action.payload;
    break;
  }
  return state
}

// STEP 1 create the store
const store = createStore(reducer);

store.subscribe(function(){
  console.log('current state is: ',
store.getState());
  console.log('current price: ',
store.getState().price);
})

// STEP 2 create and dispatch actions
store.dispatch({
  type:"POST_BOOK",
  payload: {
    id: 1,
    title:'this is the book title',
    description: 'this is the book
description',
    price: 33.33
  }
})
//-----------------------------------------
```

type**:** is a Redux keyword and therefore you cannot change them, but you can replace the word payload if you wish.

payload: can be named as you wish and it is optional

TEST THE REDUX COUNTER APP

10. compile the code and launch the app :
    - $ **webpack (run this command in an other terminal window)**
    - $ **node server.js**
    - **go to http://localhost:3000**
    - **open chrome developer tool to observe how redux changes the application state.**

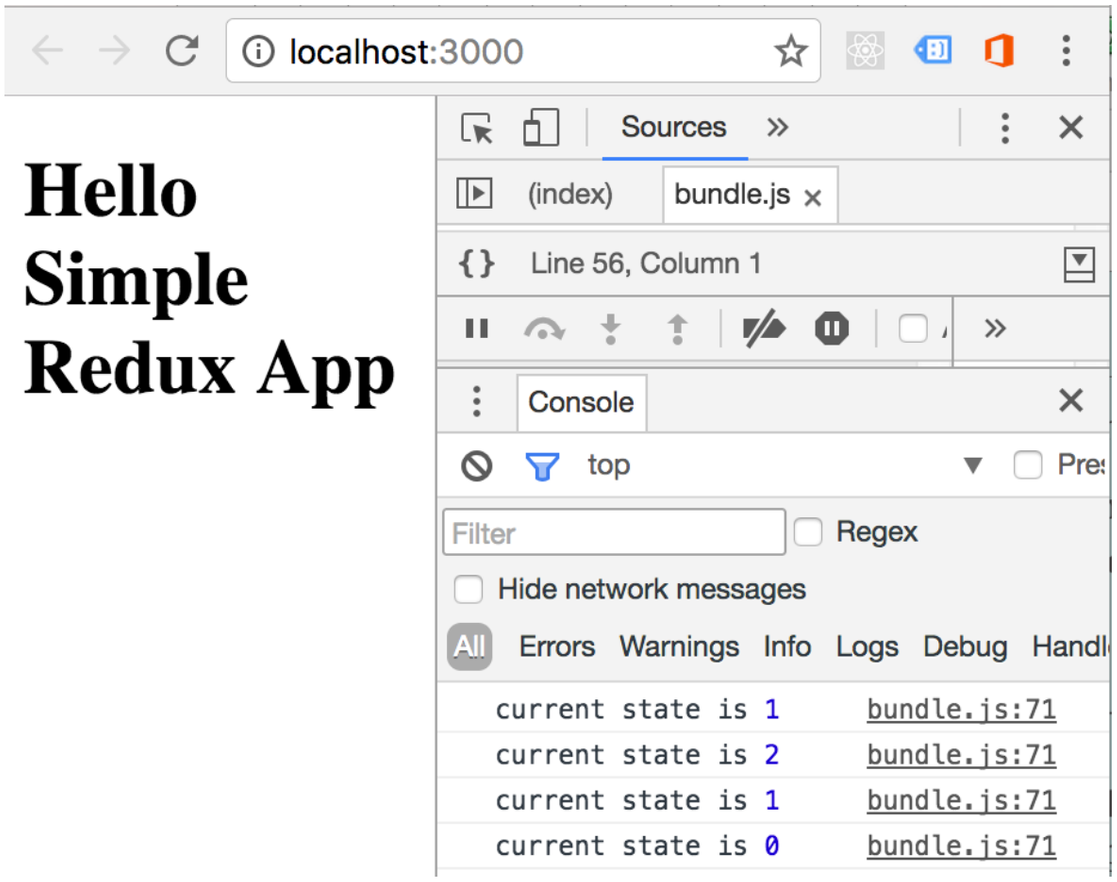


ADD 'DECREMENT' ACTIONS IN THE APP

11. edit app.js:

- **dispatch two new actions as in the example below**
- **modify the reducer as in the example below**

```
'use strict'
import {createStore} from 'redux';

// STEP 3 define reducers
const reducer = function(state=0, action){
  switch (action.type) {
    case "INCREMENT":
    return state + action.payload
      break;
    case "DECREMENT":
    return state - 1;
      break;
  }
  return state
}
// STEP 1 Create a store and subscribe for
any changes
const store = createStore(reducer);

store.subscribe(function(){
  console.log('current state is: ' +
store.getState());
})
// STEP 2 create and dispatch actions
store.dispatch({type:"INCREMENT",
payload:1});
store.dispatch({type:"INCREMENT",
payload:1});
store.dispatch({type:"DECREMENT"});
store.dispatch({type:"DECREMENT"});
```



PASSING OBJECTS TO THE STATE

In this example we will now pass an object to the reducer:

PASS AN OBJECT TO THE STATE

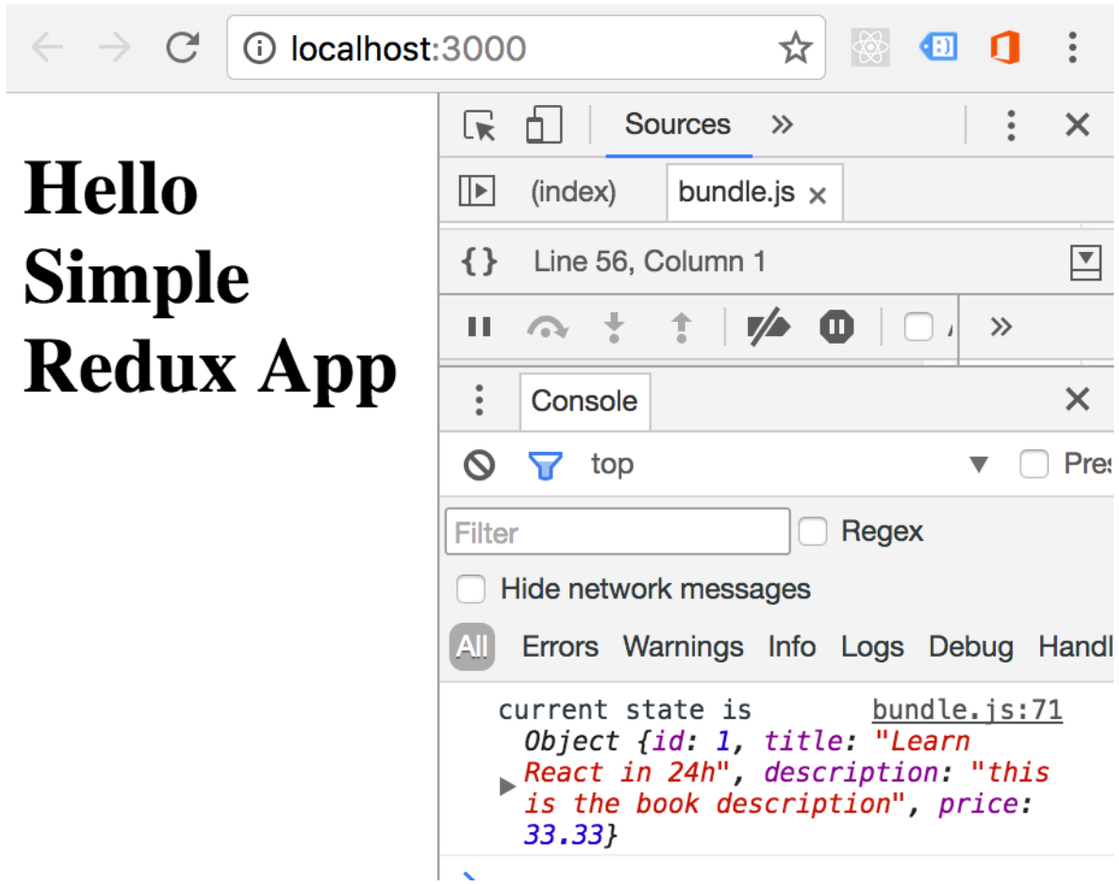
12. edit app.js:

- **remove of all actions and make a new one called: "POST_BOOK" that passes a book object:** *{id: 1, title: "Learn React in 24h", description: "this is the book description", price: 33.33}*
- **remove all reducers and write a new one for the "POST_BOOK" action type**
- **in the reducer, set the initial state to an empty object:** *state={}*

```
"use strict"
import {createStore} from 'redux';

//STEP 3 define reducers
const reducer = function(state={}, action){
  switch(action.type){
    case "POST_BOOK":
    return state = action.payload;
    break;
  }
  return state
}

// STEP 1 create the store
```

```
const store = createStore(reducer);

store.subscribe(function(){
console.log('current state is: ',
store.getState());
})

// STEP 2 create and dispatch actions
store.dispatch({
  type:"POST_BOOK",
  payload: {
    id: 1,
    title:'this is the book title',
    description: 'this is the book
description',
    price: 33.33
  }
})

//----------------------------------------
```

When you run this example, you will see that the state is now capturing the entire object (if you want to see all object properties in the log, remember to replace "+", with the comma ",", otherwise the output of the log will be stringified):



Now, you can access the object and each of his properties.

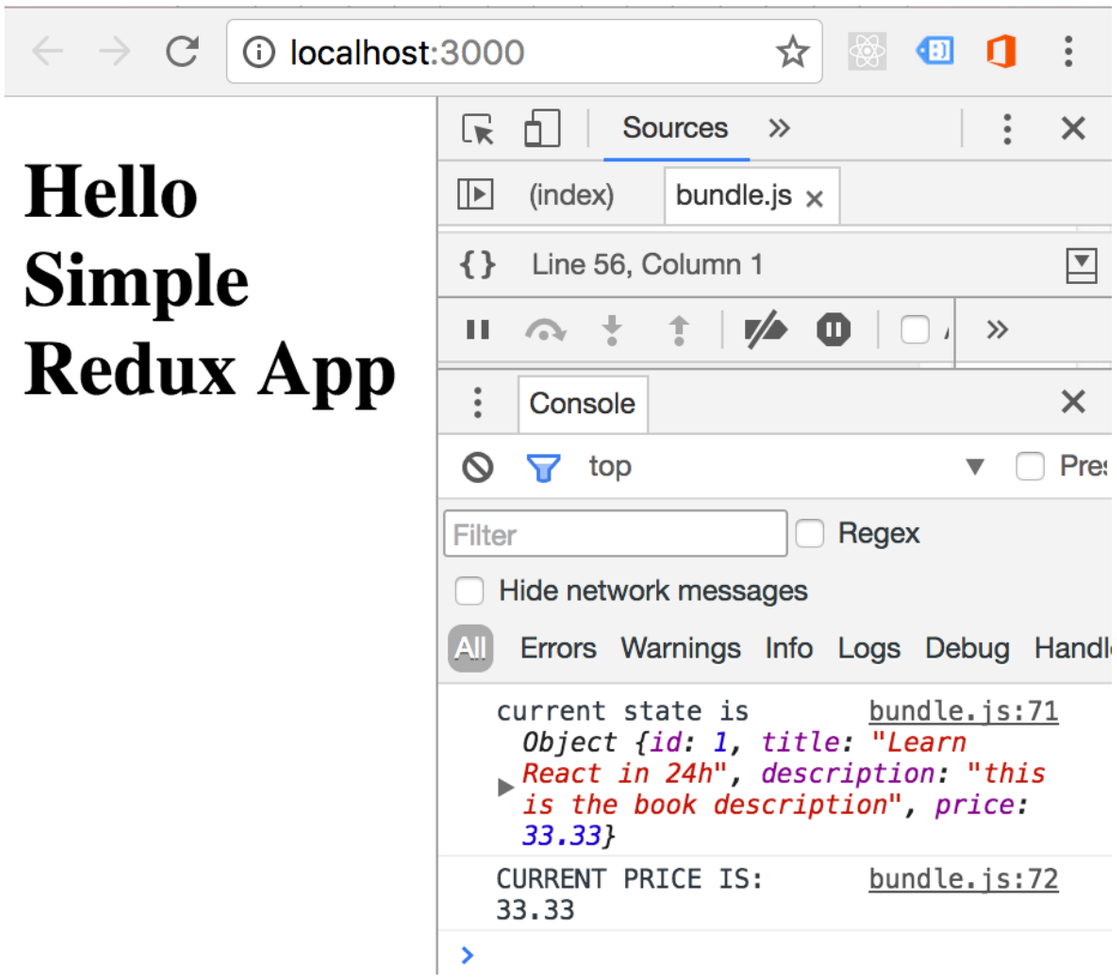
ACCESS ONE PROPERTY IN THE STATE

12. edit app.js:

- **add a log inside the subscribe function to see how "price" property is recorded in the state:** *console.log('CURRENT PRICE IS: ' + store.getState().price);*

```
18  // SUBSCRIBE some tasks to the store
19  store.subscribe(function() {
20    console.log("current state is", store.getState());
21    console.log('CURRENT PRICE IS: ' + store.getState().price);
22  })
```

and if you run the app, you should see that any object property can be easily accessed from the state:



At this point You are ready to make a little more complex operations with Redux.

PASS AN ARRAY OF OBJECTS TO THE STATE

A common scenario in Redux is to pass array of objects to the state (i.e. data from an API)

13. edit app.js:

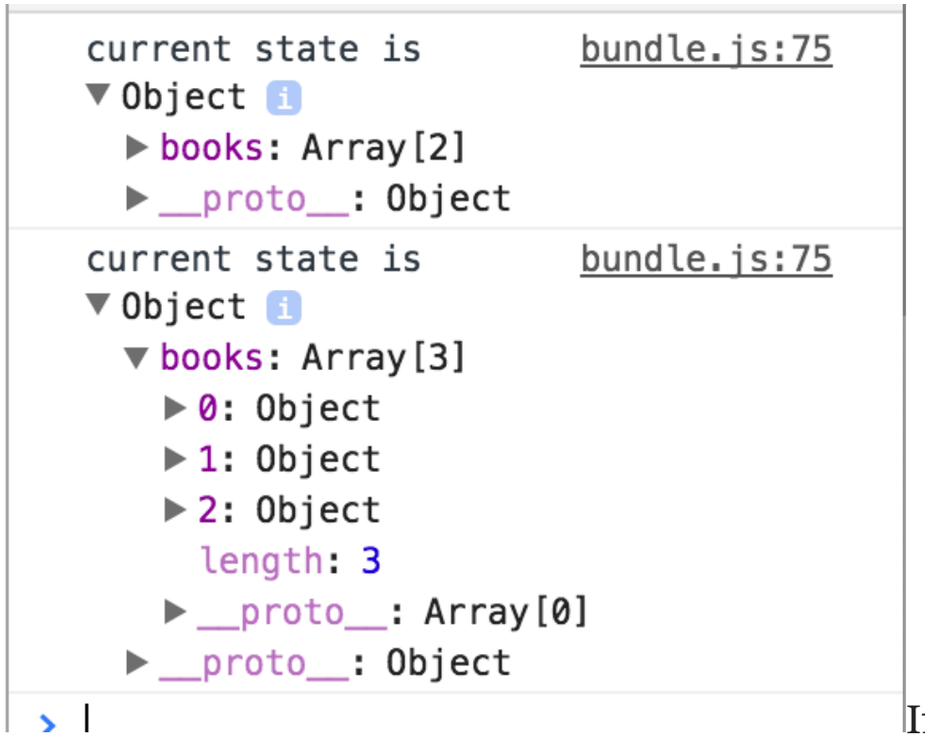
- **in the reducer, set the initial state equal to an object containing an array of books object:** *state={ books:[] }*
- **add an other book in POST_BOOK**
- **Since you will post the books as an array, we will wrap them inside an array**
- **To see how the store changes across multiple cycles as it would do in a real scenario, you will also dispatch a third book right after the previous dispatch.**
- ***Finally, I will rewrite our reducer as follow:***

  let books = state.books.concat(action.payload);

  return {books}

```
"use strict"
import {createStore} from 'redux';

//STEP 3 define reducers
```

```
const reducer = function(state=[], action){
  switch(action.type){
    case "POST_BOOK":
    let books = state.concat(action.payload)
    return books;
    break;
  }
  return state
}

// STEP 1 create the store
const store = createStore(reducer);

store.subscribe(function(){
  console.log('current state is: ',
store.getState());
  //console.log('current price: ',
store.getState()[1].price);
})

// STEP 2 create and dispatch actions
store.dispatch({
  type:"POST_BOOK",
  payload: [{
    id: 1,
    title:'this is the book title',
    description: 'this is the book
description',
    price: 33.33
  },
  {
    id: 2,
    title:'this is the second book title',
    description: 'this is the second book
description',
    price: 50
  }]
})

// DISPATCH an other book
store.dispatch({
  type:"POST_BOOK",
  payload: {
    id: 3,
    title:'this is the third book title',
    description: 'this is the third book
description',
    price: 40
  }
})

//----------------------------------------
```



If you look at the result you see that the state was updated correctly.

More often people uses the spread operator in place of the concat function or in place of “object.assign” in case of operations with object rather than arrays. The spread operator is more concise but in order to use it, it is required babel-preset-stage-1 (which we already installed and included the preset in webpack config). Here how it looks using the spread operator:

```
//STEP 3 define reducers
const reducer = function(state={books:[]},
action){
  switch(action.type){
    case "POST_BOOK":
    // let books =
state.books.concat(action.payload);
    // return {books};
    return {books:[...state.books,
...action.payload]}
    break;
  }
  return state
}

//IMPORTANT: remember to update your second
dispatch to be an array as below
// DISPATCH an other book
store.dispatch({
  type:"POST_BOOK",
  payload: [{
    id: 3,
    title:'this is the third book title',
    description: 'this is the third book
description',
    price: 40
  }]
})
//----------------------------------------
```

Now, before we move to more complex example, there is something critical we need to know.

IMMUTABLE STATE

In the last video we said that our reducers must not mutate the state. In order to understand what ***"mutate the state"*** means, we firstly need to know what a pure function is.

Redux, indeed, relies on "pure functions" for preventing state mutation. Ok... so...what pure functions are and why do we need them in Redux?

A **pure function** is a function that, ***given the same input, will always return the same output and does not have any observable side effects.***

look at this example...:

app.js | pure_function.js

```
// PURE
//Given the same input, will always return the same output
function checkAge(age) {
  let minimum = 18;
  return age >= minimum;
};

// IMPURE
//given the same input, will NOT necessary return the same output
let minimum = 18;

function checkAge(age) {
  return age >= minimum;
};
```

The first one is a pure function. Given the same input, for instance "16", the function will always return the same output which is "FALSE" in the case of 16.

The second function, although looks almost the same, it is an i**n-pure function**, because the returned result will change based on the value of minimum age variable that is defined outside the scope of the function!

I want you to think what the problem is... From the application state will depends what data will be returned to the users and therefore how your UI will behave.

You want to have your state to be "predictable" and so "reliable". You don't want your state mutating accidentally overtime by any part of your code.

For instance, if you build a shopping cart, **you want to be confident** that your app will always display precisely the users requested quantity of products in the basket, not one unit more or less.

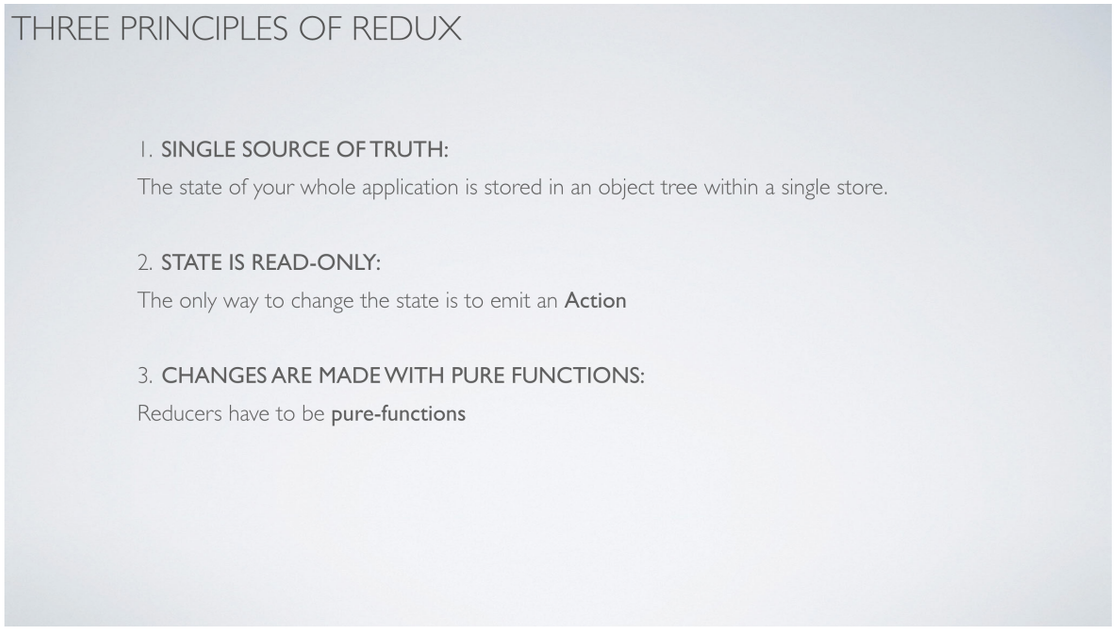
You want to be confident, that every time a user "**input**" one unit of product, your basket adds exactly that one unit.

What would happen if, somewhere in your code, you have a function or a variable that at certain conditions alter the result of your basket? You will get a massive bug, that sometimes could be even hard to spot or root-cause.

So... how would you mitigate such risks to mutate your application state? A good solution is to use pure functions,

because ***iven the same input, will*** **always get a predictable output.**

Actually, this is the way how Redux works and the reason why reducers have to be pure function!



Redux is based on three principles:

1. SINGLE SOURCE OF TRUTH:

The state of your whole application is stored in **an object tree within a single store**. So you will have a single state object that for instance contains: a books array, an other array for categories, a boolean value or a string or an integer..all together under on object state.

2. STATE IS READ-ONLY:

The only way to change the state is to **emit an Action**

3. CHANGES ARE MADE WITH PURE FUNCTIONS:

Reducers have to be pure-functions

Finally, if you STILL don't feel sure or comfortable how you should approach Redux to prevent state mutation, don't worry, later in the course we will write many different operations and we will consider many different scenarios. But in principle, when making operations with Arrays, you should use **concat**, **slice** or the **spread operators**.

NEVER USE PUSH or SPLICE!

Instead, when making operations with objects, you can use **Object.Assign** or again the **spread**.

CRUD OPERATIONS WITH REDUX

Ok, now you are ready to deal with CRUD operations in Redux (Create, Read, Update, Delete).

You already saw how to post new data, let's look now how to delete an item in an array.

DELETE

14. edit app.js:

- **Remove the second POST_BOOK action**
- **Add a new action called: DELETE_BOOK:** *store.dispatch({ type: "DELETE_BOOK", payload: { id: 1 } })*
- **Add a new reducer for the actionType DELETE_BOOK as in the code below.**

this is how the Action looks like:

```
store.dispatch({
  type: "DELETE_BOOK",
  payload: {id: 1}
})
```

this is how the Reducers will looks like:

```
"use strict"
import {createStore} from 'redux';

//STEP 3 define reducers
const reducer = function(state={books:[]},
action){
  switch(action.type){
    case "POST_BOOK":
    // let books =
state.books.concat(action.payload);
    // return {books};
    return {books:[...state.books,
...action.payload]}
    break;
     case "DELETE_BOOK":
    // Create a copy of the current array of
books
    const currentBookToDelete =
[...state.books]
    // Determine at which index in books
array is the book to be deleted
    const indexToDelete =
currentBookToDelete.findIndex(
      function(book){
        return book.id === action.payload.id;
      }
    )
    //use slice to remove the book at the
specified index
    return {books:
[...currentBookToDelete.slice(0,
indexToDelete),
...currentBookToDelete.slice(indexToDelete +
1)]}
    break;
  }
  return state
}
```

```
// STEP 1 create the store
const store = createStore(reducer);

store.subscribe(function(){
  console.log('current state is: ',
store.getState());
  //console.log('current price: ',
store.getState()[1].price);
})

// STEP 2 create and dispatch actions
store.dispatch({
  type:"POST_BOOK",
  payload: [{
    id: 1,
    title:'this is the book title',
    description: 'this is the book
description',
    price: 33.33
  },
  {
    id: 2,
    title:'this is the second book title',
    description: 'this is the second book
description',
    price: 50
  }]
})

// DELETE a book
store.dispatch({
  type:"DELETE_BOOK",
  payload: { id: 1}
})

//----------------------------------------
```

Inspecting our Chrome console, you should see that the state reports only the id=2 book after we launched the Delete action:

```
current state is        bundle.js:89
▶ Object {books: Array[2]}
current state is        bundle.js:89
▼ Object i
  ▼ books: Array[1]
    ▼ 0: Object
        description: "this is the book
        id: 2
        price: 45
        title: "Redux in 24h"
      ▶ __proto__: Object
      length: 1
    ▶ __proto__: Array[0]
  ▶ __proto__: Object
```

The findIndex() method returns an index of the first element in the array that satisfies the provided testing function. Otherwise -1 is returned.

arr.findIndex(callback[, thisArg])

if it is not clear how it works.. could have written like this:

```
25        function whereIsMyBook(book) {
26          return book.id === action.payload.id
27        }
28
29        currentBooktoDelete.findIndex(whereIsMyBook)
30
```

and if it is not yet, clear… try to visualize this method in this way:

```
25        function whereIsMyBook(book) {
26          return book.id === action.payload.id
27        }
28
29        [1, 2].findIndex(whereIsMyBook)
30
```

where, 1 and 2 are the current “id’s” in our books array.

the “book” argument is just how I decided to call the elements in the array, but I could even have written like that and it would work:

```
25        function whereIsMyBook(testBook) {
26          return testBook.id === action.payload.id
27        }
28
29        [1, 2].findIndex(whereIsMyBook)
30
```

UPDATE

EXAMPLE 08

We will update the book with id = 2

UPDATE BOOKS

15. edit app.js:

- **Add a new action called: UPDATE_BOOK**
- **Add a new reducer for the actionType UPDATE_BOOK as in the cade below.**

this is how the Action looks like:

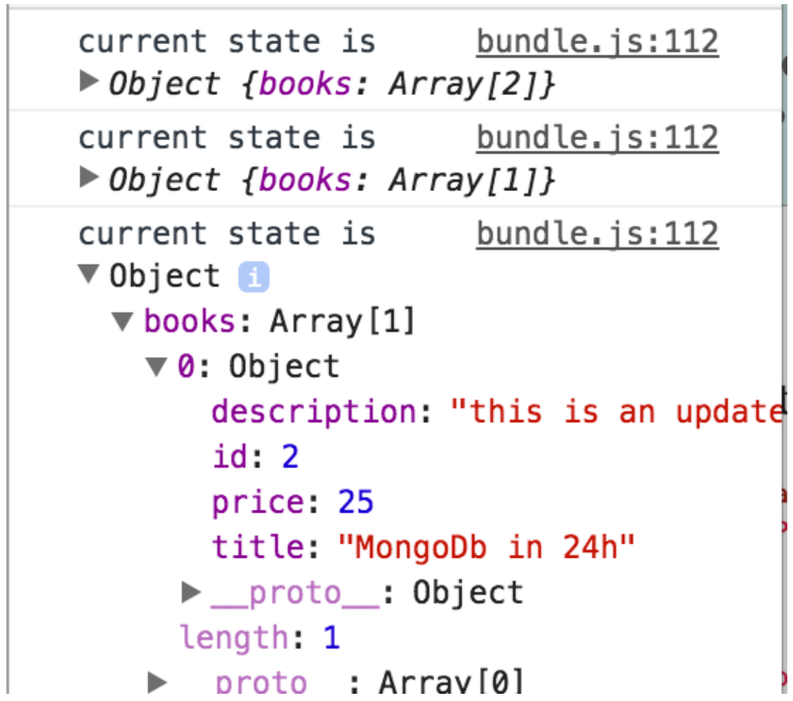
```
// UPDATE a book
store.dispatch({
  type:"UPDATE_BOOK",
  payload:{
    id: 2,
    title:'Learn React in 24h'
  }
})
//-----------------------------------------
```

this is how the Reducers will looks like:

```
case "UPDATE_BOOK":
```

```
    // Create a copy of the current array of
books
    const currentBookToUpdate =
[...state.books]
    // Determine at which index in books
array is the book to be deleted
    const indexToUpdate =
currentBookToUpdate.findIndex(
      function(book){
        return book.id === action.payload.id;
      }
    )
    // Create a new book object with the new
values and with the same array index of the
item we want to replace. To achieve this we
will use ...spread but we could use concat
methos too
    const newBookToUpdate = {
      ...currentBookToUpdate[indexToUpdate],
      title: action.payload.title
    }
    // This Log has the purpose to show you
how newBookToUpdate looks like
    console.log("what is it newBookToUpdate",
newBookToUpdate);
    //use slice to remove the book at the
specified index, replace with the new object
and concatenate witht he rest of items in the
array
    return {books:
[...currentBookToUpdate.slice(0,
indexToUpdate), newBookToUpdate,
...currentBookToUpdate.slice(indexToUpdate +
1)]}
    break;

//----------------------------------------
```

and the result:



Finally, in this example we updated only the title property, but maybe it is worth mentioning that you can use the same pattern when you want to replace an entire item.

COMBINE-REDUCERS

In this course, we will be creating a small shopping-cart, so we need actions and reducer for adding, removing or editing products in the basket. .

So far, we wrote everything in one file and I think it was the best way for you to understand Redux by visualizing the entire Redux process and parts in one sight. But now, as you can see, even with few actions and reducers our code is becoming quite verbose and if we add other actions and reducers for the cart, it will be hard to maintain the code

A good approach is to separate all your reducers and actions in different files and use a Redux method called "combineReducers" to get all of them inside a single state object.

I **warn** you that we will now go through the tedious and boring process of splitting our code in multiple files and therefore it's easy to make mistakes during the process! So try to be very careful when writing your code and linking all modules. Fortunately, if you ever get lost, you not only have this video but also the script in attachment to help you on following all necessary steps correctly.

SPLITTING THE CODE

16. in the projects:

- Create two folders, one named "***actions***" and the other "***reducers***".
- In reducers folder create the following files: "***booksReducers.js***" , "***cartReducers.js***" and "***index.js***".
-

in index.js:

- **Import "combineReducers" from Redux**
- ***import {booksReducers} and {cartReducers};***
- **export default the two combine reducers**
- **Combine the two reducers using the following code:** *const reducers = combineReducers({books: bookReducers, cart: cartReducers})*
- ***Remember to replace "reducer" with "reducerS" in createStore method***
- ***Add an action called ADD_TO_CART***

SPLITTING THE CODE

in booksReducers.js:

- **in App.js Rename the book reducer as:** *bookReducers and* ***Cut*** *the reducer* ***and Paste it*** *inside booksReducers.js and make the function available from outside the module by writing:* ***"export function booksReducers()...."***

index.js:

```
"use strict"
import {combineReducers} from 'redux';

// HERE IMPORT REDUCERS TO BE COMBINED
import {booksReducers} from
'./booksReducers';
import {cartReducers} from './cartReducers';

//HERE COMBINE THE REDUCERS
export default combineReducers({
  books: booksReducers,
  cart: cartReducers
})

//----------------------------------------
```

booksReducers.js:

```
"use strict"
//BOOKS REDUCERS
export function booksReducers
(state={books:[]}, action){
  switch(action.type){
    case "POST_BOOK":
    // let books =
state.books.concat(action.payload);
    // return {books};
    return {books:[...state.books,
...action.payload]}
    break;
    case "DELETE_BOOK":
    // Create a copy of the current array of
books
    const currentBookToDelete =
[...state.books]
    // Determine at which index in books
array is the book to be deleted
    const indexToDelete =
currentBookToDelete.findIndex(
      function(book){
        return book.id === action.payload.id;
      }
    )
```

```
    //use slice to remove the book at the
specified index
    return {books:
[...currentBookToDelete.slice(0, indexToDelete),
...currentBookToDelete.slice(indexToDelete +
1)]}
    break;

    case "UPDATE_BOOK":
    // Create a copy of the current array of
books
    const currentBookToUpdate = [...state.books]
    // Determine at which index in books array
is the book to be deleted
    const indexToUpdate =
currentBookToUpdate.findIndex(
      function(book){
        return book.id === action.payload.id;
      }
    )
    // Create a new book object with the new
values and with the same array index of the item
we want to replace. To achieve this we will use
...spread but we could use concat methos too
    const newBookToUpdate = {
      ...currentBookToUpdate[indexToUpdate],
      title: action.payload.title
    }
    // This Log has the purpose to show you how
newBookToUpdate looks like
    console.log("what is it newBookToUpdate",
newBookToUpdate);
```

```
    //use slice to remove the book at the
specified index, replace with the new object and
concatenate witht he rest of items in the array
    return {books:
[...currentBookToUpdate.slice(0, indexToUpdate),
newBookToUpdate,
...currentBookToUpdate.slice(indexToUpdate +
1)]}
    break;
  }
  return state
}

//----------------------------------------
```

SPLITTING THE CODE

in app.js:

- **import the combinedReducers from reducers/index as following:**

import reducers from ‘./reducers/index

Save all files and refresh to make sure Redux is working as before.

At this point we can add actions and reducers for our cart too.

CREATE ACTION FOR CART

in app.js :

- **dispatch an "ADD_TO_CART" action to add a product to the cart**

app.js:

```
//-->> CART ACTIONS <<--
// ADD TO CART
store.dispatch({
  type:"ADD_TO_CART",
  payload: {id: 2}
})

//----------------------------------------
```

CREATE CART REDUCER

in cartReducers.js :

- **write an ADD_TO_CART reducer**

cartReducers.js:

```
"use strict"
//CART REDUCERS
export function cartReducers(state={cart:[]},
action){

  switch(action.type){
    case "ADD_TO_CART":
    return {cart:[...state.cart,
...action.payload]}
    break;
  }
  return state
}
//----------------------------------------
```

ADD CART REDUCERS TO COMBINE-REDUCERS

in reducers/index.js.js :

- **import cartReducers.js and combine cart with books**

index.js:

```
"use strict"
import {combineReducers} from 'redux';

// HERE IMPORT REDUCERS TO BE COMBINED
```

```
import {booksReducers} from
'./booksReducers';
import {cartReducers} from './cartReducers';

//HERE COMBINE THE REDUCERS
export default combineReducers({
  books: booksReducers,
  cart: cartReducers
})

//----------------------------------------
```

If save and refresh, you should see now that we have a cart array visible in our state:

```
current state is:  ▶Object {books: Object, cart: Object}
current state is:  ▼Object i
                     ▶books: Object
                     ▼cart: Object
                       ▼cart: Array[1]
                         ▼0: Object
                           id: 2
                         ▶ proto : Object
```

```
current state is    bundle.js:128
▼Object i
  ▶books: Object
  ▶categories: Object
  ▶__proto__: Object
```

You should also have noticed that after combining the reducers our state object not only contains cart, but in order to access for instance to books array, we firstly need to access the books object.

You probably have noticed that also our actions became quite verbose. In a real application we need to keep them in order, so we will split the actions as well in two separate files and we will fire our dispatch function from app.js

SPLIT THE ACTIONS

inside action folder:

- **Add two files: booksActions.js and cartActions.js**

booksActions.js:

```
"use strict"
// POST A BOOK
export function postBooks(book){
  return {
          type:"POST_BOOK",
          payload: book
        }
}

// DELETE A BOOK
export function deleteBooks(id){
  return {
          type:"DELETE_BOOK",
```

```
        payload: id
      }
}

// UPDATE A BOOK
export function updateBooks(book){
  return {
        type:"UPDATE_BOOK",
        payload: book
      }
}

//----------------------------------------
```

You can notice that each function takes an argument to be passed as payload from our main component

STEPS CHECK-LIST

20. edit app.js:

- **Import books and cart actions**
- **edit the dispatched actions:**

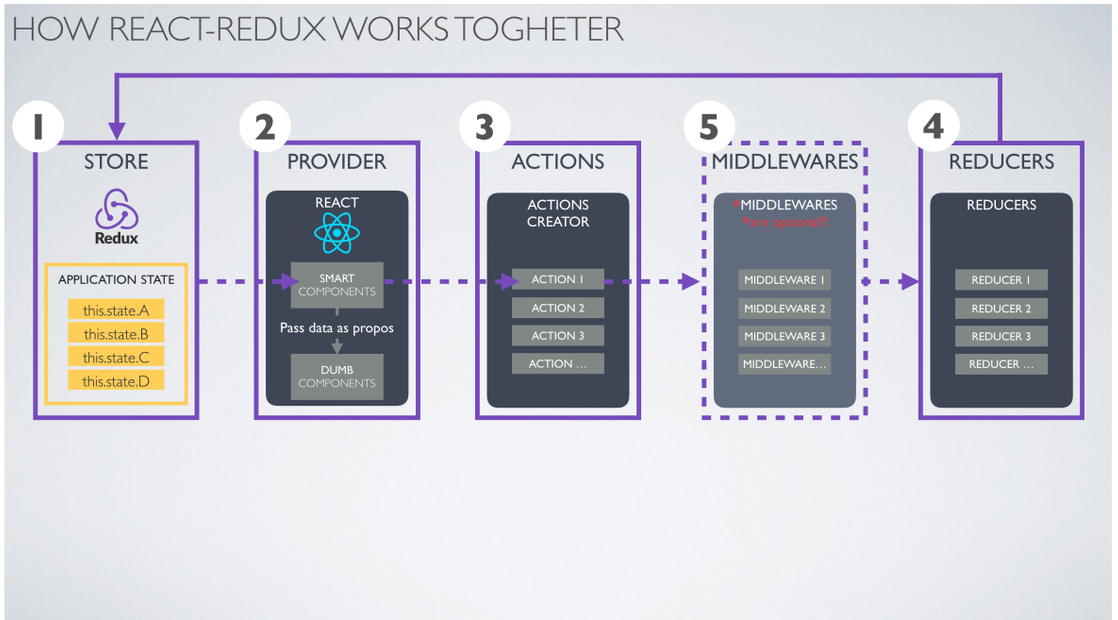
app.js:

```
// IMPORT ACTIONS
import {addToCart} from
'./actions/cartActions';
import {postBooks, deleteBooks, updateBooks}
from './actions/booksActions';

and edit the actions:
// STEP 2 create and dispatch actions
store.dispatch(postBooks(
    [{
      id: 1,
      title:'this is the book title',
      description: 'this is the book
description',
      price: 33.33
    },
    {
      id: 2,
      title:'this is the second book title',
      description: 'this is the second book
description',
      price: 50
    }]
))

// DELETE a book
store.dispatch(deleteBooks(
  {id: 1}
))
// UPDATE a book
store.dispatch(updateBooks(
  {
    id: 2,
    title:'Learn React in 24h'
```

```
  }
))

//-->> CART ACTIONS <<--
// ADD to cart
store.dispatch(addToCart([{id: 1}]))

//----------------------------------------
```

Finally, save and refresh to make sure everything is working correctly! You should have the same result as before splitting the actions code

MIDDLEWARES



A Redux Middleware sits between the dispatch of an action and the reducer, basically, it allows us to execute some tasks after an action has been dispatched and the reducer is executed.

We can use third parties Middlewares or we can even build a custom one for our own needs. For instance, we could create a middleare that fires a new action when a specific action has been dispatched and so creating chains of actions.

Let's use a third party middleware called: **redux-logger**. Redux-logger is a middleware that capture all actions and logs nicely the previous state and the next one, providing a great visibility on how the store is behaving.

STEPS CHECK-LIST

21. npm:

- **install redux-logger:** *npm install --save-dev redux-logger*
- *import {**applyMiddleware**, createStore} from* 'redux';
- *import logger from '**redux-logger**';*
- ***write the following code**: const middleware = applyMiddleware(logger());*
- **include the** *middelware* **constant as second argument in your createStore method as following:** *const store = createStore(reducers, **middleware**);*

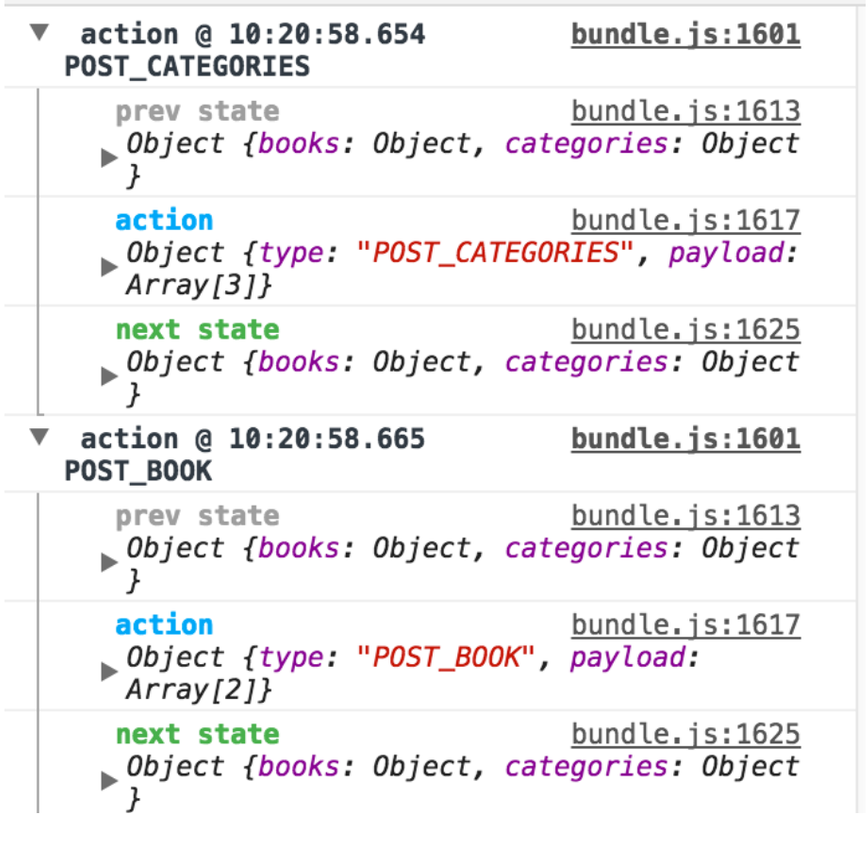
WARNING!!!

Latest version of redux-logger (V3.0 and above) exports the logger function by default and therefore you have to replace "logger()" with "logger".

Hereby, the correct code to use if you have redux-logger@3.0 or above:

```
//STEP 1 create store
const middleware = applyMiddleware(logger);
const store = createStore(reducers, middleware);
```

If you run the app, you will have now beautiful logs that shows you previous state, action dispatched and its payload and the next state too:



Later in the course I will add an other middleware for managing asynchronous api calls, but for now let's not overwhelm your brain with too many new notions you already reached a milestone! At this point you know already everything it is needed to build entire applications with Redux

In the next session we will learn how to make React and Redux work together and later we will also learn how to build our own APIs and keep the APIs data in the state with Redux!

Chapter 2

Learn React by building a Shopping-cart

Before we dig into MongoDB, we will create a more robust set-up of our webserver in Express which will host the entire application.

SECTION 1

REACT-REDUX TOGETHER

CREATE A SIMPLE REACT APP

1. Let's get started by installing : **npm install --save react@15.4.1 react-dom@15.4.1 react-redux@5.0.5 react-router@3.0.4**
2. Inside SRC create a folder: **components**
3. Inside components create also a folder: **pages**
4. Inside "pages" folder create a file: **booksList.js** . It will display a list of books
5. In booksList.js, **import 'react'** and **write a simple BooksList class to display an H1 tag with 'Hello React'**

WARNING!!!

After the make of this course was released react-router V4.0 and therefore make sure you specify the react-router version @2.8.1 when you install the npm package

booksList.js

```
"use strict"
import React from 'react';

class BooksList extends React.Component{
  render(){
    return(
      <div>
        <h1>Hello React</h1>
      </div>
    )
  }
}
export defafault BooksList;
//----------------------------------------
```

CREATE A SIMPLE REACT APP

In app.js,

1. Import react, ***render*** from ***react-dom*** and **your newly created {BookList} component**
2. Test it and make sure is working

app.js:

```
"use strict"
// REACT
import React from 'react';
import {render} from 'react-dom';

import {applyMiddleware, createStore} from
'redux';
import logger from 'redux-logger';

// IMPORT COMBINED REDUCERS
import reducers from './reducers/index';
// IMPORT ACTIONS
```

```
import {addToCart} from
'./actions/cartActions';
import {postBooks, deleteBooks, updateBooks}
from './actions/booksActions';

// STEP 1 create the store
const middleware =applyMiddleware(logger());
const store = createStore(reducers,
middleware);

// store.subscribe(function(){
//   console.log('current state is: ',
store.getState());
//   //console.log('current price: ',
store.getState()[1].price);
// })
import BooksList from
'./components/pages/bookslist';

render(
  <BooksList />,
document.getElementById('app')
);

// STEP 2 create and dispatch actions
store.dispatch(postBooks(
    [{
      id: 1,
      title:'this is the book title',
      description: 'this is the book
description',
      price: 33.33
    },
    {
      id: 2,
      title:'this is the second book title',
      description: 'this is the second book
description',
      price: 50
    }]
))

// DELETE a book
store.dispatch(deleteBooks(
  {id: 1}
))
// UPDATE a book
store.dispatch(updateBooks(
  {
    id: 2,
    title:'Learn React in 24h'
  }
))

//-->> CART ACTIONS <<--
// ADD to cart
store.dispatch(addToCart([{id: 1}]))

//----------------------------------------
```

CREATE A SIMPLE REACT-REDUX APP

In app.js,

1. Import react ***Provider component from react-redux***,
2. Use the React-Dom render component to wrap your React Application into the redux Provider component
3. In app.js, remove store.subscribe
4. In app.js, comment or remove: deleteBooks, updatedBooks and addToCart dispatcher.
5. Remove the the H1 tag related to Redux from index.html

HOW TO ACCESS THE STORE FROM REDUX

In booksList.js

1. Import all components you need to Link React with Redux: {***Connect***} from react-redux,
2. Import {***postBooks***} from './actions/booksActions';
3. Write the ***mapStateToProps*** method to get the state mapped in React
4. Connect together MapStateToProps and the React component with: ***export default connect(mapStateToProps)(BooksList);***
5. Render a dynamic div with by using map method to extract all objects from books array (this.props.books)

app.js:

```
"use strict"
// REACT
import React from 'react';
import {render} from 'react-dom';
import {Provider} from 'react-redux';
```

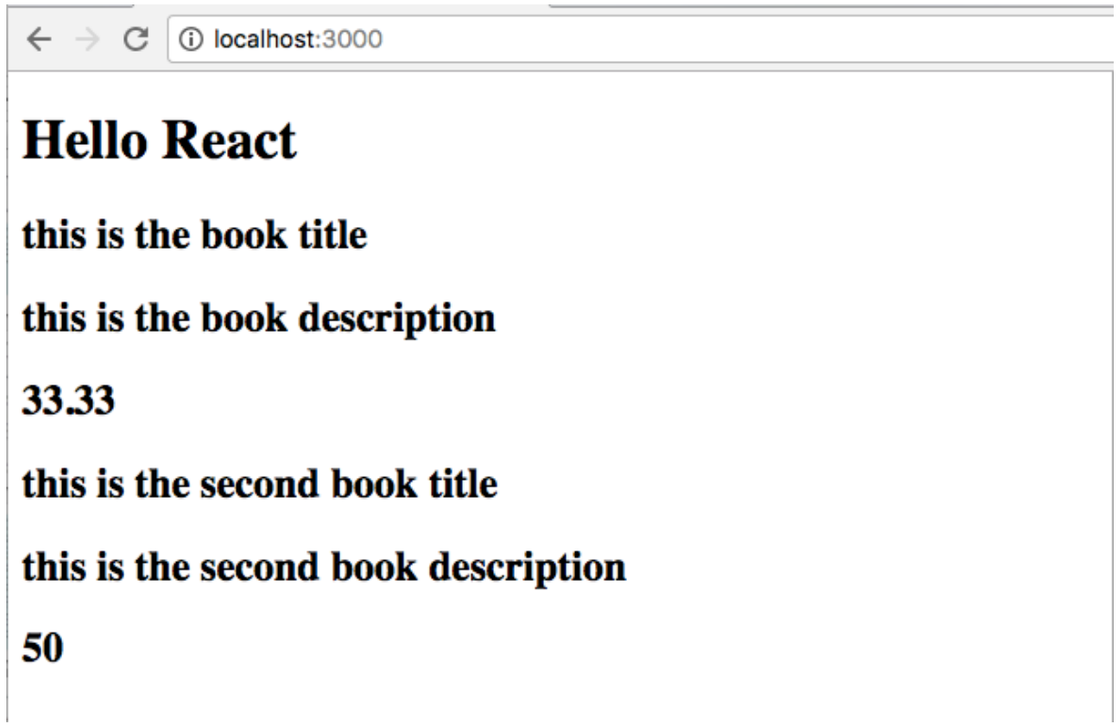
```
import {applyMiddleware, createStore} from
'redux';
import logger from 'redux-logger';

// IMPORT COMBINED REDUCERS
import reducers from './reducers/index';
// IMPORT ACTIONS
import {addToCart} from
'./actions/cartActions';
import {postBooks, deleteBooks, updateBooks}
from './actions/booksActions';

// STEP 1 create the store
const middleware =applyMiddleware(logger());
const store = createStore(reducers,
middleware);

import BooksList from
'./components/pages/bookslist';

render(
  <Provider store={store}>
    <BooksList />
  </Provider>, document.getElementById('app')
);

// STEP 2 create and dispatch actions
store.dispatch(postBooks(
    [{
      id: 1,
      title:'this is the book title',
      description: 'this is the book
description',
      price: 33.33
    },
    {
      id: 2,
      title:'this is the second book title',
      description: 'this is the second book
description',
      price: 50
    }]
))

//----------------------------------------
```

bookList.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';

class BooksList extends React.Component{
  render(){
    const booksList =
this.props.books.map(function(booksArr){
      return(
        <div key={booksArr.id}>
          <h2>{booksArr.title}</h2>
          <h2>{booksArr.description}</h2>
          <h2>{booksArr.price}</h2>
```

```
          </div>
        )
      })
      return(
        <div>
          <h1>Hello React</h1>
          {booksList}
        </div>
      )
    }
}
function mapStateToProps(state){
  return{
    books: state.books.books
  }
}
export default
connect(mapStateToProps)(BooksList);

//----------------------------------------
```

If you test this code, you should see:



Fantastic! That's it... you just were able to access data from the Store to your react and you can do that from any other component you will be creating!

Now, we are just missing the very last bit.

How do we dispatch actions from our smart component??

HOW TO DISPATCH ACTIONS FROM REACT

1. Since we will fire our actions from our react component, now you you have to delete getBooks action from app.js, as at the moment it is still fired by redux.
2. you can remove from your Import statement in app.js
3. In bookList.js, if you forgot to import {***getBook***} action, pls do it.
4. In order to dispatch action from your bookList component, go to bookList.js and add a ***mapDispatchToProps*** method which returns a dispatcher.
5. You add ***mapDispatchToProps*** as second argument of connect method
6. Supposing this was an API call, we would probably dispatch the action from componentDidMount, so in there we write: ***this.props.getBooks();***

booksList.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {getBooks} from
'../../actions/booksActions';
import {bindActionCreators} from 'redux';

class BooksList extends React.Component{
  componentDidMount(){
    this.props.getBooks(
          [{
            id: 1,
            title:'this is the book title',
            description: 'this is the book
description',
            price: 43.33
          },
          {
            id: 2,
            title:'this is the second book
title',
            description: 'this is the second
book description',
            price: 60
          }]
    )
  }
  render(){
    const booksList =
this.props.books.map(function(booksArr){
      return(
```

```
          <div key={booksArr.id}>
            <h2>{booksArr.title}</h2>
            <h2>{booksArr.description}</h2>
            <h2>{booksArr.price}</h2>
          </div>
        )
      })
      return(
        <div>
          <h1>Hello React</h1>
          {booksList}
        </div>
      )
    }
}
function mapStateToProps(state){
  return{
    books: state.books.books
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    getBooks:getBooks
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksList);

//----------------------------------------
```

Now that your React component is working completely independent from the store code, you want to better structure the file having the store related code inside a file called store.js where you create and store and add middlewares. so you can quickly do that in this way:

ADD REACT COMPONENTS

You can now add a little bit of initial styling:

INSTALLING REACT-BOOTSTRAP

1. npm install --save react-bootstrap
2. add CDN in the “<HEAD></HEAD>” of the index.html file

index.html:

```
<!DOCTYPE html>
<HTML>
  <HEAD>
    <TITLE>Hello Redux</TITLE>
    <meta name="viewport"
content="width=device-width,
initial-scale=1.0 maximum-scale=1.0" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootstr
```

```
ap/3.3.7/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg3
2OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
  </HEAD>
  <BODY>
    <DIV id="app">

    </DIV>
    <SCRIPT src="bundle.js"></SCRIPT>
  </BODY>
</HTML>
//----------------------------------------
```

Why are we using react-bootstrap rather then the original one? React-Bootstrap has all the original features written purely as react component. Basically, it will help you to write a little less code and waste time between your css and your react files and likely will save you from few bugs.

Style booklist and create bookitem components

1. In bookList.js, import {Grid, Row, Col, Button} from react-bootstrap
2. Edit the code to create a grid
3. in SRC folder add bookItem.js
4. In bookItem, import {Well, Button} from react-bootstrap and add a Well in the code

bookList.js:

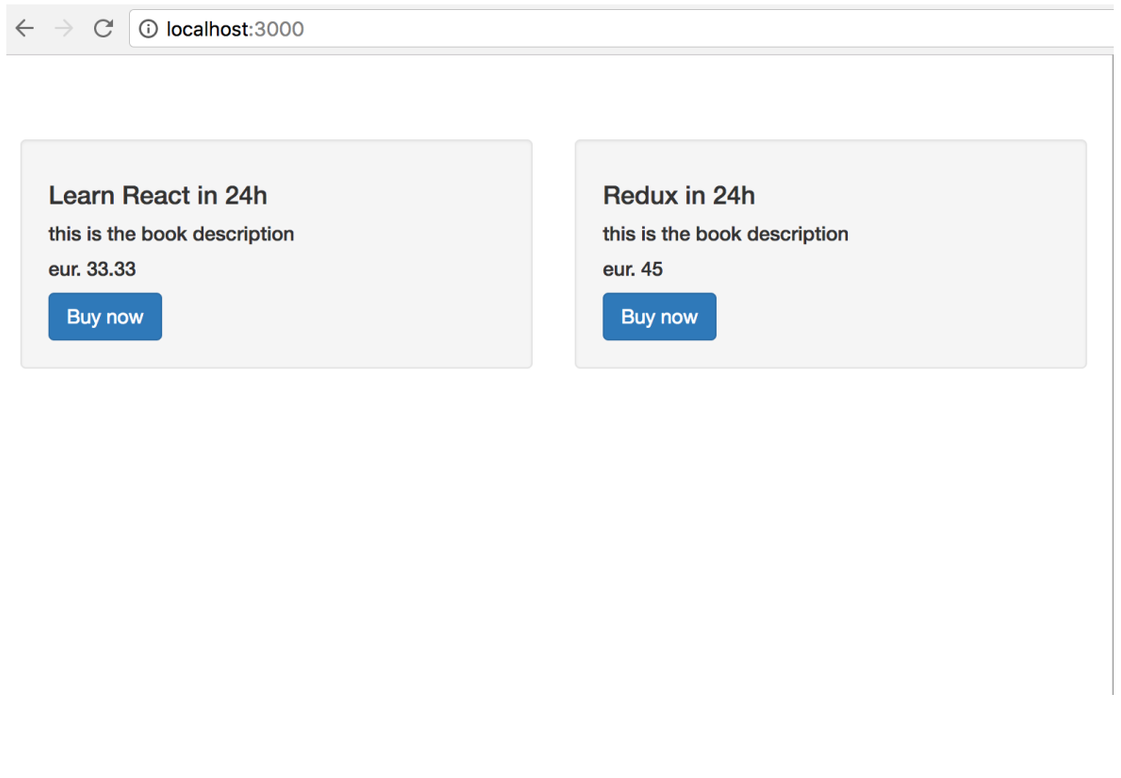
```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {getBooks} from
'../../actions/booksActions';
import {bindActionCreators} from 'redux';
import {Grid, Col, Row, Button} from
'react-bootstrap';

import BookItem from './bookItem';

class BooksList extends React.Component{
  componentDidMount(){
```

```
    this.props.getBooks(
          [{
            id: 1,
            title:'this is the book title',
            description: 'this is the book
description',
            price: 43.33
          },
          {
            id: 2,
            title:'this is the second book
title',
            description: 'this is the second
book description',
            price: 60
          }]
    )
  }
  render(){
    const booksList =
this.props.books.map(function(booksArr){
      return(
        <Col xs={12} sm={6} md={4}
key={booksArr.id}>
          <BookItem
                id= {booksArr.id}
                title={booksArr.title}

description={booksArr.description}
                price={booksArr.price}/>
        </Col>
      )
    })
    return(
      <Grid>
        <Row>
            {booksList}
        </Row>
      </Grid>
    )
  }
}
function mapStateToProps(state){
  return{
    books: state.books.books
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    getBooks:getBooks
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksList);
//---------------------------------------
```

bookItem.js:

```
"use strict"
import React from 'react';
import {Row, Col, Well, Button} from
'react-bootstrap';

class BookItem extends React.Component{
  render(){
    return(
      <Well>
        <Row>
          <Col xs={12}>
            <h6>{this.props.title}</h6>
            <p>{this.props.description}</p>
            <h6>usd. {this.props.price}</h6>
            <Button bsStyle='primary'>Buy
now</Button>

          </Col>
        </Row>
      </Well>
    )
  }
}
export default BookItem;

//----------------------------------------
```

You should see something like this:



Add BooksForm Component

1. Add src/booksForm.js
2. Import {Well, FormControl, FormGroup, ControlLabel, Button} from React-Bootstrap
3. Add four Input Controls
4. Import the control in bookList.js and place it in a new row

booksForm.js:

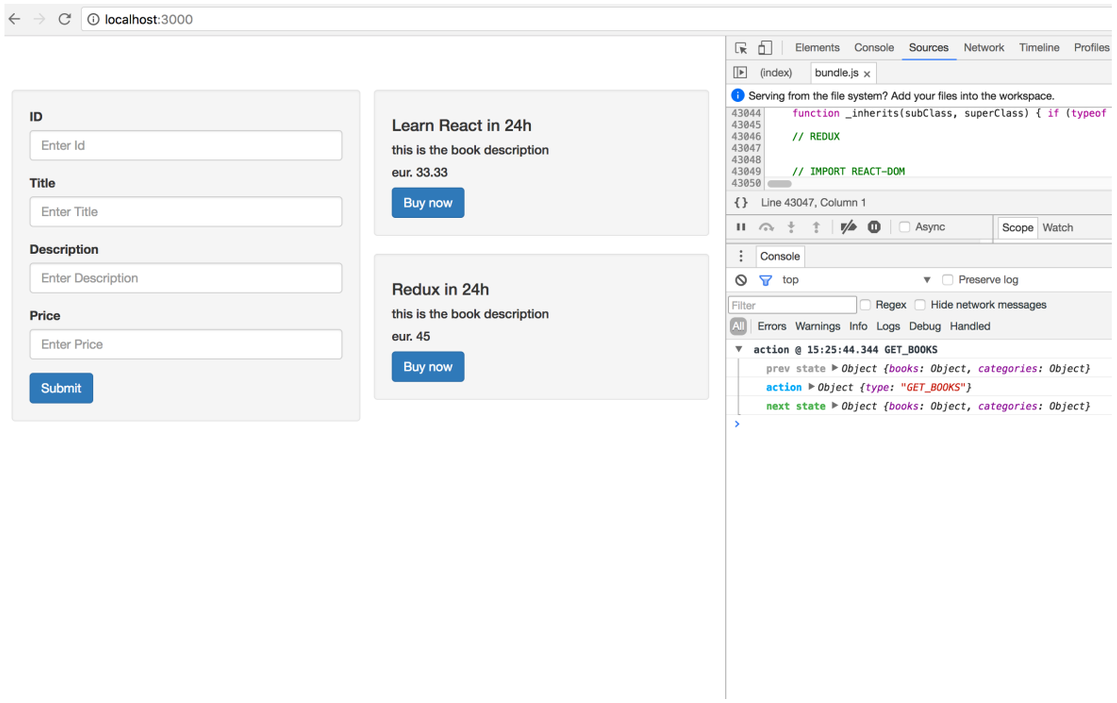
```
"use strict"
import React from 'react';
import {Well, Panel, FormControl, FormGroup,
ControlLabel, Button} from 'react-bootstrap';

class BooksForm extends React.Component{
  render(){
    return(
      <Well>
        <Panel>
          <FormGroup controlId="title">

<ControlLabel>Title</ControlLabel>
              <FormControl
                  type="text"
                  placeholder="Enter Title"
                  ref="title" />
            </FormGroup>
            <FormGroup controlId="description">

<ControlLabel>Description</ControlLabel>
              <FormControl
                  type="text"
                  placeholder="Enter
Description"
                  ref="description" />
            </FormGroup>
            <FormGroup controlId="price">

<ControlLabel>Price</ControlLabel>
              <FormControl
                  type="text"
                  placeholder="Enter Price"
                  ref="price" />
            </FormGroup>
            <Button
onClick={this.handleSubmit.bind(this)}
bsStyle="primary">Save book</Button>
          </Panel>
        </Well>
    )
  }
}
export default BooksForm;

//---------------------------------------
```

bookList.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {getBooks} from
'../../actions/booksActions';
import {bindActionCreators} from 'redux';
import {Grid, Col, Row, Button} from
'react-bootstrap';
```

```
import BookItem from './bookItem';
import BooksForm from './BooksForm';


class BooksList extends React.Component{
  componentDidMount(){
    this.props.getBooks()
  }
  render(){
    const booksList =
this.props.books.map(function(booksArr){
      return(
        <Col xs={12} sm={6} md={4}
key={booksArr.id}>
          <BookItem
                id= {booksArr.id}
                title={booksArr.title}

description={booksArr.description}
                price={booksArr.price}/>
        </Col>
      )
    })
    return(
      <Grid>
        <Row>
          <Col xs={12} sm={6}>
            <BooksForm />
          </Col>
            {booksList}
        </Row>
      </Grid>
    )
  }
}
function mapStateToProps(state){
  return{
    books: state.books.books
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    getBooks:getBooks
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksList);

//----------------------------------------
```

This should be the result (in a large screen)



Connect BooksForm to redux

1. Import the necessary redux components: import {***connect***} from 'react-redux', import { ***bindActionCreators*** } from 'redux'; import {***postBooks***} from './actions/booksActions';
2. Add ***mapDispatchToProps*** (since we will fire actions out from this component)
3. Add the connect function at the bottom: ***export default connect(null, mapDispatchToProps)(BooksForm);***
4. Add handleSubmit function and pass the inputs using title findDOMNode ie.:***findDOMNode(this.refs.title).value***.
5. Import ***import {findDOMNode} from***

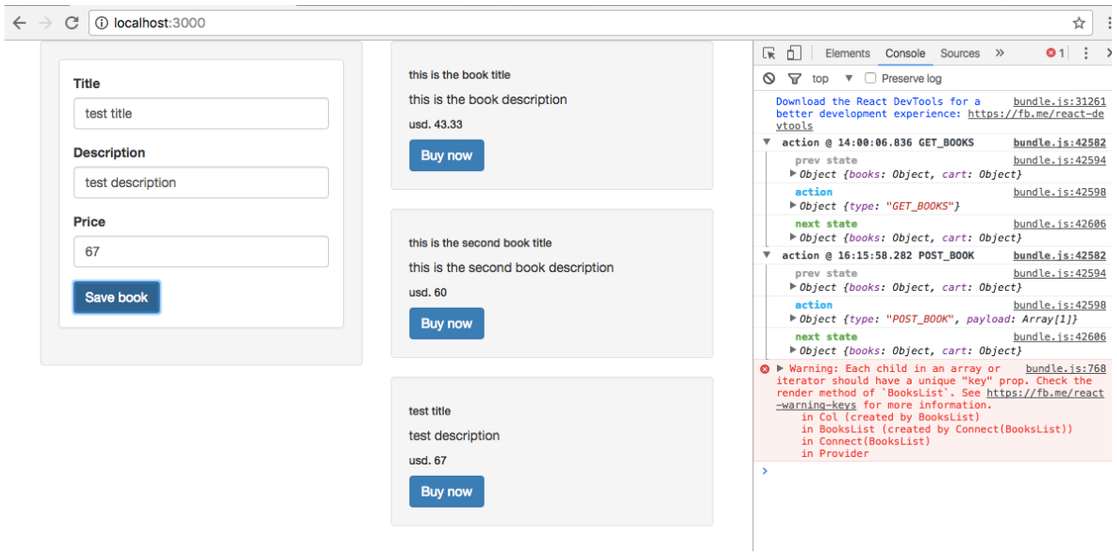
Do not get confused by the findDOMNode method we used to get the reference from our input form. **Why are we getting the imports using findDOMNode rather then this.refs.title.value?** Not a big deal, this is the way react-Bootstrap is able to reference their formComtrol. No worries, this is the exact equivalent of this.refs.id.values in terms of performances.

booksForm.js:

```
"use strict"
import React from 'react';
import {Well, Panel, FormControl, FormGroup,
ControlLabel, Button} from 'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';
import {findDOMNode} from 'react-dom';
import {postBooks} from
'../../actions/booksActions';

class BooksForm extends React.Component{

  handleSubmit(){
    const book=[{
      title:
findDOMNode(this.refs.title).value,
      description:
findDOMNode(this.refs.description).value,
      price:
findDOMNode(this.refs.price).value,
    }]
    this.props.postBooks(book);
  }

  render(){
    return(
      <Well>
        <Panel>
          <FormGroup controlId="title">

<ControlLabel>Title</ControlLabel>
            <FormControl
                type="text"
                placeholder="Enter Title"
                ref="title" />
          </FormGroup>
          <FormGroup controlId="description">

<ControlLabel>Description</ControlLabel>
            <FormControl
                type="text"
                placeholder="Enter
Description"
                ref="description" />
          </FormGroup>
          <FormGroup controlId="price">

<ControlLabel>Price</ControlLabel>
            <FormControl
                type="text"
                placeholder="Enter Price"
                ref="price" />
          </FormGroup>
          <Button
onClick={this.handleSubmit.bind(this)}
bsStyle="primary">Save book</Button>
        </Panel>
      </Well>
    )
```

```
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({postBooks},
dispatch)
}
export default connect(null,
mapDispatchToProps)(BooksForm);

//----------------------------------------
```

The result:



*You will get a warning: “***_Each child in an array or iterator should have a unique "key" prop_**”. Do not worry, you get this error because we are not specifying an ID for the book we are posting. Later we will post new books in a MongoDB database and Mongo will automatically insert an ID

If you run the app, you will see that it quickly post a new product and your state is always updated.

Obviously, since we are not yet storing this changes in a database and our data is coming from our initialState set in the reducer, **if you refresh the page your newly added books will disappear.**

CREATE THE CART COMPONENT - PART1

1. Add src/cart.js
2. Import {Panel, Col, Row, Well, Button} from React-Bootstrap
3. This will be a smart component so import also:
4. import {connect} from 'react-redux';
5. import {addToCart} from './actions/cartActions';
6. Make bookItem to be a smart component adding {connect} and {bindActionCreator}
7. Add a onClick event to the button
8. Add a handleCart() function to dispatch addToCart action
9. Change ADD_TO_CART reducer
10. Finally, import the Cart component in booksList to render it together with the other

Cart.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {Panel, Col, Row, Well, Button} from
'react-bootstrap';

class Cart extends React.Component{
  render(){
    if(this.props.cart[0]){
      return this.renderCart();
    } else {
      return this.renderEmpty();
    }
  }
  renderEmpty(){
    return(<div></div>)
  }

  renderCart(){
    const cartItemsList =
this.props.cart.map(function(cartArr){
      return(
        <Panel key={cartArr.id}>
          <Row>
            <Col xs={12} sm={4}>
              <h6>{cartArr.title}</h6>
            </Col>
          </Row>
        </Panel>
      )
    })
```

```
    return(
      <Panel header="Cart" bsStyle="primary">
        {cartItemsList}
      </Panel>
    )
  }
}
function mapStateToProps(state){
  return{
    cart: state.cart.cart
  }
}
export default
connect(mapStateToProps)(Cart);

//----------------------------------------
```

bookItem.js:

```
"use strict"
import React from 'react';
import {Row, Col, Well, Button} from
'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux'
import {addToCart} from
'../../actions/cartActions';

class BookItem extends React.Component{

  handleCart(){
    const book = [...this.props.cart, {
      id:this.props.id,
      title:this.props.title,
      description:this.props.description,
      price:this.props.price
    }]
    this.props.addToCart(book);
  }

  render(){
    return(
      <Well>
        <Row>
          <Col xs={12}>
            <h6>{this.props.title}</h6>
            <p>{this.props.description}</p>
            <h6>usd. {this.props.price}</h6>
            <Button
onClick={this.handleCart.bind(this)}
bsStyle='primary'>Buy now</Button>

          </Col>
        </Row>
      </Well>
    )
  }
}
function mapStateToProps(state){
```

```
  return{
    cart:state.cart.cart
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    addToCart:addToCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BookItem);

//----------------------------------------
```

bookList.js:

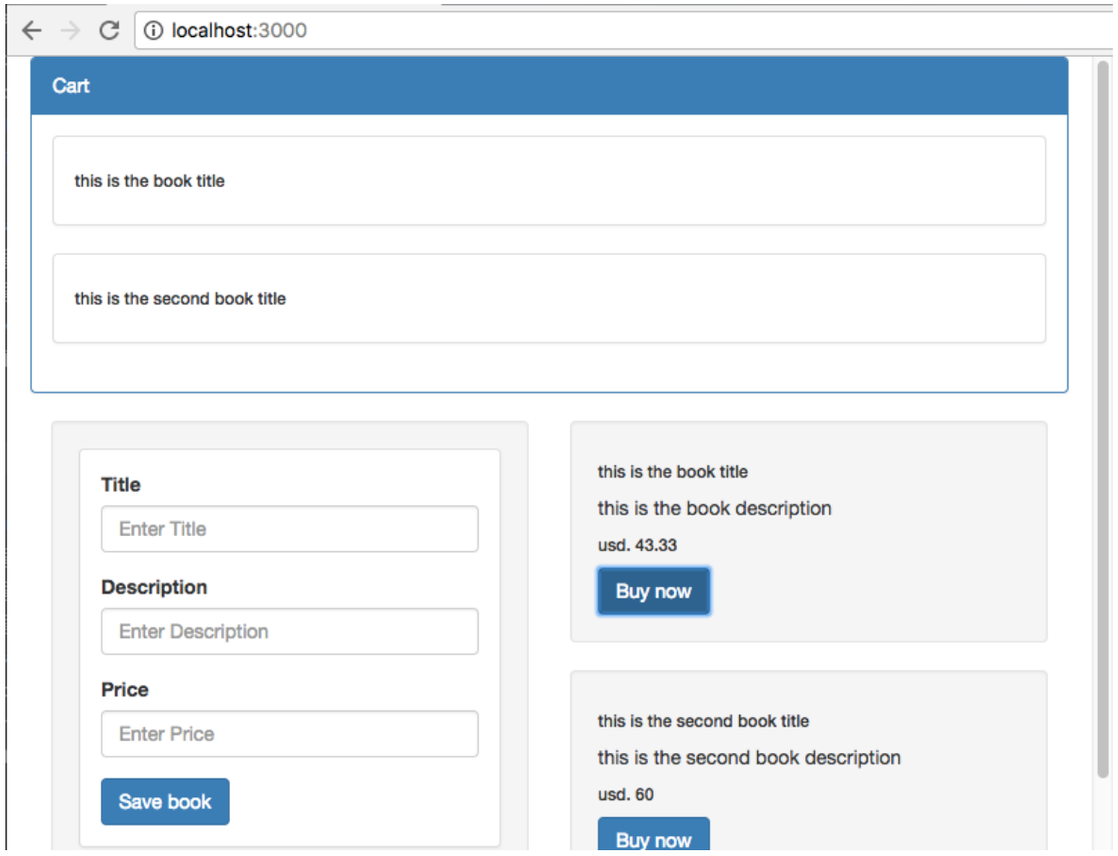
IMPORT cart.js in bookList.js:

```
return(
      <Grid>
        <Row>
          <Cart />
        </Row>
        <Row>
          <Col xs={12} sm={6}>
            <BooksForm />
          </Col>
            {booksList}
        </Row>
      </Grid>
    )

//----------------------------------------
```

cartReducers.js:

```
case "ADD_TO_CART":
    return {...state, ...action.payload}
    break;

//----------------------------------------
```

When you press the “Buy now” button you should see the cart rendering the result:



CREATE THE CART COMPONENT - PART2

1. Add src/cart.js
2. Import: ButtonGroup, Label from react-bootstrap
3. Add the following code in cart.js
4. save and test

cart.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {Panel, Col, Row, Well, Button,
ButtonGroup, Label} from 'react-bootstrap';

class Cart extends React.Component{
  render(){
    if(this.props.cart[0]){
      return this.renderCart();
    } else {
      return this.renderEmpty();
    }
  }
  renderEmpty(){
```

```
    return(<div></div>)
  }

  renderCart(){
    const cartItemsList =
this.props.cart.map(function(cartArr){
      return(
        <Panel key={cartArr.id}>
          <Row>
            <Col xs={12} sm={4}>
              <h6>{cartArr.title}</h6><span>
</span>
            </Col>
            <Col xs={12} sm={2}>
              <h6>usd. {cartArr.price}</h6>
            </Col>
            <Col xs={12} sm={2}>
              <h6>qty. <Label
bsStyle="success"></Label></h6>
            </Col>
            <Col xs={6} sm={4}>
              <ButtonGroup
style={{minWidth:'300px'}}>
                <Button bsStyle="default"
bsSize="small">-</Button>
                <Button bsStyle="default"
bsSize="small">+</Button>
                <span>      </span>
                <Button bsStyle="danger"
bsSize="small">DELETE</Button>
              </ButtonGroup>
            </Col>
          </Row>
        </Panel>
      )
    })
    return(
      <Panel header="Cart" bsStyle="primary">
        {cartItemsList}
      </Panel>
    )
  }
}
function mapStateToProps(state){
  return{
    cart: state.cart.cart
  }
}
export default
connect(mapStateToProps)(Cart);

//----------------------------------------
```

ADD UPDATE QUANTITIES IN THE CART

1. In bookItem.js inside handleCart function, add "quantity" property and set it equal to 1
2. in cart.js render quantity property in the label for showing the quantity.
3. test and make sure quantity is now visible

bookItem.js:

```
handleCart(){
    const book = [...this.props.cart, {
      _id:this.props._id,
      title:this.props.title,
      description:this.props.description,
      price:this.props.price,
      quantity:1
    }]

//----------------------------------------
```

cart.js:

```
<Col xs={12} sm={2}>
<h6>qty. <Label bsStyle="success"
>{cartArr.quantity}</Label></h6>
 </Col>

//----------------------------------------
```

ADD UPDATE QUANTITIES IN THE CART

1. Add updateCart Action
2. Add the reducer
3. In bookItem.js improve handleCart function to update the quantity in case in cart there is an other item with the same _id
4. Save and Refresh and check if quantity are increased when user press the Buy button multiple times

cartActions.js:

```
// UPDATE CART
export function updateCart(_id, unit){
  return {
    type:"UPDATE_CART",
    _id: _id,
```

```
    unit: unit
  }
}
//-----------------------------------------
```

cartReducers.js:

```
case "UPDATE_CART":
    // Create a copy of the current array of
books
    const currentBookToUpdate =
[...state.cart]
    // Determine at which index in books
array is the book to be deleted
    const indexToUpdate =
currentBookToUpdate.findIndex(
      function(book){
        return book._id === action._id;
      }
    )

    const newBookToUpdate = {
      ...currentBookToUpdate[indexToUpdate],
      quantity:
currentBookToUpdate[indexToUpdate].quantity +
action.unit
    }

    let cartUpdate =
[...currentBookToUpdate.slice(0,
indexToUpdate), newBookToUpdate,
...currentBookToUpdate.slice(indexToUpdate +
1)]

    return {...state,
      cart:cartUpdate
    }
    break;

//-----------------------------------------
```

bookItem.js:

```
handleCart(){
    const book = [...this.props.cart, {
      _id:this.props._id,
      title:this.props.title,
      description:this.props.description,
      price:this.props.price,
      quantity:1
    }]

    // CHECK IF CART IS EMPTY
    if(this.props.cart.length > 0) {
      // CART IS NOT EMPTY
      let _id = this.props._id;

      let cartIndex =
this.props.cart.findIndex(function(cart){
```

```
        return cart._id === _id;
      })
      // IF RETURNS -1 THERE ARE NO ITEMS
WITH SAME ID
      if (cartIndex === -1){
        this.props.addToCart(book);
      } else {
        // WE NEED TO UPDATE QUANTITY
        this.props.updateCart(_id, 1)
      }

    } else {
      // CART IS EMPTY
      this.props.addToCart(book);
    }

  }

//----------------------------------------
```

ADD UPDATE QUANTITIES IN THE CART

1. In cart.js, add a onIncrement(_id) and onDecrement(_id, quantity) functions and write the code below
2. Add onClick event to + and - buttons in cart.js to fire respectively onIncrement and onDecrement functions and passing the cartArr._id and in case of decrement also the cartArr.quantity
3. import updateCart action and add the action inside the mapDispatchToProps function
4. save and refresh, users should be now able to increment and decrement quantities by pressing the + and - buttons in the cart

cart.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
```

```
import {Panel, Col, Row, Well, Button,
ButtonGroup, Label} from 'react-bootstrap';
import {bindActionCreators} from 'redux';
import {deleteCartItem, updateCart} from
'../../actions/cartActions';

class Cart extends React.Component{

  onDelete(_id){
    // Create a copy of the current array of
books
    const currentBookToDelete =
this.props.cart;
    // Determine at which index in books
array is the book to be deleted
    const indexToDelete =
currentBookToDelete.findIndex(
      function(cart){
        return cart._id === _id;
      }
    )
    //use slice to remove the book at the
specified index
    let cartAfterDelete =
[...currentBookToDelete.slice(0,
indexToDelete),
...currentBookToDelete.slice(indexToDelete +
1)]


this.props.deleteCartItem(cartAfterDelete);
  }
  onIncrement(_id){
    this.props.updateCart(_id, 1);
  }
  onDecrement(_id, quantity){
    if(quantity > 1){
      this.props.updateCart(_id, -1);
    }
  }

  render(){
    if(this.props.cart[0]){
      return this.renderCart();
    } else {
      return this.renderEmpty();
    }
  }
  renderEmpty(){
    return(<div></div>)
  }

  renderCart(){
    const cartItemsList =
this.props.cart.map(function(cartArr){
      return(
        <Panel key={cartArr._id}>
          <Row>
            <Col xs={12} sm={4}>
              <h6>{cartArr.title}</h6><span>
</span>
            </Col>
```

```
            <Col xs={12} sm={2}>
              <h6>usd. {cartArr.price}</h6>
            </Col>
            <Col xs={12} sm={2}>
              <h6>qty. <Label
bsStyle="success">{cartArr.quantity}</Label><
/h6>
            </Col>
            <Col xs={6} sm={4}>
              <ButtonGroup
style={{minWidth:'300px'}}>
                <Button
onClick={this.onDecrement.bind(this,
cartArr._id, cartArr.quantity)}
bsStyle="default" bsSize="small">-</Button>
                <Button
onClick={this.onIncrement.bind(this,
cartArr._id)} bsStyle="default"
bsSize="small">+</Button>
                <span>     </span>
                <Button
onClick={this.onDelete.bind(this,
cartArr._id)} bsStyle="danger"
bsSize="small">DELETE</Button>
              </ButtonGroup>
            </Col>
          </Row>
        </Panel>
      )
    }, this)
    return(
      <Panel header="Cart" bsStyle="primary">
        {cartItemsList}
      </Panel>
    )
  }
}
function mapStateToProps(state){
  return{
    cart: state.cart.cart
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    deleteCartItem:deleteCartItem,
    updateCart:updateCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(Cart);

//----------------------------------------
```

ADD MODAL WINDOWS TO CONFIRM PURCHASE

1. Add an <H6> tag for the “Total amount”
2. Add a “Proceed to checkout” button with sbStyle=”success” and assize=”small”
3. Copy the Modal dialog code from this script
4. import “Modal” from react-bootstrap
5. paste the modal dialog code after the checkout button
6. remember to add “.bind(this)” to the onHide and onClose events in the dialog
7. write the React constructor to initialize the state of the dialog to: showModal:false
8. add a open() and close() functions whose set the showModal state respectively to “true” and “false”
9. add the onClick event in the checkout button that fires the open() function
10. add some contents in the Modal body as in this script
11. save and test it

cart.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {Modal, Panel, Col, Row, Well, Button,
ButtonGroup, Label} from 'react-bootstrap';
import {bindActionCreators} from 'redux';
import {deleteCartItem, updateCart} from
'../../actions/cartActions';

class Cart extends React.Component{

  onDelete(_id){
    // Create a copy of the current array of
books
    const currentBookToDelete =
this.props.cart;
    // Determine at which index in books
array is the book to be deleted
    const indexToDelete =
currentBookToDelete.findIndex(
      function(cart){
        return cart._id === _id;
      }
    )
    //use slice to remove the book at the
specified index
    let cartAfterDelete =
[...currentBookToDelete.slice(0,
```

```
indexToDelete),
...currentBookToDelete.slice(indexToDelete +
1)]


this.props.deleteCartItem(cartAfterDelete);
  }
  onIncrement(_id){
    this.props.updateCart(_id, 1);
  }
  onDecrement(_id, quantity){
    if(quantity > 1){
      this.props.updateCart(_id, -1);
    }
  }
  constructor(){
    super();
    this.state = {
      showModal:false
    }
  }
  open(){
    this.setState({showModal:true})
  }
  close(){
    this.setState({showModal:false})
  }

  render(){
    if(this.props.cart[0]){
      return this.renderCart();
    } else {
      return this.renderEmpty();
    }
  }
  renderEmpty(){
    return(<div></div>)
  }

  renderCart(){
    const cartItemsList =
this.props.cart.map(function(cartArr){
      return(
        <Panel key={cartArr._id}>
          <Row>
            <Col xs={12} sm={4}>
              <h6>{cartArr.title}</h6><span>
</span>
            </Col>
            <Col xs={12} sm={2}>
              <h6>usd. {cartArr.price}</h6>
            </Col>
            <Col xs={12} sm={2}>
              <h6>qty. <Label
bsStyle="success">{cartArr.quantity}</Label><
/h6>
            </Col>
            <Col xs={6} sm={4}>
              <ButtonGroup
style={{minWidth:'300px'}}>
                <Button
onClick={this.onDecrement.bind(this,
```

```
cartArr._id, cartArr.quantity)}
bsStyle="default" bsSize="small">-</Button>
                  <Button
onClick={this.onIncrement.bind(this,
cartArr._id)} bsStyle="default"
bsSize="small">+</Button>
                  <span>     </span>
                  <Button
onClick={this.onDelete.bind(this,
cartArr._id)} bsStyle="danger"
bsSize="small">DELETE</Button>
                </ButtonGroup>
              </Col>
            </Row>
          </Panel>
        )
      }, this)
      return(
        <Panel header="Cart" bsStyle="primary">
          {cartItemsList}
          <Row>
            <Col xs={12}>
              <h6>Total amount:</h6>
              <Button
onClick={this.open.bind(this)}
bsStyle="success" bsSize="small">
                PROCEED TO CHECKOUT
              </Button>
            </Col>
          </Row>
```

```
        <Modal show={this.state.showModal}
onHide={this.close.bind(this)}>
            <Modal.Header closeButton>
              <Modal.Title>Thank
you!</Modal.Title>
            </Modal.Header>
            <Modal.Body>
              <h6>Your order has been
saved</h6>
              <p>You will receive an email
confirmation</p>
            </Modal.Body>
            <Modal.Footer>
              <Col xs={6}>
                <h6>total $:</h6>
              </Col>
              <Button
onClick={this.close.bind(this)}>Close</Button
>
            </Modal.Footer>
          </Modal>
        </Panel>
      )
    }
}
function mapStateToProps(state){
  return{
    cart: state.cart.cart
  }
}
function mapDispatchToProps(dispatch){
```

```
  return bindActionCreators({
    deleteCartItem:deleteCartItem,
    updateCart:updateCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(Cart);

//----------------------------------------
```

CALCULATE TOTALS

1. In cartReducers add the totals function
2. in all reducers in cartReducers add totalAmount and TotalQty
3. In cart.js add totalQuantity in mapStateToProps function

cartReducers.js:

```
"use strict"
// CART REDUCERS
export function cartReducers(state={cart:[]},
action) {
  switch(action.type){
    case "ADD_TO_CART":
    return {...state,
      cart:action.payload,
      totalAmount:
totals(action.payload).amount,
      totalQty: totals(action.payload).qty
    }
    break;
    case "UPDATE_CART":
```

```
    // Create a copy of the current array of
books
    const currentBookToUpdate =
[...state.cart]
    // Determine at which index in books
array is the book to be deleted
    const indexToUpdate =
currentBookToUpdate.findIndex(
      function(book){
        return book._id === action._id;
      }
    )

    const newBookToUpdate = {
      ...currentBookToUpdate[indexToUpdate],
      quantity:
currentBookToUpdate[indexToUpdate].quantity +
action.unit
    }

    let cartUpdate =
[...currentBookToUpdate.slice(0,
indexToUpdate), newBookToUpdate,
...currentBookToUpdate.slice(indexToUpdate +
1)]

    return {...state,
      cart:cartUpdate,
      totalAmount: totals(cartUpdate).amount,
      totalQty: totals(cartUpdate).qty
    }
```

```
    break;
    case "DELETE_CART_ITEM":
    return {...state,
      cart:action.payload,
      totalAmount:
totals(action.payload).amount,
      totalQty: totals(action.payload).qty
    }
    break;
  }
  return state
}

// CALCULATE TOTALS
export function totals(payloadArr){

  const totalAmount =
payloadArr.map(function(cartArr){
    return cartArr.price * cartArr.quantity;
  }).reduce(function(a, b) {
    return a + b;
  }, 0); //start summing from index0


  const totalQty =
payloadArr.map(function(qty){
    return qty.quantity;
  }).reduce(function(a, b) {
    return a + b;
  }, 0);
```

```
  return {amount:totalAmount.toFixed(2),
qty:totalQty}
}

//----------------------------------------
```

cart.js:

```
//...
<h6>Total amount:
{this.props.totalAmount}</h6>
//....
<h6>total $: {this.props.totalAmount}</h6>
//..
function mapStateToProps(state){
  return{
    cart: state.cart.cart,
    totalAmount: state.cart.totalAmount,
  }
}
//...

//----------------------------------------
```

CREATE THE FORM TO DELETE BOOKS

In booksForm.js:

1. add the mapStateToProps function to access books from the state
2. Import and Add in MapDispatToProps deleteBooks action
3. Add a new Panel below the form to submit new books
4. inside the panel add A booksForm with a FormControl with a componentClass of “select”.
5. in FormControl add a ref=”delete” to capture the selected bookId
6. Below the formGroup add a new Button with sbStyle=”danger”
7. Add onDelete() function to dispatch the deleteBooks action

In BooksReduders.js;

1. In findIndex function in DELETE_BOOK reducer, remove the “_id” after the action.payload and replace the: “===” with “==” or convert book._id toString().

booksForm.js:

```
"use strict"
import React from 'react';
import {Well, Panel, FormControl, FormGroup,
ControlLabel, Button} from 'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';
import {findDOMNode} from 'react-dom';
import {postBooks, deleteBooks} from
'../../actions/booksActions';


class BooksForm extends React.Component{

  handleSubmit(){
    const book=[{
      title:
findDOMNode(this.refs.title).value,
      description:
findDOMNode(this.refs.description).value,
      price:
findDOMNode(this.refs.price).value,
    }]
    this.props.postBooks(book);
  }

  onDelete(){
    let bookId =
findDOMNode(this.refs.delete).value;
    this.props.deleteBooks(bookId);
  }

  render(){

    const booksList =
this.props.books.map(function(booksArr){
      return (
        <option key={booksArr._id}>
{booksArr._id}</option>
      )
    })

    return(
      <Well>
        <Panel>
          <FormGroup controlId="title">

<ControlLabel>Title</ControlLabel>
            <FormControl
                type="text"
                placeholder="Enter Title"
                ref="title" />
          </FormGroup>
          <FormGroup controlId="description">

<ControlLabel>Description</ControlLabel>
            <FormControl
                type="text"
```

```
                placeholder="Enter
Description"
                ref="description" />
            </FormGroup>
            <FormGroup controlId="price">

<ControlLabel>Price</ControlLabel>
              <FormControl
                type="text"
                placeholder="Enter Price"
                ref="price" />
            </FormGroup>
            <Button
onClick={this.handleSubmit.bind(this)}
bsStyle="primary">Save book</Button>
          </Panel>
          <Panel >
            <FormGroup
controlId="formControlsSelect">
              <ControlLabel>Select a book id to
delete</ControlLabel>
              <FormControl ref="delete"
componentClass="select" placeholder="select">
                <option
value="select">select</option>
                  {booksList}
              </FormControl>
            </FormGroup>
            <Button
onClick={this.onDelete.bind(this)}
bsStyle="danger">Delete book</Button>
          </Panel>
        </Well>
      )
    }
}
function mapStateToProps(state){
  return {
    books: state.books.books
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    postBooks,
    deleteBooks
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksForm);

//----------------------------------------
```

booksReducers.js:

```
case "DELETE_BOOK":
    // Create a copy of the current array of
books
    const currentBookToDelete =
[...state.books]
```

```
    // Determine at which index in books
array is the book to be deleted
    const indexToDelete =
currentBookToDelete.findIndex(
      function(book){
        return book._id == action.payload;
      }
    )
    //use slice to remove the book at the
specified index
    return {books:
[...currentBookToDelete.slice(0,
indexToDelete),
...currentBookToDelete.slice(indexToDelete +
1)]}
    break;

//----------------------------------------
```

Create NavBar and Footer:

1. Copy the code of Responsive Navbar component from React-Bootstrap website
2. Create a menu.js inside Components folder
3. Import Menu component in Ap.pjs
4. save & Refresh to make sure the Menu bar is visible
5. Create a style.css file inside Public folder
6. Insert the Link to style.css in index.html
7. Save & Refresh to make sure the Css is available in our web app
8. Create footer.js inside Components folder
9. Save & refresh to check if the `Footer is now in place

Menu.js:

```
"use strict"
import React from 'react';
import {Nav, NavItem, Navbar, Badge} from
'react-bootstrap';
```

```
class Menu extends React.Component{
  render(){
    return(
      <Navbar inverse fixedTop>
    <Navbar.Header>
      <Navbar.Brand>
        <a href="/">React-Bootstrap</a>
      </Navbar.Brand>
      <Navbar.Toggle />
    </Navbar.Header>
    <Navbar.Collapse>
      <Nav>
        <NavItem eventKey={1}
href="/about">About</NavItem>
        <NavItem eventKey={2}
href="/contacts">Contact Us</NavItem>

      </Nav>
      <Nav pullRight>
        <NavItem eventKey={1}
href="/admin">Admin</NavItem>
        <NavItem eventKey={2}
href="/cart">Your Cart
            <Badge
className="badge">1</Badge></NavItem>
      </Nav>
    </Navbar.Collapse>
  </Navbar>
    );
  }
```

```
}
export default Menu


//-------------------------------------------------------------
```

Footer.js:

```
"use strict"
import React from 'react';

class Footer extends React.Component{
  render(){
    return(
        <footer className="footer
text-center">
          <div className="container">
            <p
className="footer-text">Copyright 2017
BooksShop. All rights reserved</p>
          </div>
        </footer>
    );
  }
}
export default Footer


//-------------------------------------------------------------
```

Appjs:

```
import Menu from './components/menu.js';
import Footer from './components/footer.js';

render(
  <Provider store={store}>
      <div>
        <Menu />
        <Footer />
      </div>
  </Provider>, document.getElementById('app')
);

//------------------------------------------------------------------
```

WARNING!!!

After the make of this course was released react-router V4.0 and therefore make sure you specify the react-router version @3.0.2 when you install the npm package

REACT-ROUTER:

1. Install react-router: **npm i --save react-router@3.0.4**
2. Create main.js inside src folder

in main.js:

1. Import Menu and Footer components
2. Inside render method insert Menu, {this.props.children} and footer

In app.js:

1. Import react-router
2. remove Menu and Footer components (both the import statement and from the render method
3. Import Cart, BooksForm and Main components
4. Create a constant Route with all routing set and pass it to the render method

in Server.js:

1. Change the app.get url to “*” in order to accept all routes

main.js:

```
"use strict"
import React from 'react';
import Menu from './components/menu';
import Footer from './components/footer';

class Main extends React.Component{
  render(){
    return(
      <div>
        <Menu />
          {this.props.children}
        <Footer />
      </div>
    );
  }
}
export default Main
//----------------------------------------------------------------
```

App.js:

```
"use strict"
// REACT
import React from 'react';
import {render} from 'react-dom';
import {Provider} from 'react-redux';
// REACT-ROUTER
import {Router, Route, IndexRoute,
browserHistory} from 'react-router';

import {applyMiddleware, createStore} from
'redux';
import logger from 'redux-logger';

// IMPORT COMBINED REDUCERS
import reducers from './reducers/index';
// IMPORT ACTIONS
import {addToCart} from
'./actions/cartActions';
// STEP 1 create the store
const middleware =applyMiddleware(logger());
const store = createStore(reducers,
middleware);

import BooksList from
'./components/pages/bookslist';
import Cart from './components/pages/cart';
import BooksForm from
'./components/pages/booksForm';
import Main from './main';

const Routes = (
  <Provider store={store}>
      <Router history={browserHistory}>
        <Route path="/" component={Main}>
            <IndexRoute
component={BooksList}/>
```

```
            <Route path="/admin"
component={BooksForm}/>
            <Route path="/cart"
component={Cart}/>
          </Route>
        </Router>
    </Provider>
)

render(
  Routes, document.getElementById('app')
);


//---------------------------------------------------------------------
```

server.js:

```
app.get('*', function(req, res){
  res.sendFile(path.resolve(__dirname,
'public', 'index.html'))
})
//---------------------------------------------------------------------
```

CONNECT THE BADGE TO THE STATE

in main.js

1. Edit main.js to receive totalQty props from state and pass it down to menu.js

in menu.js

1. Ad a conditional statement to display the number of item in the cart only if this.props.cartItemNumber props are greater than 0

main.js:

main.js:

```
"use strict"
import React from 'react';
import Menu from './components/menu';
import Footer from './components/footer';

import{connect} from 'react-redux';

class Main extends React.Component{
  render(){
    return(
      <div>
        <Menu
cartItemsNumber={this.props.totalQty} />
          {this.props.children}
```

```
        <Footer />
      </div>
    );
  }
}

function mapStateToProps(state){
  return {
    totalQty: state.cart.totalQty
  }
}
export default
connect(mapStateToProps)(Main);
//-------------------------------------------------------------
```

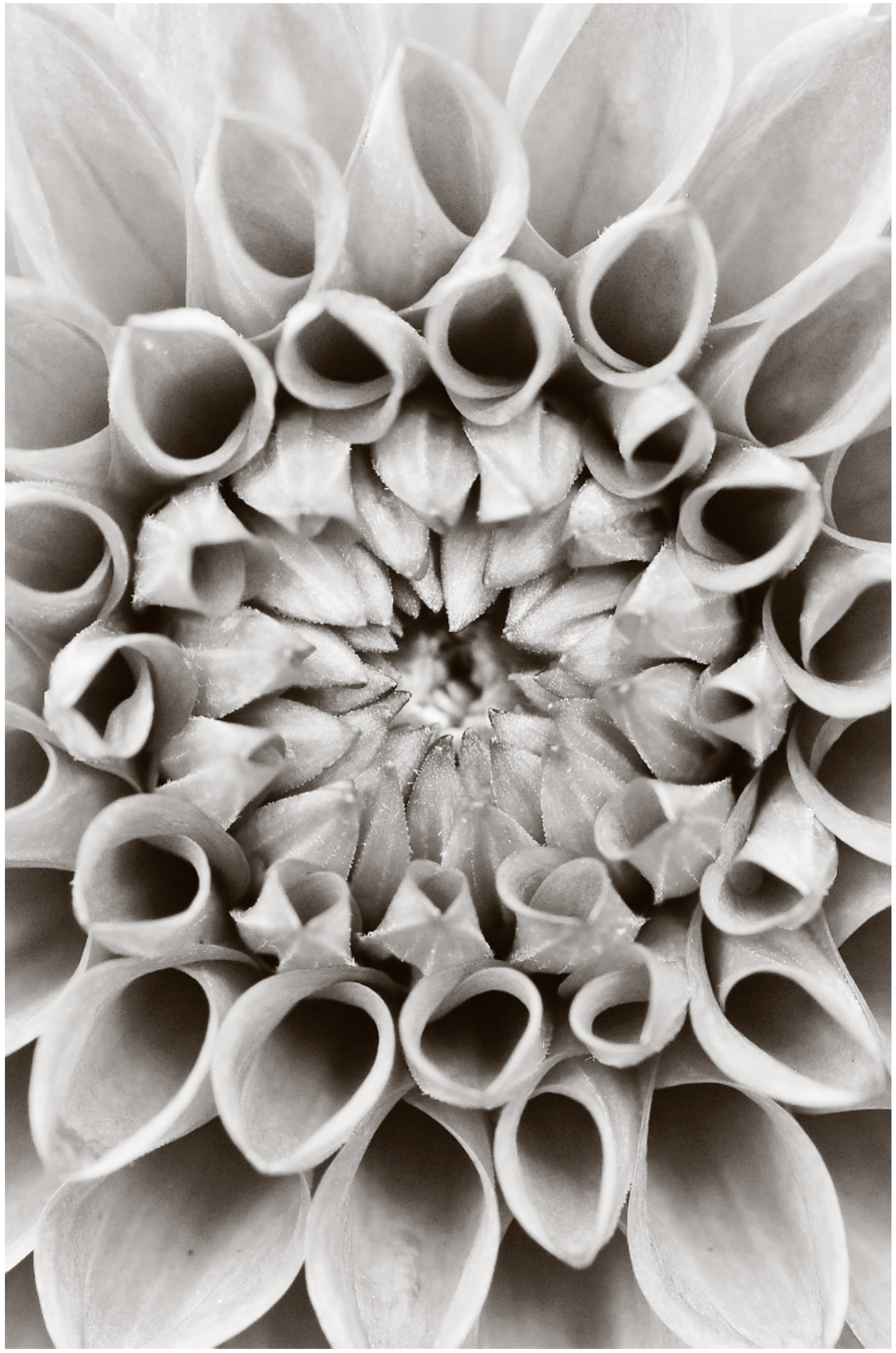
menu.js:

```
"use strict"
import React from 'react';
import {Nav, NavItem, Navbar, Badge} from
'react-bootstrap';

class Menu extends React.Component{
  render(){
    return(
      <Navbar inverse fixedTop>
    <Navbar.Header>
      <Navbar.Brand>
        <a href="/">React-Bootstrap</a>
      </Navbar.Brand>
      <Navbar.Toggle />
    </Navbar.Header>
    <Navbar.Collapse>
      <Nav>
        <NavItem eventKey={1}
href="/about">About</NavItem>
        <NavItem eventKey={2}
href="/contacts">Contact Us</NavItem>

      </Nav>
      <Nav pullRight>
        <NavItem eventKey={1}
href="/admin">Admin</NavItem>
        <NavItem eventKey={2}
href="/cart">Your Cart
{ (this.props.cartItemsNumber > 0)?(<Badge
className="badge">{this.props.cartItemsNumber
}</Badge>):('')}
          </NavItem>
      </Nav>
    </Navbar.Collapse>
  </Navbar>
    );
  }
}
export default Menu
//-------------------------------------------------------------------------
```

CHAPTER 3

Learn how to build an API service

In this chapter we will build a scalable Node-Express back-end with a web-server, an Api server and a proxy server.
We will also learn how to set-up a MongoDB database and use the learnings to build the database to feed the API service for the Shopping-cart application



SET-UP NODE-EXPRESS TO RUN A COMPLEX ARCHITECTURE

For our Shopping-cart we will build a relatively complex back-end architecture.

It wouldn't be a scalable approach to run everything in one single server, for this reason we will set up two separate servers: one to be used as web-service and the other to run our API as service. We will also implement an http-proxy in order to be compliant with browser security policy for Cross-Origin-Resource-Sharing.

In order to have a robust Express implementation, capable to be scaled up and handle and log all main errors, we will install Express-Generator.

SET-UP EXPRESS-GENERATOR:

1. ***Duplicate your package.json file and rename it. In fact, Express-Generator will overwrite package.json and you will loose your current list of dependencies.**
2. **In case you forgot to make a back-up , you will anyway find a copy of the final json file below :-)**
3. If you never installed Express command tool use the command: **npm install express-generator -g** *(in the doubt, you can run this command anyway)*
4. Run the command: ***express*** *(pls, in terminal check that you are inside reduxApp folder !!!)*
5. Download the new dependencies created by Express-Generator in the package.json file**:** **npm install**
6. Copy the old dependencies from your back-up copy of the package.json file *REMEMBER to add a "comma" when you paste the old dependencies int he new file.
7. From the back-up, copy the devDependences too
8. Edit app.js as below
9. Restart the server REMEMBER TO START YOU SERVER USING **npm start** COMMAND OR ALTERNATIVELY: ***node ./bin/www***

package.json:

```
{
  "name": "reduxapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "devDependencies": {
    "redux-logger": "^2.8.1",
    "webpack": "^2.2.1"
  },
  "dependencies": {
    "body-parser": "~1.16.0",
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.0",
    "express": "~4.14.1",
    "jade": "~1.11.0",
    "morgan": "~1.7.0",
    "serve-favicon": "~2.3.2",
    "babel-core": "^6.23.1",
    "babel-loader": "^6.3.2",
    "babel-preset-es2015": "^6.22.0",
    "babel-preset-react": "^6.23.0",
    "babel-preset-stage-1": "^6.22.0",
    "express": "^4.14.1",
    "react": "^15.4.2",
    "react-bootstrap": "^0.30.7",
    "react-dom": "^15.4.2",
    "react-redux": "^5.0.3",
    "react-router": "^3.0.2",
    "redux": "^3.6.0"
  }
}
//-------------------------------------------------------------
```

app.js:

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
// app.set('views', path.join(__dirname,
'views'));
// app.set('view engine', 'jade');

// uncomment after placing your favicon in
/public
//app.use(favicon(path.join(__dirname,
'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
```

```
app.use(bodyParser.urlencoded({ extended:
false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname,
'public')));

app.get('*', function(req, res){
  res.sendFile(path.resolve(__dirname,
'public', 'index.html'))
})

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in
development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') ===
'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
```

```
module.exports = app;
//---------------------------------------------------------------
```

NOSQL DATABASES

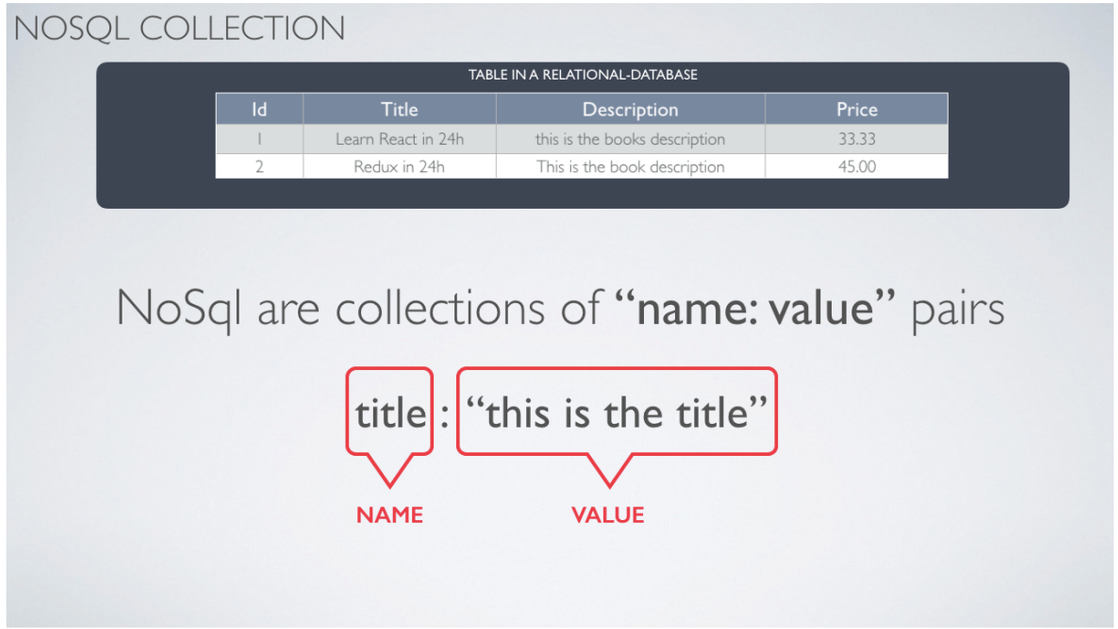
Thanks to their performances and flexibility to scale up, NoSQL have gained a lot of traction in the last few years.

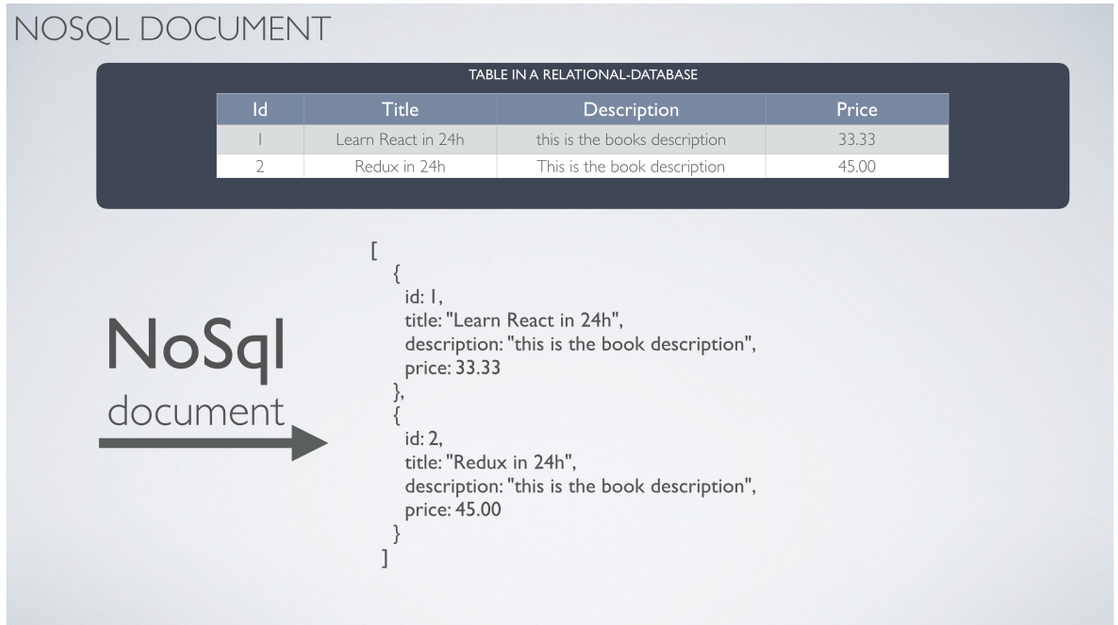
But what a NoSQL database is?

In a relational database, data is stored in tables that are related to each other to avoid repeating data and saving space.

In recent years cloud space became significantly cheaper and space is no longer a concern.

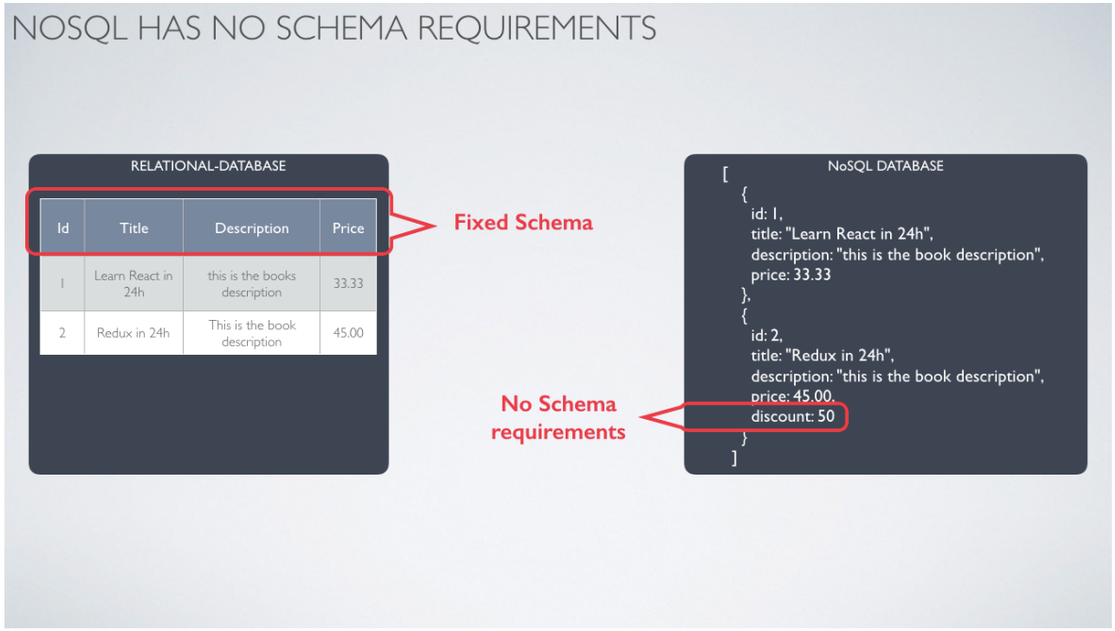
In a NoSql database, instead of tables, data is stored in “name-value” pair format called Json which you can easily read and write from your code because is a subset of javascript. In fact, if you take a closer look of a json file you would see that it contains javascript objects and arrays.



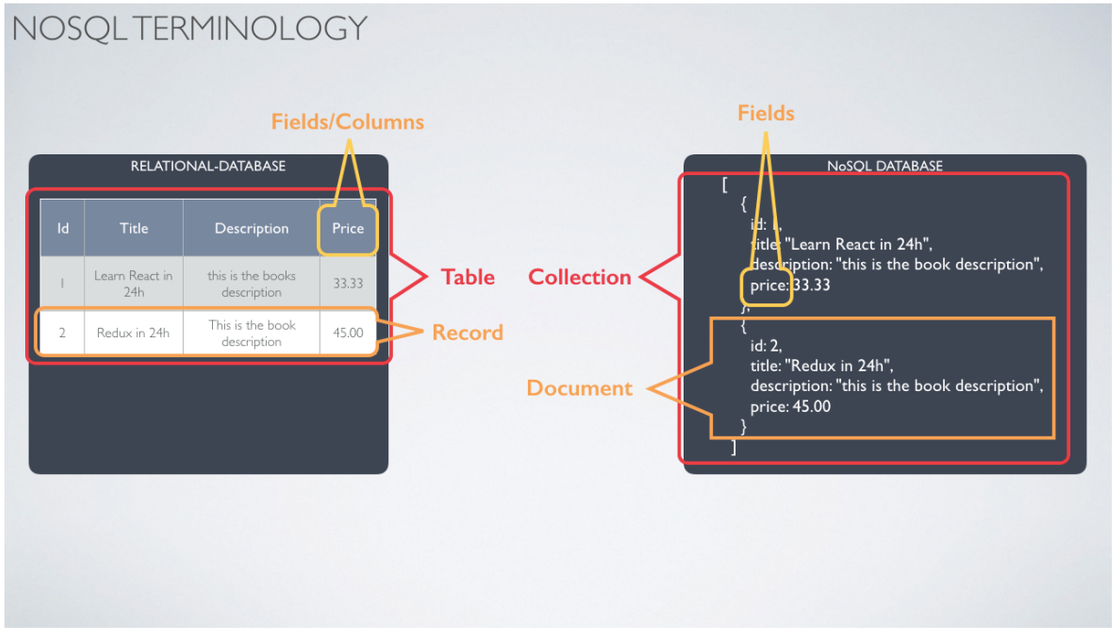


With a NoSQL database you don’t have the constraints related of being interfaced with a specific relational-database. You just create and consume data with great flexibility and performances.

In fact, NoSQL databases don't need to be tight to a fixed schema.



NOSQL TERMINOLOGY:



INSTALLATION OF MONGODB

You can install MongoDB in your computer or use third party solutions in the cloud to host your database.

We will first learn how to install and run MongoDB, but when we will deploy our shopping cart on a server online, we will need to use a third party cloud server.

INSTALLING MONGODB

1. Download MongoDb Community Server from mongoldb.com website.
2. Extract the file.
3. If you are **on Windows** just double click the executable and follow the instructions and specifying to install the complete version
4. **In a Mac**, rename your folder to something easy for you to remember (i.e. mongo) and drag & drop the folder in your user directory.
5. Create a folder where to save your databases. For instance you can create a "dbs" folder inside Mongo directory, but you can name this folder as you prefer.
6. From Terminal (or Command prompt in Windows) go inside the bin directory (i.e. /Mongo/bin) and start Mongo server launching mongoD using the following command:

./mongod --dbpath <your db absolute path> (i.e.: ./mongoD --dbpath ~/mongo/dbs). In Windows you use the same command but launching **mongo.exe in place of ./mongod**

At this point your server should start and you should see a message saying: "Waiting for connection on port 27017"

If you get any error check that your absolute path to the db folder you created is correct. Make sure you are launching mongod from inside your bin folder.

INSTALLING ROBOMONGO

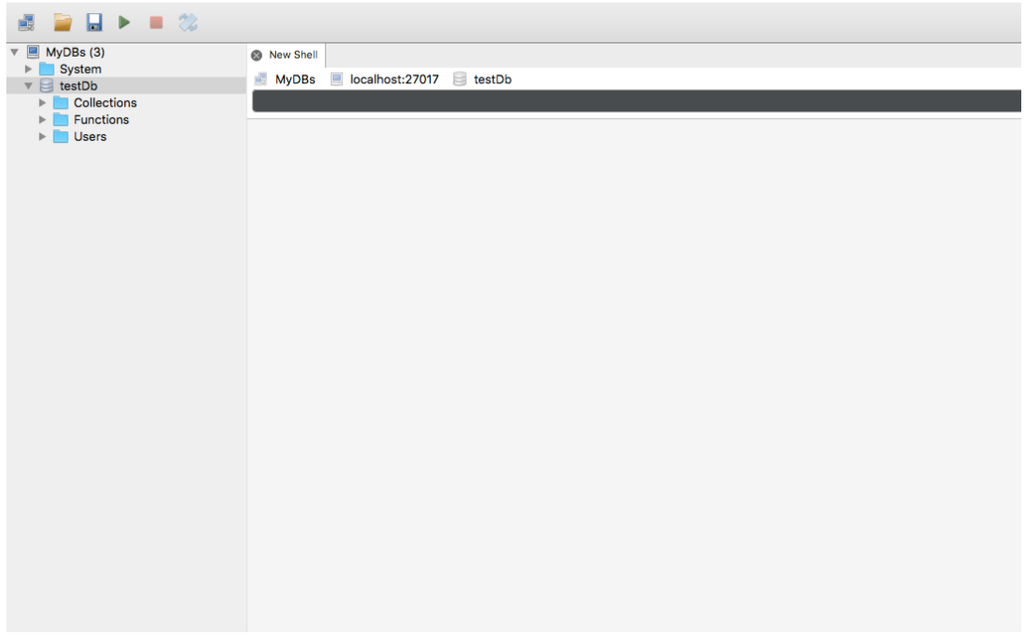
1. Download Robomongo from robomongo.org
2. Install Robomongo
3. Click on "create" to create a new connection
4. Give the connection a name and press "save" and then "connect"
5. At this point, you should be able to access your db

CRUD OPERATIONS IN MONGODB

In this lecture you will learn how to create and manage MongoDB databases.

CRUD OPERATIONS IN MONGODB

1. Start up you MongoDB server (./mongod --dbpath <your db absolute path>)
2. Open Robomongo and select your MongoDB connection and click on “connect”
3. Right click on the connection name and select “Create Database”
4. Give a name to the database i.e.: testDB
5. If your newly created database is not displayed, right-click and select: “refresh”
6. Right-click on the database name and select: “Open Shell”



CREATE OPERATION IN MONGODB

1. In the shell command tool.
2. Add the below query to insert a new record in the database and select “Execute” to run the
3. once you verify that one record was inserted, launch the same query but this time insert an array

Insert one book:

```
db.books.insert(
  {
```

```
    "title": "first book title",
    "description": "first book description",
    "price": 11
    }
)
```

MongoDB will automatically create a new collection “books”.

If you double click on the “books” collection name in Robomongo, it will automatically open a new tab and launch a query to read the data in “books” collection.

You should see that the record was inserted.

Insert an array of books:

```
db.books.insert(
  [
    {
     "title": "second book title",
     "description": "second book description",
     "price": 22
     },
     {
     "title": "second book title",
     "description": "second book description",
     "price": 33
     }
  ]
)
```

UPDATE OPERATION IN MONGODB

1. In the shell command tool.
2. Add the below query to update the title of one books and select “Execute” to run the query

Update title to one books in your database:

```
db.books.update(
   {"_id" : ObjectId("<HERE AN ID FROM YOUR DB>")},
   {$set: {"title": "title was updated"}}
)
```

3. In the shell command tool.
4. Add the below query. Since the “discount” field doesn’t exist, the $set operator will create the field automatically

Add a new field using the $set operator inside an update method:

```
db.books.update(
   {"_id" : ObjectId("<HERE AN ID FROM YOUR DB>")},
   {$set: {"discount": 20}}
)
```


6. In the shell command tool.
7. Change the last two digits in the _id of the books you just updated.
8. In your update action edit the title
9. Add the below query. Since the filter criteria will not match any _id, the “upset” option will create a new record

Add a new record if filter criteria doesn’t match, by adding “upset: true” option:

```
db.books.update(
   {"_id" : ObjectId("<HERE AN ID FROM YOUR DB>")},
   {$set: {"title": "this book didn't exist"}},
   {upsert: true}
)
```

UPDATE OPERATION IN MONGODB

10. In the shell command tool.
11. Set price $gt20 (greater then 20) as update criteria
12. Set a discout equal to 55 as “update action”
13. Add the below query. with the “multi: true” option, in order to update multiple records

Add a new field “discount” and set it to 55 in all records where price is “greater than” 20:

```
db.books.update(
   {"price" : {$gt:20}},
   {$set: {"discount": 55}},
   {multi: true}
)
```

DELETE OPERATION IN MONGODB

1. In the shell command tool.
2. use the “remove” method and specify an _id to be deleted
3. Add the below query.

Deletes a the record with the specified _id:

```
db.books.remove(
   {"_id" : ObjectId("<HERE AN ID FROM YOUR DB>")}
)
```

CREATE OUR API - POST METHOD

1. Install Mongoose: ***npm i --save mongoose***
2. Install Nodemon: ***npm i --save-dev nodemon***
3. Restart the server using the command: ***nodemon***
4. In the main app directory create a folder called: ***models***
5. Inside models, create a file file: ***books.js***
6. ***Edit books.js*** as below.
7. ***Edit app.js*** adding the code below

books.js:

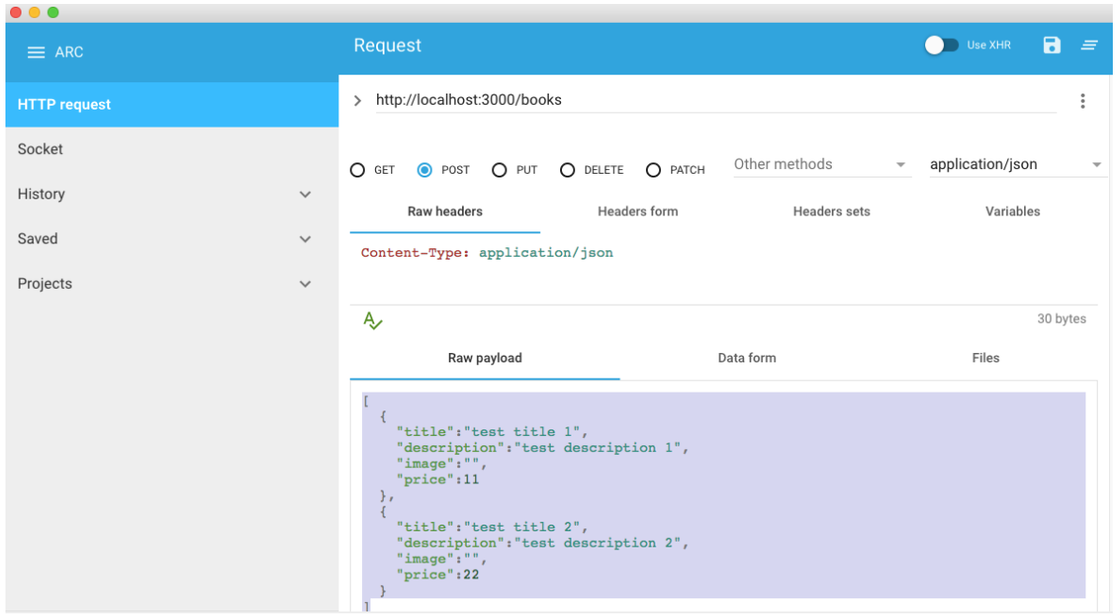
```
"use strict"
var mongoose = require('mongoose');

var booksSchema = mongoose.Schema({
  title: String,
  description: String,
  image: String,
  price: Number
});

var Books = mongoose.model('Books',
booksSchema);
module.exports = Books;
```

App.js:

```
// APIs
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/b
ookshop');

Books = require('./models/books.js');

//-----POST BOOKS----------
app.post('/books', function(req, res){
  var book = req.body;
  Books.create(book, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
});

// END APIs
```

TEST API

1. Download and Install **Advance Rest Client** from Chrome web-store
2. In ARC add the API web address to be tested: ***localhost:3000/books***
3. Select the method to be tested: ***POST***
4. Select "***application/json***" as content-type
5. Add an array of books in the raw data (below an example)
6. Press "***send***" button




```
[
  {
    "title":"test title 1",
    "description":"test description 1",
    "image":"",
    "price":11
  },
  {
    "title":"test title 2",
    "description":"test description 2",
    "image":"",
    "price":22
  }
]
```

app.js:

```
//------->>> GET BOOKS <<<-------
app.get('/books', function(req, res){
  Books.find(function(err, books){
    if(err){
      throw err;
```

```
    }
    res.json(books);
  })
});
```

CREATE OUR API - GET METHOD

1. ***Edit app.js*** adding the code below
2. In ARC add the API web address to be tested: ***localhost:3000/books***
3. Select the method to be tested: ***POST***
4. Select "***application/json***" as content-type
5. Press "***send***" button

app.js:

```
//----->>>> GET BOOKS <<<---------
app.get('/books', function(req, res){
  Books.find(function(err, books){
    if(err){
      throw err;
    }
    res.json(books)
  })
});
```

CREATE OUR API - DELETE METHOD

1. ***Edit app.js*** adding the code below
2. In ARC add the API web address to be tested: ***localhost:3000/books/<paste here the _id of the book you want to delete>***
3. Select the method to be tested: ***DELETE***
4. Select "***application/json***" as content-type

app.js:

```
//---->>> DELETE BOOKS <<<------
app.delete('/books/:_id', function(req, res){
  var query = {_id: req.params._id};

  Books.remove(query, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
});
```

CREATE OUR API - UPDATE METHOD

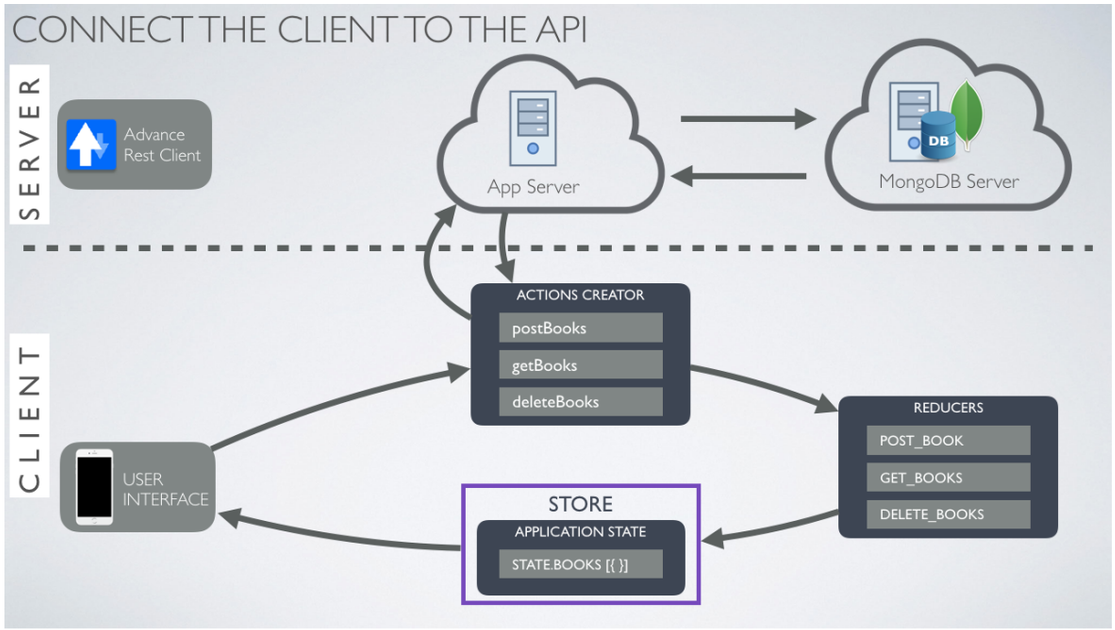
1. ***Edit app.js*** adding the code below
2. In ARC add the API web address to be tested: ***localhost:3000/books/<paste here the _id of the book you want to update>***
3. Select the method to be tested: ***PUT***
4. In the raw data , add a book object with new values for the property you want to update
5. Select "***application/json***" as content-type
6. Press "***send***" button

app.js:

```
//---->>> UPDATE BOOKS <<<------
app.put('/books/:_id', function(req, res){
  var book = req.body;
  var query = req.params._id;
  // if the field doesn't exist $set will set
a new field
  var update = {
    '$set':{
      title:book.title,
      description:book.description,
      image:book.image,
      price:book.price
    }
  };
    // When true returns the updated document
    var options = {new: true};

    Books.findOneAndUpdate(query, update,
options, function(err, books){
      if(err){
        throw err;
      }
      res.json(books);
    })

})
```

CONNECTING THE API TO THE APP

At this point we need to enable the Action Creator to make XmlHttpRequest to the API in order to integrate the Client app with the newly created API:



CONNECT THE API TO THE APP - PART 2

1. install Axios: **npm i --save axios**
2. install Redux-Thunk: **npm i --save redux-thunk**
3. To avoid confusion with the server app, **rename app.js** inside the src folder as: **client.js**
4. **Edit postBooks** function in booksActions

In client.js:

5. **import thunk** from 'redux-thunk'
6. add thunk to applyMiddleware method
7. **Test it** by adding a new book

booksActions:

```
// POST A BOOK
export function postBooks(book){
  return function(dispatch){
    axios.post("/books", book)
      .then(function(response){
        dispatch({type:"POST_BOOK",
payload:response.data})
```

```
      })
      .catch(function(err){
        dispatch({type:"POST_BOOK_REJECTED",
payload:"there was an error while posting a
new book"})
      })
  }
}
```

client.js:

```
"use strict"
// REACT
import React from 'react';
import {render} from 'react-dom';
import {Provider} from 'react-redux';
// REACT-ROUTER
import {Router, Route, IndexRoute,
browserHistory} from 'react-router';

import {applyMiddleware, createStore} from
'redux';
import logger from 'redux-logger';
import thunk from 'redux-thunk';

// IMPORT COMBINED REDUCERS
import reducers from './reducers/index';
// IMPORT ACTIONS
import {addToCart} from
'./actions/cartActions';
// STEP 1 create the store
const middleware =applyMiddleware(thunk,
logger());
const store = createStore(reducers,
middleware);

import BooksList from
'./components/pages/booksList';
import Cart from './components/pages/cart';
import BooksForm from
'./components/pages/booksForm';
import Main from './main';

const Routes = (
  <Provider store={store}>
      <Router history={browserHistory}>
        <Route path="/" component={Main}>
            <IndexRoute
component={BooksList}/>
            <Route path="/admin"
component={BooksForm}/>
            <Route path="/cart"
component={Cart}/>
        </Route>
      </Router>
  </Provider>
)

render(
  Routes, document.getElementById('app')
);
```

CONNECT THE API TO THE APP - PART 3

in booksReducers.js:

1. **Change book initial state to be an empty array of books: state:{books:[]}**
2. In the GET_BOOKS reducers **replace the "state.books" with "action.payload"**

in booksActions.js:

3. **Edit getBooks** function in booksActions
4. **Test it** to see if you get books saved in the database visible in the UI
5. **Edit deleteBooks** function in booksActions
6. **Test it** by deleting a book

booksReducers.js:

```
//BOOKS REDUCERS
export function booksReducers(state={
  books:[ ]
}, action){
  switch(action.type){
    case "GET_BOOKS":
      return {...state,
books:[...action.payload]}
      break;
```

..... interrupted.....

booksActions:

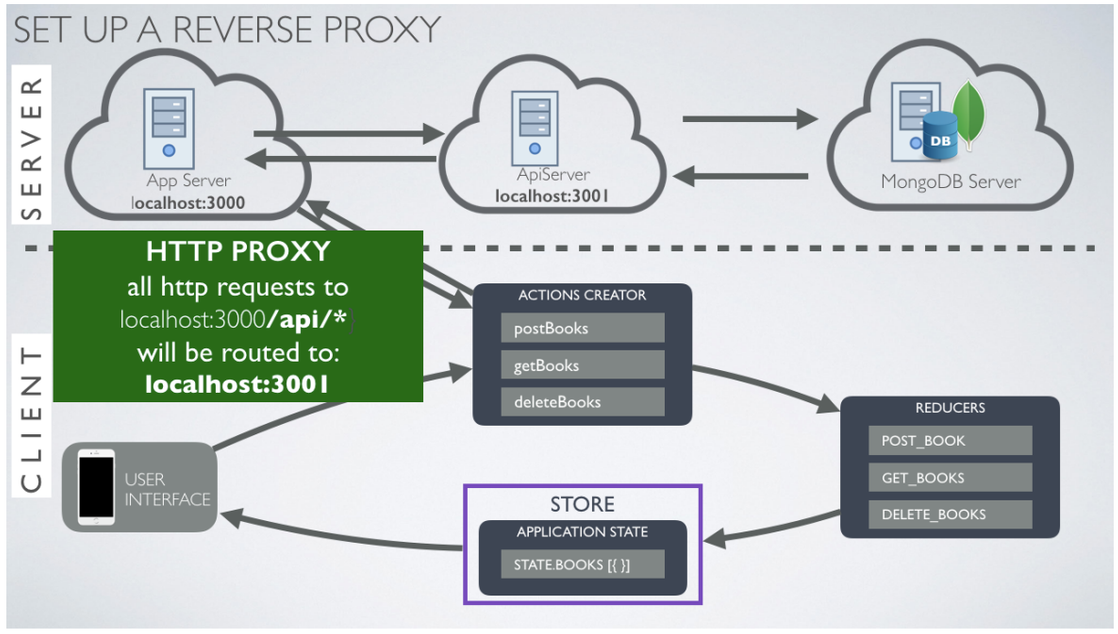
```
"use strict"
import axios from 'axios';
// GET A BOOK
export function getBooks(){
  return function(dispatch){
    axios.get("/books")
      .then(function(response){
        dispatch({type:"GET_BOOKS",
payload:response.data})
      })
      .catch(function(err){
        dispatch({type:"GET_BOOKS_REJECTED",
payload:err})
      })
  }
}
// POST A BOOK
export function postBooks(book){
  return function(dispatch){
    axios.post("/books", book)
      .then(function(response){
```

```
        dispatch({type:"POST_BOOK",
payload:response.data})
      })
      .catch(function(err){
        dispatch({type:"POST_BOOK_REJECTED",
payload:"there was an error while posting a
new book"})
      })
  }
}

// DELETE A BOOK
export function deleteBooks(id){
  return function(dispatch){
    axios.delete("/books/" + id)
      .then(function(response){
        dispatch({type:"DELETE_BOOK",
payload:id})
      })
      .catch(function(err){

dispatch({type:"DELETE_BOOK_REJECTED",
payload:err})
      })
  }
}
```

SET UP A REVERSE PROXY

A reverse proxy is a type of proxy server that retrieves data on behalf of a client from one or more servers.

Basically, we will create an additional server for our API and we will set up a proxy in the web-server to retrieve the data from the API-server.



SET UP A REVERSE PROXY`Z

1. **right-click on app.js and duplicate the file and name it: apiServer.js**
2. **Clean the apiServer.js** file from all the requires and middleware we don't need
3. Set the apiServer to listen on port 3001
4. From Terminal run: **npm i --save http-proxy**

in app.js:

5. Clean the app.js file from all the requires and middleware we don't need
6. **Require "http-proxy"** and create the proxy to intercept all request to /api

in booksAction.js:

7. Edit all API call URL to **include "/api/"**

in package.json:

8. Edit add the command to start apiServer in "start": **"start": "node ./bin/www & node apiServer.js"**
9. **Make sure you run webpack and nodemon!!**

apiServer.js:

```
"use strict"
var express = require('express');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended:
false }));
app.use(cookieParser());


// APIs
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/b
ookshop');

var Books = require('./models/books.js');

//---->>> POST BOOKS <<<-----
app.post('/books', function(req, res){
  var book = req.body;

  Books.create(book, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
```

```
});

//----->>>> GET BOOKS <<<---------
app.get('/books', function(req, res){
  Books.find(function(err, books){
    if(err){
      throw err;
    }
    res.json(books)
  })
});

//---->>> DELETE BOOKS <<<------
app.delete('/books/:_id', function(req, res){
  var query = {_id: req.params._id};

  Books.remove(query, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
});

//---->>> UPDATE BOOKS <<<------
app.put('/books/:_id', function(req, res){
  var book = req.body;
  var query = req.params._id;
  // if the field doesn't exist $set will set
a new field
  var update = {
    '$set':{
      title:book.title,
      description:book.description,
      image:book.image,
      price:book.price
    }
  };
    // When true returns the updated document
    var options = {new: true};

    Books.findOneAndUpdate(query, update,
options, function(err, books){
      if(err){
        throw err;
      }
      res.json(books);
    })

})

// END APIs

app.listen(3001, function(err){
  if(err){
    return console.log(err);
  }
  console.log('API Sever is listening on
http://localhost:3001');
});
//-------------------------------------------------------------
```

app.js:

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
//PROXY
var httpProxy = require('http-proxy');

var app = express();

app.use(logger('dev'));

//PROXY TO API
const apiProxy =
httpProxy.createProxyServer({
  target:"http://localhost:3001"
});
app.use('/api', function(req, res){
  apiProxy.web(req, res);
})
// END PROXY

// uncomment after placing your favicon in
/public
//app.use(favicon(path.join(__dirname,
'public', 'favicon.ico')));
app.use(express.static(path.join(__dirname,
'public')));
app.get('*', function(req, res){
  res.sendFile(path.resolve(__dirname,
'public', 'index.html'))
})

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in
development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') ===
'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
//----------------------------------------------------------------
```

booksActions.js:

```
"use strict"
import axios from 'axios';
// GET A BOOK
export function getBooks(){
  return function(dispatch){
    axios.get("/api/books")
      .then(function(response){
        dispatch({type:"GET_BOOKS",
payload:response.data})
      })
      .catch(function(err){
        dispatch({type:"GET_BOOKS_REJECTED",
payload:err})
      })
  }
}
// POST A BOOK
export function postBooks(book){
  return function(dispatch){
    axios.post("/api/books", book)
      .then(function(response){
        dispatch({type:"POST_BOOK",
payload:response.data})
      })
      .catch(function(err){
        dispatch({type:"POST_BOOK_REJECTED",
payload:"there was an error while posting a
new book"})
      })
  }
}
// DELETE A BOOK
export function deleteBooks(id){
  return function(dispatch){
    axios.delete("/api/books/" + id)
      .then(function(response){
        dispatch({type:"DELETE_BOOK",
payload:id})
      })
      .catch(function(err){

dispatch({type:"DELETE_BOOK_REJECTED",
payload:err})
      })
  }
}

// UPDATE A BOOK
export function updateBooks(book){
  return {
          type:"UPDATE_BOOK",
          payload: book
        }
}
//-------------------------------------------------------------
```

package.json:

```
 "scripts": {
"start": "node ./bin/www & node apiServer.js"
 },
```

MAKE A PERSISTENT SHOPPING-CART

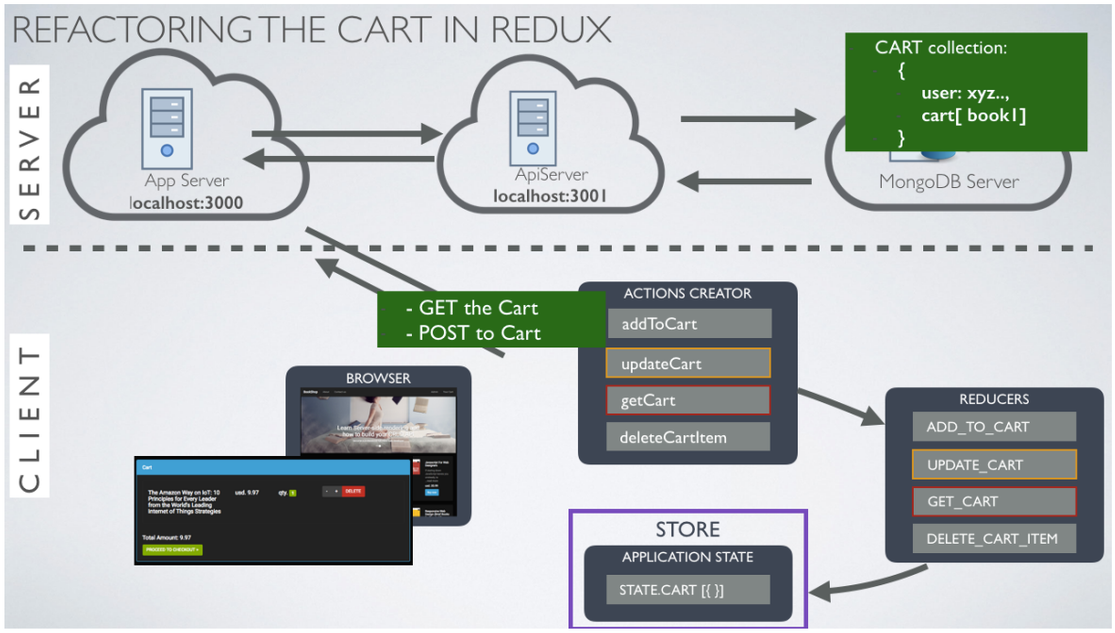
You might have noticed that if you refresh the browser cart data is lost.

In order to create a persistent shopping-cart we will need to identify anonymous user by tracking the sessions.

Each session will have a Session Id that will be stored in the client browser to recognize the specific user.

Beside tracking the session we will store cart data in MongoDB associated to the session ID.

To make this work, we will need to set up sessions management in Express, add new requests to our API that will deal with the users cart data and finally refactor our cartActions to post and get data from the API and forward the responses to the Reducer to align the cart data in the state with the one in the database.



MAKE A PERSISTENT SHOPPING-CART

1. From Terminal run: **npm i --save connect-mongo**
2. From Terminal run: **npm i --save express-session**

in apiServer.js:

3. Require connect-mongo and express-session
4. edit apiServer.js as below

in cartAction.js:

5. Edit all cartActions.js file as below

in cartReducers:

6. Edit cartReducers as below

in cart.js:

7. import getCart and dispatch the getCart action from componentDidMount()

in bookItem.js:

8. Edit bookItem as below

apiServer.js:

```
"use strict"
var express = require('express');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
const session = require('express-session');
const MongoStore =
require('connect-mongo')(session);

var app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended:
false }));
app.use(cookieParser());


// APIs
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/b
ookshop');

var db = mongoose.connection;
db.on('error', console.error.bind(console, '#
MongoDB - connection error: '));

// --->>> SET UP SESSIONS <<<----
app.use(session({
  secret: 'mySecretString',
  saveUninitialized: false,
  resave:false,
  store: new MongoStore({mongooseConnection:
db, ttl: 2 * 24 * 60 * 60})
  //ttl: 2 days * 24 hours * 60 minutes * 60
seconds
}))
  // SAVE SESSION CART API
  app.post('/cart', function(req, res){
    var cart = req.body;
    req.session.cart = cart;
    req.session.save(function(err){
      if(err){
        throw err;
      }
      res.json(req.session.cart);
    })
  });
  // GET SESSION CART API
  app.get('/cart', function(req, res){
    if(typeof req.session.cart !==
'undefined'){
      res.json(req.session.cart);
    }
  });
//--->>> END SESSION SET UP <<<----


var Books = require('./models/books.js');

//--->>> POST BOOKS <<<-----
app.post('/books', function(req, res){
  var book = req.body;
```

```
  Books.create(book, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
});

//----->>>> GET BOOKS <<<---------
app.get('/books', function(req, res){
  Books.find(function(err, books){
    if(err){
      throw err;
    }
    res.json(books)
  })
});

//---->>> DELETE BOOKS <<<------
app.delete('/books/:_id', function(req, res){
  var query = {_id: req.params._id};

  Books.remove(query, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
});
```

```
//---->>> UPDATE BOOKS <<<------
app.put('/books/:_id', function(req, res){
  var book = req.body;
  var query = req.params._id;
  // if the field doesn't exist $set will set
a new field
  var update = {
    '$set':{
      title:book.title,
      description:book.description,
      image:book.image,
      price:book.price
    }
  };
    // When true returns the updated document
    var options = {new: true};

    Books.findOneAndUpdate(query, update,
options, function(err, books){
      if(err){
        throw err;
      }
      res.json(books);
    })

})

// END APIs

app.listen(3001, function(err){
  if(err){
```

```
    return console.log(err);
  }
  console.log('API Sever is listening on
http://localhost:3001');
});
//-------------------------------------------------------------------
```

cartActions.js:

```
"use strict"
import axios from 'axios';
// GET CART
export function getCart(){
  return function(dispatch){
    axios.get('/api/cart')
      .then(function(response){
        dispatch({type:"GET_CART",
payload:response.data})
      })
      .catch(function(err){
        dispatch({type:"GET_CART_REJECTED",
msg:"error when getting the cart from
session"})
      })
  }
}
// ADD TO CART
export function addToCart(cart){
  return function(dispatch){
    axios.post("/api/cart", cart)
      .then(function(response){
        dispatch({type:"ADD_TO_CART",
payload:response.data})
      })
      .catch(function(err){
        dispatch({type:"ADD_TO_CART_REJECTED", msg:
'error when adding to the cart'})
      })
  }
}
// UPDATE CART
export function updateCart(_id, unit, cart){
  // Create a copy of the current array of
books
  const currentBookToUpdate = cart
  // Determine at which index in books array
is the book to be deleted
  const indexToUpdate =
currentBookToUpdate.findIndex(
    function(book){
      return book._id === _id;
    }
  )

  const newBookToUpdate = {
    ...currentBookToUpdate[indexToUpdate],
    quantity:
currentBookToUpdate[indexToUpdate].quantity +
unit
  }
```

```
  let cartUpdate =
[...currentBookToUpdate.slice(0,
indexToUpdate), newBookToUpdate,
...currentBookToUpdate.slice(indexToUpdate +
1)]

  return function(dispatch){
    axios.post("/api/cart", cartUpdate)
      .then(function(response){
        dispatch({type:"UPDATE_CART",
payload:response.data})
      })
      .catch(function(err){

dispatch({type:"UPDATE_CART_REJECTED", msg:
'error when adding to the cart'})
      })
  }
}
// DELETE FROM CART
export function deleteCartItem(cart){
  return function(dispatch){
    axios.post("/api/cart", cart)
      .then(function(response){
        dispatch({type:"DELETE_CART_ITEM",
payload:response.data})
      })
      .catch(function(err){

dispatch({type:"DELETE_CART_ITEM_REJECTED",
msg: 'error when deleting an item from the
cart'})
      })
  }
}
//-------------------------------------------------------------
```

cartReducers.js:

```
"use strict"

// CART REDUCERS
export function cartReducers(state={cart:[]},
action) {
  switch(action.type){
    case "GET_CART":
    return{...state,
      cart:action.payload,

totalAmount:totals(action.payload).amount,
      totalQty: totals(action.payload).qty
    }
    break;
    case "ADD_TO_CART":
    return {...state,
      cart:action.payload,
      totalAmount:
totals(action.payload).amount,
      totalQty: totals(action.payload).qty
    }
```

```
    break;
    case "UPDATE_CART":
    return {...state,
      cart:action.payload,
      totalAmount:
totals(action.payload).amount,
      totalQty: totals(action.payload).qty
    }
    break;
    case "DELETE_CART_ITEM":
    return {...state,
      cart:action.payload,
      totalAmount:
totals(action.payload).amount,
      totalQty: totals(action.payload).qty
    }
    break;
  }
  return state
}

// CALCULATE TOTALS
export function totals(payloadArr){

  const totalAmount =
payloadArr.map(function(cartArr){
    return cartArr.price * cartArr.quantity;
  }).reduce(function(a, b) {
    return a + b;
  }, 0); //start summing from index0

  const totalQty =
payloadArr.map(function(qty){
    return qty.quantity;
  }).reduce(function(a, b) {
    return a + b;
  }, 0);

  return {amount:totalAmount.toFixed(2),
qty:totalQty}
}
//-------------------------------------------------------------
```

cart.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {Modal, Panel, Col, Row, Well, Button,
ButtonGroup, Label} from 'react-bootstrap';
import {bindActionCreators} from 'redux';
import {deleteCartItem, updateCart, getCart}
from '../../actions/cartActions';

class Cart extends React.Component{
  componentDidMount(){
    this.props.getCart();
  }
  onDelete(_id){
```

```
    // Create a copy of the current array of
books
    const currentBookToDelete =
this.props.cart;
    // Determine at which index in books
array is the book to be deleted
    const indexToDelete =
currentBookToDelete.findIndex(
      function(cart){
        return cart._id === _id;
      }
    )
    //use slice to remove the book at the
specified index
    let cartAfterDelete =
[...currentBookToDelete.slice(0,
indexToDelete),
...currentBookToDelete.slice(indexToDelete +
1)]


this.props.deleteCartItem(cartAfterDelete);
  }
  onIncrement(_id){
    this.props.updateCart(_id, 1,
this.props.cart);
  }
  onDecrement(_id, quantity){
    if(quantity > 1){
      this.props.updateCart(_id, -1,
this.props.cart);
  }
}
constructor(){
  super();
  this.state = {
    showModal:false
  }
}
open(){
  this.setState({showModal:true})
}
close(){
  this.setState({showModal:false})
}

render(){
  if(this.props.cart[0]){
    return this.renderCart();
  } else {
    return this.renderEmpty();
  }
}
renderEmpty(){
  return(<div></div>)
}

renderCart(){
  const cartItemsList =
this.props.cart.map(function(cartArr){
    return(
      <Panel key={cartArr._id}>
```

```
            <Row>
              <Col xs={12} sm={4}>
                <h6>{cartArr.title}</h6><span>
</span>
              </Col>
              <Col xs={12} sm={2}>
                <h6>usd. {cartArr.price}</h6>
              </Col>
              <Col xs={12} sm={2}>
                <h6>qty. <Label
bsStyle="success">{cartArr.quantity}</Label><
/h6>
              </Col>
              <Col xs={6} sm={4}>
                <ButtonGroup
style={{minWidth:'300px'}}>
                  <Button
onClick={this.onDecrement.bind(this,
cartArr._id, cartArr.quantity)}
bsStyle="default" bsSize="small">-</Button>
                  <Button
onClick={this.onIncrement.bind(this,
cartArr._id)} bsStyle="default"
bsSize="small">+</Button>
                  <span>     </span>
                  <Button
onClick={this.onDelete.bind(this,
cartArr._id)} bsStyle="danger"
bsSize="small">DELETE</Button>
                </ButtonGroup>
              </Col>
            </Row>
          </Panel>
        )
      }, this)
      return(
        <Panel header="Cart" bsStyle="primary">
          {cartItemsList}
          <Row>
            <Col xs={12}>
              <h6>Total amount:
{this.props.totalAmount}</h6>
              <Button
onClick={this.open.bind(this)}
bsStyle="success" bsSize="small">
                PROCEED TO CHECKOUT
              </Button>
            </Col>
          </Row>
          <Modal show={this.state.showModal}
onHide={this.close.bind(this)}>
            <Modal.Header closeButton>
              <Modal.Title>Thank
you!</Modal.Title>
            </Modal.Header>
            <Modal.Body>
              <h6>Your order has been
saved</h6>
              <p>You will receive an email
confirmation</p>
            </Modal.Body>
            <Modal.Footer>
```

```
            <Col xs={6}>
              <h6>total $:
{this.props.totalAmount}</h6>
            </Col>
            <Button
onClick={this.close.bind(this)}>Close</Button
>
          </Modal.Footer>
        </Modal>
      </Panel>
    )
  }
}
function mapStateToProps(state){
  return{
    cart: state.cart.cart,
    totalAmount: state.cart.totalAmount,
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    deleteCartItem:deleteCartItem,
    updateCart:updateCart,
    getCart:getCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(Cart);
//-------------------------------------------------------------
```

bookItem.js:

```
"use strict"
import React from 'react';
import {Row, Col, Well, Button} from
'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux'
import {addToCart, updateCart} from
'../../actions/cartActions';

class BookItem extends React.Component{

  handleCart(){
    const book = [...this.props.cart, {
      _id:this.props._id,
      title:this.props.title,
      description:this.props.description,
      price:this.props.price,
      quantity:1
    }]

    // CHECK IF CART IS EMPTY
    if(this.props.cart.length > 0) {
      // CART IS NOT EMPTY
      let _id = this.props._id;

      let cartIndex =
this.props.cart.findIndex(function(cart){
        return cart._id === _id;
      })
```

```
      // IF RETURNS -1 THERE ARE NO ITEMS
WITH SAME ID
      if (cartIndex === -1){
        this.props.addToCart(book);
      } else {
        // WE NEED TO UPDATE QUANTITY
        this.props.updateCart(_id, 1,
this.props.cart);
      }

    } else {
      // CART IS EMPTY
      this.props.addToCart(book);
    }

  }

  render(){
    return(
      <Well>
        <Row>
          <Col xs={12}>
            <h6>{this.props.title}</h6>
            <p>{this.props.description}</p>
            <h6>usd. {this.props.price}</h6>
            <Button
onClick={this.handleCart.bind(this)}
bsStyle='primary'>Buy now</Button>

          </Col>
        </Row>
        </Well>
      )
    }
}
function mapStateToProps(state){
  return{
    cart:state.cart.cart
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    addToCart:addToCart,
    updateCart:updateCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BookItem);
//-------------------------------------------------------------
```

BOOKSFORM - IMAGE SELECTOR

in apiServer.js

1. Add the API to read images file names from images directory in the server

in booksForm.js:

2. import InputGroup with DropDownButton, FromGroup, MenuItem, Col and Row from React-bootstrap
3. edit the code as below

in booksList.js:

4. remove the booksForm from the component
5. Add the “images” property to be passed down to

apiServer.js:

```
"use strict"
var express = require('express');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
const session = require('express-session');
const MongoStore =
require('connect-mongo')(session);
```

```
var app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended:
false }));
app.use(cookieParser());


// APIs
var mongoose = require('mongoose');
mongoose.connect('mongodb://testUser:test@ds0
43012.mlab.com:43012/bookshop')

var db = mongoose.connection;
db.on('error', console.error.bind(console, '#
MongoDB - connection error: '));

// --->>> SET UP SESSIONS <<<----
app.use(session({
  secret: 'mySecretString',
  saveUninitialized: false,
  resave:false,
  store: new MongoStore({mongooseConnection:
db, ttl: 2 * 24 * 60 * 60})
  //ttl: 2 days * 24 hours * 60 minutes * 60
seconds
}))
  // SAVE SESSION CART API
  app.post('/cart', function(req, res){
    var cart = req.body;
```

```
    req.session.cart = cart;
    req.session.save(function(err){
      if(err){
        throw err;
      }
      res.json(req.session.cart);
    })
  });
  // GET SESSION CART API
  app.get('/cart', function(req, res){
    if(typeof req.session.cart !==
'undefined'){
      res.json(req.session.cart);
    }
  });
//--->>> END SESSION SET UP <<<----


var Books = require('./models/books.js');

//---->>> POST BOOKS <<<-----
app.post('/books', function(req, res){
  var book = req.body;

  Books.create(book, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
});
```

```
//----->>>> GET BOOKS <<<---------
app.get('/books', function(req, res){
  Books.find(function(err, books){
    if(err){
      throw err;
    }
    res.json(books)
  })
});

//---->>> DELETE BOOKS <<<------
app.delete('/books/:_id', function(req, res){
  var query = {_id: req.params._id};

  Books.remove(query, function(err, books){
    if(err){
      throw err;
    }
    res.json(books);
  })
});

//---->>> UPDATE BOOKS <<<------
app.put('/books/:_id', function(req, res){
  var book = req.body;
  var query = req.params._id;
  // if the field doesn't exist $set will set
a new field
  var update = {
    '$set':{
```

```
      title:book.title,
      description:book.description,
      image:book.image,
      price:book.price
    }
  };
    // When true returns the updated document
    var options = {new: true};

    Books.findOneAndUpdate(query, update,
options, function(err, books){
      if(err){
        throw err;
      }
      res.json(books);
    })

})

  // --->>> GET BOOKS IMAGES API <<<------
  app.get('/images', function(req, res){

    const imgFolder = __dirname +
'/public/images/';
    // REQUIRE FILE SYSTEM
    const fs = require('fs');
    //READ ALL FILES IN THE DIRECTORY
    fs.readdir(imgFolder, function(err,
files){
      if(err){
        return console.error(err);
      }
      //CREATE AN EMPTY ARRAY
      const filesArr = [];
      var i = 1;
      // ITERATE ALL IMAGES IN THE DIRECTORY
AND ADD TO THE THE ARRAY
      files.forEach(function(file){
        filesArr.push({name: file});
        i++
      });
      // SEND THE JSON RESPONSE WITH THE
ARARY
      res.json(filesArr);
    })
  })

// END APIs

app.listen(3001, function(err){
  if(err){
    return console.log(err);
  }
  console.log('API Sever is listening on
http://localhost:3001');
});
//-------------------------------------------------------------
```

booksForm.js:

```
"use strict"
import React from 'react';
import {MenuItem, InputGroup, DropdownButton,
Image, Col, Row, Well, Panel, FormControl,
FormGroup, ControlLabel, Button} from
'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';
import {findDOMNode} from 'react-dom';
import {postBooks, deleteBooks, getBooks}
from '../../actions/booksActions';
import axios from 'axios';

class BooksForm extends React.Component{
  constructor() {
    super();
    this.state = {
      images:[{}],
      img:''
    }
  }
  componentDidMount(){
    this.props.getBooks();
    //GET IMAGES FROM API
    axios.get('/api/images')
      .then(function(response){

this.setState({images:response.data});
      }.bind(this))
      .catch(function(err){
        this.setState({images:'error loading
image files from the server', img:''})
      }.bind(this))
  }
  handleSubmit(){
    const book=[{
      title:
findDOMNode(this.refs.title).value,
      description:
findDOMNode(this.refs.description).value,

images:findDOMNode(this.refs.image).value,
      price:
findDOMNode(this.refs.price).value,
    }]
    this.props.postBooks(book);
  }

  onDelete(){
    let bookId =
findDOMNode(this.refs.delete).value;

    this.props.deleteBooks(bookId);
  }
  handleSelect(img){
    this.setState({
      img: '/images/'+ img
    })
  }
  render(){
```

```
    const booksList =
this.props.books.map(function(booksArr){
      return (
        <option key={booksArr._id}>
{booksArr._id}</option>
      )
    })

    const imgList =
this.state.images.map(function(imgArr, i){
      return(
        <MenuItem key={i}
eventKey={imgArr.name}

onClick={this.handleSelect.bind(this,
imgArr.name)}>{imgArr.name}</MenuItem>
      )
    }, this)

    return(
      <Well>
        <Row>
          <Col xs={12} sm={6}>
            <Panel>
              <InputGroup>
                <FormControl type="text"
ref="image" value={this.state.img} />
                <DropdownButton

componentClass={InputGroup.Button}
                  id="input-dropdown-addon"
                  title="Select an image"
                  bsStyle="primary">
                  {imgList}
                </DropdownButton>
              </InputGroup>
              <Image src={this.state.img}
responsive/>
            </Panel>
          </Col>
          <Col xs={12} sm={6}>
            <Panel>
              <FormGroup controlId="title">

<ControlLabel>Title</ControlLabel>
                <FormControl
                  type="text"
                  placeholder="Enter Title"
                  ref="title" />
              </FormGroup>
              <FormGroup
controlId="description">

<ControlLabel>Description</ControlLabel>
                <FormControl
                  type="text"
                  placeholder="Enter
Description"
                  ref="description" />
              </FormGroup>
              <FormGroup controlId="price">
```

```
<ControlLabel>Price</ControlLabel>
                    <FormControl
                        type="text"
                        placeholder="Enter Price"
                        ref="price" />
                </FormGroup>
                <Button
onClick={this.handleSubmit.bind(this)}
bsStyle="primary">Save book</Button>
            </Panel>
            <Panel>
              <FormGroup
controlId="formControlsSelect">
                    <ControlLabel>Select a book
id to delete</ControlLabel>
                    <FormControl ref="delete"
componentClass="select" placeholder="select">
                        <option
value="select">select</option>
                        {booksList}
                    </FormControl>
                </FormGroup>
                <Button
onClick={this.onDelete.bind(this)}
bsStyle="danger">Delete book</Button>
            </Panel>
          </Col>
        </Row>

      </Well>
```

```
    )
  }
}
function mapStateToProps(state){
  return {
    books: state.books.books
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    postBooks,
    deleteBooks,
    getBooks
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksForm);
//----------------------------------------------------------
```

booksList.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {getBooks} from
'../../actions/booksActions';
import {bindActionCreators} from 'redux';
import {Grid, Col, Row, Button} from
'react-bootstrap';
```

```
import BookItem from './bookItem';
import BooksForm from './booksForm';
import Cart from './cart';


class BooksList extends React.Component{
  componentDidMount(){
    this.props.getBooks()
  }
  render(){
    const booksList =
this.props.books.map(function(booksArr){
      return(
        <Col xs={12} sm={6} md={4}
key={booksArr._id}>
          <BookItem
                _id= {booksArr._id}
                title={booksArr.title}

description={booksArr.description}
                images={booksArr.images}
                price={booksArr.price}/>
        </Col>
      )
    })
    return(
        <Grid>
          <Row>
            <Cart />
          </Row>
          <Row>
                {booksList}
          </Row>
        </Grid>
    )
  }
}
function mapStateToProps(state){
  return{
    books: state.books.books
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    getBooks:getBooks
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksList);
//-----------------------------------------------------------------
```

bookItem.js:

```
"use strict"
import React from 'react';
import {Image, Row, Col, Well, Button} from
'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux'
import {addToCart, updateCart} from
'../../actions/cartActions';
```

```
class BookItem extends React.Component{

  handleCart(){
    const book = [...this.props.cart, {
      _id:this.props._id,
      title:this.props.title,
      description:this.props.description,
      images: this.props.images,
      price:this.props.price,
      quantity:1
    }]

    // CHECK IF CART IS EMPTY
    if(this.props.cart.length > 0) {
      // CART IS NOT EMPTY
      let _id = this.props._id;

      let cartIndex =
this.props.cart.findIndex(function(cart){
        return cart._id === _id;
      })
      // IF RETURNS -1 THERE ARE NO ITEMS
WITH SAME ID
      if (cartIndex === -1){
        this.props.addToCart(book);
      } else {
        // WE NEED TO UPDATE QUANTITY
        this.props.updateCart(_id, 1,
this.props.cart);
      }
    } else {
      // CART IS EMPTY
      this.props.addToCart(book);
    }

  }

  render(){
    return(
      <Well>
        <Row>
          <Col xs={12} sm={4}>
            <Image src={this.props.images}
responsive />
          </Col>
          <Col xs={6} sm={8}>
            <h6>{this.props.title}</h6>
            <p>{this.props.description}</p>
            <h6>usd. {this.props.price}</h6>
            <Button
onClick={this.handleCart.bind(this)}
bsStyle='primary'>Buy now</Button>

          </Col>
        </Row>
      </Well>
    )
  }
}
function mapStateToProps(state){
```

```
  return{
    cart:state.cart.cart
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    addToCart:addToCart,
    updateCart:updateCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BookItem);
//-------------------------------------------------------------
```

BOOKSFORM - BUTTON FEEDBACK

in booksReducers.js:

1. Add POST_BOOK_REJECTED reducer with two new properties: “msg” and “style”
2. Add “msg” and “style” properties in POST_BOOK reducer
3. Add a new reducer: RESET_BUTTON

in booksActions.js:

4. Add a new action: resetButton()

in booksForm.js:

5. Edit the “Save book” button as below
6. Add “msg” and “style” state properties in MapStateToProps function
7. import restButton action and include in MapDispatchToProps function

booksForm.js:

```
"use strict"
import React from 'react';
```

```
import {MenuItem, InputGroup, DropdownButton,
Image, Col, Row, Well, Panel, FormControl,
FormGroup, ControlLabel, Button} from
'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';
import {findDOMNode} from 'react-dom';
import {postBooks, deleteBooks, getBooks,
resetButton} from
'../../actions/booksActions';
import axios from 'axios';

class BooksForm extends React.Component{
  constructor() {
    super();
    this.state = {
      images:[{}],
      img:''
    }
  }
  componentDidMount(){
    this.props.getBooks();
    //GET IMAGES FROM API
    axios.get('/api/images')
      .then(function(response){

this.setState({images:response.data});
      }.bind(this))
      .catch(function(err){
        this.setState({images:'error loading
image files from the server', img:''})
      }.bind(this))
  }
  handleSubmit(){
    const book=[{
      title:
findDOMNode(this.refs.title).value,
      description:
findDOMNode(this.refs.description).value,

images:findDOMNode(this.refs.image).value,
      price:
findDOMNode(this.refs.price).value,
    }]
    this.props.postBooks(book);
  }

  onDelete(){
    let bookId =
findDOMNode(this.refs.delete).value;

    this.props.deleteBooks(bookId);
  }

  handleSelect(img){
    this.setState({
      img: '/images/'+ img
    })
  }

  resetForm(){
    //RESET THE Button
```

```
    this.props.resetButton();

    findDOMNode(this.refs.title).value = '';
    findDOMNode(this.refs.description).value
= '';
    findDOMNode(this.refs.price).value = '';
    this.setState({img:''});
  }

  render(){

    const booksList =
this.props.books.map(function(booksArr){
      return (
        <option key={booksArr._id}>
{booksArr._id}</option>
      )
    })

    const imgList =
this.state.images.map(function(imgArr, i){
      return(
        <MenuItem key={i}
eventKey={imgArr.name}

onClick={this.handleSelect.bind(this,
imgArr.name)}>{imgArr.name}</MenuItem>
      )
    }, this)

    return(
```

```
      <Well>
        <Row>
          <Col xs={12} sm={6}>
            <Panel>
              <InputGroup>
                <FormControl type="text"
ref="image" value={this.state.img} />
                <DropdownButton

componentClass={InputGroup.Button}
                  id="input-dropdown-addon"
                  title="Select an image"
                  bsStyle="primary">
                  {imgList}
                </DropdownButton>
              </InputGroup>
              <Image src={this.state.img}
responsive/>
            </Panel>
          </Col>
          <Col xs={12} sm={6}>
            <Panel>
              <FormGroup controlId="title">

<ControlLabel>Title</ControlLabel>
                <FormControl
                    type="text"
                    placeholder="Enter Title"
                    ref="title" />
              </FormGroup>
```

```
                <FormGroup
controlId="description">

<ControlLabel>Description</ControlLabel>
                  <FormControl
                      type="text"
                      placeholder="Enter
Description"
                      ref="description" />
                </FormGroup>
                <FormGroup controlId="price">

<ControlLabel>Price</ControlLabel>
                  <FormControl
                      type="text"
                      placeholder="Enter Price"
                      ref="price" />
                </FormGroup>
                <Button

onClick={(!this.props.msg)?(this.handleSubmit
.bind(this)):(this.resetForm.bind(this))}

bsStyle={(!this.props.style)?("primary"):(thi
s.props.style)}>
                    {(!this.props.msg)?("Save
book"):(this.props.msg)}
                  </Button>
              </Panel>
              <Panel>
                <FormGroup
controlId="formControlsSelect">
                  <ControlLabel>Select a book
id to delete</ControlLabel>
                  <FormControl ref="delete"
componentClass="select" placeholder="select">
                    <option
value="select">select</option>
                      {booksList}
                  </FormControl>
                </FormGroup>
                <Button
onClick={this.onDelete.bind(this)}
bsStyle="danger">Delete book</Button>
              </Panel>
            </Col>
          </Row>

        </Well>
      )
    }
}
function mapStateToProps(state){
  return {
    books: state.books.books,
    msg: state.books.msg,
    style: state.books.style
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
```

```
    postBooks,
    deleteBooks,
    getBooks,
    resetButton
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksForm);
//-------------------------------------------------------------
```

booksActions.js:

```
// RESET FORM BUTTON
export function resetButton(){
  return {
          type:"RESET_BUTTON"
        }
}
//-------------------------------------------------------------
```

booksReducers:js

```
"use strict"

//BOOKS REDUCERS
export function booksReducers(state={
  books:[]
}, action){
  switch(action.type){
    case "GET_BOOKS":
    // let books =
state.books.concat(action.payload);
    // return {books};
    return {...state,
books:[...action.payload]}
    break;
    case "POST_BOOK":
    return {...state, books:[...state.books,
...action.payload], msg:'Saved! Click to
continue', style:'success'}
    break;
    case "POST_BOOK_REJECTED":
    return {...state, msg:'Please, try
again', style:'danger'}
    break;
    case "RESET_BUTTON":
    return {...state, msg:null,
style:'primary'}
    break;
    case "DELETE_BOOK":
    // Create a copy of the current array of
books
    const currentBookToDelete =
[...state.books]
    // Determine at which index in books
array is the book to be deleted
    const indexToDelete =
currentBookToDelete.findIndex(
      function(book){
        return book._id == action.payload;
```

```
      }
    )
    //use slice to remove the book at the
specified index
    return {books:
[...currentBookToDelete.slice(0,
indexToDelete),
...currentBookToDelete.slice(indexToDelete +
1)]}
    break;

    case "UPDATE_BOOK":
    // Create a copy of the current array of
books
    const currentBookToUpdate =
[...state.books]
    // Determine at which index in books
array is the book to be deleted
    const indexToUpdate =
currentBookToUpdate.findIndex(
      function(book){
        return book._id ===
action.payload._id;
      }
    )
    // Create a new book object with the new
values and with the same array index of the
item we want to replace. To achieve this we
will use ...spread but we could use concat
methos too
    const newBookToUpdate = {
      ...currentBookToUpdate[indexToUpdate],
      title: action.payload.title
    }
    // This Log has the purpose to show you
how newBookToUpdate looks like
    console.log("what is it newBookToUpdate",
newBookToUpdate);
    //use slice to remove the book at the
specified index, replace with the new object
and concatenate witht he rest of items in the
array
    return {books:
[...currentBookToUpdate.slice(0,
indexToUpdate), newBookToUpdate,
...currentBookToUpdate.slice(indexToUpdate +
1)]}
    break;
  }
  return state
}
//-----------------------------------------------------------------
```

COMPLETING THE CLIENT APP

Apply new Theme:

1. Go to bootstrapcdn.com/bootswatch
2. copy the html tag for the Cyborg theme and paste it in index.html

Implement validationState:

3. in booksReducers.js, add validation property (or any other name you prefer) to POST_BOOK, POST_BOOK_REJECTED, RESET_BUTTON reducers
4. in booksForm.js, add "validation" property in MapStateToProps function and add **validationState** FormGroup property to the "title", "description" and "price" FormGroup.

Add Carousel in booksList:

5. Copy the "Uncontrolled Carousel" component code from React-Bootstrap website
6. Import Carousel component in booksList
7. in the render method add a new Row and paste the Carousel code linking to image you want to be displayed in the Carousel

Fixing the cart Badge:

1. in main.js, import "bindActionCreators" and "getCart" action
2. Add a mapDispatchToProps function dispatching getCart action
3. Dispatch getCart action from ComponentDidMount

Add a "…read more" button in bookItem component:

4. Add an initial state for "isClicked" property and set it to "false"
5. Add a "onReadMore" function that set the state to "true"
6. Add the code below for the description and a new html button
7. Add "button.link" CSS in style.css

Edit all APIs to log errors rather than throwing errors

8. in apiServer.js replace all "throw err" with a log and indicating to which API is related the error

index.html:

```
<!DOCTYPE html>
<HTML>
  <HEAD>
    <TITLE>Hello Redux</TITLE>
    <meta name="viewport"
content="width=device-width,
initial-scale=1.0 maximum-scale=1.0" />
    <link
href="https://maxcdn.bootstrapcdn.com/bootswa
tch/3.3.7/cyborg/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-D9XILkoivXN+bcvB2kSOowkIvIc
BbNdoDQvfBNsxYAIieZbx8/SI4NeUvrRGCpDi"
crossorigin="anonymous">
    <link rel="stylesheet" href="style.css"
type="text/css"  />
  </HEAD>
  <BODY>
    <DIV id="app">

    </DIV>
    <SCRIPT src="bundle.js"></SCRIPT>
  </BODY>
</HTML>
//-------------------------------------------------------------
```

booksReducers.js:

```
   case "POST_BOOK":
    return {...state, books:[...state.books,
...action.payload], msg:'Saved! Click to
continue', style:'success',
validation:'success'}
    break;
    case "POST_BOOK_REJECTED":
    return {...state, msg:'Please, try
again', style:'danger', validation:'error'}
    break;
    case "RESET_BUTTON":
    return {...state, msg:null,
style:'primary', validation:null}
    break;
//-------------------------------------------------------------
```

booksForm.js:

```
"use strict"
import React from 'react';
import {MenuItem, InputGroup, DropdownButton,
Image, Col, Row, Well, Panel, FormControl,
FormGroup, ControlLabel, Button} from
'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';
import {findDOMNode} from 'react-dom';
import {postBooks, deleteBooks, getBooks,
resetButton} from
'../../actions/booksActions';
```

```
import axios from 'axios';

class BooksForm extends React.Component{
  constructor() {
    super();
    this.state = {
      images:[{}],
      img:''
    }
  }
  componentDidMount(){
    this.props.getBooks();
    //GET IMAGES FROM API
    axios.get('/api/images')
      .then(function(response){

this.setState({images:response.data});
      }.bind(this))
      .catch(function(err){
        this.setState({images:'error loading
image files from the server', img:''})
      }.bind(this))
  }
  handleSubmit(){
    const book=[{
      title:
findDOMNode(this.refs.title).value,
      description:
findDOMNode(this.refs.description).value,

images:findDOMNode(this.refs.image).value,
      price:
findDOMNode(this.refs.price).value,
    }]
    this.props.postBooks(book);
  }

  onDelete(){
    let bookId =
findDOMNode(this.refs.delete).value;

    this.props.deleteBooks(bookId);
  }

  handleSelect(img){
    this.setState({
      img: '/images/'+ img
    })
  }

  resetForm(){
    //RESET THE Button
    this.props.resetButton();

    findDOMNode(this.refs.title).value = '';
    findDOMNode(this.refs.description).value
= '';
    findDOMNode(this.refs.price).value = '';
    this.setState({img:''});
  }

  render(){
```

```
    const booksList =
this.props.books.map(function(booksArr){
      return (
        <option key={booksArr._id}>
{booksArr._id}</option>
      )
    })

    const imgList =
this.state.images.map(function(imgArr, i){
      return(
        <MenuItem key={i}
eventKey={imgArr.name}

onClick={this.handleSelect.bind(this,
imgArr.name)}>{imgArr.name}</MenuItem>
      )
    }, this)

    return(
      <Well>
        <Row>
          <Col xs={12} sm={6}>
            <Panel>
              <InputGroup>
                <FormControl type="text"
ref="image" value={this.state.img} />
                <DropdownButton

componentClass={InputGroup.Button}
                  id="input-dropdown-addon"
                  title="Select an image"
                  bsStyle="primary">
                  {imgList}
                </DropdownButton>
              </InputGroup>
              <Image src={this.state.img}
responsive/>
            </Panel>
          </Col>
          <Col xs={12} sm={6}>
            <Panel>
              <FormGroup controlId="title"
validationState={this.props.validation}>

<ControlLabel>Title</ControlLabel>
                <FormControl
                  type="text"
                  placeholder="Enter Title"
                  ref="title" />
                  <FormControl.Feedback/>
              </FormGroup>
              <FormGroup
controlId="description"
validationState={this.props.validation}>

<ControlLabel>Description</ControlLabel>
                <FormControl
                  type="text"
                  placeholder="Enter
Description"
```

```
                    ref="description" />
                    <FormControl.Feedback/>
                </FormGroup>
                <FormGroup controlId="price"
validationState={this.props.validation}>

<ControlLabel>Price</ControlLabel>
                  <FormControl
                    type="text"
                    placeholder="Enter Price"
                    ref="price" />
                    <FormControl.Feedback/>
                </FormGroup>
                <Button

onClick={(!this.props.msg)?(this.handleSubmit
.bind(this)):(this.resetForm.bind(this))}

bsStyle={(!this.props.style)?("primary"):(thi
s.props.style)}>
                    {(!this.props.msg)?("Save
book"):(this.props.msg)}
                  </Button>
                </Panel>
                <Panel>
                  <FormGroup
controlId="formControlsSelect">
                    <ControlLabel>Select a book
id to delete</ControlLabel>
                    <FormControl ref="delete"
componentClass="select" placeholder="select">
                      <option
value="select">select</option>
                      {booksList}
                    </FormControl>
                  </FormGroup>
                  <Button
onClick={this.onDelete.bind(this)}
bsStyle="danger">Delete book</Button>
                </Panel>
              </Col>
            </Row>

          </Well>
      )
    }
}
function mapStateToProps(state){
  return {
    books: state.books.books,
    msg: state.books.msg,
    style: state.books.style,
    validation: state.books.validation
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    postBooks,
    deleteBooks,
    getBooks,
    resetButton
  }, dispatch)
```

```
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksForm);
//-------------------------------------------------------------
```

booksList.js:

```
"use strict"
import React from 'react';
import {connect} from 'react-redux';
import {getBooks} from
'../../actions/booksActions';
import {bindActionCreators} from 'redux';
import {Carousel, Grid, Col, Row, Button}
from 'react-bootstrap';

import BookItem from './bookItem';
import BooksForm from './booksForm';
import Cart from './cart';


class BooksList extends React.Component{
  componentDidMount(){
    this.props.getBooks()
  }
  render(){
    const booksList =
this.props.books.map(function(booksArr){
      return(
        <Col xs={12} sm={6} md={4}
key={booksArr._id}>
          <BookItem
                _id= {booksArr._id}
                title={booksArr.title}

description={booksArr.description}
                images={booksArr.images}
                price={booksArr.price}/>
        </Col>
      )
    })
    return(
        <Grid>
          <Row>
            <Carousel>
              <Carousel.Item>
                <img width={900}
height={300} alt="900x300"
src="/images/home1.jpg"/>
                <Carousel.Caption>
                  <h3>First slide label</h3>
                  <p>Nulla vitae elit
libero, a pharetra augue mollis interdum.</p>
                </Carousel.Caption>
              </Carousel.Item>
              <Carousel.Item>
                <img width={900}
height={300} alt="900x300"
src="/images/home2.jpg"/>
                <Carousel.Caption>
```

```
                        <h3>Second slide
label</h3>
                        <p>Lorem ipsum dolor sit
amet, consectetur adipiscing elit.</p>
                      </Carousel.Caption>
                    </Carousel.Item>
                  </Carousel>
                </Row>
                <Row>
                  <Cart />
                </Row>
                <Row style={{marginTop:'15px'}}>
                    {booksList}
                </Row>
              </Grid>
    )
  }
}
function mapStateToProps(state){
  return{
    books: state.books.books
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    getBooks:getBooks
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BooksList);
//-----------------------------------------------------------------
```

main.js:

```
"use strict"
import React from 'react';
import Menu from './components/menu';
import Footer from './components/footer';

import{connect} from 'react-redux';
import {bindActionCreators} from 'redux';
import {getCart} from
'../src/actions/cartActions';

class Main extends React.Component{
  componentDidMount(){
    this.props.getCart();
  }
  render(){
    return(
      <div>
        <Menu
cartItemsNumber={this.props.totalQty} />
          {this.props.children}
        <Footer />
      </div>
    );
  }
}

function mapStateToProps(state){
  return {
```

```
    totalQty: state.cart.totalQty
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    getCart:getCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(Main);
//-------------------------------------------------------------------
```

bookItem.js:

```
"use strict"
import React from 'react';
import {Image, Row, Col, Well, Button} from
'react-bootstrap';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux'
import {addToCart, updateCart} from
'../../actions/cartActions';

class BookItem extends React.Component{

  handleCart(){
    const book = [...this.props.cart, {
      _id:this.props._id,
      title:this.props.title,
      description:this.props.description,
      images: this.props.images,
      price:this.props.price,
      quantity:1
    }]

    // CHECK IF CART IS EMPTY
    if(this.props.cart.length > 0) {
      // CART IS NOT EMPTY
      let _id = this.props._id;

      let cartIndex =
this.props.cart.findIndex(function(cart){
        return cart._id === _id;
      })
      // IF RETURNS -1 THERE ARE NO ITEMS
WITH SAME ID
      if (cartIndex === -1){
        this.props.addToCart(book);
      } else {
        // WE NEED TO UPDATE QUANTITY
        this.props.updateCart(_id, 1,
this.props.cart);
      }

    } else {
      // CART IS EMPTY
      this.props.addToCart(book);
    }

  }
  constructor(){
```

```
    super();
    this.state = {
      isClicked:false
    };
  }
  onReadMore(){
    this.setState({isClicked:true})
  }

  render(){
    return(
      <Well>
        <Row>
          <Col xs={12} sm={4}>
            <Image src={this.props.images}
responsive />
          </Col>
          <Col xs={6} sm={8}>
            <h6>{this.props.title}</h6>

<p>{(this.props.description.length > 50 &&
this.state.isClicked ===
false)?(this.props.description.substring(0,
50)):(this.props.description)}
              <button className='link'
onClick={this.onReadMore.bind(this)}>
                {(this.state.isClicked ===
false && this.props.description !== null &&
this.props.description.length > 50)?('...read
more'):('')}
              </button>
              </p>
              <h6>usd. {this.props.price}</h6>
              <Button
onClick={this.handleCart.bind(this)}
bsStyle='primary'>Buy now</Button>

          </Col>
        </Row>
      </Well>
    )
  }
}
function mapStateToProps(state){
  return{
    cart:state.cart.cart
  }
}
function mapDispatchToProps(dispatch){
  return bindActionCreators({
    addToCart:addToCart,
    updateCart:updateCart
  }, dispatch)
}
export default connect(mapStateToProps,
mapDispatchToProps)(BookItem);
//-------------------------------------------------------------
```

style.css:

```
body {
```

```
  margin-top: 50px;
  margin-bottom: 50px;
}

.container {
  width: auto;
  max-width: 900px;
  padding: 0 15px;
}
.container .footer-text{
  margin: 20px 0;
}
.badge{
  background-color: #e4aa48;
}
button.link {
  background: none;
  border: none;
  margin:0;
  padding: 3;
  outline: none;
  outline-offset: 0;
  color: white
}
//-----------------------------------------------------------------
```

apiserver.js:

```
"use strict"
var express = require('express');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
const session = require('express-session');
const MongoStore =
require('connect-mongo')(session);

var app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended:
false }));
app.use(cookieParser());


// APIs
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/b
ookshop');

var db = mongoose.connection;
db.on('error', console.error.bind(console, '#
MongoDB - connection error: '));

// --->>> SET UP SESSIONS <<<----
app.use(session({
  secret: 'mySecretString',
  saveUninitialized: false,
  resave:false,
  store: new MongoStore({mongooseConnection:
db, ttl: 2 * 24 * 60 * 60})
```

```
  //ttl: 2 days * 24 hours * 60 minutes * 60
seconds
}))
  // SAVE SESSION CART API
  app.post('/cart', function(req, res){
    var cart = req.body;
    req.session.cart = cart;
    req.session.save(function(err){
      if(err){
        console.log('# API POST CART SESSION:
', err);
      }
      res.json(req.session.cart);
    })
  });
  // GET SESSION CART API
  app.get('/cart', function(req, res){
    if(typeof req.session.cart !==
'undefined'){
      res.json(req.session.cart);
    }
  });
//--->>> END SESSION SET UP <<<----


var Books = require('./models/books.js');

//---->>> POST BOOKS <<<-----
app.post('/books', function(req, res){
  var book = req.body;
```

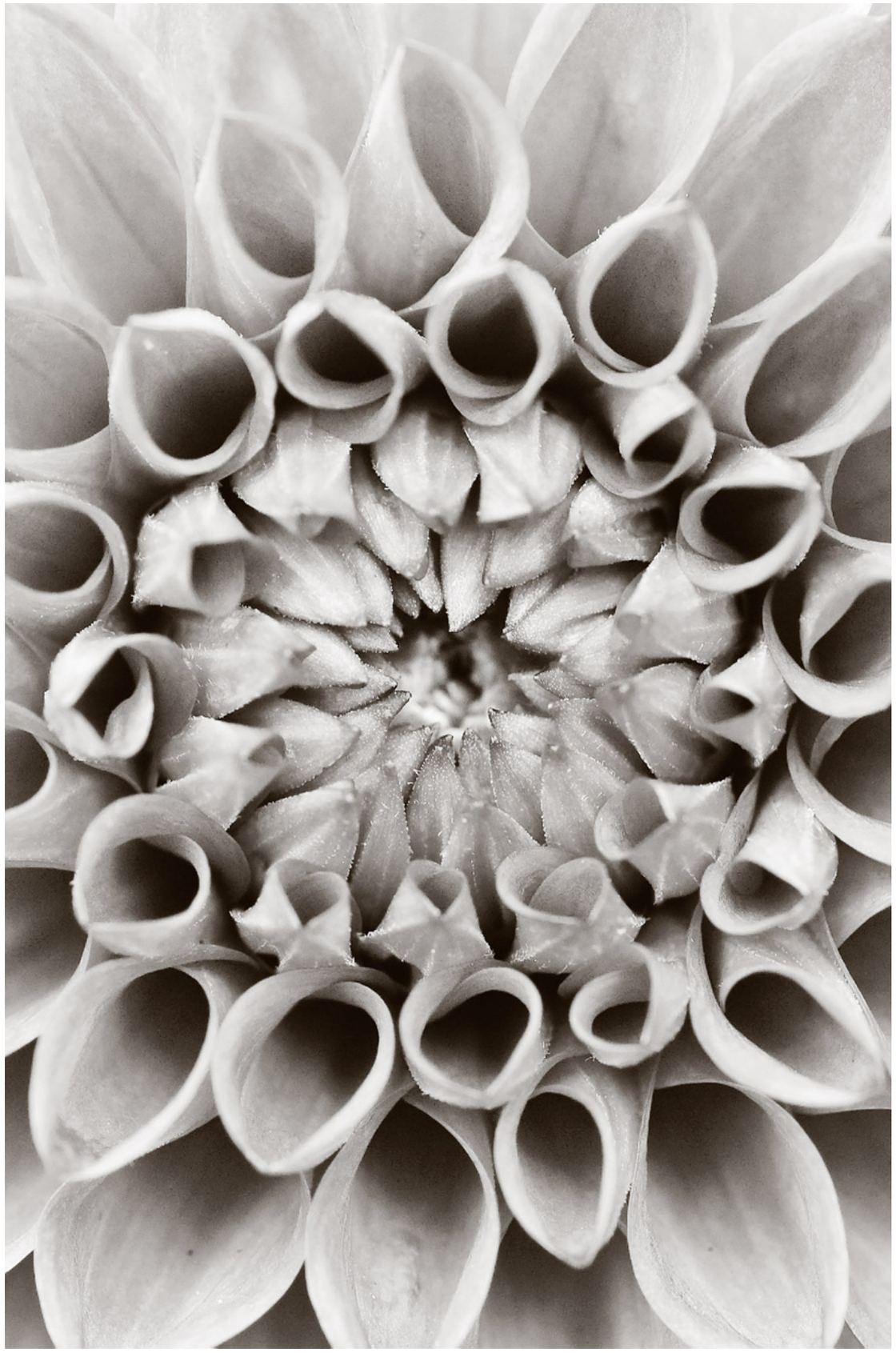
```
  //console.log('test title',
req.body[0].title);
  Books.create(book, function(err, books){
    if(err){
      console.log('# API POST BOOKS: ', err);
    }
    //res.json({books,
"title":req.body[0].title});
    res.json(books);
  })
});
// // INPUT VALIDATION
// function handleValidation(input){
//
// }


//----->>>> GET BOOKS <<<---------
app.get('/books', function(req, res){
  Books.find(function(err, books){
    if(err){
      console.log('# API GET BOOKS: ', err);
    }
    res.json(books)
  })
});

//---->>> DELETE BOOKS <<<------
app.delete('/books/:_id', function(req, res){
  var query = {_id: req.params._id};

  Books.remove(query, function(err, books){
```

```
    if(err){
      console.log('# API DELETE BOOKS: ',
err);
    }
    res.json(books);
  })
});

//--->>> UPDATE BOOKS <<<------
app.put('/books/:_id', function(req, res){
  var book = req.body;
  var query = req.params._id;
  // if the field doesn't exist $set will set
a new field
  var update = {
    '$set':{
      title:book.title,
      description:book.description,
      image:book.image,
      price:book.price
    }
  };
    // When true returns the updated document
    var options = {new: true};

    Books.findOneAndUpdate(query, update,
options, function(err, books){
      if(err){
        console.log('# API UPDATE BOOKS: ',
err);
      }
      res.json(books);
    })
})

// --->>> GET BOOKS IMAGES API <<<------
app.get('/images', function(req, res){

  const imgFolder = __dirname +
'/public/images/';
  // REQUIRE FILE SYSTEM
  const fs = require('fs');
  //READ ALL FILES IN THE DIRECTORY
  fs.readdir(imgFolder, function(err, files){
    if(err){
      return console.error(err);
    }
    //CREATE AN EMPTY ARRAY
    const filesArr = [];
    var i = 1;
    // ITERATE ALL IMAGES IN THE DIRECTORY
AND ADD TO THE THE ARRAY
    files.forEach(function(file){
      filesArr.push({name: file});
      i++
    });
    // SEND THE JSON RESPONSE WITH THE ARARY
    res.json(filesArr);
  })
})
```

```
// END APIs

app.listen(3001, function(err){
  if(err){
    return console.log(err);
  }
  console.log('API Sever is listening on
http://localhost:3001');
});
//-------------------------------------------------------------
```

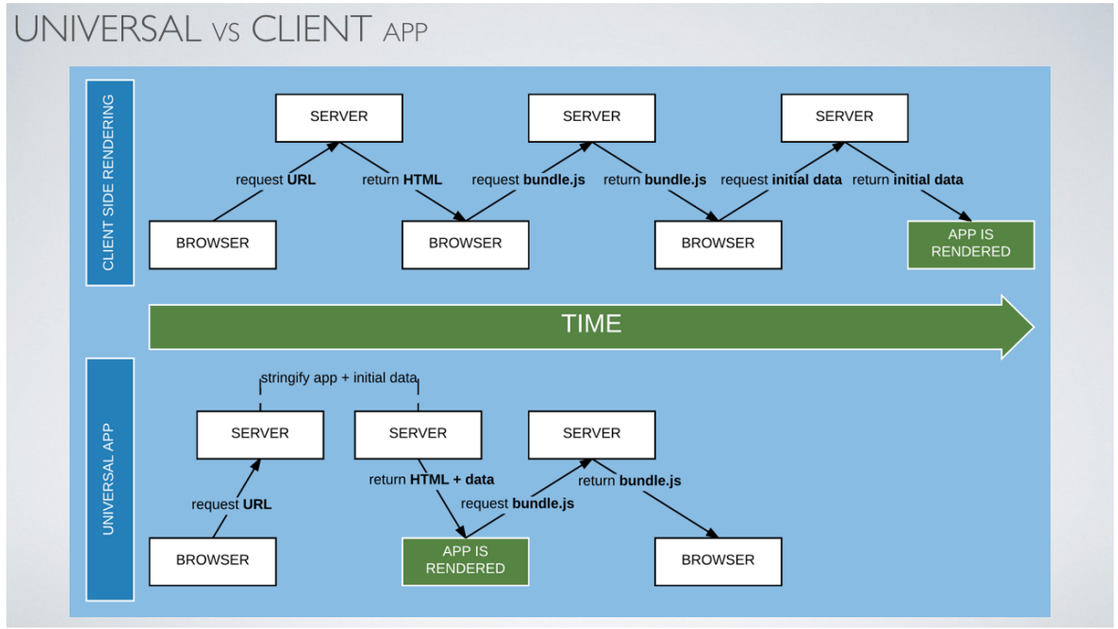
Chapter 4

MAKE A UNIVERSAL APP



In this chapter we will convert our fully client rendered app to be universal. We will first understand what a Universal app is and why we need and then we will implement the necessary changes and deploy the app in Heroku and AWS

Universal App



A Universal app delivers to the users the html and initial data with the first server response and for this reason the users will have the perception that the app is loading much faster than a fully client one.

Also, since the data is rendered by the server the app can be indexed by search engines without problems and it will be SEO compliant.

MAKE THE APP UNIVERSAL - PART 1

1. *Make a back-up of the project folder of the Client app and name it:"BooksShop_Client".*
2. Rename the other project folder as:"BooksShop_Universal" and open it in Atom.
3. From Terminal install EJS: **npm i --save ejs**
4. In project directory, open "Views", remove all .jade files in the folder and add: index.ejs
5. Copy the html code from index.html to index.ejs
6. Rename index.html to something else (i.e. NOindex.html) so to be ignored by the server.

in app.js:

7. Remove app.get('*'.....)
8. Add app.set.... to set EJS to be the template engine
9. Add app.use(requestHandler)
10. Add new file requestHandler.js and the code below.
11. import requestHandler.js in app.js
12. Request Babel-core and presets from the top of the file

app.js:

```
require('babel-core/register')({
  "presets":["es2015", "react", "stage-1"]
});

var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
//PROXY
var httpProxy = require('http-proxy');
// REQUEST HANDLER FOR SERVER-SIDE RENDERING
var requestHandler =
require('./requestHandler.js');


var app = express();

app.use(logger('dev'));
```

```
//PROXY TO API
const apiProxy =
httpProxy.createProxyServer({
  target:"http://localhost:3001"
});
app.use('/api', function(req, res){
  apiProxy.web(req, res);
})
// END PROXY ///

// uncomment after placing your favicon in
/public
//app.use(favicon(path.join(__dirname,
'public', 'favicon.ico')));
app.use(express.static(path.join(__dirname,
'public')));

app.set('view engine', 'ejs');

app.use(requestHandler);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in
development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') ===
'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;


//-------------------------------------------------------------
```

index.ejs:

```
<!DOCTYPE html>
<HTML>
  <HEAD>
    <TITLE>Hello Redux</TITLE>
    <meta name="viewport"
content="width=device-width,
initial-scale=1.0 maximum-scale=1.0" />

    <link
href="https://maxcdn.bootstrapcdn.com/bootswa
tch/3.3.7/cyborg/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-D9XILkoivXN+bcvB2kSOowkIvIc
```

```
BbNdoDQvfBNsxYAIieZbx8/SI4NeUvrRGCpDi"
crossorigin="anonymous">
    <link rel="stylesheet" href="style.css"
type="text/css"  />
  </HEAD>
  <BODY>
    <DIV id="app"><%- myHtml -%></DIV>
    <SCRIPT src="bundle.js"></SCRIPT>
  </BODY>
</HTML>
//--------------------------------------------------------------------
```

requestHandler.js:

```
"use strict"
import axios from 'axios';

function handleRender(req, res){
  axios.get('http://localhost:3001/books')
    .then(function(response){
        var myHtml =
JSON.stringify(response.data);
        res.render('index', {myHtml});
    })
    .catch(function(err){
      console.log('#Initial Server-side
rendering error', err);
    })
}

module.exports = handleRender;
//--------------------------------------------------------------------
```

MAKE THE APP UNIVERSAL - PART 2

1. *Duplicate client.js and name the new files: "routes.js"*

in routes.js:

2. Export default the module
3. Remove the Provider tags and all Redux import statements .

in client.js:

4. Remove all react-router related tags and import statements
5. Get initial state from window.INITIAL_STATE global variable and pass it to createStore method

in requestHandler.js:

6. Create the Redux store and pass the API response to the initial state
7. Stringily the initial state
8. implement a react-router "match" method
9. update index.ejs to include the new markups from requestHandler

10. *Write all necessary import statements*
11. *remember to import reducers and routes modules too.*
12. *remember to compile bundle.js before testing the app*

routes.js:

```
"use strict"
// REACT
import React from 'react';
import {render} from 'react-dom';
// REACT-ROUTER
import {Router, Route, IndexRoute,
browserHistory} from 'react-router';

import BooksList from
'./components/pages/booksList';
import Cart from './components/pages/cart';
import BooksForm from
'./components/pages/booksForm';
import Main from './main';

const routes = (
      <Router history={browserHistory}>
        <Route path="/" component={Main}>
            <IndexRoute
component={BooksList}/>
            <Route path="/admin"
component={BooksForm}/>
            <Route path="/cart"
component={Cart}/>
        </Route>
      </Router>
);

export default routes;
//-----------------------------------------------------------
```

client.js:

```
"use strict"
// REACT
import React from 'react';
import {render} from 'react-dom';
import {Provider} from 'react-redux';
// REACT-ROUTER
import {Router, Route, IndexRoute,
browserHistory} from 'react-router';

import {applyMiddleware, createStore} from
'redux';
import logger from 'redux-logger';
import thunk from 'redux-thunk';
```

```
// IMPORT COMBINED REDUCERS
import reducers from './reducers/index';
// IMPORT ACTIONS
import {addToCart} from
'./actions/cartActions';
// STEP 1 create the store
const middleware =applyMiddleware(thunk,
logger());
// WE WILL PASS INITIAL STATE FROM SERVER
STORE
const initialState = window.INITIAL_STATE;
const store =
createStore(reducers,initialState,
middleware);

import routes from './routes'
const Routes = (
  <Provider store={store}>
    {routes}
  </Provider>
)

render(
  Routes, document.getElementById('app')
);
//-----------------------------------------------------------------
```

requestHandler.js:

```
"use strict"
import axios from 'axios';
import React from 'react';
import {createStore} from 'redux';
import {Provider} from 'react-redux';
import {renderToString} from
'react-dom/server';
import {match, RouterContext} from
'react-router';

import reducers from './src/reducers/index';
import routes from './src/routes';

function handleRender(req, res){
  axios.get('http://localhost:3001/books')
    .then(function(response){
      // var myHtml =
JSON.stringify(response.data);
      // res.render('index', {myHtml});

      // STEP-1 CREATE A REDUX STORE ON THE
SERVER
      const store = createStore(reducers,
{"books":{"books":response.data}})
      // STEP-2 GET INITIAL STATE FROM THE
STORE
      const initialState =
JSON.stringify(store.getState()).replace(/<\/
script/g, '<\\/script').replace(/<!--/g,
'<\\!--');
```

```
        // STEP-3 IMPLEMENT REACT-ROUTER ON
THE SERVER TO INTERCEPT CLIENT REQUESTs AND
DEFINE WHAT TO DO WITH THEM
        const Routes ={
          routes:routes,
          location:req.url
        }
        match(Routes, function(error,
redirect, props){
          if(error){
            res.status(500).send("Error
fullfilling the request");
          } else if(redirect){
            res.status(302, redirect.pathname
+ redirect.search)
          } else if(props){
            const reactComponent =
renderToString(
              <Provider store={store}>
                <RouterContext {...props}/>
              </Provider>
            )
            res.status(200).render('index',
{reactComponent, initialState})
          } else {
            res.status(404).send('Not Found')
          }
        })

    })
    .catch(function(err){
      console.log('#Initial Server-side
rendering error', err);
    })
}

module.exports = handleRender;
//-----------------------------------------------------------------
```

index.ejs:

```
<!DOCTYPE html>
<HTML>
  <HEAD>
    <TITLE>Hello Redux</TITLE>
    <meta name="viewport"
content="width=device-width,
initial-scale=1.0 maximum-scale=1.0" />

    <link
href="https://maxcdn.bootstrapcdn.com/bootswa
tch/3.3.7/cyborg/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-D9XILkoivXN+bcvB2kSOowkIvIc
BbNdoDQvfBNsxYAIieZbx8/SI4NeUvrRGCpDi"
crossorigin="anonymous">
    <link rel="stylesheet" href="style.css"
type="text/css"  />
  </HEAD>
  <BODY>
```

```
    <DIV id="app"><%- reactComponent
-%></DIV>
    <script>window.INITIAL_STATE=<%-
initialState -%></script>
    <SCRIPT src="bundle.js"></SCRIPT>
  </BODY>
</HTML>
//----------------------------------------------------------------
```

DEPLOY THE DB ONLINE

1. *Sign up on mlab.com*
2. *Select "create new" MondoDB Deployment*
3. *Choose "Single Node" and "Sandbox" plan and choose a name for your database*
4. *Create a user for the database*
5. *From Terminal, move to bin folder inside the Mongo directory and run the following command to dum the entire local db:* **./mongodump --db DB_PATH --out <output directory>**
6. From TOOLS tab in mlab.com copy the command to import your localdb and run it in Terminal after you have replaced the correct user, password, local directory and added "./" in front of mongorestore: **./mongorestore -h ds043012.mlab.com:43012 -d bookshop -u <user> -p <password> <input directory>**

7. *Copy the connection string from the entry page of the database in mlab.com to mongoose.connection in apiServer.js and replacing the correct user, password in the string:* mongoose.connect(**mongodb://<dbuser>:<dbpassword>@ds043012.mlab.com:43012/bookshop**)
8. Make sure you restart the server
9. Test to see if the app is connected tot he remote database hosted by mlab.com

SET UP GIT

1. If you don't have Git, download git software from git.scm.com
2. After the installation verify you have Git insgtalled correctly by running the command: "**git --version**" in Terminal
3. In Terminal, go to the project directory and run the command: "**git init**" to create an empty git repository for your project files
4. run the command: "**git status**" and in Git should print in red color all your project files
5. In order to exclude the node_modules from the repository, create a new file called: "**.gitignore**" and write: "**node_modules/**"
6. To add the file in the repository, run the command: "**git add .**"
7. Now , if you run "**git status**" again all files should be printed in green color, indicating that these files are now tracked by Git
8. If you are happy with the current version of the file, run the command: **commit -a -b '<a message to remember this specific change>' .**

9. *If you run again the command "git status" you should get a message saying that there are no changes to commit.*
10. *From now on you can use Git to keep track of your changes. If you add new files, use the command "**git add .**" and then **commit -a -b '<a message>'**. When you change make changes, you can run **commit -a -b '< a message>'**.*

DEPLOY THE APP IN HEROKU

1. Register an account in heroku.com
2. install Heroic CLI following the instruction in toolbelt.heroku.com
3. if you already used Heroic in the past, in Terminal run the command: "**heroku update**"
4. Ensure you commit the latest changes in the code running: "**git status**" and **commit the changes** (if any)
5. Run the command: "**heroku login**" and input email and password you registered in Heroku
6. Create an Empty repository running the command: "**heroku create**"
7. Deploy the code to heroku with the command: "**git push heroku master**"
8. open your website with the command: "**heroku open**"

DEPLOY THE APP IN AWS ELASTIC BEANSTALK

1. Register an account in aws.amazon.com
2. **install Elastic Beanstalk CLI** following the instruction in: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install.html
3. If you don't have a user, create one in IAM following the instruction in the lecture
4. In the App project folder, create a new directory called: **.ebextensions** and inside it, create a new file: **nodecommand.config** with the below
5. Initialize an Elastic Beanstalk repository in your project folder by running in Terminal the command: **ed init** and choose the country that will host your created environment and answer to the rest of questions. The first time you create an environment from a computer you will be asked for the **aws-access-id** and the **secrete-access-key** you get when you create a new user in AWS
6. run the command: **eb create <a name for your environment>**
7. open the website with: **eb open**

nodecommand.config:

```
option_settings:
  aws:elasticbeanstalk:container:nodejs:
    NodeCommand: "npm start"

//-------------------------------------------------------------------
-
```