Removing the Middle Layer with React, CouchDB and Pouch.js

Saurabh Manoj Machave, Nitin Srinivas k, Sachin S, Tushar Jain

Department of Computer Science and Engineering
The National Institute of Engineering, Mysore, India

---***---

Abstract - *There are various ways of developing on the web, with thousands of frameworks destined to run on both client and server sides that hope to make the job of the developer easy with their innovative use of design and APIs (Application Program Interfaces). There are many such frameworks out there, none can be said to be the best, but simply depending on the tastes of the developers and their requirements of their project. One such use case is when the development must occur fast, yet the project must be scalable enough with use of databases, a responsive and a reactive UI/UX. In a field such a web development where there are multiple ways of writing the backend by using any method and language, and several components of the frontend that need to react in a particular way that the backend requires it to, it indeed results in a very complex equation if programmed without the right tools. Not just complex, but when we put time constraints on the development period things become even more tense. To combat such issues, the developer must equip themselves with the right tools for the job, and why are these tools the right ones? This paper explains these technicalities.*

***Key Words*: Developmental agility, Reactive user interfaces, HTTP, NodeJS, Pouch.js**

1.INTRODUCTION

In terms of an MVC (Model-View-Controller) Architecture, the middle layer in web development consists of a controller and a view rendering engine which takes parameters from the controller and renders the view. The controller acts as an interface between the database and the view, it queries the database for useful information depending on the requirement of the end user (or moreover based on the how the controller has been programmed to respond to the various events), fetches the information, parses the result, performs computation and passes it as a parameter to the Views to get rendered onto the screen of the end-user.

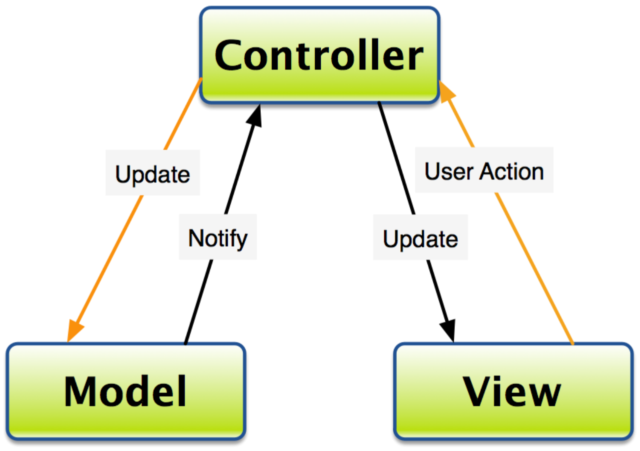
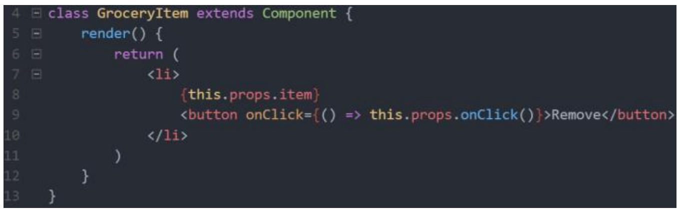


Figure 1*:* Model-View-Controller

The middle layer is usually hosted on a server which listens to HTTP requests and responds accordingly depending on the HTTP verbs and the path along with the headers and cookies accompanying the request. It plays a crucial role in the process, which in a way it acts as a gateway for the end-user to use the services that the web application provides. But what if we did not have to write this layer at all? It would certainly save a ton of time at the least. Granted, but this notion does not imply that the code that should have gone to the middle layer should simply be ignored. To achieve similar functionality, we need that piece of code somewhere in the project. The bottom-line is that to make things simpler we simply just remove the middle layer, or to be precise (we "move" it to a different layer, which we will describe shortly).

React is a frontend framework created by Facebook, which is designed to bring an Object Oriented Paradigm to web development. As mentioned earlier, the web as a platform is very scattered, with separate components such as HTML, CSS and JavaScript along with the backend. React aims at integrating the HTML and JavaScript portions to make a neat reactive model which responds to changes in the structure of the Document Object Model (DOM) and user inputs. It finds a way to integrate the two components using ECMAScript 2015 (Transpiled using Babel) into the same context without getting syntactical errors. This allows users to make modifications to HTML elements with ease. The response to events and rendering of webpages takes place in a waterfall model which starts at the topmost point and goes to the leaves.

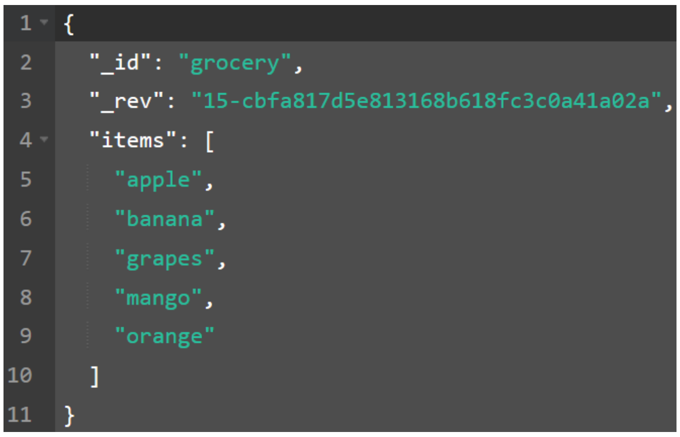


```
class GroceryItem extends Component {
    render() {
        return (
            <li>
                {this.props.item}
                <button onClick={() => this.props.onClick()}>Remove</button>
            </li>
        )
    }
}
```

Figure 2: Example of combining JavaScript with HTML

CouchDB is a NoSQL database which stores data in JSON (JavaScript Object Notation) format. It consists of databases which can have permissions set for them by administrators. For example, only such and such users can pull data from or modify a certain database. There is no concept of tables, such as in SQL databases and a NoSQL database called MongoDB. We can store documents directly into the database. To clarify that a document belongs to a table, we may give it a certain property which identifies this behaviour.

Every document has an *_id* field which uniquely identifies that document in that database. This field can be set either by the user or it is automatically set by CouchDB. We can modify documents too, and by doing so a *_rev* field is generated. This keeps track of the revision of that particular document. If a document is to be updated, its old *_rev* and new *_rev* need to agree with each other.



```
{
  "_id": "grocery",
  "_rev": "15-cbfa817d5e813168b618fc3c0a41a02a",
  "items": [
    "apple",
    "banana",
    "grapes",
    "mango",
    "orange"
  ]
}
```

Figure 3: An example of a CouchDB document

The administrators of the database can allocate the required permissions to users, or they can even allocate 'roles' which work like 'class' attribute in HTML/CSS. If a group of users need access to more than one databases which are common to all of them, rather than giving them each permissions to access each database in question, a 'role' can be given to these users and the 'role' can be added to the databases.

The most interesting feature of CouchDB is to be able to make these transactions with REST API which it supports for all operations. It returns the response in a JSON format. This great built-in feature of CouchDB allows it to be used with ease. Another great feature that it has is a built-in is Fauxton, which is a user interface to view the database, configure it, create design documents and everything possible that you can think of. We can even create user sessions to make the transaction process persistent.

2. BUILDING THE FRONT END

It is no surprise that all of the codebase is actually located in the front end layer, where React is located. If you intent to make your website a multi-page one, you can proceed to use a library called react router which enables different pages to load by modifying the DOM and using the History API to keep track of which pages you have visited. In simple words, your website consists simply of one HTML page, and JavaScript takes care of all the page swapping and rendering. React is best used in an editor like Visual Studio Code or Sublime Text so that the lint tool can give you useful suggestions about syntactical and semantical errors. The best way to start off with a React project is to install a NodeJS app called create-react-app which helps you download the necessary libraries and files to get started off with your project. Once you start running it, any changes you make in the working directory get compiled and reflected in your browser.

In order to be enabled to use Pouch.js in React, we can include it in any of the modules that we build with a simple import statement. After that, we are able to use it normally just like any other module. This applies to any other Javascript library. We can import a CSS file in a similar way, but we do not need to specify which module needs to be imported, since it assumes those properties for that page.

React consists of Components which can be inherited and used as user-defined components. We can pass data to these components in the form of properties (called props). These are used like read-only variables inside the component, and they directly reference the argument.

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Mark" />
```

Figure 4: An example of how props are passed to child components

React has several concepts to understand, such as when a component is rendered by the parent for the first time, it calls the constructor of that component. But if that component needs to be rendered again by the parent, the constructor is not called again and even the properties are not changed from the first time it was called. But if you want to supply the child component with modified props, you can create a method inside the child called componentWillReceiveProps with a parameter to get the latest props.

React provides a way to store the states of variables that may change over a period of time. You can define which variables to store in the state class member. If React detects that the state has been changed in any way, it initiates a full re-render of all the components from the current till the bottom of the tree. Just changing the state using an equals sign does not trigger this operation. A special function called setState is used to both set the state and initiate the rendering process. The parent components can also provide methods as arguments to their child components, which serve as callbacks that the child can call at times when a certain event occurs, etc and even pass information up along the tree.

```
handleClick(i) {
  const squares = this.state.squares.slice();
  squares[i] = this.state.xIsNext ? 'X' : 'O';
  this.setState({
    squares: squares,
    xIsNext: !this.state.xIsNext,
  });
}
```

Figure 5: An example of how states are used and set

Pouch.js should be installed as a local package using npm or cloned from its GitHub repository. It provides API for full-scale out-of-the-box management of the database. It includes everything from Creating/Updating documents, connecting to the database with options like the authentication, deleting documents, creating and executing map/reduce queries. Most of these queries return a Promise. A Promise is an object in Javascript that promises to return the result some time in the future whenever it becomes available. It is used with asynchronous requests such as network requests or file system management. In our case, the Promise will return the document in the future for a fetching query or several documents in a all_docs query. To simplify the use of Pouch.js, you can create a small library with several functions utilizing the Pouch.js library but which will shorten your code length (because of reuse). One of the vital functions that is used often is a function to save and fetch the connection to the database, so that you need not type in your credentials every time.

3. PREPARING THE DATABASE

CouchDB requires you to create a username and password for the root administrator during the initial usage. This operator has maximum privileges and has access to all data across every database. There are a couple of databases created prior to actually using CouchDB which store the internal data. One such database is the _users database which consists of information relating to each user. Upon receiving a request, CouchDB checks whether the credentials sent along with the request are correct and then gives/denies the user based on those. As mentioned earlier, each user can be given a role so as to provide them access to databases.

Fauxton can be used to make the initial setup of the database so that it becomes simpler in general and gives the user an idea of how the system works in general and what are all the options available to them. One of the options are to create views. Views are like tables but which are not respective to the documents available in a database. They can be even created using the information contained in the documents, or can consist of a select number of documents from the entire database. It all depends on the application. Since the only functions available are to fetch a document specified by their _id and on the other hand to fetch all the documents in the database, creating views is absolutely essential to fetch exactly the data that is required.

```
curl -X GET http://127.0.0.1:5984/_all_dbs
```

```
["baseball"]
```

Figure 6: An example of a CouchDB request and response

As shown in Figure 6 to retrieve data from the database requires you to make a request to the correct hostname and port with a path that signifies which function you want to use.

The return value is in JSON format.

REFERENCES

1. The Pouch.js main website https://pouchdb.com/ consists of all the help taken for using their library.
2. The main site of React https://facebook.github.io/react/ consists of tutorials, examples and documentation regarding the use of React.
3. CouchDB can be downloaded from their main site https://couchdb.apache.org/ and the documentation page is linked there as well.
4. Create-react-app is useful for starting out with React https://github.com/facebookincubator/create-react-app/
5. Special mention goes out to NodeJS and its package manager NPM for package management.
6. A hearty shout out to Stack Overflow https://stackoverflow.com/ as a whole for generally coming to the rescue in times of need.