Asabeneh Shitahun Yetayeh

Developing Dynamic Single Page Web Applications Using Meteor

Comparing JavaScript Frameworks: Blaze and React

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

2.6.2017

Author	Asabeneh Shitahun Yetayeh
Title	Developing Dynamic Single Page Web Applications Using Meteor
Number of Pages	31 pages + 3 appendices
Date	2 June, 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Patrick Ausderau, Lecturer Sonja Holappa, Lecturer

This paper studies Meteor which is a JavaScript full-stack framework to develop interactive single page web applications. Meteor allows building web applications entirely in JavaScript. Meteor uses Blaze, React or AngularJS as a view layer and Node.js and MongoDB as a back-end. The main purpose of this study is to compare the performance of Blaze and React.

A multi-user Blaze and React web applications with similar HTML and CSS were developed. Both applications were deployed on Heroku's web server and compared based on the size of the development community, the programming pattern, the available packages and the page loading speed.

This study showed that React has a higher number of development community and available packages than Blaze. The speed test showed that React web application is faster and lighter than Blaze. Both Blaze and React have similar features and user interfaces.

This paper concludes that Meteor with React is a better option to develop interactive single page applications because of higher popularity, smaller page size, faster page loading speed and more available packages. For next study, this paper recommends to integrate VueJS as Meteor view layer which will give developers more options to choose from and in return it will increase Meteor's development community.

Keywords	Blaze, JavaScript, Meteor, MongoDB, Node.js, React, Spacebar, Template, VueJS, Web application

Contents

Abbreviations

CLI	Command Line Interface.
CSS	Cascading Style Sheet.
DDP	Distributed Data Protocol.
DOM	Document Object Model.
ES	ECMA Script 6.
HTML	HyperText Markup Language.
JS	JavaScript.
JSX	JavaScript XML.
MDG	Meteor Development Group.
NPM	Node Package Management.
OS	Operating System.
RDB	relational database.
SCSS	Super Set of CSS.
UI	User Interface.
UX	User Experience.

Glossary

CRUD	In computer programming, create, read, update, and delete are the four basic functions of persistent storage. CRUD refers to all of the major functions that are implemented in relational database applications.
DOM	DOM is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document [1]..
JSON	JSON is a lightweight data-interchange format. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999.
JSX	is a XML-like syntax extension to ECMAScript without any defined semantics. .
NoSQL	referring to ”non SQL”, ”non relational” or ”not only SQL” database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational database..
NPM	default package manager for the JavaScript run-time environment,Node.js.
SEO	SEO is a marketing discipline focused on growing visibility in organic (non-paid) search engine results of a website [2]..

SQL	Structured Query Language is a domain-specific language used in programming and designed for managing data held in a relational database management system.
UI	An interface is a set of commands, menus or features through which a user communicates with an application.
UX	UX design is the process used to determine what the experience will be like when a user interacts with the user interface of the application.
XML	XML is a software and hardware-independent tool for storing and transporting structured data..

1 Introduction

JavaScript (JS) has increased in popularity in recent years and has been the leading programming language for four consecutive years [3] and is the most used language on Github [4] [5]. It is also becoming a default choice to develop dynamic, single page web applications [6]. A number of JS frameworks has been evolved in the last few years [6]. Most of them are for client side development. Just to mention few of the competitive open sources JS frameworks which came to the web programmers' table in the last few years: AngularJS, React, Ember, Polymer, Backbone and Knockout. Different frameworks come with various methods to solve the common underlying problem of rendering complex user interfaces dynamically and making single page applications loading faster [6]

The emergence of different JS frameworks is a big storm in the web development community, as switching from one web technology to another is very frustrating for web developers. These entire front-end technologies do not solve all the problems of the web developers. There should be a framework which connects front-end with the back-end. Node.js evolved as a server side JS run time environment makes it possible to use JS in the back-end [7]. This has led to the emergence of full-stack applications entirely written in JS [4]. In contrast to front-end JS frameworks, there are only a few full-stack JS frameworks in existence. Some of these are Meteor, Sails.js, Express.js, Feather.js, Derby, and Wakanda [8].

This paper focuses on Meteor which is an open-source full-stack JS framework to develop interactive single page web applications [9]. Meteor uses only JS both in the front-end and back-end. Meteor has its own native front-end framework, Blaze. In addition, Meteor can also be integrated with other front-end frameworks such as AngularJS and React [10].

This paper compares Blaze, the default Meteor view layer with React as a newly integrated view layer to Meteor. Interactive web applications using Blaze and React which implement CRUD operations in the back-end is developed and deployed on Heroku. Popularity, page loading speed and page size of the applications are evaluated.

2 Meteor a JavaScript Full-Stack Framework

2.1 Meteor

Meteor JS framework uses only JS to build modern, real-time single page web applications for both desktop and mobile platforms [10]. Meteor is the most well-known isomorphic open-source JS framework [9]. Meteor is a very simple environment to build modern web applications using only one language, JS [11]. Meteor is beyond a JS coding framework, as it offers an innovative way to develop scale-able, rich, interactive single page web applications [12].

Meteor can be a good choice for both beginners and advanced developers [11]. Meteor full-stack framework can be the best choice of framework for many reasons. Meteor applications are written in JS, so no need to learn another server specific programming language. It provides an easy way to install a development environment which includes a web server and database server. It includes a smart way to use packaging management system which could be easily incorporated to the application. Meteor provides very reactive data sources, with a neat distributed data model, meaning applications will feel reactive to multiple users and Meteor can generate native iOS and Android applications from the same JS code base. Meteor broke down the barrier between client and server by implementing an isomorphic model which means the same code base on the server side and the client side [9] [10].

The complete architecture of Meteor is illustrated in figure 1, showing how the client-side and sever-side communicates. Meteor initially used to use only Blaze as a view layer technology. In 2016, it included React and AngularJS as its view layer technologies. Currently, it uses Blaze, React and AngularJS as a view layer framework and Node.js and MongoDB in the back-end [10]. The upper layer of figure 1 shows that Meteor can be used to build both web or mobile applications. Below the upper layer, the three Meteor front-end technologies such as Blaze, AngularJS and React are represented. Below the view layer technologies, there is Client Data Cache. Every user connecting to a Meteor application will have the copy of server side data as Client Data Cache which is also know as Minimongo. In the middle of figure 1, there is Distributed Data Protocol (DDP). Meteor uses DDP to communicate with the clients. As it is depicted in figure 1, DDP is the boundary line between the client and server side.

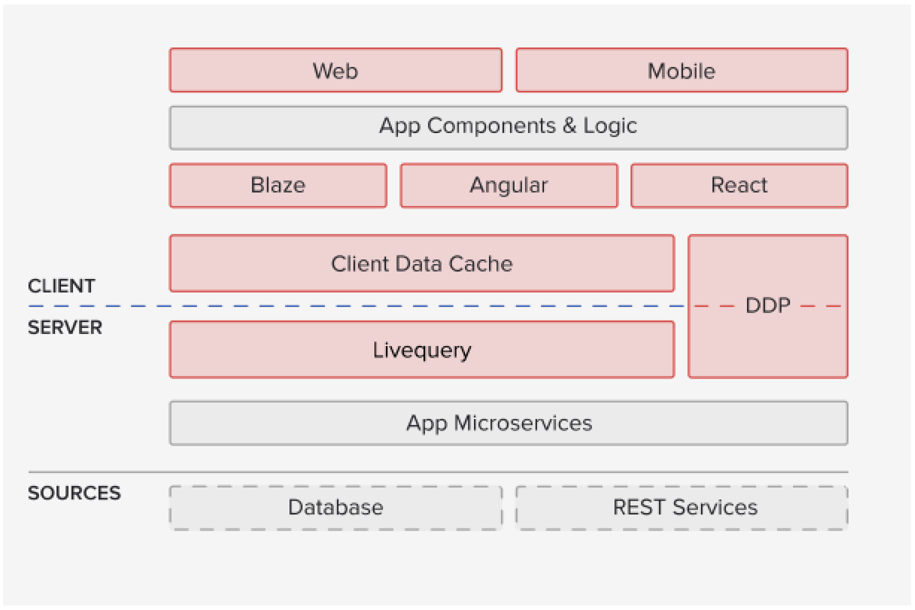


Figure 1: The Meteor architecture (Copied from fjaguero.com(2017) [13])

Meteor provides three alternative view layer technologies as it is depicted in figure 1, it is possible to use either Blaze, React or AngularJS in a Meteor project. Among the alternative view layer options Meteor provides, Blaze and React view layer were used for this study. In the coming section Blaze, the default Meteor front-end technology, will be discussed.

2.2 Front-end Technologies

2.2.1 Blaze

Blaze is a Meteor's default and built-in reactive rendering library which uses reactive HTML templates. Instead of using a combination of traditional templates and jQuery, Blaze eliminates the need for all the update logic in the application that listens for data changes and manipulates the DOM by familiarizing template directives and integrating with Tracker's transparent reactivity and Minimongo's database cursors so that the DOM updates automatically when there is a change. [14] In contrast to React, Blaze emphasizes HyperText

Markup Language (HTML) templates rather than JS component classes. Templates are more approachable than JS code and easier to read, write, and style with Cascading Style Sheet (CSS). Instead of using Tracker, React relies on a combination of explicit *setState* calls and data-model diffing in order to achieve efficient rendering [14].

The Blaze template language is inspired by Handlebars and written in Spacebars, a variant of Handlebars [14]. The Blaze template consists of a template tag and name attribute. In Meteor, templates are used to create a connection between the projects HTML interface and projects JS code [11]. To create a template, it needs a template tag, template name attribute and HTML elements inside the template tag. The code in listing 1 is an example of a template with a name leaderboard having two HTML elements with static data. The Spacebar engine helps to inject dynamic data into the template.

```
<template name="leaderboard">
    <h4>player #1 - point 5</h4>
    <h4>player #2 - point 4</h4>
</template>
```

Listing 1: An example of Blaze template

Blaze uses Spacebar to handle events and to inject data into the HTML interface of the template. Spacebar consists of HTML interspersed with template tags, which are delimited by two curly braces [14]. Spacebar uses the Spacebar template helpers and events to manipulate data and to handle events. As listing 2 illustrates, Spacebar helpers can contain variables, data and functions. This Spacebar has an array variable named players.

```
Template.leaderboard.helpers({
players: [
    { name:"player #1", score: 5 },
    { name:"player #2", score: 4 }
]
});
```

Listing 2: Spacebar Helper Function

The array data in listing 2 has two data, the name of the player and score of the respective player. To insert the players data into the leaderboard template, Spacebar uses `{{#each players}} {{name}} - {{score}} {{/each}}` iteration methods. This mechanism is similar to `forEach` in JS or `map` in ECMA Script 6 (ES). React uses map to loop through data and insert into JSX elements or components. As listing 3 shows the power of Spacebar to iterate array data into the template dynamically. The codes in listing 3 is an example of template with Spacebar.

```
<template name="leaderboard">
    {{#each players}}
        <h4>{{name}} - {{score}}</h4>
    {{/each}}
</template>
```

Listing 3: An example Spacebar iterating data into a blaze template

In addition to variables and arrays, Helpers can also contain JS functions. Helper functions are JS functions that are attached to templates and allow executing code inside the template interface [11]. To create a helper function, it needs the keyword template, the name of the template and helpers keyword. The codes from line 6 - 11 in listing 4 are examples of helpers and from 13 - 20 are event handler of the leaderboard template. The codes in listing 4 have both Helpers and Events. Helpers functions are to store and extract and Events are to handle data related to that specific events.

```
1  Template.leaderboard.helpers({
2  players: [
3      { name: "Player #1", score: 5 },
4      { name: "Player #2", score: 4 }
5  ],
6  'playersName':function(){
7      return players.name;
8  },
9  'playersScore':function(){
10     return players.score;
11 }});
12
13 Template.leaderboard.events({//template events
14 'click button':function(){
15     alert("you have clicked a button");
16 },
17 'submit input':function(){
18     alert("you are about to submit an input");
19 }
20 });
```

Listing 4: Template Helpers and Events function

As listing 4 shows, in the `leaderboard` helper there are a players array data, and two functions. In the `leaderboard` events function there are two events. In the next section, the second view layer technology of Meteor which is React will be discussed.

2.2.2 React

React is a JS front-end library and it is one of the most popular front-end technology to design web applications [15]. React uses JSX, a preprocessor step that adds XML syntax to JS. It is possible to use React without JSX but JSX makes React more elegant.

React makes building interactive User Interface (UI)s simple. Designing a simple view for each state in an application, React will efficiently update and render the right components when data changes [15]. React builds encapsulated components which manage their own state, then compose them to make complex UIs.

React component logic is written in JS not in templates. Therefore, it is easy to pass rich data through an application and keep state out of the DOM [15]. The following syntax: `const element = <h1> Hello World </h1>` is an example of JSX which is not an HTML nor JS string declaration. Every JSX elements and components will be rendered in the root `div` element. The root `div` is usually written like this: `<div id = "root"></div>`. It is possible to embed any JS expression in JSX by wrapping it in curly braces. The JS function `fullName()` in listing 5 is embedded in to the JavaScript XML (JSX) element.

```
function fullName(player){  // function to format fullname
    return player.firstName +  ' ' + player.lastName;
}
const player = { // Simple JavaScript Object
    firstName:"Firstname",
    lastName :"Lastname"
};

//JSX element
const element = (<h1> Hello, {fullName(player)} </h1>);

//Rendering JSX element to the root
ReactDOM.render(element, document.getElementById('root'));

const player = {
    name:"Player #1",
    score:5
};
```

Listing 5: Rendering JSX element in a root div.

As listing 5 shows that React source code consists of a function, JSX element, JS object and ReactDOM. The JS function, `fullName(player)` passes the `player` data to the

JSX element and the JSX will be rendered in the `ReactDOM` at line 13. JSX element may contain an attribute. The attribute could be an expression or a string. JSX attribute could be static data as this one `const element = (<h1 name = "name"> Hello</h1>)` or a dynamic data like this one, `const element = (<h1 name = {player.name}>Hello</h1>)`. JSX tag may contain children. All HTML elements in JSX should be enclosed by a common parent HTML element if there are multiple HTML elements. This is `const element = (<div> <h1> Name </h1> <h1> score </h1></div>)` JSX element which has two `h1` HTML elements enclosed by a parent `div`element.

React uses the JSX components instead of JSX elements to be productive. Components makes source code split into independent, reusable pieces. Components are like JS functions which are responsible for a certain task. As demonstrated in listing 6, React class component can be exported and used anywhere in a project file.

```
export default class Player extends React.Component{
    render(){
        return(
        <div>
            <h1> Player name</h1>
            <h1> Player score</h1>
        </div>
    );
}
};
```

Listing 6: An example of React component.

The React component in listing 6 is a re-usable component which can be used like this, `const element = <Player name = "name" />` in a project. The name attribute in the `<Player name = "name" />` is a `props` which can be static data or data which can be changed dynamically. Player is now a reusable component which can be used anywhere in the project. In this example, the component has only static data.

To put dynamic data to the UI, React uses `setState` method plays a great role. The main core of every React component is its *state*, an object that determines how the component renders and behaves. React state objects are what allow creating components that are dynamic and interactive. [15] If the components have states they change UI according to the state. The code illustrated in listing 7 is an example of a component with component state. State could be a single value or a set of values in an object or array.

```
export default class Player extends React.Component{
    constructor(props){
        super(props);
        this.state = {score:0};
    }
    render(){
        return(
            <div>
                <h1> Player name</h1>
                <h1> Player score</h1>
            </div>
        );
    }
};
```

Listing 7: States in React component.

States and props play a great role in changing data dynamical. JSX components usually render to UI. React uses a built-in function to prioritize rendering and to destroy the rendered component. It is very important to free up resources used by components by destroying the rendered components. React uses built-in functions to monitor the components' life cycle. The two commonly used React built-in functions are `componentDidMount()` and `componentWillUnMount()`. The `componentDidMount()` hook runs after the component output has been rendered to the Document Object Model (DOM). The `componentWillUnMount()` life-cycle hook will destroy the rendered components. The next section will discuss the server-side of Meteor.

2.3 Back-end Technologies

2.3.1 Node.js

Node.js is an open-source, cross-platform JS run-time environment for developing different applications. It has been around since 2009 and brought a big change in the web programming world [16]. Node.js introduced a package manager called NPM in January 2010. Node Package Management (NPM) makes publishing and sharing of JS source codes easier and is designed to simplify installation, updating and un-installation of libraries [7]. Data should not be only requested from the client, to make the web reactive, data should also be pushed from the server to the client and Node.js is good at in real-time web applications by employing push technology over web-sockets [17].

The Node.js run-time environment interprets JS by taking full advantage of Google's V8 JS engine. V8 JS engine allows Node.js to create a run-time environment that delivers JS from the server to the client very fast. V8 translates JS into native machine code, instead of working overtime to interpret it as byte code, giving Node.js its swiftness. The responsive speed, combined with asynchronous programming make Node.js so responsive and desirable in building dynamic and reactive applications. Meteor took advantage of Node.js greatness by using it in the back-end to deliver content to the client. [16]

2.3.2 MongoDB

According to MongoDB's website definition, MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling [18]. MongoDB is a NoSQL database system which is used in many applications that offers a very flexible structure that can range from basic databases to complex.

MongoDB does not require a pre-defined schema to start and it can be scaled up at any time [11]. MongoDB uses JSON type syntax is called Binary JSON (BSON) [18]. In contrast to the well-known relational database (RDB), SQL, MongoDB does not use table, row and column. Instead it uses collection, document and field [18]. As table 1 shows some of MongoDB and SQL terminologies are different from each other [11].

Table 1: MongoDB and SQL terminologies (Modified from https://www.mongodb.com/ (2017) [18]) .

MongoDB	SQL
Database	Database
Collection	Table
Index	Index
Document	Row
Field	Column
Embed Documents and Linking	Join
Primary Key	Primary
Aggregation	Group by

MongoDB databases and SQL databases are similar, but there is a subtle difference [11].

As table 1 illustrates that SQL stores data in tables whereas MongoDB in collections. MongoDB is the default database system for Meteor [11]. Installation of MongoDB is not necessary to create and run Meteor projects. Meteor has its own local MongoDB package [10]. Meteor does not support other database systems [11]. By default, every Meteor projects automatically creates its own MongoDB database and a MongoDB setup configuration is not required. Whenever a Meteor project is running the local server is running and MongoDB starts.

3 Technologies and Implementation

3.1 Development Environment

3.1.1 System Setup

A Windows 7 64-bit Operating System (OS) computer, an Atom text editor, Git version control, Meteor Installer, Heroku’s Command Line Interface (CLI) and Heroku’s free hosting plan were used during the development process. The Atom text editor was used to write source codes while Git version control was used to control the different version of the projects and also to push the project source code to Heroku. Heroku was used to deploy the application in a hosting plan and to check the performance of Blaze and React web applications. Meteor uses CLI commands and the Windows OS CLI was used to create projects, run projects, install, un-install packages and test meteor projects.

3.1.2 Designing User Interface

Before starting coding, a similar wire-frame or mock-up for both Blaze and React web applications were designed. The login page requires a user e-mail and password. This page has two buttons:one to submit user e-mail and password and the other is a link to the sign up page. Similar to the login page, the sign up page also requires a user e-mail and password and it has two buttons, one button to submit user information and the other is a link to the login page.

As figure 2 illustrates, the login and signup page of Blaze with Meteor and React with Meteor web applications is designed using wireframe. The UI of the web follows the common pattern of a signup and login page design.

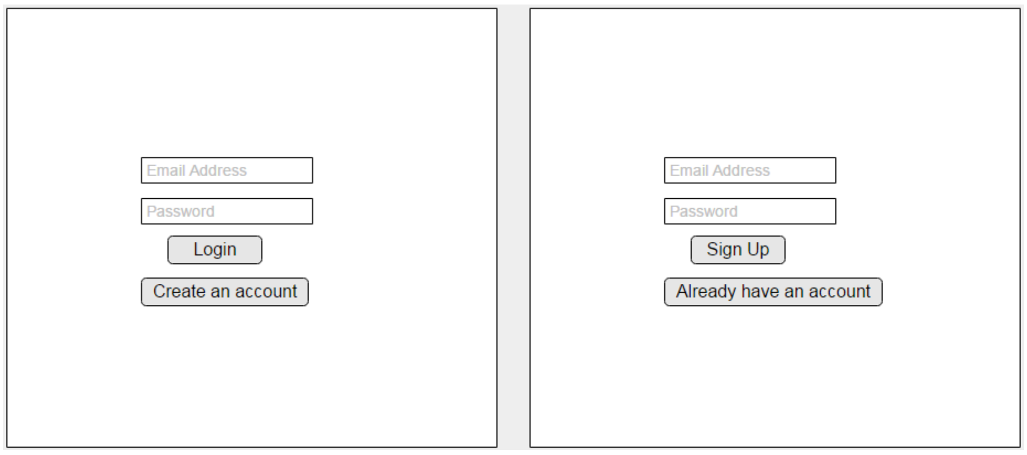


Figure 2: A sign up and login page both of both Blaze and React web applications

After users logged in, the landing page will be accessible as seen in figure 3. A user can add players, remove players, give and take points from players. A player with highest rank has to move to the first place and he will be the leader.

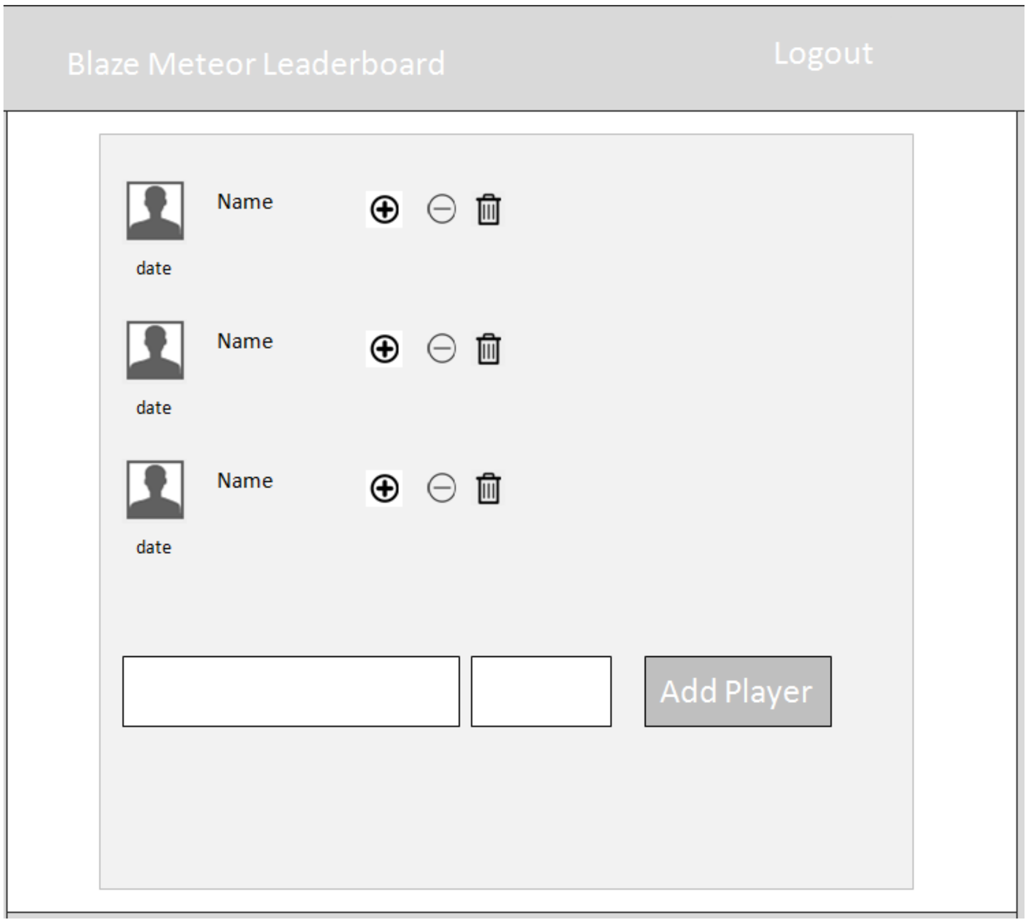


Figure 3: Blaze with Meteor web application landing page

As seen in figure 3, a logged in user can access all features of the application. A user can do the CRUD operations for this interface. A user can leave the page by logging out from this page.

3.1.3 Software Installation

The Atom text editor was installed by downloading Atom Windows installer 64-bit from atom.io website. Git version control application was installed by downloading Git Windows installer 64-bit from git-scm.com website. The latest version of Meteor, version 1.4.3.2 was installed on a Windows 7 64-bit OS computer from the official website of Meteor. Meteor was installed on Windows simply by downloading the official Meteor installer and running the InstallMeteor.exe. Heroku CLI was installed by downloading Heroku CLI Windows installer 64-bit.

3.2 Meteor Projects

3.2.1 Creating Meteor Projects

After setting up the development environment, a Meteor project was created. Creating meteor project is simple and fast. As illustrated in listing 8, a project named Leaderboard-blaze was created using the CLI. The project was created and launched using the CLI commands illustrated in listing 8. Meteor itself sets up everything and gives a boilerplate application during project creation and this boilerplate code can run without any additional setup and modification.

```
C:\Users\Asab\Desktop>meteor create Leaderboard-blaze
C:\Users\Asab\Desktop>cd Leaderboard-blaze
C:\Users\Asab\Desktop\Leaderboard-blaze>meteor run
[[[[[ C:\Users\Asab\Desktop\Leaderboard-blaze ]]]]]
=> Started proxy.
=> Started MongoDB.
=> Started your app.
=> App running at: http://localhost:3000/
Type Control-C twice to stop.
```

Listing 8: Creating a meteor project with a name Leaderboard-blaze

The shell script in listing 8 shows that the application is running on a local machine at `http://localhost:3000/` and MongoDB is initiated. While the local machine server is running it is possible to modify the project and developing the application. The running project can be stopped by typing `Ctr + C` twice.

A Meteor project creates a `client`, `server`, `.meteor` and `npm module` folders and `package.json` file. Meteor local packages such as Blaze-templating engine, MongoDB and others are stored in `.meteor` folder. Codes which run in the client will be in the `client` folder, codes which run in the sever will be in the `server` folder. The `package.json` file stores the dependencies of modules installed in the application. `npm module` is used to install or remove different packages.

The shell command in listing 9 represents the file structure of a Meteor web application project. A similar file structure were used for both Blaze and React except the projects' name. The name of the Blaze with Meteor application is Leaderboard-blaze whereas the React with Meteor application is Leaderboard-react.

```
Directory of C:\Users\Asab\Desktop\Leaderboard-blaze
04/05/2017  11:03 PM                  14 .gitignore
04/05/2017  11:04 PM    <DIR>          .meteor
04/05/2017  11:03 PM    <DIR>          client
04/05/2017  11:04 PM    <DIR>          node_modules
04/05/2017  11:03 PM                 190 package.json
04/05/2017  11:03 PM    <DIR>          server
```

Listing 9: File structure of Blaze with Meteor web application

Meteor uses lazy-loading system which means all files in `client` folder will be rendered without being imported. Therefore, after a Meteor project is created, the file structure in listing 9 has to be customized to avoid lazy-loading. According to Meteor's official file structure, files which are going to be imported will be in the `imports` folder. In the next section, the file structure of both Blaze and React with Meteor web application will be presented.

3.2.2 Meteor Projects File Structure

As it is explained in sub section 3.2.1, Meteor application has a default file structure. The default file structure is not compulsory but it is important to follow the common practice.

In this paper, the common Meteor file structure was used. The file structure of Blaze with Meteor web application is presented in figure 4. It is similar to listing 9 but with additional folders. The initial Meteor boilerplate does not have an `imports` folder. Files in the `imports` folder can be rendered only if they are imported into the main client folder. Inside the `imports` folder, there are `api`, `other client` and `ui`. The `api` folder contains the player collection. The `client` folder, inside the `imports` folder contains styles CSS and Super Set of CSS (SCSS) file. The `ui` folder contains Blaze templates and template helpers to render the UI of the web application. All the folder and sources codes of the Blaze with Meteor web application are found on github under Meteor-Blaze-Leaderboard [1] repository and in Appendix 3.

[1]https://github.com/Asabeneh/Meteor-Blaze-Leaderboard

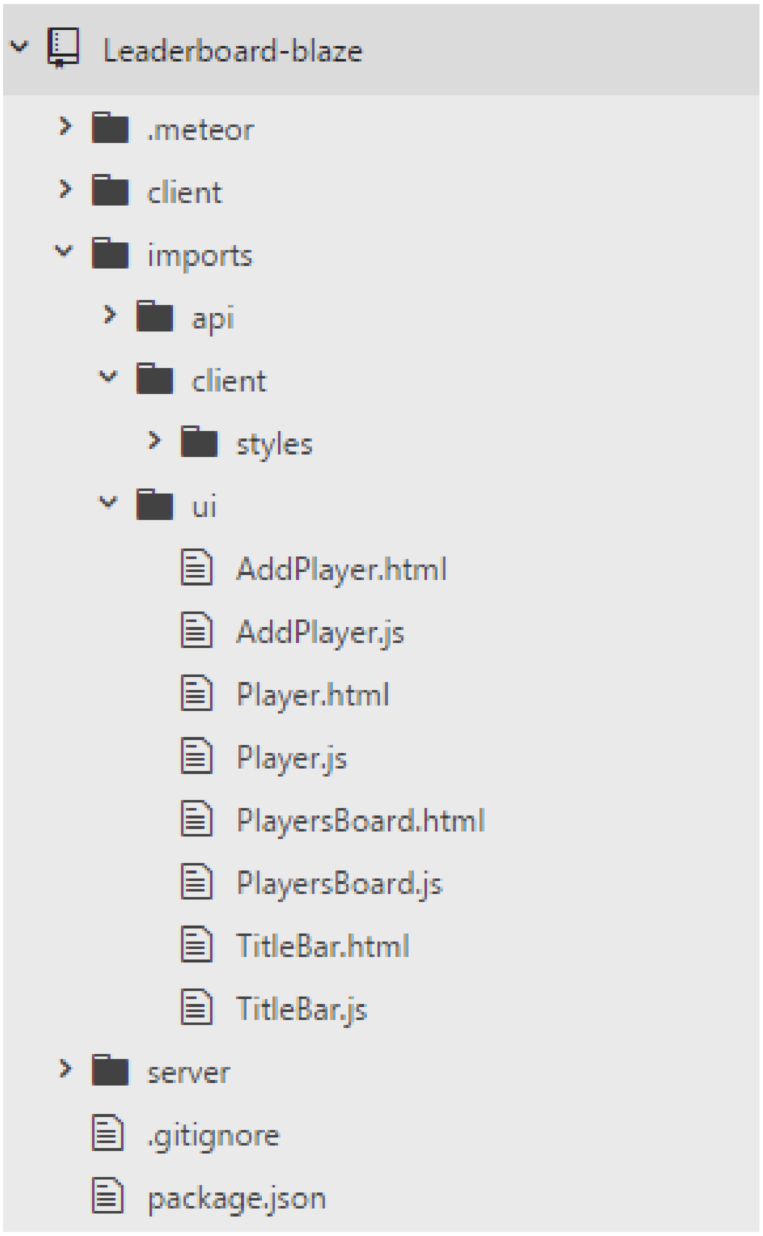
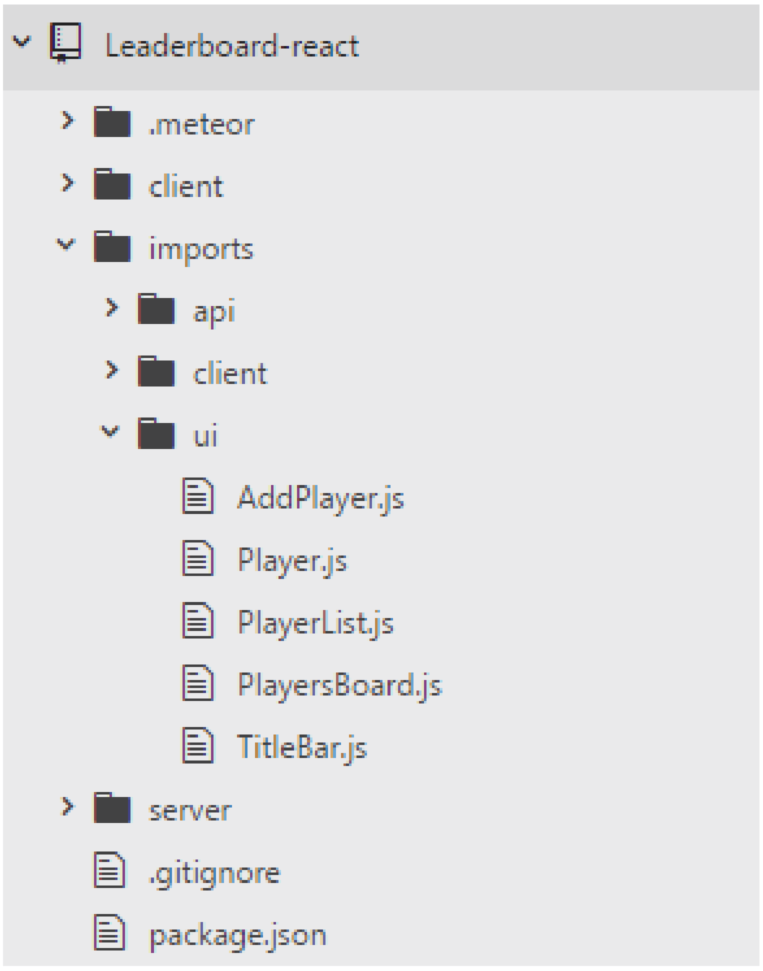


Figure 4: Files in the Blaze with Meteor web application

As in figure 5 , the folder and file structure represents React with Meteor web application. It has similar structure with the Blaze with Meter web application but only differ inside the `ui` folder. Here, React uses JSX to render data inside HTML. Therefore, only JS files are included. However, in Blaze there are separate template files in addition to JS files. All the folder and sources codes of the Blaze with Meteor web application will be found on github under React-Meteor-Leaderboard2 repository [2] and Appendix 3.

[2]https://github.com/Asabeneh/React-Meteor-Leaderboard2



```
Leaderboard-react
  .meteor
  client
  imports
    api
    client
    ui
      AddPlayer.js
      Player.js
      PlayerList.js
      PlayersBoard.js
      TitleBar.js
  server
  .gitignore
  package.json
```

Figure 5: Files in the React with Meteor web application

3.2.3 Meteor Packages

Packages are client side or server side JS libraries available for Meteor projects. There are two types of packages: Meteor official packages and third party packages. The former packages are developed by the Meteor Development Group (MDG) and they are installed on CLI of the project directory using the command in listing 10. The latter packages are provided by third parties. The command `meteor add packagename` adds different Meteor official packages to meteor project whereas if the package is a third party package, `meteor add packagename:thirdPartyName` is used to add third party packages into the project. For instance, to install one of Meteor's official package, the account package, `meteor add accounts-password` and to install bootstrap a third-party package, `meteor add twbs:bootstrap` are used.

All Meteor packages are available on Meteor's package management website, atmospherejs.com. Currently, there are more than 12,000 meteor official and third party packages altogether [10]. In this paper different packages were used for both Blaze and React web applications. Most of the packages are third-party packages.

As listing 10 shows, both Blaze and React applications used similar packages. The `account-password`, `bootstrap`, `fontawesome`, `scss`, `simple-schema`, `moment` and `numeral` were installed in the applications. The packages are added the the project as shown in listing 10.

```
    meteor add accounts-password
    meteor add twbs:bootstrap
    meteor add fortawesome:fontawesome
    meteor add fourseven:scss
    meteor add momentjs:moment
    meteor add aldeed:simple-schema
```

Listing 10: Packages installed in Blaze and React Meteor web applications

As listing 10 demonstrated, different packages were used for these projects. An `account-password` adds a user account management package. The `fontawesome` adds fontawesome package to the application which enables adding web icons such as delete icon, plus icon and minus icon. The `scss` package adds SCSS to the application which is a special type of CSS styling. The `simple-schema` package adds simple-schema package which helps to model the schema for data validation. The `moment` adds moment package to the application which has different time formats. The `numeral` adds numeral package to the application which is used to change the ranking system from cardinal to ordinal numbers.

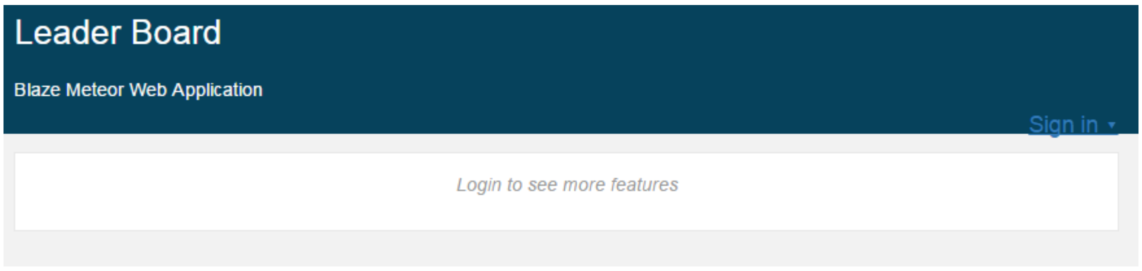
4 Results

4.1 Web Applications

4.1.1 Blaze Applications

Meteor leaderboard web applications were built using both Blaze and React. Leaderboard is an application which puts items, users, lists to the top based on their values. In this case, the leaderboard gives ranks to players which are added by a user. The Leaderboard application used to be Meteor's framework sample tutorial to learn the framework. Now, it is replaced by simple-to-do [10]. Both Blaze[3] and React [4] web applications were deployed on Heroku and and are available on the links in the footnote.

The Blaze web application gave the UI depicted in figure 6, illustrating the sign up and log in page of the application. A user has to sign up to access features and functions of the application. Since the user is not yet signed up or logged in, it informs the user to log in and to access more features.


Leader Board

Blaze Meteor Web Application

Sign in ▾

Login to see more features

Figure 6: Blaze with Meteor web application log in page

A logged in user can access the application features. As figure 7 shows, the landing page of the application which was produced by Blaze with Meteor application. The landing page has a header, informative message below the header if there is no data entered, two data input and a button to add data. In this case, the user has not added any data yet. At the top, it informs the user to start entering data. A user can add a player name

[3]https://asab-blaze.herokuapp.com/
[4]https://asab-react.herokuapp.com/

alone or player with value. Other features of the application such as remove players, give point, take point and remove all players will be visible when the user adds data to the application.

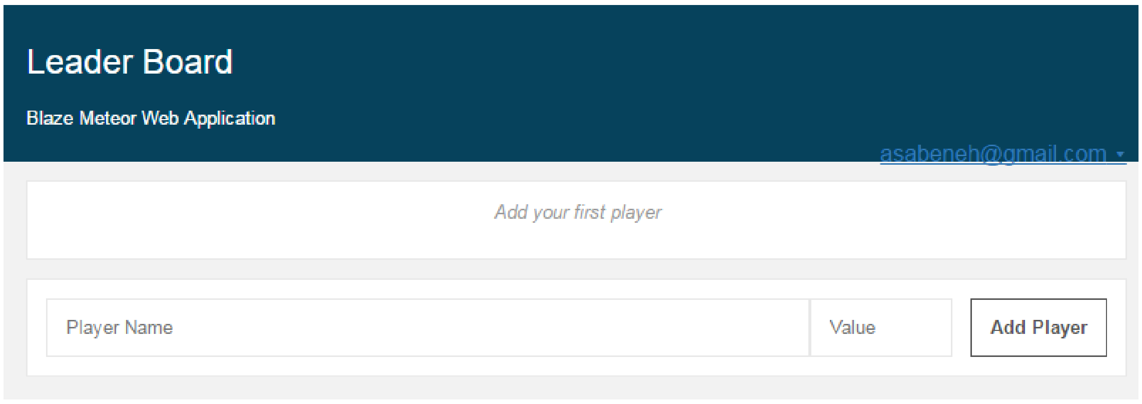
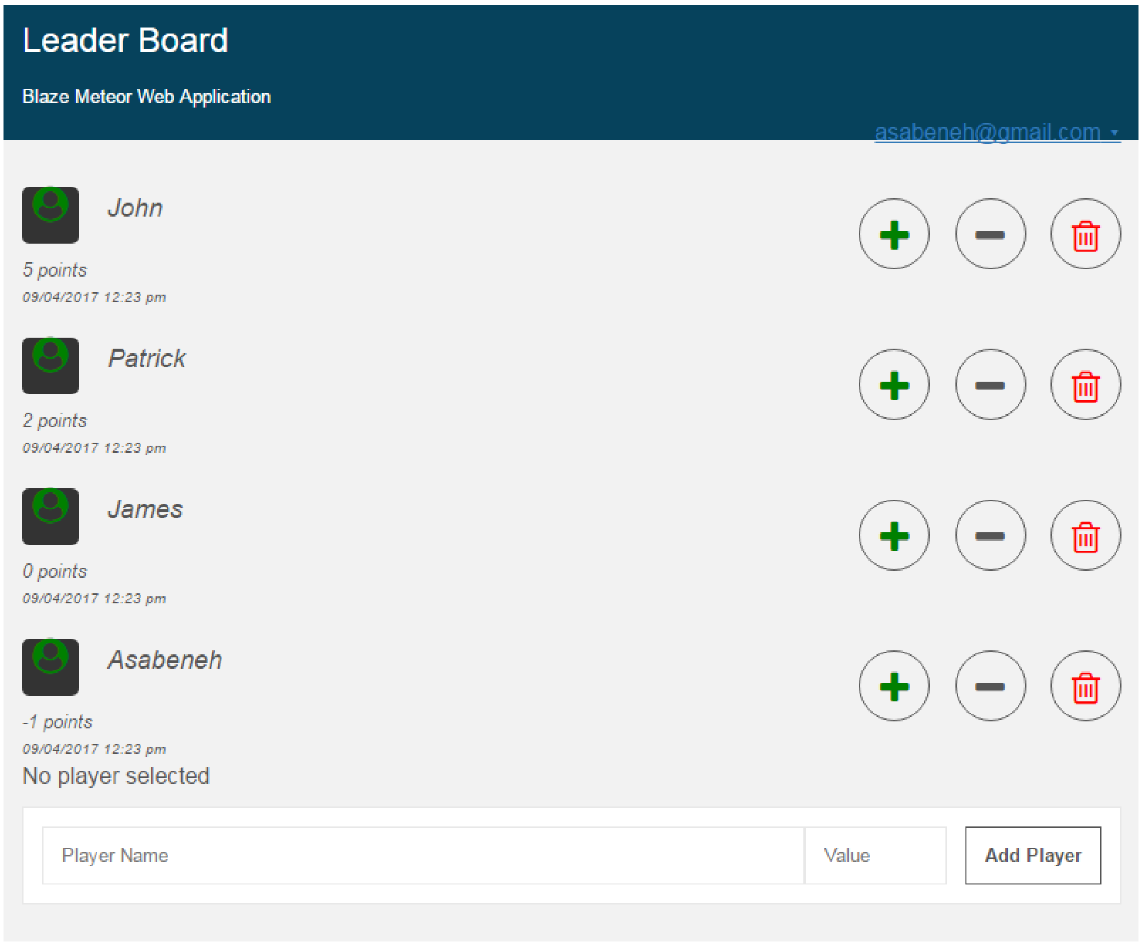


Figure 7: Blaze with Meteor web application without data

After a user logged in, it starts adding data to the application. In figure 8, the user added four players. Now, the application in figure 8 shows plus button, minus button and remove button to perform CRUD operations.


Leader Board

Blaze Meteor Web Application

asabeneh@gmail.com

John

5 points

09/04/2017 12:23 pm

Patrick

2 points

09/04/2017 12:23 pm

James

0 points

09/04/2017 12:23 pm

Asabeneh

-1 points

09/04/2017 12:23 pm

No player selected

Figure 8: Blaze with Meteor web application with data

A user can add additional players, remove players, give points to players and take points from players using the available features in figure 8. Players with the highest point will move to the top while players with equal point will be arranged in ascending order of their names.

4.1.2 React Applications

As figure 9 illustrates, the landing page of the React web application which is almost similar to Blaze web application. The application does not have any data yet and it informs a user to add data. Since there is no data, the other available features are not visible.

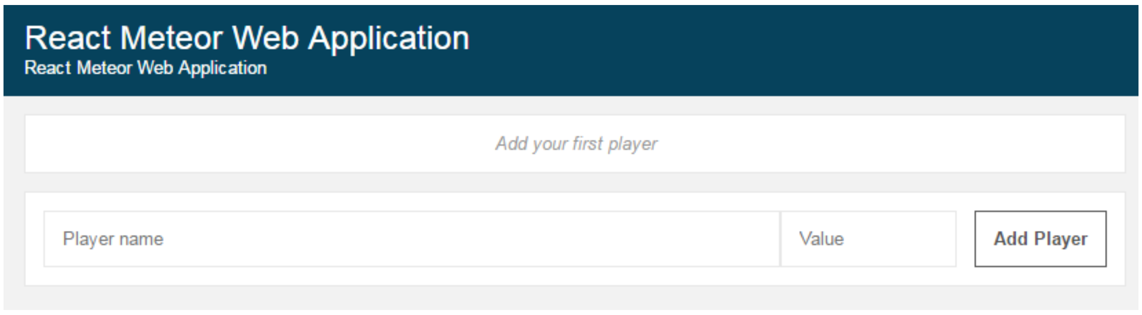


Figure 9: React Meteor web application without data

The landing page UI in figure 9 has two input fields. One of the field is to add a player name and the other to add value of the player. Next to the value field, there is an add button to insert the input data to the database collection.

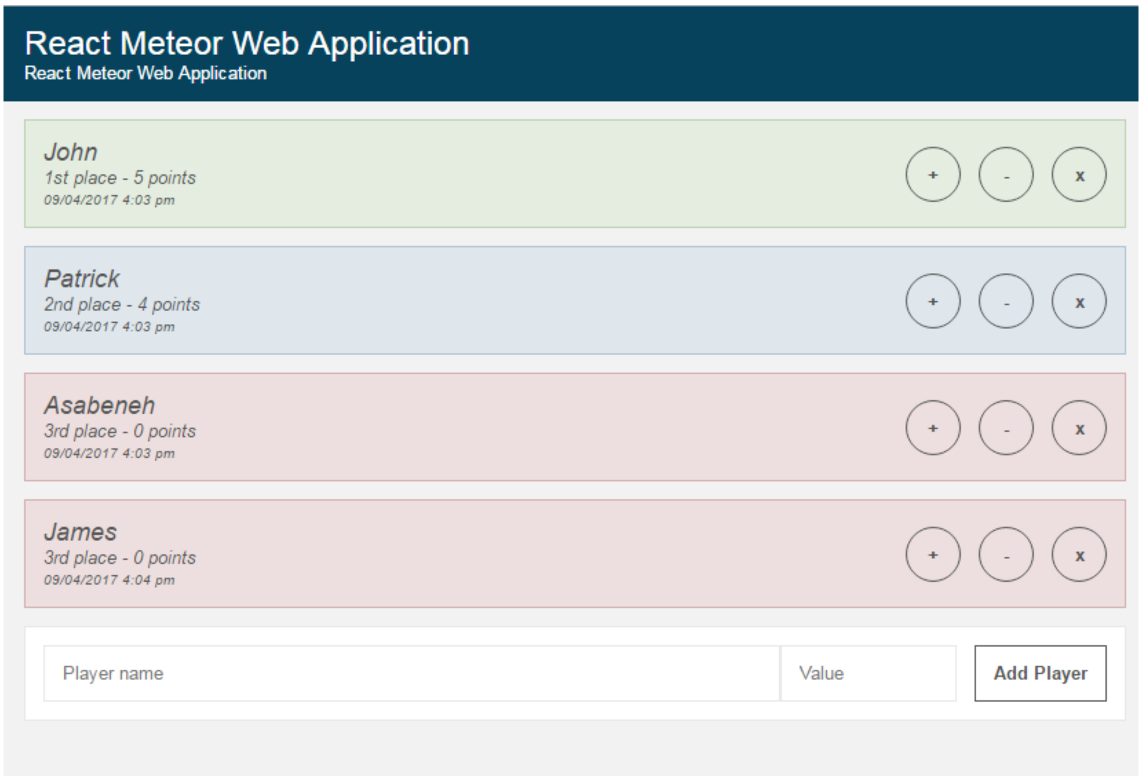


Figure 10: React Meteor web application with data

As figure 10 shows the plus button, minus button and remove button will be available for use when data is entered on the React web application. Now, a user can add more players, remove players, give points to players or delete players. Players with the highest points move to the top and he will be the leader.

4.2 Comparison Parameters

4.2.1 Development Community

React, Blaze and Meteor Stack Overflow search, Github Watch, Github Stars and Github Forks results are presented in table 2 which can be used to know about the development community. The data were collected from Stack Overflow and github websites.

Table 2: Development community (Source from Github repositories of Blaze, React and Meteor (Data gathered from github(2017)) [3] [19] [20] [21]).

Technologies	Stack Overflow	Github Watch	Github Stars	Github Fork
Blaze	3,616	43	301	69
React	95,857	4,322	63,387	11,722
Meteor	57,772	1.921	36,993	4,653

In table 2, a Stack Over flow search using the key word React is 25 times higher than Blaze. Meteor’s Stack Over flow search result is lower than React but higher than Blaze. Github Watch of React is 100 times higher than Blaze and 2 times higher than Meteor. This indicates that, React Github repositories are followed by higher number of developer than Blaze or Meteor.

4.2.2 Programming Pattern

Blaze and React have certain similarities when the programming pattern is considered. Blaze uses templates for view and Spacebar engine for logic. The Spacebar engine injects data to templates. Therefore, there should be a separate template file and JS file. These reduce convenience during programming and make the file size larger.

React uses JSX to write HTML element with JS and it is possible to mix the UI and the logic together in one file. This reduces the file size and increases convenience during programming.

4.2.3 Page Loading Speed

John Stevens, a web design blogger recommends Google Page Insights[5], Pingdom [6], GTmetrix [7] and WebPageTest [8] on-line open source test tools to test the speed of a Website [22].

Pingdom, GMtetrix and WebPageTest testings tools were implemented. The data show in table 3 is produced by the performance test using Pingdom. Pingdom counter-checked against all websites that are checked using Pingdom test and provides the grade of the web application in alphabet and percentage, page loading speed and page size.

Table 3: Performance Test Using Pingdom Website Speed Test

Web Application	Performance	Load time	Page size	Relative Speed
Blaze	92 A	1.52 s	348.2 kB	80
React	92 A	1.28 s	228.3 kB	85

As table 3 shows, both Blaze and React web application gave 'A' grade and 92% average performance. The full page loading of React is a bit faster than Blaze but both have fast loading speed. The page size of Blaze is much larger than React.

GTmetrix gives practical insights on how to optimize the page speed. As table 4 illustrates the GTmetrix result of Blaze and React web applications.

Table 4: Performance Test Using GTmetrix

Web Application	Performance	Load time	Page size	YSlow Score
Blaze	71 A	2.3 s	350 kB	69
React	71 A	2.0 s	230 kB	69

As table 4 shows Blaze and React web application have 'A' grade and 71% average performance. Similar to the Pingdom test result in table 3, the full page loading time of the React web application is a bit faster than Blaze and React page size is smaller than Blaze. WebPageTest: enables testing on different platforms using realistic data. As table

[5]https://developers.google.com/speed/pagespeed/insights/
[6]https://www.pingdom.com/
[7]https://gtmetrix.com/
[8]https://www.webpagetest.org/

5 illustrates the WebPageTest result of the Blaze and React web applications.

Table 5: Performance Test Using WEBPAGETEST

Web Application	Performance	Load Time	Page Size	Speed Index
Blaze	A	1.79 s	357 kB	813
React	A	1.49 s	237 kB	857

Similar to Pingdom and GTmetrix test results, the WEBPAGETEST test in table 5 showed that both React and Blaze gave a ”A” performance. The full page loading time and file size of React is a bit smaller than Blaze.To summarize website speed test result, React has a faster page loading speed and small file size in all the three page loading performance tests.

5 Discussions

In this chapter, the results from Chapter 4 will be discussed in detail and evaluated step by step. First, the Blaze with Meteor and React with Meteor web applications UI and UX will be discussed.

Both Blaze and React web applications gave the expected result and showing almost all features of UI are available in both applications. Some packages are easy to use in Blaze, however, when it comes to React it might be a bit tricky. For instance, to add a web icons by adding `fontawesome` package is straight forward in Blaze but to add the same package takes a long time and it need a lots of tweaking. The difference can be seen by looking at figure 7 which is the landing page of Blaze with Meteor web application and figure 10 which is the landing page of React with Meteor web application. Both Blaze and React with Meteor can give a high standard web application with good UI and User Experience (UX). According to this study, it could be possible to design web applications which have similar UIs and UXs both in Blaze and React.

Knowing the development community of a certain language is an important step before choosing a framework. According to Stack Overflow search result, Github Watch, Github Stars and Github Forks, one can give an estimate of the community activity of a certain framework. As table 2 depicts that React has a higher number of Stack Over flow search results, higher number of Github Stars and Github Forks than Blaze or Blaze and Meteor altogether. This showed that React has a bigger and more active community than Blaze or Blaze and Meteor together.

As illustrated in table 2, a Stack Over flow search using the key word React is much higher than Blaze or Meteor. On the official React github repository, React has 20 times more stars than Blaze [20]. This clearly indicates that a large number of developers and companies are using React. Furthermore, after Meteor started integrating React and Angular, Blaze developers are switching from Blaze to React or AngularJS. Though Blaze is not as popular as React, Blaze with Meteor can produce any kind of application with high productivity and simplicity. Now, let us examine, the programming pattern of Blaze and React.

Blaze and React follow a different programming pattern. Blaze uses templates for view and Spacebar engine for logic. The Spacebar engine injects data to the templates. Therefore, there should be a separate template file and JS file which is cumbersome to move from a template file to JS file in a Blaze application. These reduce convenience during programming and makes the file size larger and in return the loading speed gets lower. However, React uses RJX to write HTML elements with JS and it is possible to mix the UI and the logic together in one file. This reduces the file size and increases convenience during programming and this makes the page load faster. The folder structure of both the Blaze and React application is presented in figure 4 and figure 5 respectively.

Website loading speed improves UX and it has to be taken into account during web application developments. In addition to UX, SEO or ranking by Google is also affected by the speed of the page loading [22]. Performance testing before launching will help to improve UX. In addition to that, sites which load faster do have a high SEO rank [22] [9]. “An application that can only run in the client-side cannot serve HTML to crawlers, so it will have poor SEO by default. Web crawlers function by making a request to a web server and interpreting the result; but if the server returns a blank page, it’s not of much value” [9]. As table 3 shows, the performance of both Blaze and React web application give ’A’ grade and with 92% average performance . The full page loading time of React is lower than Blaze whereas the page size of Blaze web application is higher than React in Pingdom loading speed test. The Blaze file is larger than React because Blaze uses a separate file for template and JS source code. In addition, the React web application relative speed is higher than Blaze.

GTmetrix test produced a ’A’ grade and 71% average performance for both Blaze and React. Similar to the Pingdom test result, the GTmetrix test also showed that the React web application is lighter and faster than the Blaze web application. Generally, the loading time of both Blaze and React in the GTmetrix test is higher than the Pingdom speed test.

As the Pingdom and GTmeterics speed test, the WEBPAGETEST in table 5 also showed similar findings. The WEBPAGETEST testing produced a ’A’ grade performance for both Blaze and React web applications. Like the other tests, the React web application is lighter and faster than Blaze.

To summarize, the three different speed testing tools, the Pingdoom, GTmetrics and WEB-PAGETEST, showed a similar result for both Blaze and React. The result of the three tests showed similar trend for both Blaze and React web applications.The GTmetrix test showed an exceptionally high page loading time for both Blaze and React web applications.

6 Conclusions

JS is a very popular programming language and JS frameworks are evolving continuously to make single page web application rendering fast. There are quite a lot of front-end JS frameworks but there only very few full-stack JS. React has a more active development community than Blaze. When it comes to convenience of programming, React is more friendly than Blaze because both the JS and HTML are written as JSX in one file whereas Blaze needs a separate file for template and JS file.

In terms of the number of available packages, React has more packages than Blaze. Blaze can use all packages meant for Meteor whereas React can use the Meteor packages and its own packages. According to Stack Overflow search result and Github, React is more popular than Blaze. React has a vibrant and active development community which is not as comparable to Blaze. Based on the page loading speed test, React is lighter and faster than Blaze. The React web application loading speed is faster than Blaze but the difference is not significant.

This study showed that React with meteor is a better choice than Blaze with meteor. The number of packages available for React is much greater than Blaze. The programming pattern is better and the number of files and file sizes which could be used during development is much less for React. The loading performance of the React web application is better than Blaze web application. The development community of React is much larger than Blaze. In addition, Blaze users are switching to React and therefore, combining Meteor with React is a better choice to develop an interactive, dynamic, single page applications. This paper recommends two points to study in the future. First, it would be worth studying the performance of Angular and React when used in Meteor applications. Second, this study suggests integrating VueJS front-end framework as an alternative view layer for Meteor applications.

References

1 W3C. Document Object Model (DOM). w3C; 2017. Available from: https://www.w3.org/DOM/ [cited April 04,2017].

2 Fishkin R. The Beginners Guide to SEO.; 2015. Available from: https://moz.com/beginners-guide-to-seo [cited April 04,2017].

3 Stack Overflow. Developer Survey Results 2016. Stack Exchange, Inc.; 2016. Available from: http://stackoverflow.com/insights/survey/2016 [cited April 04,2017].

4 Alimadadi S, Mesbah A, Pattabiraman K. Understanding Asynchronous Interactions in Full-Stack JavaScript. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE); 2016. p. 1169–1180.

5 GitHub. The state of the Octoverse 2016. GitHub, Inc.; 2016. Available from: https://octoverse.github.com/ [cited April 04,2017].

6 Arora S. JavaScript Frameworks: The Best 10 for Modern Web Apps; 2016. Available from: http://noeticforce.com/best-Javascript-frameworks-for-single-page-modern-web-applications [cited April 03,2017].

7 Wen B. 6 things you should know about Node.js; 2013. Available from: http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html [cited April 03,2017].

8 Slant. What are the best full-stack Node.js frameworks? Slant; 2017. Available from: https://www.slant.co/topics/2428/~full-stack-node-js-frameworks [cited April 04,2017].

9 Brehm S. Isomorphic JavaScript: The Future of Web Apps; 2013. Available from: http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/ [cited April 04,2017].

10 Meteor. Meteor Documentation. METEOR; 2017. Available from: http://docs.meteor.com/#/full/ [cited April 03,2017].

11 Turnbull D. Your First Meteor Application; 2014-2015. April 03,2017. Available from: http://meteortips.com/.

12 Li S. Instant web applications with Meteor; 2013. Available from: https://www.ibm.com/developerworks/library/wa-meteor-webapps/ [cited April 04,2017].

13 Agüero F. 5 reasons why I choose React-Meteor for my projects; 2016. Available from: http://fjaguero.com/blog/5-reasons-why-i-choose-react-meteor-for-my-projects/ [cited April 04,2017].

14 Meteor. Blaze. METEOR; 2017. Available from: http://blazejs.org/ [cited April 03,2017].

15 Facebook. React. Facebook; 2017. Available from: https://facebook.github.io/react/docs/hello-world.html [cited April 03,2017].

16 Orsini L. What You Need To Know About Node.js; 2013. Available from: http://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/ [cited April 04,2017].

17 Hennessy K. MeteorJS: Making Single Page Applications that are Fun to Build; 2014. Available from: https://blog.appliedis.com/2014/05/15/meteorjs-making-single-page-applications-that-are-fun-to-build/ [cited April 03,2017].

18 MongoDB Inc. Introduction to MongoDB. MongoDB Inc.; 2017. Available from: https://docs.mongodb.com/manual/introduction/ [cited April 04,2017].

19 Facebook, Inc . Facebook/React Repository. Facebook Inc.; 2016. Available from: https://github.com/facebook/react [cited April 05,2017].

20 Meteor Developer Group. Meteor/meteor Repository. METEOR; 2017. Available from: https://github.com/meteor/meteor [cited April 05,2017].

21 Meteor Developer Group. Meteor/Blaze Repository. METEOR; 2017. Available from: https://github.com/meteor/blaze [cited April 05,2017].

22 Stevens J. How slow is too slow in 2016?; 2016. Available from: http://www.webdesignerdepot.com/2016/02/how-slow-is-too-slow-in-2016/ [cited April 05,2017].

1 Meteor Packages

```
meteor search // to search all the packages
meteor seach accounts-password
meteor search twbs
meteor add twbs:bootstrap  // this add bootstrap
meteor add accounts-ui //user authentication packag
meteor list //to see the installed packages
meteor remove accounts-ui
meteor add accounts-ui-unstyled
meteor add momentjs:moment
meteor add fortawesome:fontawesome
meteor remove autopublish
meteor remove insecure
```

Listing 11: Meteor Packages

2 MongoDB Queries

```
//Information about user
Meteor.user().emails[0].address //to get emails
Meteor.user() //checkes loged in user in the console
Meteor.users.find().fetch() //dispaly the users object in the console
Meteor.user().username //to get the usename of user
Meteor.user()._id;
//Information about Collection
Players.remove({});Template
Players.find().fetch(); //to see collection data in the browser console
Players.find().count(); //to count collection data
Players.findOne()._id; //to get the id of the first data in the collection
Players.findOne(); //to see the first data information
```

Listing 12: MongoDB Queries

3 Blaze and React Source Codes

3.1 Blaze Meteor Web Application Source code

```
//AddPlayer.js
import { Meteor } from 'meteor/meteor';
import { Template } from 'meteor/templating';
import './AddPlayer.html';

Template.addPlayerForm.events({
  'submit form':function(event){
    event.preventDefault();

    var playerNameVar = event.target.playerName.value;
    var playerValueVar = event.target.playerValue.value;
    if(playerValueVar === ''){
      playerValueVar = 0;
    }else{
       playerValueVar = parseInt(event.target.playerValue.value);
    }

    var currentUserId = Meteor.userId();
    Meteor.call('createPlayer',playerNameVar,playerValueVar);
    event.target.playerName.value = "";
    event.target.playerValue.value = "";
  }
});
```

Listing 13: Blaze AddPlayer.js source code

```
//Player.js
import { Meteor } from 'meteor/meteor';
import { Template } from 'meteor/templating';
import './Player.html';
```

Listing 14: Blaze Player.js source code

```
//PlayersBoard.js

import { Meteor } from 'meteor/meteor';
import { Template } from 'meteor/templating';
import './PlayersBoard.html';
import {Players } from '../api/players';

Template.body.onCreated(function bodyOnCreated() {
  Meteor.subscribe('players');
});

```

```
Template.leaderboard.helpers({
  player: function(){
    var currentUserId = Meteor.userId();
    return
   Players.find({createdBy:currentUserId},{sort:{score:-1,name:1}});
  },
  count:function(){
    return Players.find().count();
  },
  'selectedClass':function(){
    var playerId = this._id;
    var selectedPlayer = Session.get('selectedPlayer');
    if(playerId == selectedPlayer){
      return "selected";
    }
  },

  'selectedPlayer':function(){
    var selectedPlayer = Session.get('selectedPlayer');
    return Players.findOne({_id:selectedPlayer});

  }
});

Template.leaderboard.events({
  'click .player':function(){
    var playerId = this._id;
    //console.log("You have clicked a player element");
    Session.set('selectedPlayer', playerId);

    //console.log(selectedPlayer);
    //console.log(playerId);
  },
  'click .increment':function(){
    var selectedPlayer = Session.get('selectedPlayer');
    Meteor.call('updateScore', selectedPlayer, 5);
    //Players.update({_id:selectedPlayer},{$inc:{score:5}});

  },
  'click .add__button':function(){
    var selectedPlayer = Session.get('selectedPlayer');
    Meteor.call('updateScore', selectedPlayer, 1);
  },
  'click .decrement':function(){
    var selectedPlayer = Session.get('selectedPlayer');
    Meteor.call('updateScore', selectedPlayer, -5);
        Players.update({_id:selectedPlayer}, {$inc:{score:-5}});
  },
  'click .minus__button':function(){
    var selectedPlayer = Session.get('selectedPlayer');
    Meteor.call('updateScore', selectedPlayer, -1);
    //Players.update({_id:selectedPlayer}, {$inc:{score:-5}});
  },
  'click .remove,.delete__button':function(){
    var selectedPlayer = Session.get('selectedPlayer');
    if(confirm(`Are you sure you like to Delete?`)){
     Players.remove({_id:selectedPlayer});

  }
},
  'click .removeAll':function(){
    if(confirm("Are you sure you like to delete all players?")){

    Meteor.call('removeAllPlayer');
    }
}

  });
```

Listing 15: Blaze PlayersBoard.js source code

```
//TitleBar.js
import { Meteor } from 'meteor/meteor';
import { Template } from 'meteor/templating';
import './TitleBar.html';
```

Listing 16: Blaze TitleBar.js source code

```
//players.js for MongoDB collection
import { Meteor } from 'meteor/meteor';
import { Mongo } from 'meteor/mongo';

import SimpleSchema from 'simpl-schema';
export const Players = new Mongo.Collection('players');

if(Meteor.isServer){

Meteor.publish('players',function(){
  return Players.find({});
})

}

Meteor.methods({
    'createPlayer': function(name,score){
    // check(playerNameVar, String);
    // check(playerValueVar, Number);

    var currentUserId = Meteor.userId();

    if(currentUserId){
      new SimpleSchema({
        name: {
          type: String,
          min: 2
        },
        score:{
          type:Number
        }
      }).validate({name,score});

        Players.insert({
        name,score,
        visible:true,
        createdBy:currentUserId,
        createdAt:moment().format('DD/MM/YYYY h:mm a'),
        });

    }
    },
    'removePlayer':function(selectedPlayer){
    check(selectedPlayer, String);
    var currentUserId = Meteor.userId();
    if(currentUserId){
    Players.remove({_id:selectedPlayer,createdBy:currentUserId});
    }

    },
    'removeAllPlayer':function(){
    var currentUserId = Meteor.userId();
    if(currentUserId){
    Players.remove({});
    }
    },
    'updateScore': function(selectedPlayer, scoreValue){
    check(selectedPlayer, String);
    check(scoreValue, Number);
    var currentUserId = Meteor.userId();
    if(currentUserId){
    Players.update( { _id: selectedPlayer, createdBy: currentUserId },
    { $inc: {score: scoreValue} });
    }
  }

});
```

Listing 17: Blaze player.js collection source code

```
//Blaze Templates

// main.html to render all templates
<head>
  <title>Blaze-Meteor Leaderboard</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta charset="UTF-8">
  <meta name="description" content="React Meteor Web Application">
  <meta name="keywords" content="JavaScript, SCSS, Blaze, Meteor">
  <meta name="author" content="Asabeneh">
</head>

<body>
<div>
  {{>TitleBar}}
  {{>leaderboard}}

</div>
</body>
```

Listing 18: Blaze main.html source code

```
//main.js to import Template helpers and events
import { Template } from 'meteor/templating';
import './main.html';
import '../imports/ui/PlayersBoard.js';
import '../imports/ui/TitleBar.js';
import '../imports/ui/AddPlayer.js';
```

Listing 19: Blaze main.js source code

```
//addPlayerForm template
<template name = "addPlayerForm">
<div class = "item">
  <form class = "form">
    <input  class = "form__input" type = "text" name = "playerName"
    placeholder="Player Name" required />
    <input type ="text" class = "form__input form__score" name =
    "playerValue" placeholder="Value">
    <input type = "submit" class ="button" value = "Add Player">
  </form>
</div>
</template>
```

Listing 20: Blaze AddPlayerForm.html template source code

```
<template name= "Player">
  <div>
  <div class = "player">
  <div class="player__stats">
  <h3 class = "player__name"> player name</h3>
  <p class = "player__stats">
  player stat
  </p>
  </div>

  <div class= "player__actions">
  <button class = "button button__round">
  +1</button>
  <button class = "button button__round">
  -1</button>
  <button class = "button button__round">
  X</button>
  </div>

  </div>
  </div>
</template>
```

Listing 21: Blaze Player.html template source code

```
<template name = "leaderboard">
<div>
{{#if currentUser}}
<div class ="wrapper">
<ul>
    {{#if count}}
    {{#each player}}
    <div class = "itemClassName">
    <div class = "player {{selectedClass}} {{selectedPlayer}}">
    <div class = "player__stats">
    <h3 class = "player__name"><i class="fa fa-user-circle fa-2x"
    aria-hidden="true"></i> {{name}}</h3>
    <p class = "player__stats">
    {{calculatorPlayerPositions}}{{score}} points
    </p>
    <span>{{createdAt}}</span>
    </div>
    <div class = "player__actions">
    <button class = "button button__round add__button" ><i class="fa
    fa-plus fa-2x" aria-hidden="true"></i></button>
    <button class = "button button__round minus__button" ><i class="fa
    fa-minus fa-2x" aria-hidden="true"></i></button>
    <button class = "button button__round delete__button"><i class="fa
    fa-trash-o fa-2x" aria-hidden="true"></i></button>
    </div>
    </div>
    </div>
    {{/each}}
    {{else}}
    <div class ="item">
    <p class = "item__message item__message__empty"> Add your first
    player</p>
    </div>
    {{/if}}

    {{#if selectedPlayer}}
    <li> Selected Player: {{selectedPlayer.name}}</li>

    <li> Players: {{count}}</li>
    <li>
    <button class = "btn btn-success increment"> Give 5 Points</button>
    <button class = "btn btn-warning decrement"> Take 5 Points</button>
    <button class = "btn btn-danger remove">Remove Player</button>
    <button class = "btn btn-danger removeAll">Remove All Players</button>
    </li>

    {{else}}
    {{#if count}}
    <li>No player selected</li>
    {{/if}}

    {{/if}}
</ul>
{{ > addPlayerForm}}
</div>

{{else}}
<div class = "wrapper">
<div class ="item">
<p class = "item__message item__message__empty"> Login to see more
    features</p>
</div>

</div>

{{/if}}

</div>
</template>
```

Listing 22: Blaze leaderbord template source code

```
// TitleBar template
<template name= "TitleBar">
<div class="title-bar">
<div class="wrapper">
    <h1>Leader Board</h1>
    <h2 class = "title-bar__subtitle">Blaze Meteor Web Application</h2>
    {{>loginButtons}}
</div>
</div>
</template>
```

Listing 23: Blaze TitleBar.html template source code

3.2 React Meteor Web Application Source code

```
//AddPlayer.js
import React from 'react';
import {Players} from './../api/players';
import moment from 'moment';
export default class AddPlayer extends React.Component{
  handleSubmit(e){
    e.preventDefault();
    let playerName = e.target.playerName.value;
    let playerValue = e.target.playerValue.value;
    if(playerName){
      e.target.playerName.value = "";
      e.target.playerValue.value = "";
      Players.insert({
        name:playerName,
        score:Number(playerValue),
        createdAt:moment().format('DD/MM/YYYY h:mm a')
      });
    }
  }
  render(){
    return(
      <div className = "item">
       <form  className = "form" onSubmit = {this.handleSubmit.bind(this)}>
          <input className = "form__input" type = "text" name =
    "playerName" placeholder = "Player name"/>
          <input type ="text" className = "form__input form__score" name =
    "playerValue" size = "5" placeholder="Value" />
          <button className = "button"> Add Player</button>
        </form>
      </div>
    )
    }
}
```

Listing 24: React AddPlayer.js component source code

```
//Player.js
import React from 'react';
import  {Players} from './../api/players';

export default class Player extends React.Component{
render(){
  let itemClassName = `item item--position-${this.props.player.rank}`;
  return(
  <div key = {this.props.player._id} className = {itemClassName}>
    <div className = "player">
    <div className="player__stats">
      <h3 className = "player__name"> {this.props.player.name}</h3>
      <p className = "player__stats">
```

```
        {this.props.player.position} place - {this.props.player.score}
    points
      </p>
      <span>{this.props.player.createdAt}</span>
    </div>
    <div className = "player__actions">
      <button className = "button button__round add__button" onClick =
    {()=>Players.update({_id:this.props.player._id},{$inc:{score:1}})}>+
      <i className="fa fa-plus fa-2x" aria-hidden="true"></i>
      </button>
      <button className = "button button__round minus__button" onClick =
    {()=>Players.update({_id:this.props.player._id},{$inc:{score:-1}})}>
      -<i className="fa fa-minus fa-2x" aria-hidden="true"></i></button>
      <button className = "button button__round delete__button" onClick =
    {()=>Players.remove({_id:this.props.player._id})}>
    x  <i className="fa fa-trash-o fa-2x" aria-hidden="true"></i></button>
    </div>

    </div>
  </div>

  );
}
}

Player.propTypes = {
player:React.PropTypes.object.isRequired,

}
```

Listing 25: React Player.js component source code

```
 //PlayersList

import React from 'react';
import FlipMove from 'react-flip-move';
import Player from './Player';
export default class PlayerList extends React.Component{
  renderPlayers(){
    if(this.props.players.length ===0){
    return (
      <div className ="item">
        <p className = "item__message item__message__empty"> Add your first
    player</p>
      </div>
    );
    }
    else{
      return this.props.players.map((player) =>{
          return <Player key = {player._id}  player = {player}/>
    });

    }
  }
  render(){
  return(
  <div>
    <FlipMove duration={750} easing="ease-out" maintainContainerHeight
   ={true}>
  {this.renderPlayers()}
    </FlipMove>
  </div>
  )
  }
}
PlayerList.propTypes = {
players:React.PropTypes.array.isRequired
}
```

Listing 26: React PlaeyersList.js source code

```
//PlayersBoard
import React from 'react';
import TitleBar from './TitleBar';
import AddPlayer from './AddPlayer';
import PlayerList from './PlayerList';
export default class PlayersBoard extends React.Component{
  render(){
    return(
    <div>
       <TitleBar title = {this.props.title} subtitle = "React Meteor Web
   Application"/>
      <div className = "wrapper">
        <PlayerList players = {this.props.players}/>
         <AddPlayer/>
      </div>
    </div>
    )
  }
}
PlayersBoard.propTypes = {
title:React.PropTypes.string.isRequired,
players:React.PropTypes.array.isRequired,
}
```

Listing 27: React PlaeyersBoard.js source code

```
//TitleBar.js
import React from 'react';
export default class TitleBar extends React.Component{
  rendereSubtitle(){
    if(this.props.subtitle){
      return <h2 className = "title-bar__subtitle">
   {this.props.subtitle}</h2>;
  }
  }
  render(){
    return(
    <div className = "title-bar">
      <div className = "wrapper">
       <h1>{this.props.title}</h1>
      {this.rendereSubtitle()}
      </div>
    </div>
    )
  }
}
TitleBar.propTypes = {
title:React.PropTypes.string.isRequired,
subtitle:React.PropTypes.string
};
```

Listing 28: React TitleBar.js source code