strategy | consulting | digital | technology | operations

# REACTJS – MODULE 9

## FLUX

accenture

# AGENDA

- Introduction to Flux
- Basic principle of Flux
- Flux flow
- Demo
- Activity

## Introduction to Flux

Flux is a pattern which basically implements unidirectional data flow

The unidirectional pattern gives a benefits like testing and maintaining of an application in an easy way

React can be combined with flux which gives a way to handle data inside the application

## Basic Principles Of Flux

Store

This is the area where application data/State and logic will be kept

Actions invoke changes

User actions are captured which will trigger the state change

Dispatcher

Receives the action and executes the callback to notify the store

## Store

In flux architectural pattern, stores are the area where we can keep data/state, logic and data retrieval methods of an application

Flux allows to create and maintain multiple stores for a single application

User Store

Product Store

Address Store

## Store(Cont…)

The state update will happen inside the store when it is notified by the dispatcher. In order to get notifications from dispatcher stores need to register with dispatcher

Store is not a model instead store contains model

Only stores knows how to update data in flux. React components listens to store for the state updates.

Stores emits the changes using node's event emitter

accenture

# Action

In flux pattern based on the applications requirements, multiple actions can be created

Actions are triggered by react component and notify to dispatcher about the action

Example

When the user clicks on “create user” button the corresponding action is triggered which will indicate to dispatcher that new user must be created with new state

In flux actions can also be triggered by the server whenever the call to server is made while fetching the data from the server

## Action (Cont…)

Action contains the type of the action and the data

Example

```
{
   type: Create_user
   data:   {
              name : 'Mark',
              age: 34
           }
}
```

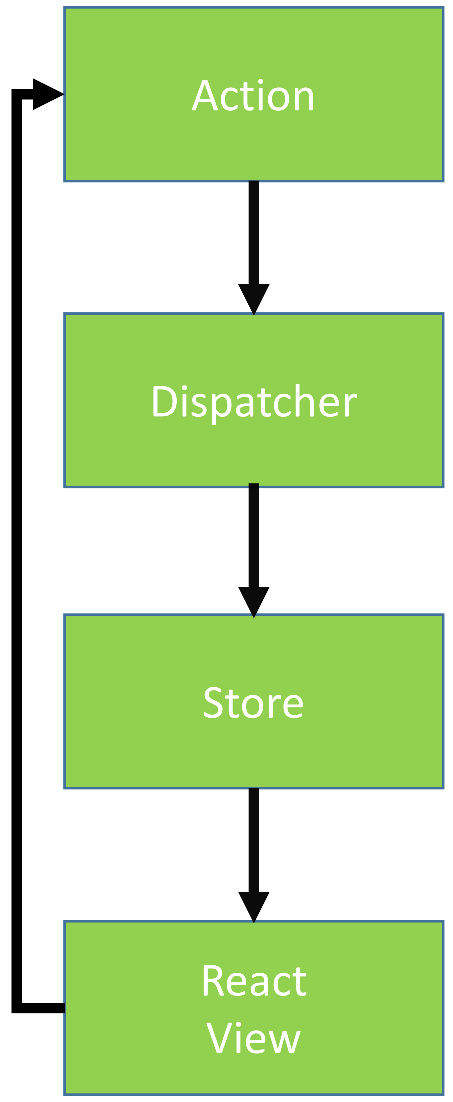
## Dispatcher

In flux there is only one centralized dispatcher can exist per application

When the actions are triggered, the dispatcher gets notified and dispatches respective actions to stores

Dispatcher simply holds the list of callbacks which are used to notify the store about the state updates based on the actions which it has received

## Flux flow



Actions are used to trigger state change request. The request will come from react components

↓

Dispatcher receives an action and the state. It uses the callback to notify the store

↓

Store gets notified about the state updates and also performs the modification of the data

↓

Once the store completes the state updates it will emit the changes using events to notify the react component about the state changes

↓

Finally react components gets rendered and UI gets updated

## How to connect React with Flux

Since Flux is a pattern which will allow to keep track of complete applications state/data in a multiple place, hence this can be used with other libraries or framework

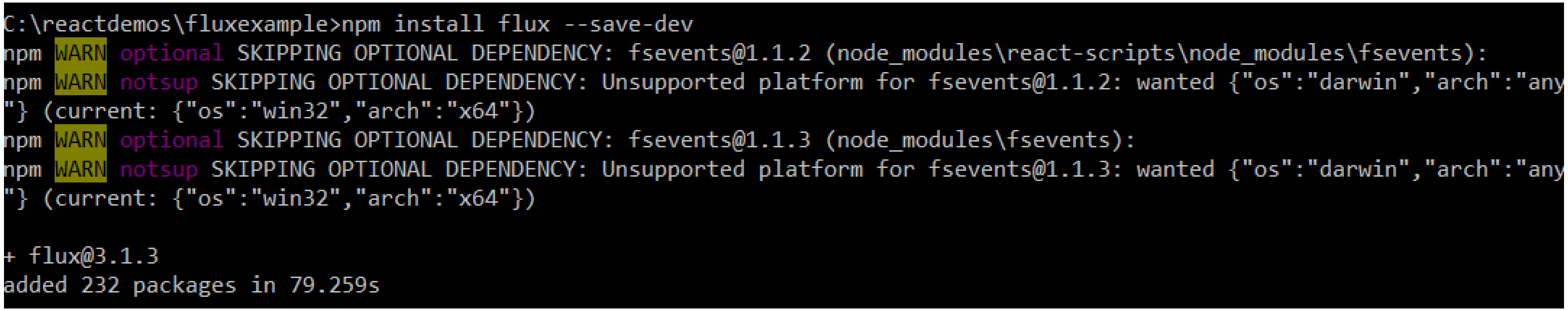
Example
Flux can be used with React library
Flux can be used with Angular framework

In this module we will be focusing on how react can be connected with Flux and utilize its features

## Demo – Flux application

Create new react application “fluxexample”

The flux can be installed using
npm install flux - -save-dev

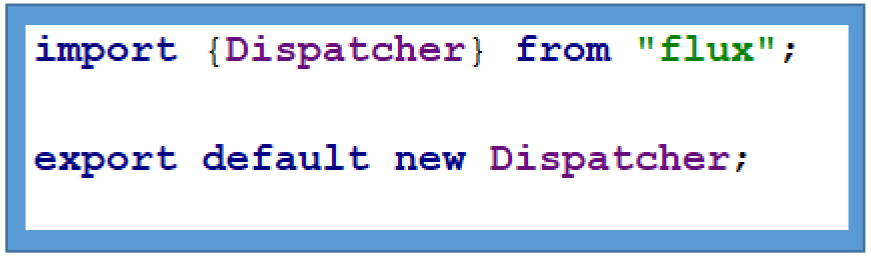


```
C:\reactdemos\fluxexample>npm install flux --save-dev
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.2 (node_modules\react-scripts\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.2: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.3 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.3: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

+ flux@3.1.3
added 232 packages in 79.259s
```

## Demo – Flux application(Cont…) - Dispatcher

Create new folder “dispatcher” inside src folder of a project. Create appDispatcher.js inside dispatcher folder

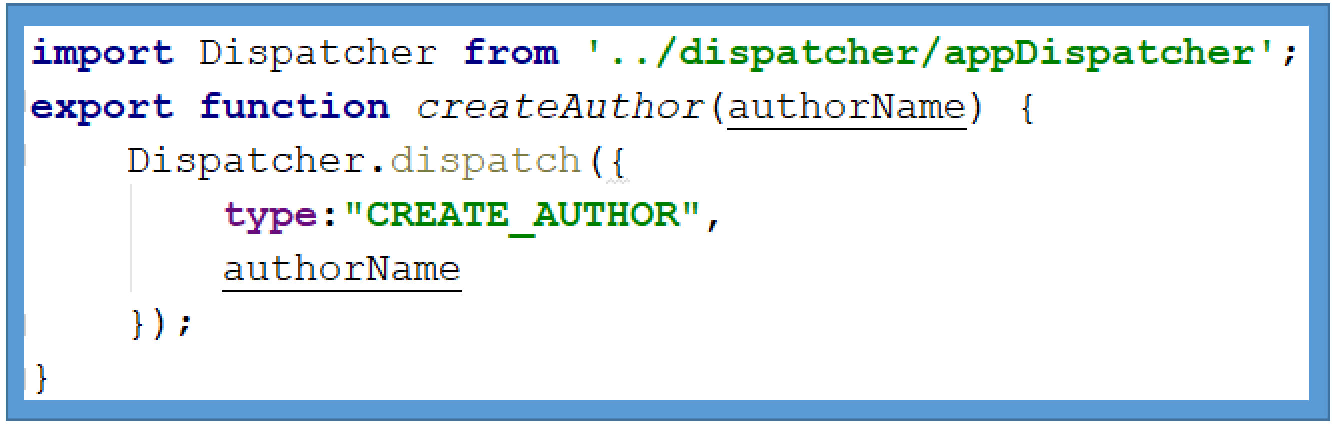
In flux pattern dispatcher is singleton. here it is created and instantiated



```
import {Dispatcher} from "flux";

export default new Dispatcher;
```

## Demo – Flux application(Cont…) - Action

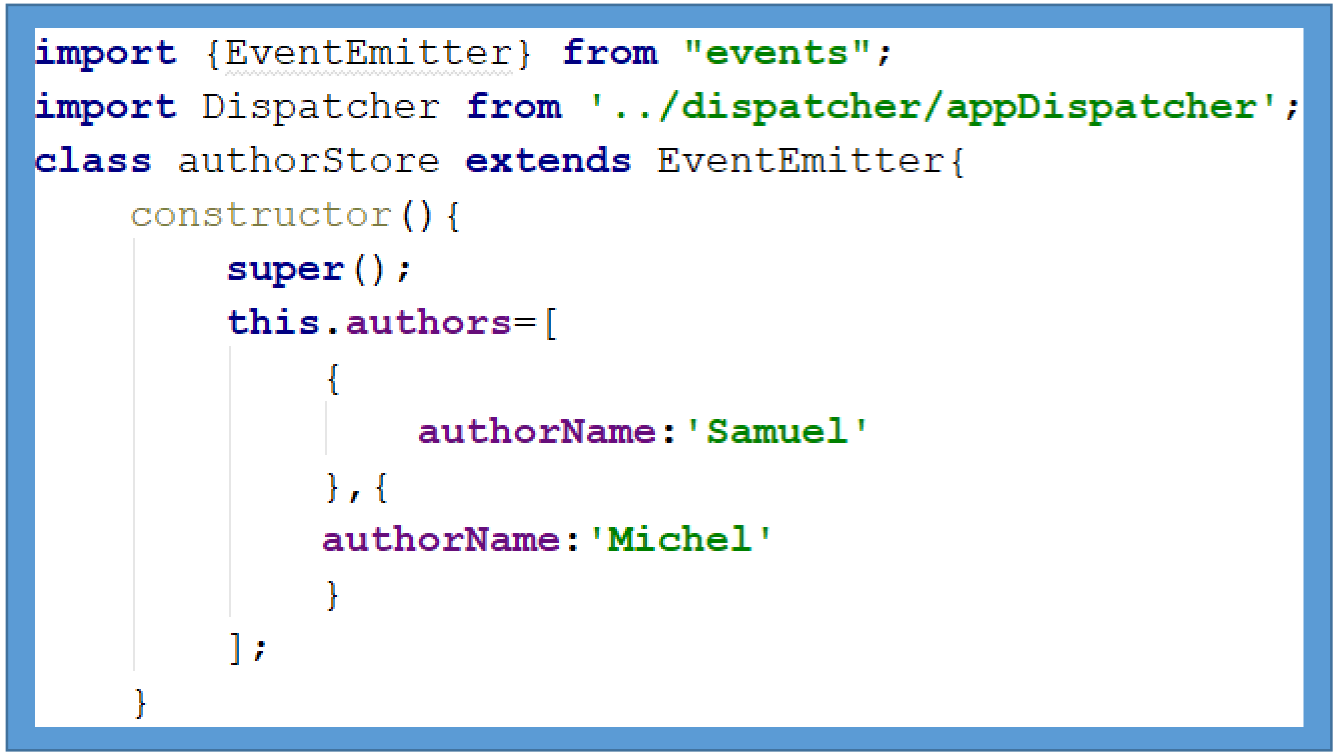
Create new folder “actions” inside src folder of a project. Create authorActions.js inside actions folder



```
import Dispatcher from '../dispatcher/appDispatcher';
export function createAuthor(authorName) {
    Dispatcher.dispatch({
        type:"CREATE_AUTHOR",
        authorName
    });
}
```

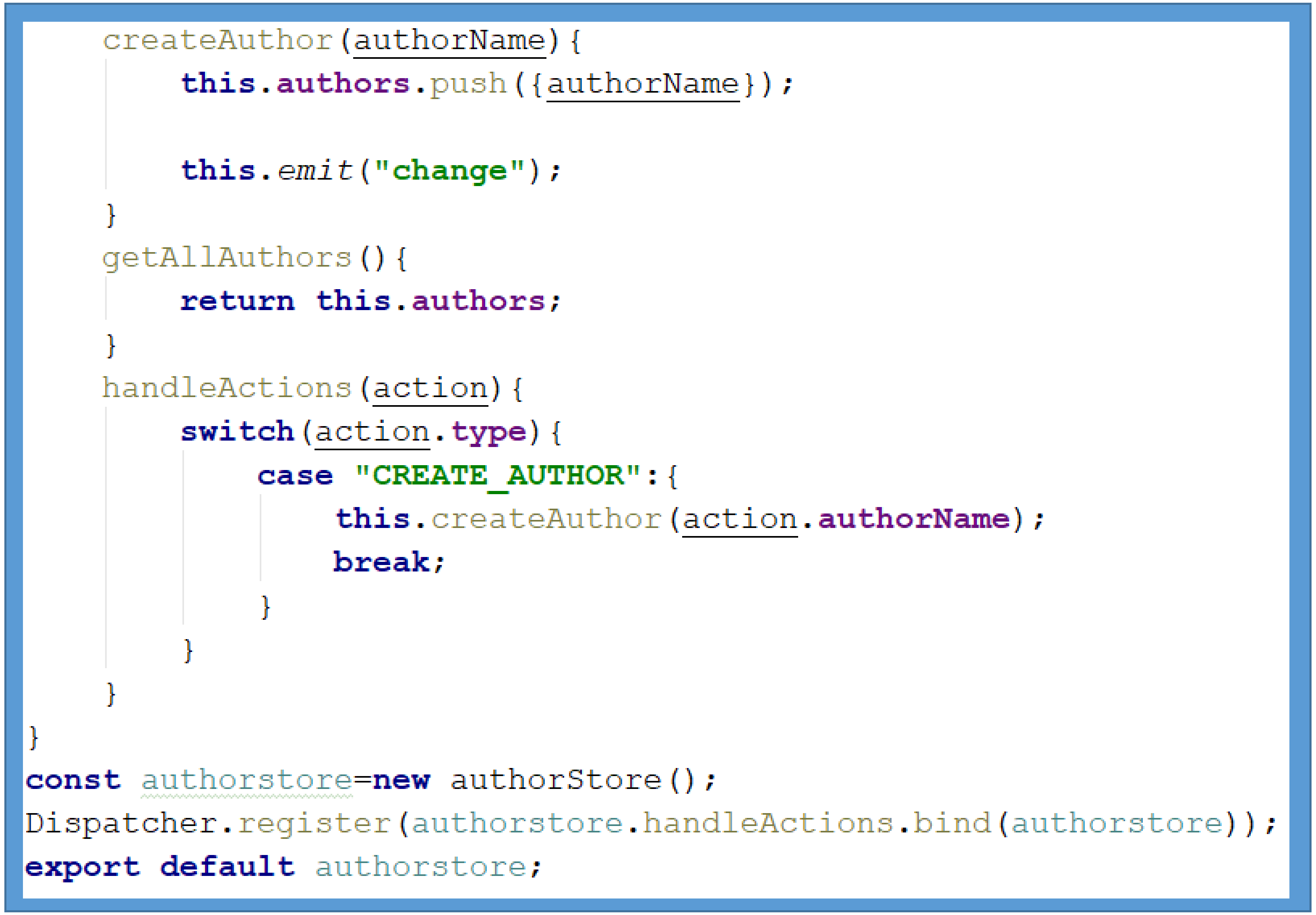
## Demo – Flux application(Cont…) - Store

Create new folder “stores” inside src folder of a project. Create authorStore.js inside stores folder



```
import {EventEmitter} from "events";
import Dispatcher from '../dispatcher/appDispatcher';
class authorStore extends EventEmitter{
    constructor(){
        super();
        this.authors=[
            {
                authorName:'Samuel'
            },{
            authorName:'Michel'
            }
        ];
    }
```

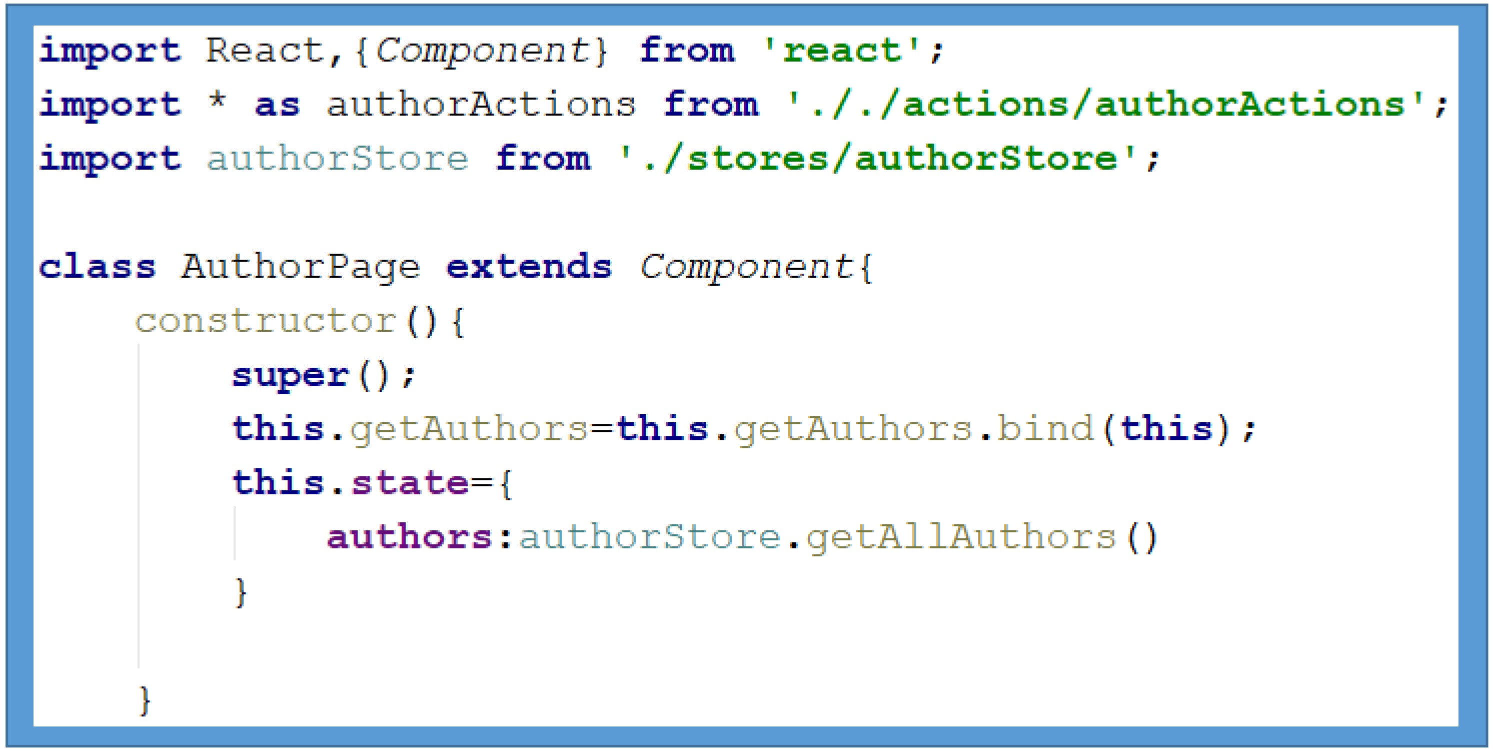
## Demo – Flux application(Cont…) - Store



```
    createAuthor(authorName){
        this.authors.push({authorName});

        this.emit("change");
    }
    getAllAuthors(){
        return this.authors;
    }
    handleActions(action){
        switch(action.type){
            case "CREATE_AUTHOR":{
                this.createAuthor(action.authorName);
                break;
            }
        }
    }
}
const authorstore=new authorStore();
Dispatcher.register(authorstore.handleActions.bind(authorstore));
export default authorstore;
```

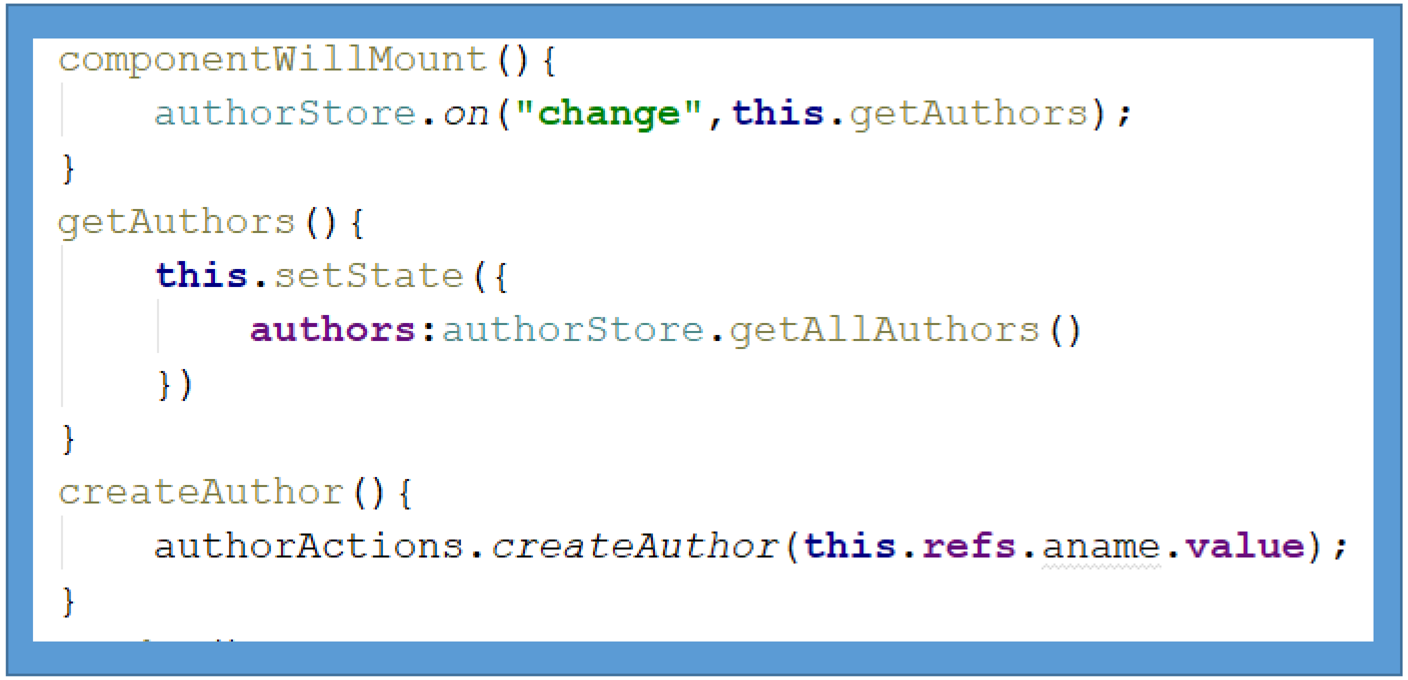
## Demo – Flux application(Cont…) - AuthorPage

Create AuthorPage.js inside src folder. This is react view



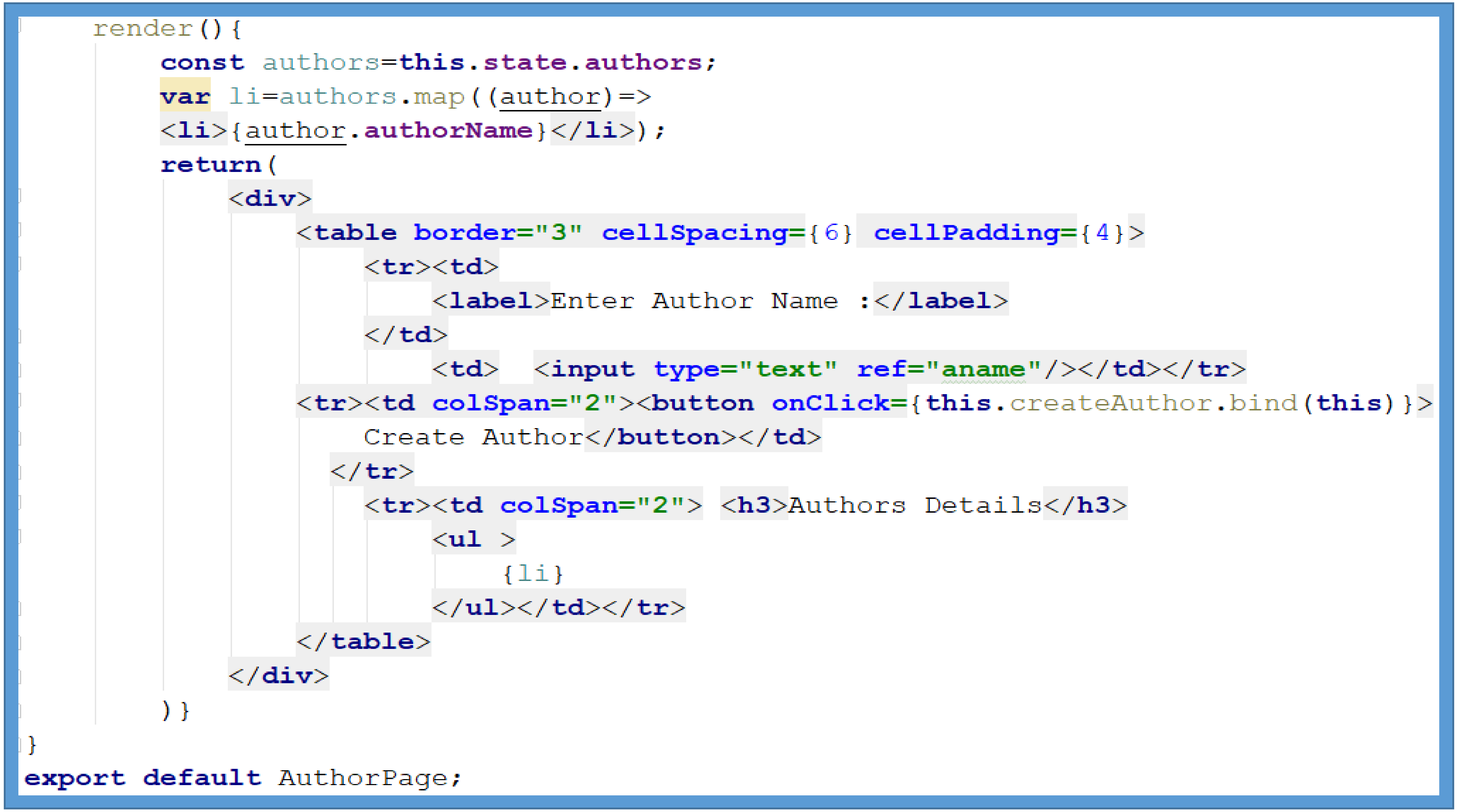
```
import React,{Component} from 'react';
import * as authorActions from './../actions/authorActions';
import authorStore from './stores/authorStore';

class AuthorPage extends Component{
    constructor(){
        super();
        this.getAuthors=this.getAuthors.bind(this);
        this.state={
            authors:authorStore.getAllAuthors()
        }

    }
```

## Demo – Flux application(Cont…) - AuthorPage



```
componentWillMount(){
    authorStore.on("change",this.getAuthors);
}
getAuthors(){
    this.setState({
        authors:authorStore.getAllAuthors()
    })
}
createAuthor(){
    authorActions.createAuthor(this.refs.aname.value);
}
```

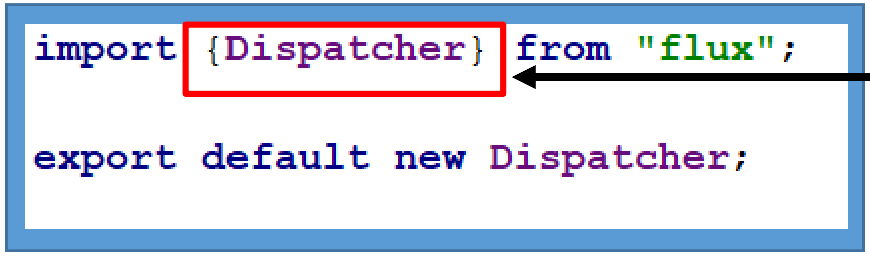
## Demo – Flux application(Cont…) - AuthorPage



```
    render(){
        const authors=this.state.authors;
        var li=authors.map((author)=>
        <li>{author.authorName}</li>);
        return(
            <div>
                <table border="3" cellSpacing={6} cellPadding={4}>
                    <tr><td>
                        <label>Enter Author Name :</label>
                    </td>
                        <td>  <input type="text" ref="aname"/></td></tr>
                <tr><td colSpan="2"><button onClick={this.createAuthor.bind(this)}>
                    Create Author</button></td>
                  </tr>
                    <tr><td colSpan="2"> <h3>Authors Details</h3>
                        <ul >
                            {li}
                        </ul></td></tr>
                </table>
            </div>
        )}
}
export default AuthorPage;
```

accenture

## Demo – Flux application Explanation

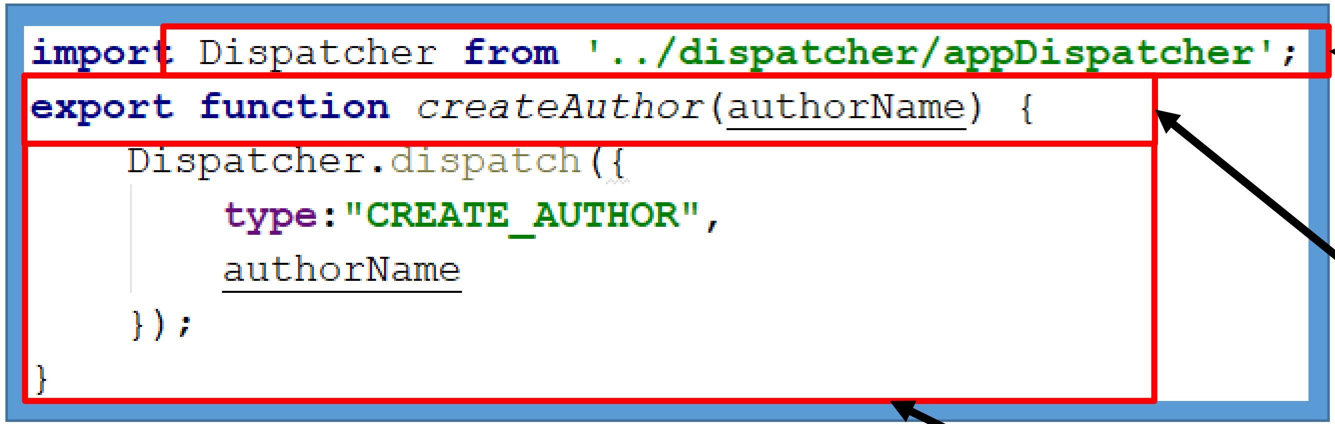


```
import {Dispatcher} from "flux";

export default new Dispatcher;
```

Importing Dispatcher from flux. It is singleton and instantiated here

accenture

## Demo – Flux application Explanation(Cont…)



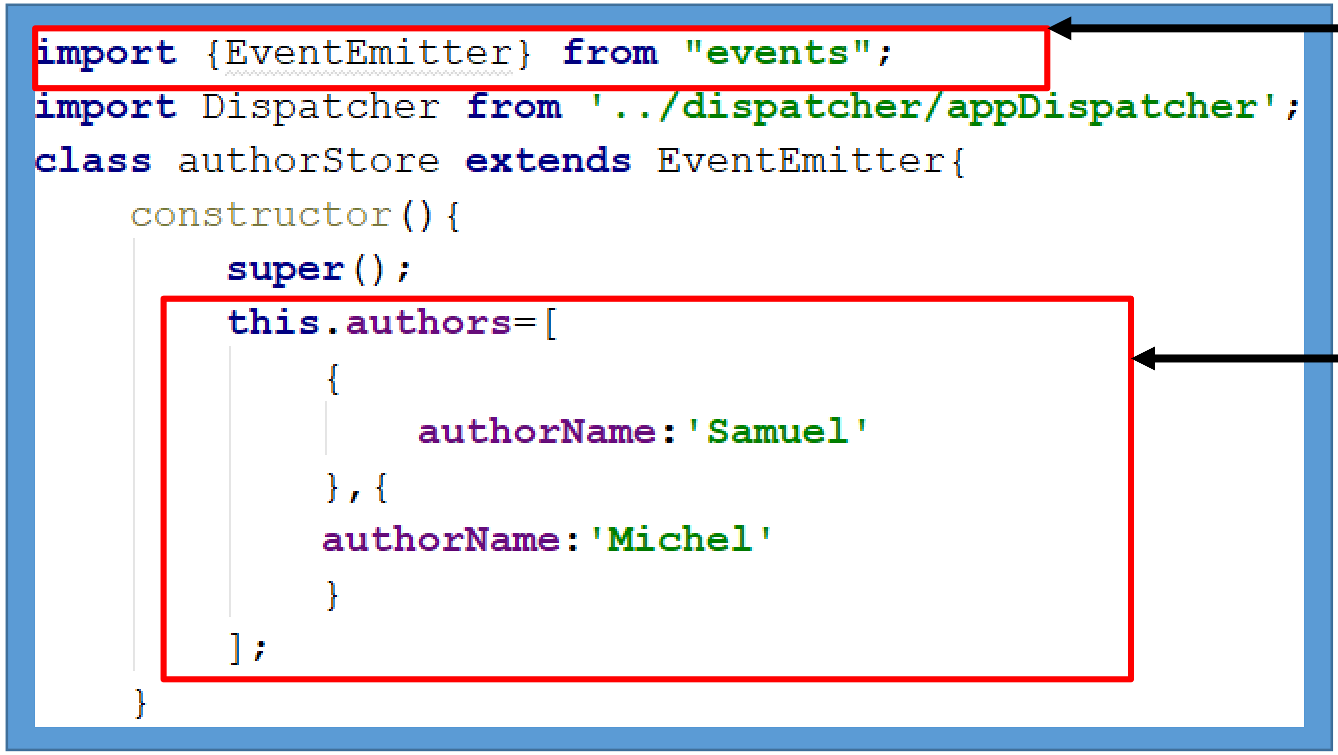
```
import Dispatcher from '../dispatcher/appDispatcher';
export function createAuthor(authorName) {
    Dispatcher.dispatch({
        type:"CREATE_AUTHOR",
        authorName
    });
}
```

Importing Dispatcher from the appDispatcher. By using dispatcher required actions can be sent to store for the state update

When the user clicks on Create Author button from the view this action will be triggered, which will also capture the author name from the view

This action will be dispatched to respective store

## Demo – Flux application Explanation(Cont…)

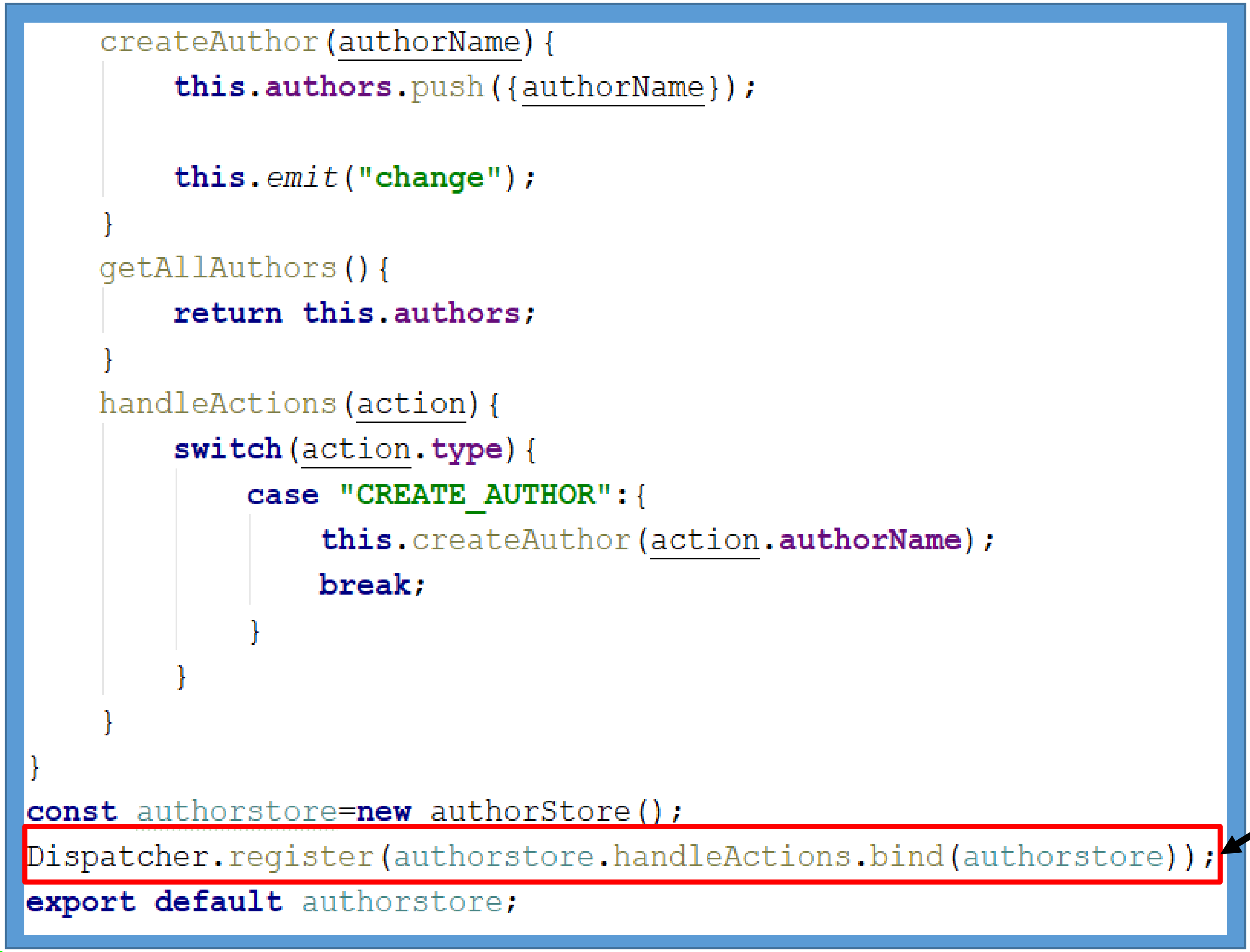


```
import {EventEmitter} from "events";
import Dispatcher from '../dispatcher/appDispatcher';
class authorStore extends EventEmitter{
    constructor(){
        super();
        this.authors=[
            {
                authorName:'Samuel'
            },{
            authorName:'Michel'
            }
        ];
    }
```

When the store modifies the data , the respective modification should be notified to react view using event emitter

The data/state of the application

## Demo – Flux application Explanation(Cont…)

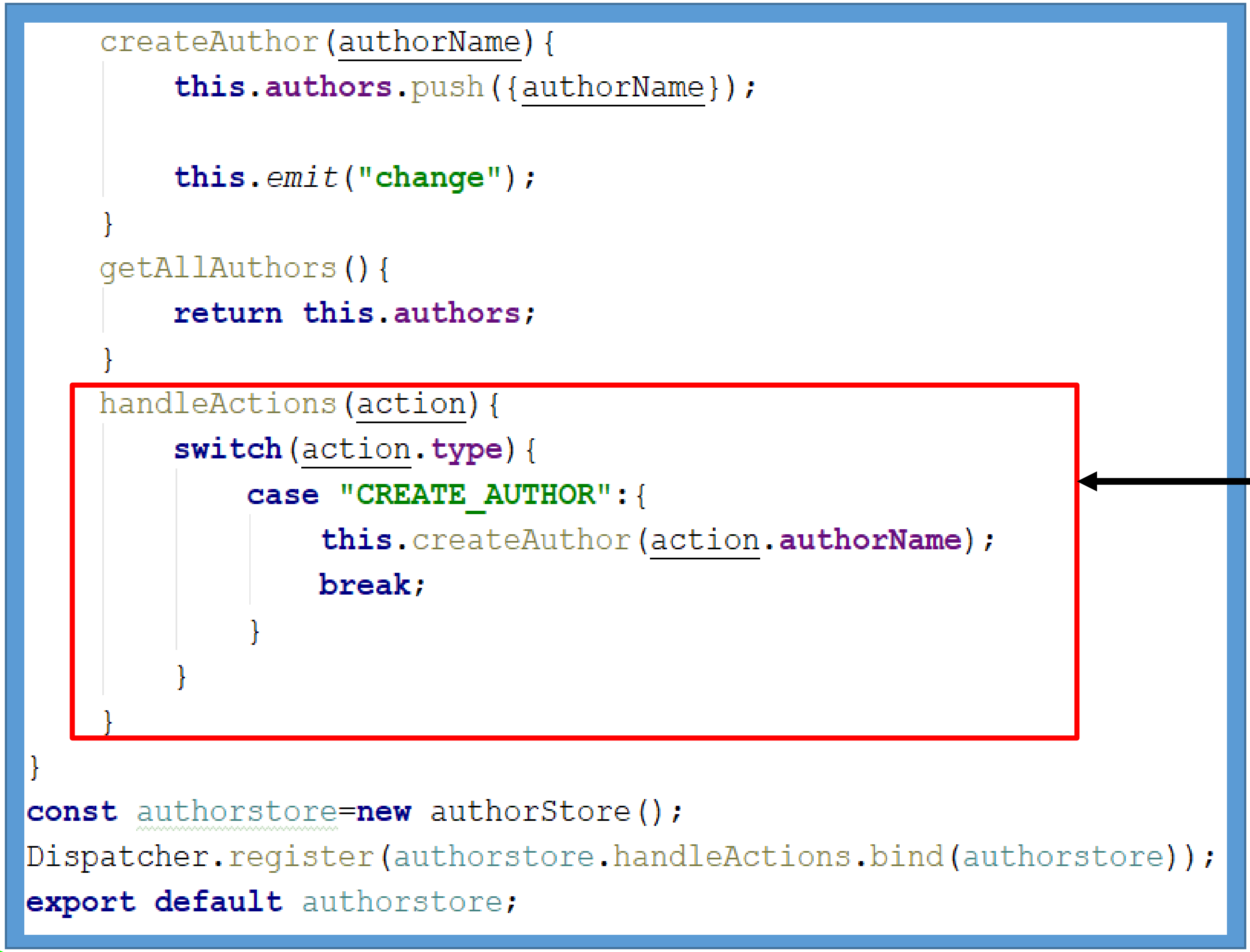


```
    createAuthor(authorName){
        this.authors.push({authorName});

        this.emit("change");
    }
    getAllAuthors(){
        return this.authors;
    }
    handleActions(action){
        switch(action.type){
            case "CREATE_AUTHOR":{
                this.createAuthor(action.authorName);
                break;
            }
        }
    }
}
const authorstore=new authorStore();
Dispatcher.register(authorstore.handleActions.bind(authorstore));
export default authorstore;
```

Store need to register to dispatcher in order to receive details about the actions

accenture

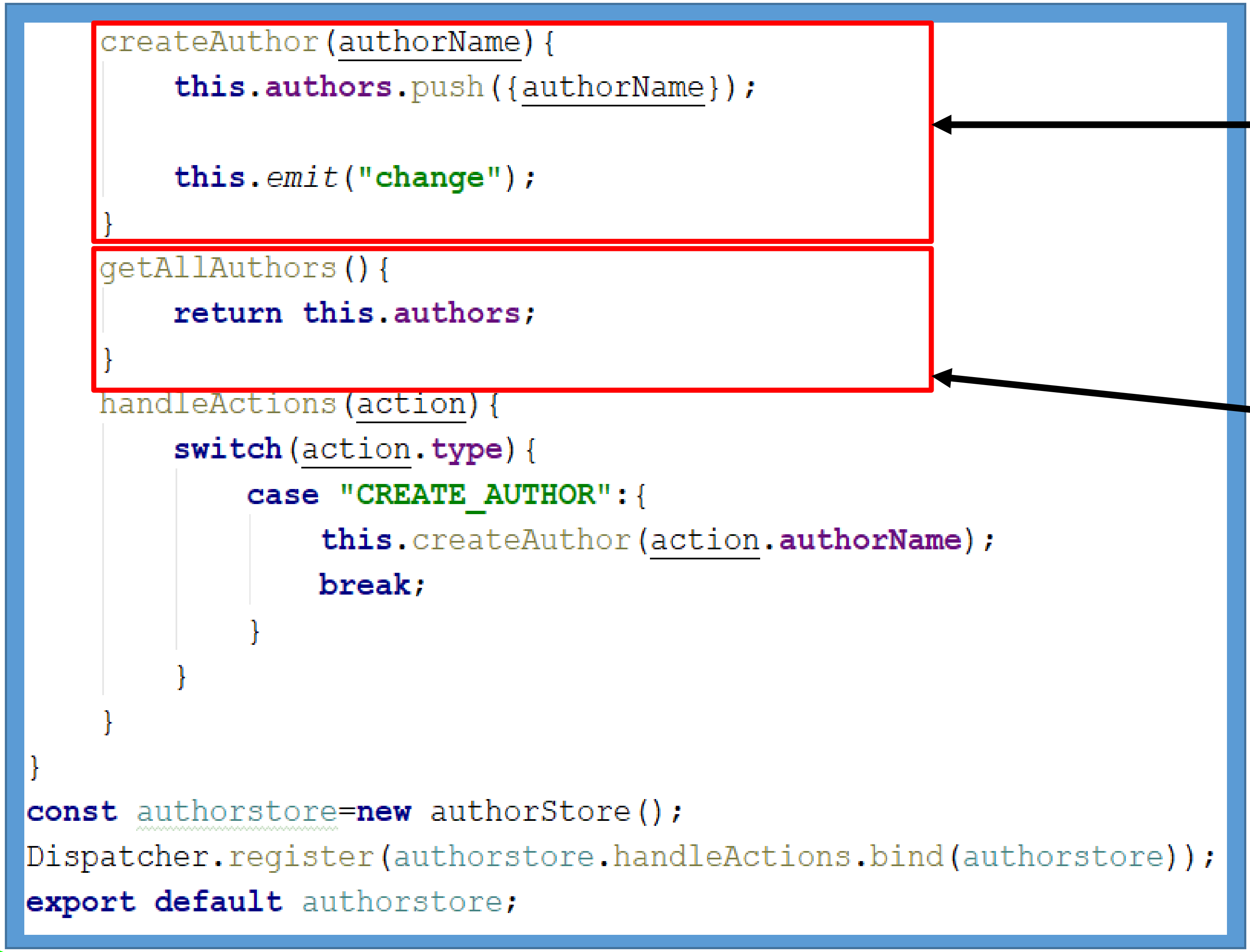
## Demo – Flux application Explanation(Cont…)



```
    createAuthor(authorName){
        this.authors.push({authorName});

        this.emit("change");
    }
    getAllAuthors(){
        return this.authors;
    }
    handleActions(action){
        switch(action.type){
            case "CREATE_AUTHOR":{
                this.createAuthor(action.authorName);
                break;
            }
        }
    }
}
const authorstore=new authorStore();
Dispatcher.register(authorstore.handleActions.bind(authorstore));
export default authorstore;
```

When the “CREATE_AUTHOR” action is handled, this will executes createAuthor method in the store

## Demo – Flux application Explanation(Cont…)



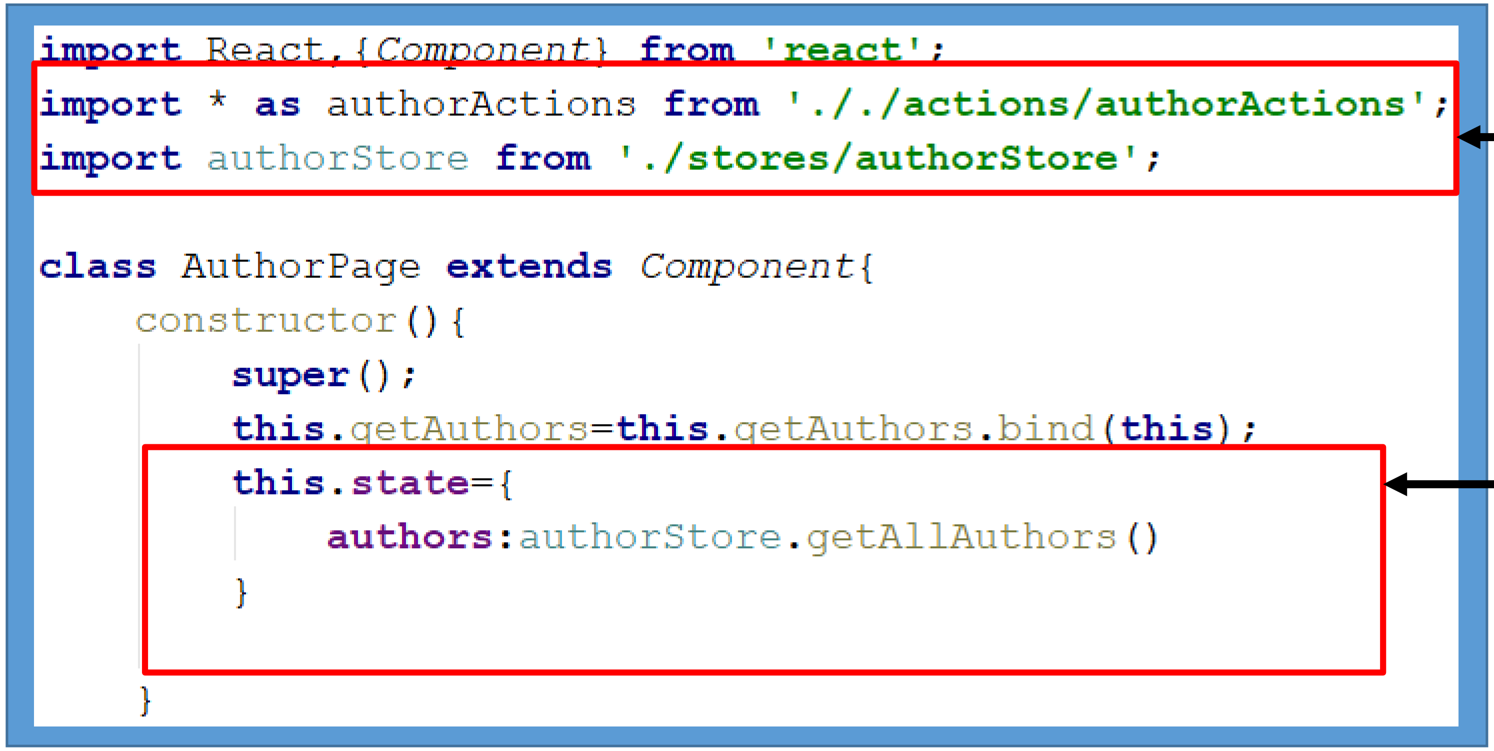
```
    createAuthor(authorName){
        this.authors.push({authorName});

        this.emit("change");
    }
    getAllAuthors(){
        return this.authors;
    }
    handleActions(action){
        switch(action.type){
            case "CREATE_AUTHOR":{
                this.createAuthor(action.authorName);
                break;
            }
        }
    }
}
const authorstore=new authorStore();
Dispatcher.register(authorstore.handleActions.bind(authorstore));
export default authorstore;
```

When the “CREATE_AUTHOR” action is triggered the dispatcher will notify the store. The store will insert new author using this method. Also emit the changes

This method will return all authors name

accenture

## Demo – Flux application Explanation(Cont…)



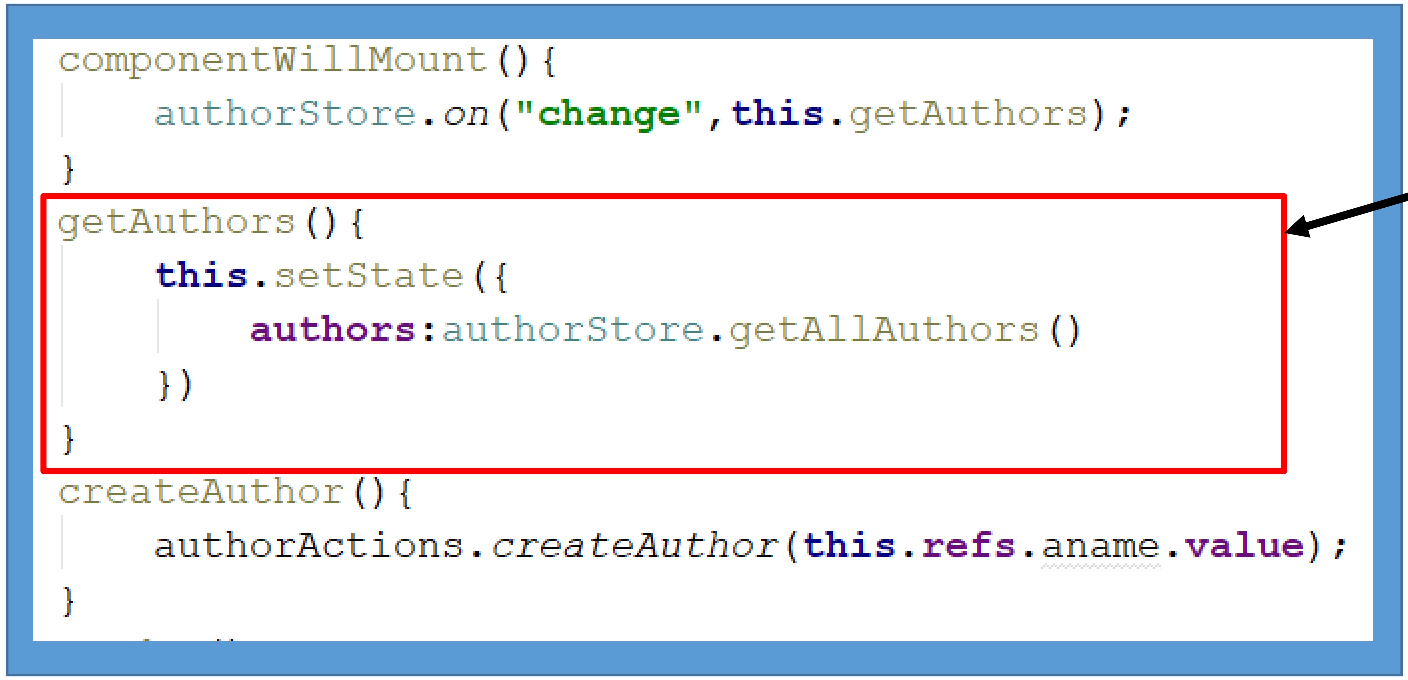
```
import React,{Component} from 'react';
import * as authorActions from './/actions/authorActions';
import authorStore from './stores/authorStore';

class AuthorPage extends Component{
    constructor(){
        super();
        this.getAuthors=this.getAuthors.bind(this);
        this.state={
            authors:authorStore.getAllAuthors()
        }

    }
```

Actions and stores are imported

The state of a component is defined inside the constructor which will get the data from the store

accenture

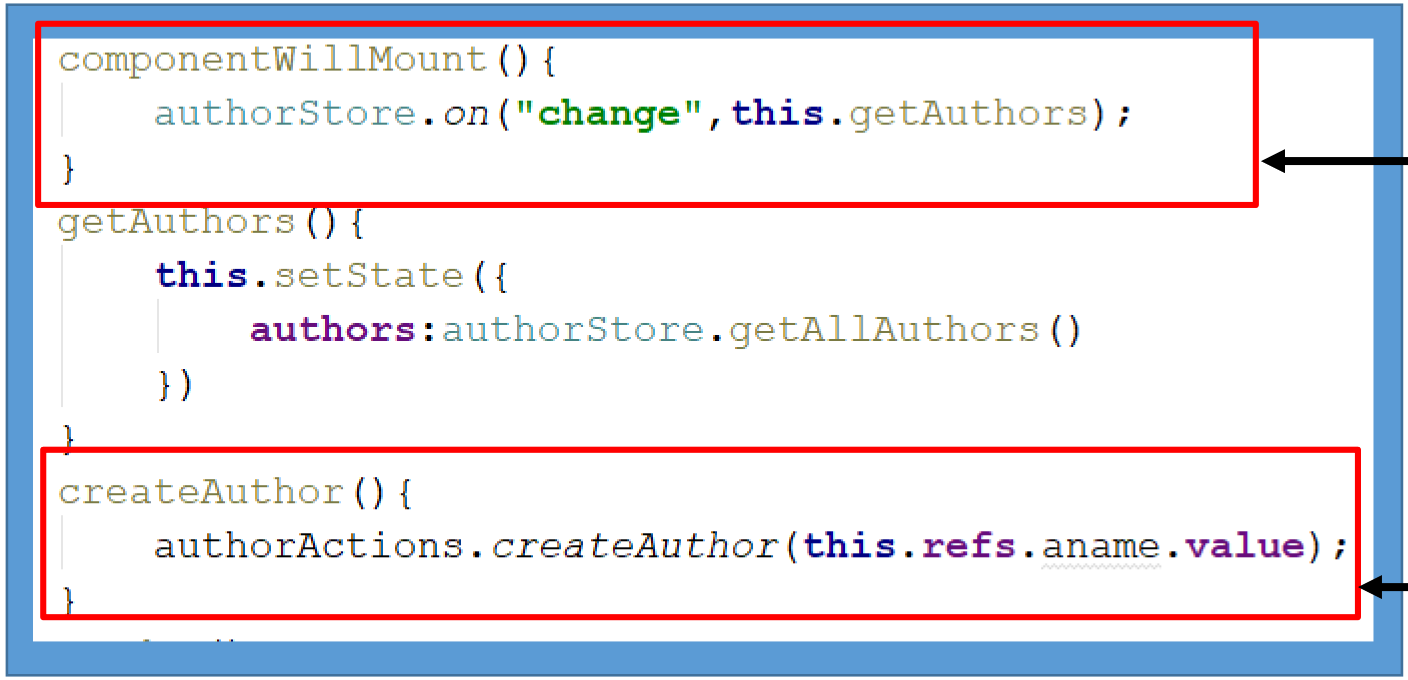
## Demo – Flux application Explanation(Cont…)



```
componentWillMount(){
    authorStore.on("change",this.getAuthors);
}
getAuthors(){
    this.setState({
        authors:authorStore.getAllAuthors()
    })
}
createAuthor(){
    authorActions.createAuthor(this.refs.aname.value);
}
```

This method will update state of a component, get all authors name from the store

# Demo – Flux application Explanation(Cont…)

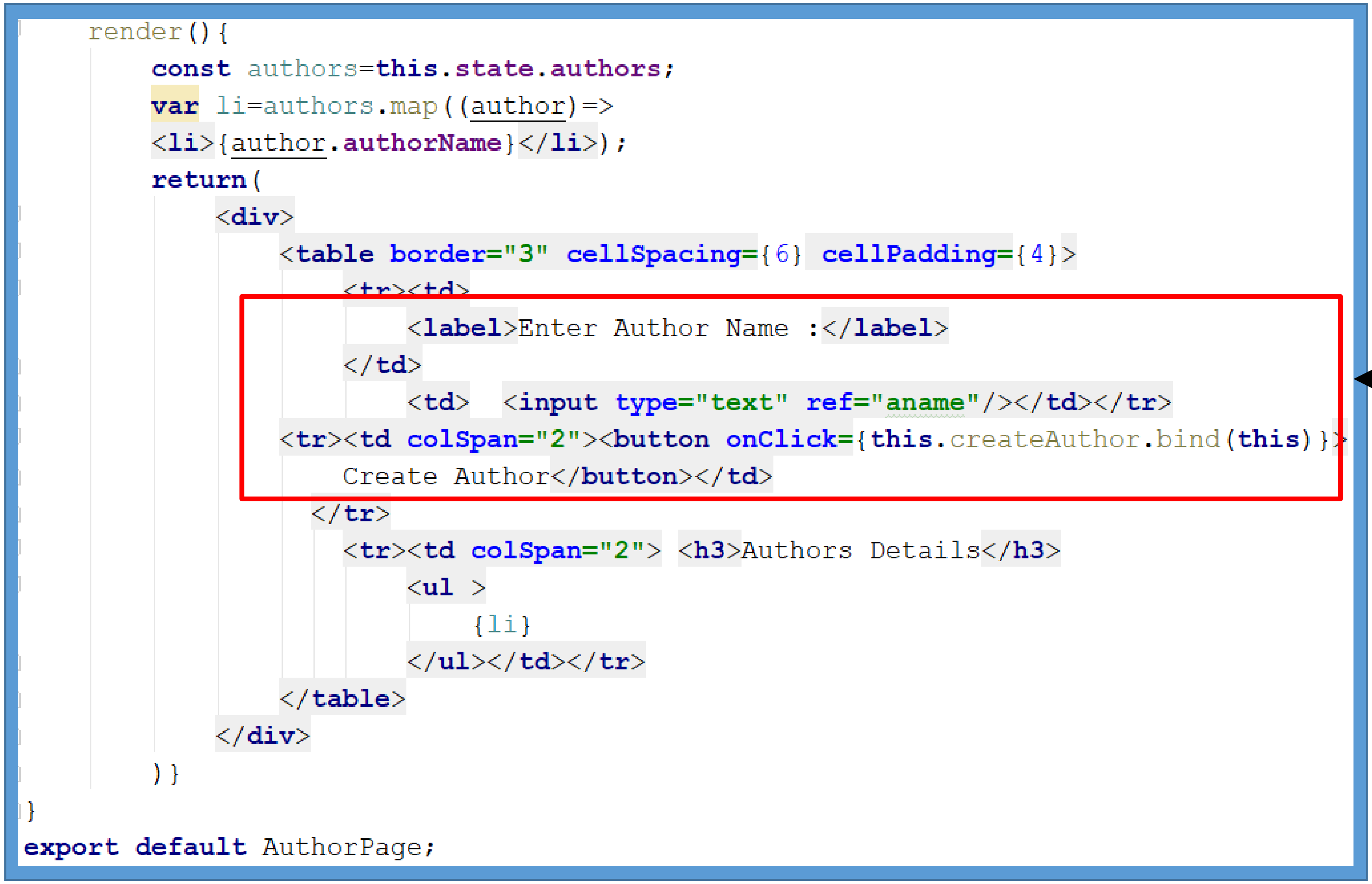


```
componentWillMount(){
    authorStore.on("change",this.getAuthors);
}
getAuthors(){
    this.setState({
        authors:authorStore.getAllAuthors()
    })
}
createAuthor(){
    authorActions.createAuthor(this.refs.aname.value);
}
```

Whenever there is any change is emitted from the store the react view will get notified

This method is invoked when the user clicks on create author button, it will also capture the text entered in the textbox, this will be sent to createAuthor action

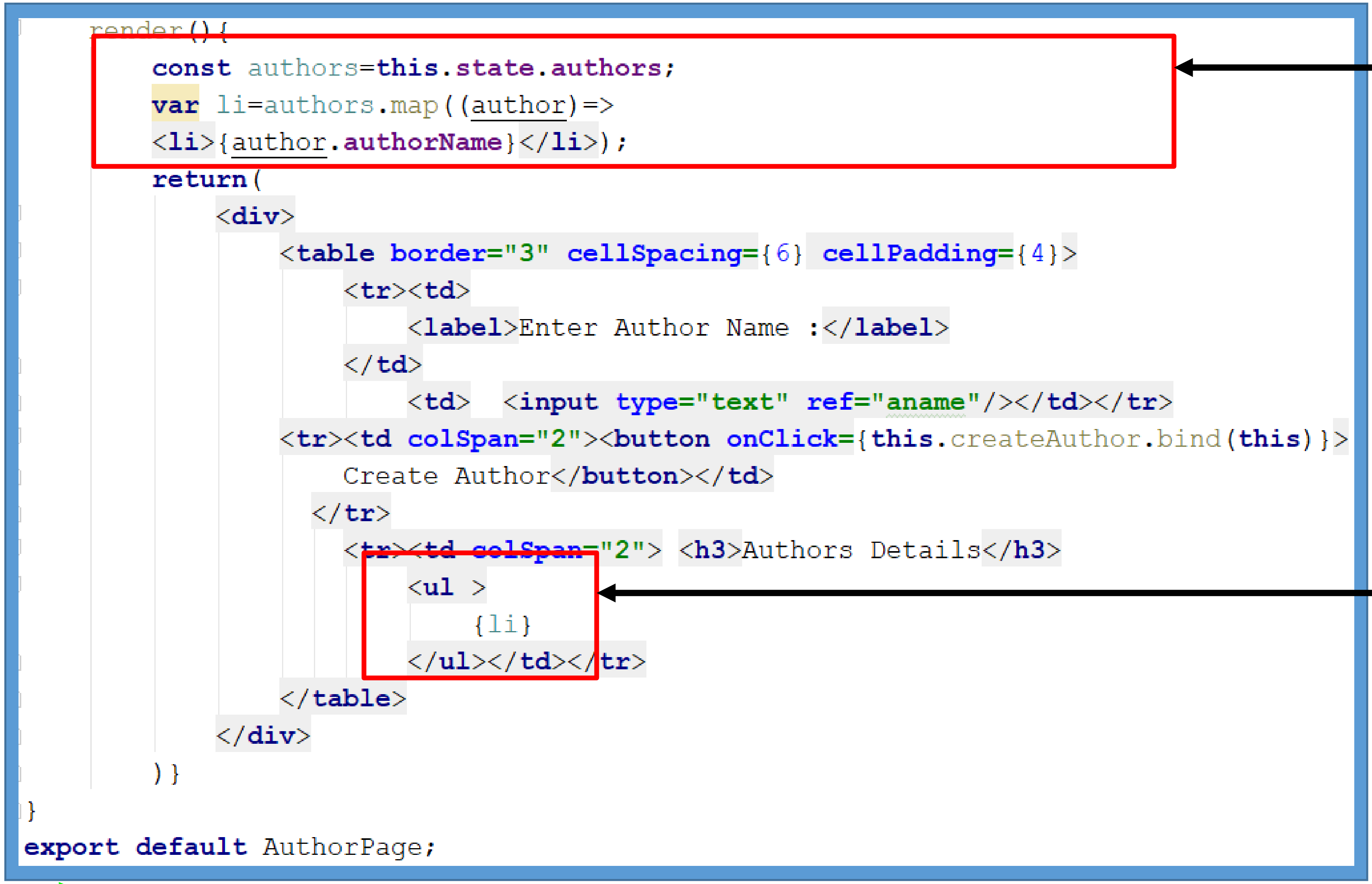
# Demo – Flux application Explanation(Cont…)



```
    render(){
        const authors=this.state.authors;
        var li=authors.map((author)=>
        <li>{author.authorName}</li>);
        return(
            <div>
                <table border="3" cellSpacing={6} cellPadding={4}>
                    <tr><td>
                        <label>Enter Author Name :</label>
                    </td>
                        <td>  <input type="text" ref="aname"/></td></tr>
                <tr><td colSpan="2"><button onClick={this.createAuthor.bind(this)}>
                    Create Author</button></td>
                  </tr>
                    <tr><td colSpan="2"> <h3>Authors Details</h3>
                        <ul >
                            {li}
                        </ul></td></tr>
                </table>
            </div>
        )}
}
export default AuthorPage;
```

When the user enters new author name in the textbox and clicks on create author button, the createAuthor method is called

# Demo – Flux application Explanation(Cont...)



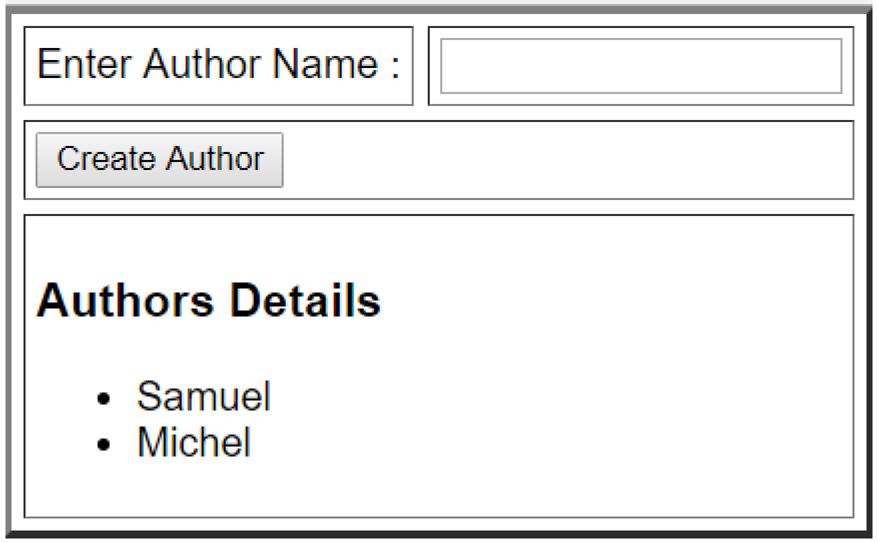
```
    render(){
        const authors=this.state.authors;
        var li=authors.map((author)=>
        <li>{author.authorName}</li>);
        return(
            <div>
                <table border="3" cellSpacing={6} cellPadding={4}>
                    <tr><td>
                        <label>Enter Author Name :</label>
                    </td>
                        <td>  <input type="text" ref="aname"/></td></tr>
                <tr><td colSpan="2"><button onClick={this.createAuthor.bind(this)}>
                    Create Author</button></td>
                  </tr>
                    <tr><td colSpan="2"> <h3>Authors Details</h3>
                        <ul >
                            {li}
                        </ul></td></tr>
                </table>
            </div>
        )}
}
export default AuthorPage;
```

Fetch all author names from the state of a component and assign it to the variable authors. Using map operator each author name is retrieved and assigned in a list format

All author names are displayed on the view

## Demo – Flux application Explanation(Cont...)

Output

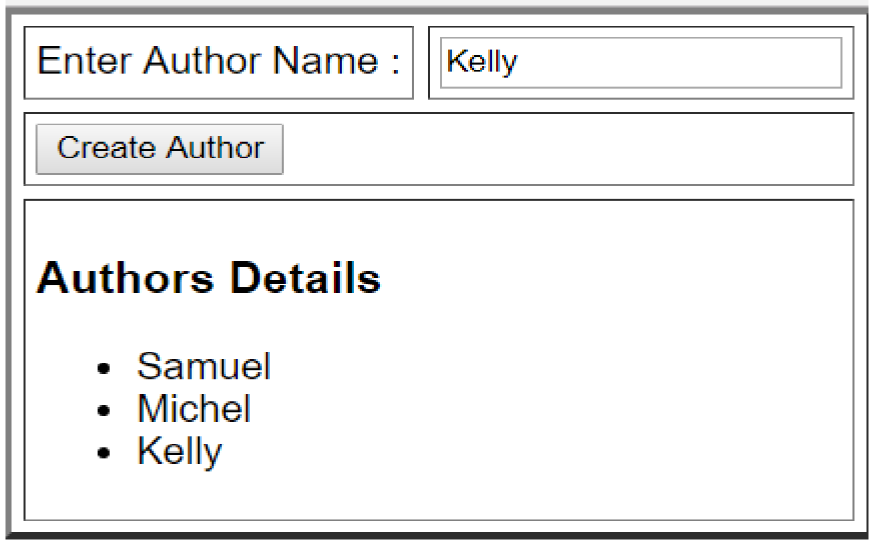

Enter Author Name :

Create Author

**Authors Details**

- Samuel
- Michel

Output after adding new author

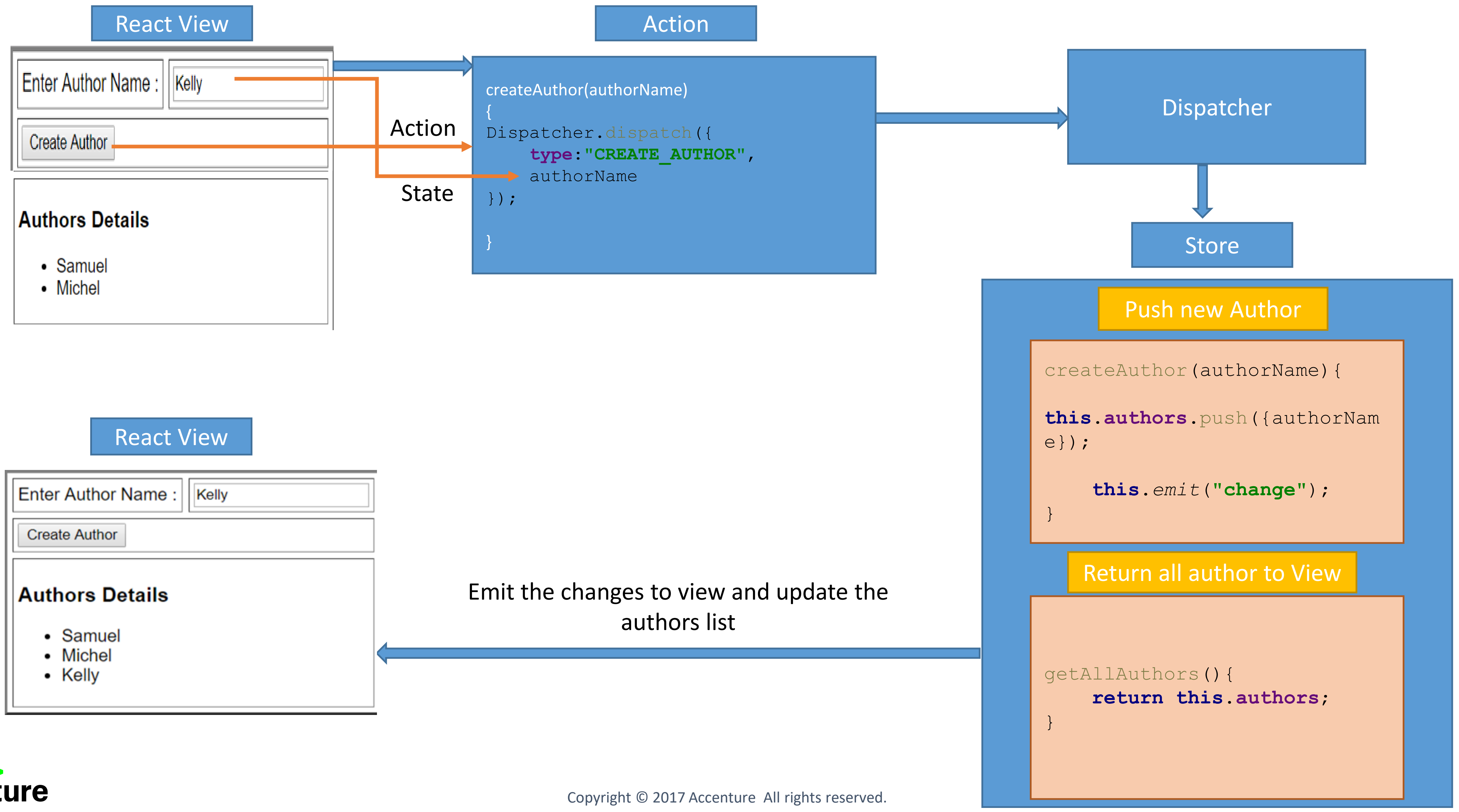

Enter Author Name : Kelly

Create Author

**Authors Details**

- Samuel
- Michel
- Kelly

# Complete Flux Example flow


React View
Action
Action
State
Dispatcher
Store
Push new Author
Return all author to View
React View
Emit the changes to view and update the authors list

Enter Author Name : Kelly

Create Author

**Authors Details**

- Samuel
- Michel

```
createAuthor(authorName)
{
Dispatcher.dispatch({
    type:"CREATE_AUTHOR",
    authorName
});

}
```

```
createAuthor(authorName){

this.authors.push({authorName});

    this.emit("change");
}
```

```
getAllAuthors(){
    return this.authors;
}
```

Enter Author Name : Kelly

Create Author

**Authors Details**

- Samuel
- Michel
- Kelly

## Activity

1. Use existing application “fluxexample” which is created using react with flux pattern

2. Create a view which will have another button “Delete Author”

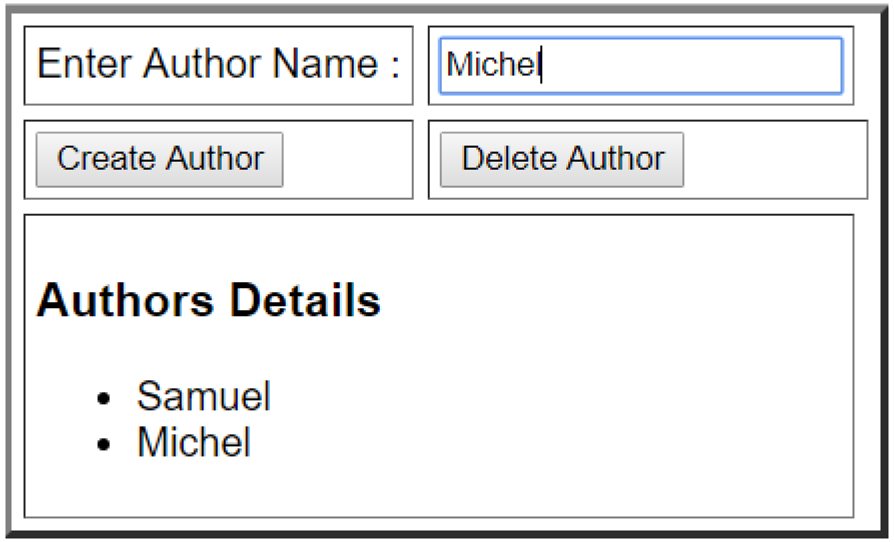
3. When the user enters author name in the textbox and clicks on delete author button, the author should be deleted

4. The view should display updated list of authors

## Activity – Expected Output

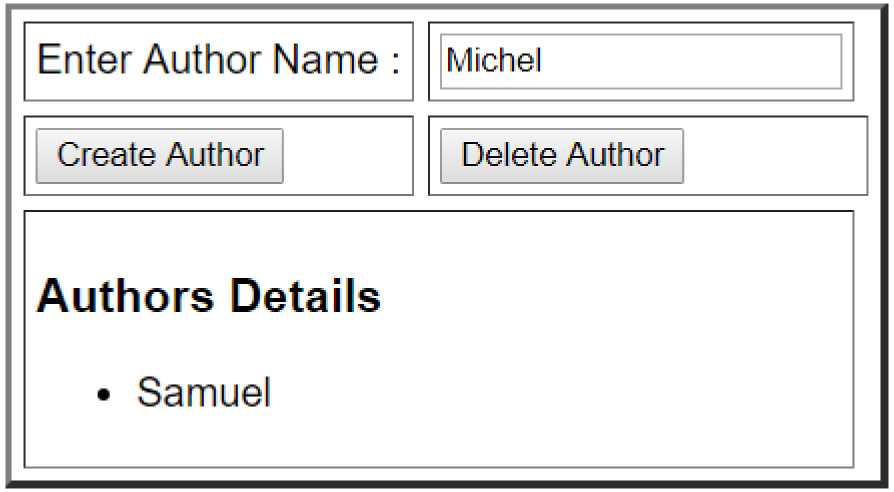
Output

Output after clicking on delete author button



**Authors Details**

- Samuel
- Michel



**Authors Details**

- Samuel

## MODULE SUMMARY

- Understand why flux is important
- Understand how flux works
- Understand
  - Action
  - Dispatcher
  - Store

accenture

## THANK YOU

accenture