Introduction

`rivarjs` is a decentralized state management library that automates changes. It harmonizes concepts from the object-oriented programming (OOP) and functional reactive programming (FRP) paradigms. At its core, `rivarjs` introduces a datatype called `RIVar`, which stands for *Reactive Instance Variable*.

The term *Reactive* signifies that *assignments* initiate dependencies, ensuring continuous updates in response to changes. The term *Instance* implies the facilitation of *information hiding*, as assignments can be performed *externally*. Subclasses assign variables of their parents, while composites assign variables of the objects they contain.

How It Works

Each variable in `rivarjs` is implemented as an *observable stream* from RxJS. Similarly, the assigned expressions for these variables are also implemented as observable streams.

To create the variable stream, the streams from the assigned expressions are merged together. This merging operation combines the individual streams into a single stream representing the variable. As a result, the variable stream will update and emit new values whenever any of the assigned expression streams produce a new value.

This design choice enables independent *assignments*, initiating dependencies for continuous updates, for shared variables within multiple contexts.

The API

The API consists of *lifting*.

1. Variables

```
var myRIVar=new RIVar();
```

2. Functions

```
var functionOverRIVars=lift((x, y) => x * y, firstRIVar, secondRIVar);
```

3. Assignments

```
myRIVar.set(functionOverRIVars);
```

It is usually readable to compose this with the previous step:
`myRIVar.set(lift((x, y) => x * y, firstRIVar, secondRIVar))`

Information Hiding

Classes can be created as a unit of information, containing *private* variables and assignments, along with *public* variables. The public visibility of variables does not compromise the ability to withstand or resist changes. The reason is that the assignments do not override previous assignments.

```
class A { // the unit of information
  constructor() {
    this.firstRIVar = new RIVar();
    // you may assign this.firstRIVar
  }
}
```

Inheritance

```
class B {
  constructor(a) {

    this.a = a;

    this.secondRIVar = new RIVar();
    this.thirdRIVar = new RIVar();

    // this resists changes!
    this.a.firstRIVar.set(lift(mul, this.secondRIVar, this.thirdRIVar));

  }
}
```

Composition

```
class B extends A {

  constructor(a) {

    this.secondRIVar = new RIVar();
    this.thirdRIVar = new RIVar();

    // this resists changes!
    this.FirstRIVar.set(lift(mul, this.secondRIVar, this.thirdRIVar));

  }
}
```

Binding

Binding in React

`React components` are connected to instances of `RIVar` by passing them as a `prop` named `rivar`.

```
<MyRIVarComponent rivar={myRIVar}/>
```

`rivarjs` provides a generic `React Component` named `RIVarComponent`.

To create your component, start by importing `RIVarComponent` from 'rivarjs/binders/react':

```
import { RIVarComponent } from 'rivarjs/binders/react';
```

To implement your custom component, you can extend the `RIVarComponent` class and override the `render()` function based on your specific requirements. Within the `render()` function, you can determine the content to be rendered based on the `this.state.value`, which corresponds to the current value of the `rivar` (from the `prop`). If a user change is detected in the value, you can call the `this.change()` method.

```
class MyRIVarComponent extends RIVarComponent {
  render() {
    return (
        <input
          value={this.state.value}
          onChange={event=>this.change(event.target.value) }  />
    );
  }
}
```

Binding with pure JavaScript

The following code initiates a connection between an instance of `RIVar` to an `HTML` element.

```
function bind(inputID, variable) {

  var input = document.getElementById(inputID);

  input.addEventListener('input', (event) => {
    const value = event.target.value;
    variable.next(new Signal(value));
    input.style.fontStyle = "normal";
  });

  variable.subscribe((signal) => {
    if (input.value !== signal.value.toString()) {
      input.value = signal.value.toString();
      input.style.fontStyle = "italic";
    }
  });

}
```

Examples

Example React

Example pure JavaScript

Installation

To use `rivarjs`, you have two options. First, you can install it using `npm` by running the following command:

```
npm install rivarjs
```

Alternatively, for an `HTML` page, you need to include the `rivarjs` script and its required dependency, `RxJS`, by adding the following script tags:

```
<script src="https://unpkg.com/rxjs@^7/dist/bundles/rxjs.umd.min.js"></script>
<script src="https://unpkg.com/rivarjs/dist/rivar.umd.js"></script>
```

Once you have `rivarjs` available, you can import the necessary elements in your `JavaScript` code using the following syntax:

```
var { RIVar, lift, Signal } = rivarjs;
```

By following these steps, you will be able to utilize the functionalities provided by the `rivarjs`.