Aprameyah
Technologies Pvt. Ltd.

Functional Reactive Programming

With Bacon.js

Functional Reactive Programming

- Functional reactive programming (FRP) is a programming paradigm for reactive programming (asynchronous dataflow programming) using the building blocks of functional programming (e.g. map, reduce, filter)

Building Blocks

- Map
  - Transform operations that change an object
- Filter
  - Decision operation that returns boolean to determine if the object needs to be in the output collection
- Reduce
  - Aggregation operation that works on a stream of data and produces a single property

Where is this useful

- When there are too many events from diverse sources manipulating too many state fields

- We bundle events into different pipes. And attach functional programs to these pipes to perform actions.

- Events evolve with map, filter and reduce

Event Stream Concept

- First concept of FRP is assemble happenings in the system as a stream of events
    - Events can come from a promise (A single event and end of stream): Bacon.fromPromise(promise)
    - From Node.js event emitters: Bacon.fromEventTarget(eventEmitter,eventName)
    - Single event from a function that takes a callback: Bacon.fromNodeCallback(f)
    - Bacon.fromPoll(interval, f): f should return Bacon.next or Bacon.end. f is called in intervals

On the Browser

- Events can also come from jquery bound controls

```
$("input[name='loginname']").asEventStream("keyup")
```

- Or from an array of data

```
return Bacon.fromArray(data.data);
```

- Once we create a stream, we can listen to the stream using onValue function

```
myStream.onValue(function(streamData) {
    console.log(streamData);
});
```

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Create an input box, listen to keyup events and log them to console

Aprameyah
Technologies Pvt. Ltd.

Transforming Events

- Map functions are used to transform events on streams

- stream.map(function(data){ return transformedData})

Lets Try It

- Transform the event objects coming out of the keyup stream to the text value contained in the event and see how the listening on the stream goes

- In the value listener, update a div next to the user input box to indicate if the username is valid or not (yes/not)

  - Transform the text value to Yes/No values depending on if the text is greater than 8 chars or not

Properties

- Properties are very similar to streams except that they have a current value & initial value.

- Properties are result of “reduce” operations on some stream

- property.sample(interval) - get current value at certain intervals

- property.sampledBy(stream) - get current value of property every time there is an event on the stream

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Convert the user stream to a property by calling - toProperty(“No”)

- This should provide an initial value to the status display

Combining Streams With Properties

- A value from a stream can be combined with a value from a property to arrive at a new stream

- stream.combine(property, function(a,b){ return a+b;})

- Create another text field for email address input and create a yes/no property for it after validating it has the @ character in it.

    - Combine this property with the username yes/no property to arrive at a joint validity decision property

Aprameyah
Technologies Pvt. Ltd.

Streams from Ajax

- JQuery ajax calls return deferred objects

```
$.ajax({ url : "isUserValid.txt"}).done(function(result)
{ console.log(result)});
```

- This deferred object can be part of a stream

- Lets check if the user is valid on every keyup event. To the earlier created username property, create a map function that returns ajax responses.

    - Base the “validity of field” decision on the Ajax response being true or false

Aprameyah
Technologies Pvt. Ltd.

FlatMaps

- flatMap is a function that creates a stream for every event in the incoming stream

- stream1.flatMap(function(event){ return subStream});

- Output is a single stream that has all elements of the sub-streams flattened

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Load the players from players.json provided in the server.
    - Convert an ajax request into a stream
      Bacon.fromPromise($.ajax({ url : “players.json"}));
    - Call flatMap on resultant stream to make the data a stream Bacon.fromArray(data.data);
    - On the stream value handler, append the user to a div

Filter Operation

- Streams can be filtered to eliminate the events not needed
- stream.filter(function(eventData){ return boolean})
- Modify the previous user listing to eliminate retired players from the display.

Aprameyah
Technologies Pvt. Ltd.

Reduce Operation

- Scan function is used to perform reduce operation based on previous stream values:
  scan(initialValue,function(prev,current){return accumulate(prev and current);});

- Display a count of players by doing a scan operation on the filtered stream

Manipulate Streams - Map, Filter, Reduce

- Methods on streams: .onValue, .onError, .onEnd,
- .map(function(value){}) - Converts events in this stream using the provided function
- .map(property) - Puts property into stream for every event on stream
- .filter(property)