Functional and Reactive programming using Facebook's React and Appian's SAIL

January 17th, 2017

Carlos Aguayo
Director, Software Development
Hello!
I am Carlos Aguayo

Appian
- 12 years
- Director, Software Development
- Front End Developer
- SAIL patent

Georgia Tech
- Masters student
- Computer Science
- Machine Learning specialization

linkedin.com/in/carlosaguayo
@carlosaguayo81
What is Appian
Story

- Think about how complex building a web application can be.
Story

- Think about how complex building a web application can be.
  - Then make it robust
Story

- Think about how complex building a web application can be.
  - Then make it robust
  - And scalable
Think about how complex building a web application can be.
○ Then make it robust
○ And scalable
○ And fast
Think about how complex building a web application can be.
  ○ Then make it robust
  ○ And scalable
  ○ And fast
  ○ Multiple servers
Think about how complex building a web application can be.

- Then make it robust
- And scalable
- And fast
- Multiple servers
- It needs to be secure
Story

- Think about how complex building a web application can be.
  - Then make it robust
  - And scalable
  - And fast
  - Multiple servers
  - It needs to be secure
  - It needs to be usable
• Think about how complex building a web application can be.
  ○ Then make it robust
  ○ And scalable
  ○ And fast
  ○ Multiple servers
  ○ It needs to be secure
  ○ It needs to be usable
  ○ It needs to be beautiful
Think about how complex building a web application can be.
- Then make it robust
- And scalable
- And fast
- Multiple servers
- It needs to be secure
- It needs to be usable
- It needs to be beautiful
- iPhone, iPad, Android
Think about how complex building a web application can be.
○ Then make it robust
○ And scalable
○ And fast
○ Multiple servers
○ It needs to be secure
○ It needs to be usable
○ It needs to be beautiful
○ iPhone, iPad, Android
○ It needs to be easy to code
Story

- Think about how complex building a web application can be.
  - Then make it robust
  - And scalable
  - And fast
  - Multiple servers
  - It needs to be secure
  - It needs to be usable
  - It needs to be beautiful
  - iPhone, iPad, Android
  - It needs to be easy to code
  - We need to be able to change it quickly
Story

- Think about how complex building a web application can be.
  - Then make it robust
  - And scalable
  - And fast
  - Multiple servers
  - It needs to be secure
  - It needs to be usable
  - It needs to be beautiful
  - iPhone, iPad, Android
  - It needs to be easy to code
  - We need to be able to change it quickly
  - It can't be expensive
Story

- Think about how complex building a web application can be.
  - Then make it robust
  - And scalable
  - And fast
  - Multiple servers
  - It needs to be secure
  - It needs to be usable
  - It needs to be beautiful
  - iPhone, iPad, Android
  - It needs to be easy to code
  - We need to be able to change it quickly
  - It can't be expensive
  - **And we want it now!**
VENTURE CAPITAL DISPATCH

Investor Says Appian Has Crossed $1 Billion Valuation

By DEBORAH GAGE
Jan 8, 2016 2:34 pm ET

Custom software maker Appian Corp. said its valuation is north of $1 billion, a milestone it has reached after 17 years on just $47.5 million in venture capital.
Overview

At this lecture, you will:

1. Learn about programming paradigms for building modern UIs
   1.1. Reactive Programming
   1.2. Functional Programming

2. See an example of how to apply these techniques
React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

https://facebook.github.io/react/
Facebook's React

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

https://facebook.github.io/react/
Hello World in React

```jsx
<HelloWorld />

<div>
  <h1>Hello World!</h1>
</div>
```

[https://facebook.github.io/react/](https://facebook.github.io/react/)
Hello World in React

```html
<HelloWorld name="MIT"/>

<div>
  <h1>Hello {this.props.name}!</h1>
</div>
```

https://facebook.github.io/react/
Hello World in React

```html
HelloWorld name="MIT"/>
```

```html
<div>
  <h1>Hello {this.props.name}!</h1>
</div>
```

https://facebook.github.io/react/
Hello World in React

```javascript
class HelloWorld extends React.Component {
  render() {
    return(
      <div>
        <h1>Hello {this.props.name}!</h1>
      </div>
    )
  }
}

ReactDOM.render(
  <HelloWorld name="MIT"/>

  document.getElementById("container")
)
```java
class HelloWorld extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello {this.props.name}!</h1>
      </div>
    );
  }
}

ReactDOM.render(
  <HelloWorld name="MIT" />,
  document.getElementById("container")
)
```

https://facebook.github.io/react/
A component with state

Hello MIT!

https://facebook.github.io/react/
A component with state

State
- name: MIT

Hello MIT!
A component with state

Change Name: ___MIT___

State
- name: MIT

Hello MIT!
A component with state

Change Name: ___Carlos___

State
- name: MIT

Hello MIT!

https://facebook.github.io/react/
A component with state

- **State**
  - name: Carlos

  **Change Name:** __Carlos__

  **Hello MIT !**
A component with state

Change Name: ___Carlos___

State
- name: Carlos

Hello Carlos!
Why is this important?

https://facebook.github.io/react/
Think of a relatively complex web application
From this...
From this ...

https://facebook.github.io/react/
To this ...

https://facebook.github.io/react/
To this ...

https://facebook.github.io/react/
Facebook's React

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

https://facebook.github.io/react/
A simple app, Tic-Tac-Toe

![Tic-Tac-Toe board]

**Winner: X**

1. **Game start**
2. **Move #1**
3. **Move #2**
4. **Move #3**
5. **Move #4**
6. **Move #5**

A simple app, Tic-Tac-Toe

First component: a Square

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

First component: a Square

Needs to:

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

First component: a Square

Needs to:
1. Be able to display an 'X' or an 'O'

https://facebook.github.io/react/tutorial/tutorial.html

Winner: X

1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

First component: a Square

Needs to:
1. Be able to display an 'X' or an 'O'
2. Report whenever it gets clicked

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

First component: a Square

Needs to:
1. Be able to display an 'X' or an 'O'
2. Report whenever it gets clicked

```javascript
class Square extends React.Component {
    render() {
        return (
            <button className="square" onClick={() => this.props.onClick()}
                {this.props.value}
            </button>
        );
    }
}
```
A simple app, Tic-Tac-Toe

First component: a Square

Needs to:
1. Be able to display an 'X' or an 'O'
2. Report whenever it gets clicked

```javascript
class Square extends React.Component {
  render() {
    return (
      <button className="square" onClick={() => this.props.onClick()}>
        {this.props.value}
      </button>
    );
  }
}
```
A simple app, Tic-Tac-Toe

Second component: a Board

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Second component: a Board

Needs to:

![Tic-Tac-Toe Board]

Winner: X

1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Second component: a Board

Needs to:
1. Render 9 Squares

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Second component: a Board

Needs to:
1. Render 9 Squares
2. Each square with a given value

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Second component: a Board

Needs to:
1. Render 9 Squares
2. Each square with a given value
3. Report whenever one of them gets clicked with its position

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple application: a tic-tac-toe board.

Second component: a `Board`

Needs to:
1. Render 9 squares
2. Each square with a given value
3. Report whenever one of them gets clicked with its position

```javascript
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={this.props.squares[i]}
      onClick={() => this.props.onClick(i)} />
  }
  render() {
    return (<
      <div className="board-row">
        {this.renderSquare(0)}
        {this.renderSquare(1)}
        {this.renderSquare(2)}
      </div>
      <div className="board-row">
        {this.renderSquare(3)}
        {this.renderSquare(4)}
        {this.renderSquare(5)}
      </div>
      <div className="board-row">
        {this.renderSquare(6)}
        {this.renderSquare(7)}
        {this.renderSquare(8)}
      </div>
      </div>
    );
  }
}
```

Winner: X

1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app

Second component: a Board

Needs to:

1. Render 9 squares
2. Each square with a given value
3. Report whenever one of them gets clicked with its position

```javascript
class Board extends React.Component {
  renderSquare(i) {
    return <Square
      value={this.props.squares[i]}
      onClick={() => this.props.onClick(i)} />
  }

  render() {
    return (<
      div className="board-row">
      {this.renderSquare(0)}
      {this.renderSquare(1)}
      {this.renderSquare(2)}
    </div>

    <div className="board-row">
      {this.renderSquare(3)}
      {this.renderSquare(4)}
      {this.renderSquare(5)}
    </div>

    <div className="board-row">
      {this.renderSquare(6)}
      {this.renderSquare(7)}
      {this.renderSquare(8)}
    </div>

    </div>
  }
}
```

Winner: X

1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple application.

Second component: a Board

Needs to:
1. Render 9 squares
2. Each square with a given value
3. Report whenever one of them gets clicked with its position
A simple app, Tic-Tac-Toe

Third component: a Game!

Winner: X

1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number
3. Keep track of who is next

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number
3. Keep track of who is next
4. Keep track of moves

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number
3. Keep track of who is next
4. Keep track of moves
5. Tell if someone has won

Winner: X

1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number
3. Keep track of who is next
4. Keep track of moves
5. Tell if someone has won

```jsx
return (  
    <div className="game">  
        <div>  
            <Board squares={current.squares} onClick={i => this.handleClick(i)} />  
        </div>  
        <div className="game-info">  
            <div>{status}</div>  
            <ol>{moves}</ol>  
        </div>  
    </div>  
);  
```

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number
3. Keep track of who is next
4. Keep track of moves
5. Tell if someone has won

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number
3. Keep track of who is next
4. Keep track of moves
5. Tell if someone has won

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)
A simple app, Tic-Tac-Toe

Third component: a Game!

Needs to:
1. Render a Board
2. Keep track of step number
3. Keep track of who is next
4. Keep track of moves
5. Tell if someone has won

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Represent each state as a list of 'X' and 'O'

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Represent each state as a list of 'X' and 'O'

'X', 'O', null,
'O', 'X', null,
null, null, 'X'

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Every time the user clicks, we are adding an entry into our history.

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
Next player: X

1. **Game start**

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

```javascript
history = [
    [null, null, null, null, null, null, null, null, null, null]
]
```

Winner: X

1. **Game start**
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Next player: O
1. Game start
2. Move #1

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

Winner: X
1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

history = [
[null, null, null, null, null, null, null, null, null, null],
[null, null, null, null, null, null, null, null, null, 'X']
]
A simple app, Tic-Tac-Toe

Next player: X

1. Game start
2. Move #1
3. Move #2

history = [
    [null, null, null, null, null, null, null, null, null],
    [null, null, null, null, null, null, null, null, 'X'],
    [null, null, null, 'O', null, null, null, null, 'X']
]

State
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

Winner: X

1. Game start
2. Move #1
3. Move #2
4. Move #3
5. Move #4
6. Move #5

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Next player: O

1. **Game start**
2. **Move #1**
3. **Move #2**
4. **Move #3**

**State**
- What step am I in? (stepNumber)
- Who's next? (xIsNext)
- Move history (history)

```
history = [
    [null, null, null, null, null, null, null, null, null, null],
    [null, null, null, null, null, null, null, null, 'X'],
    [null, null, null, 'O', null, null, null, null, 'X'],
    [null, null, null, 'O', 'X', null, null, null, 'X']
]
```

Winner: X

https://facebook.github.io/react/tutorial/tutorial.html
Let's play Tic-Tac-Toe!
A simple app, Tic-Tac-Toe

Let's play Tic-Tac-Toe!

https://facebook.github.io/react/tutorial/tutorial.html
A simple app, Tic-Tac-Toe

Let's play Tic-Tac-Toe!

React.JS is just a view layer.

Everything else is just details and better control over components:

1. How to mount and dismount
2. JavaScript ES6
3. etc.

https://facebook.github.io/react/tutorial/tutorial.html
So why should I care?
So why should I care?

React:

● Handles plumbing for you
  ○ Does all the data binding
● It's fast and extremely efficient!
● It knows exactly what to render and how to render.
● Simple is beautiful!

https://facebook.github.io/react/tutorial/tutorial.html
Thinking/Winning in React

https://facebook.github.io/react/tutorial/tutorial.html
Thinking/Winning in React

1. Start With A Mock
Thinking/Winning in React

1. Start With A Mock
2. Break The UI Into A Component Hierarchy

https://facebook.github.io/react/tutorial/tutorial.html
Thinking/Winning in React

1. Start With A Mock
2. Break The UI Into A Component Hierarchy
3. Build A Static Version

Thinking/Winning in React

1. Start With A Mock
2. Break The UI Into A Component Hierarchy
3. Build A Static Version
4. Identify the state

Thinking/Winning in React

1. Start With A Mock
2. Break The UI Into A Component Hierarchy
3. Build A Static Version
4. Identify the state
5. Identify where it should live

Thinking/Winning in React

1. Start With A Mock
2. Break The UI Into A Component Hierarchy
3. Build A Static Version
4. Identify the state
5. Identify where it should live
6. Win!

```javascript
class ProductCategoryRow extends React.Component {
  render() {
    return <tr><th colSpan="2">{this.props.category}</th></tr>;
  }
}

class ProductRow extends React.Component {
  render() {
    var name = this.props.product.stocked ?
      this.props.product.name :
      <span style={{color: 'red'}}>
        {this.props.product.name}
      </span>;
    return (
      <tr>
        <td>{name}</td>
        <td>{this.props.product.price}</td>
      </tr>
    );
  }
}

class ProductTable extends React.Component {
  render() {
    var rows = [];
    var lastCategory = null;
    this.props.products.forEach(function(product) {
      if (product.category !== lastCategory) {
        rows.push(<ProductCategoryRow category={product.category} key={product.category} />);
      }
    };
  }
```
What's Reactive Programming?
var a = 1;
var b = 2;

var c = a + b;

// c equals to 3

b = 3;

// is c equals now to 4?
What's Reactive Programming?

```javascript
var a = 1;
var b = 2;

var c = a + b;

// c equals to 3

b = 3;

// is c equals now to 4?
```

Imperative Programming
What's Reactive Programming?

Reactive Programming Simple Example
What's Reactive Programming?

Change Name: __MIT__

State
- name: MIT

Hello MIT!
Reactive programming is a programming paradigm oriented around data flows and the propagation of change.

This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow.
What's Reactive Programming?

Programming with asynchronous data streams
What's Reactive Programming?

Programming with asynchronous data streams

You don't really control when and where for data flows.
What's Reactive Programming?

Programming with asynchronous data streams

You don't really control when and where for data flows.
Why Reactive Programming?

The benefit is more evident in modern webapps and mobile apps that are highly interactive with a multitude of UI events related to data events. 10 years ago, interaction with web pages was basically about submitting a long form to the backend and performing simple rendering to the frontend. Apps have evolved to be more real-time: modifying a single form field can automatically trigger a save to the backend, "likes" to some content can be reflected in real time to other connected users, and so forth.

Apps nowadays have an abundancy of real-time events of every kind that enable a highly interactive experience to the user. We need tools for properly dealing with that, and Reactive Programming is an answer.

https://dzone.com/articles/only-introduction-reactive
Functional Reactive Programming at Appian
Functional Reactive Programming at Appian
Appian SAIL: Functional Reactive Programming

load(
    first: "John",
    last: "Smith",
    with(
        display: lower(concat(first, ".", last)),
        {
            a!textField(label: "First", value: first, saveInto: first),
            a!textField(label: "Last", value: last, saveInto: last),
            a!textField(label: "Display", value: display, readOnly: true)
        }
    )
)
Appian SAIL: Functional Reactive Programming

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>john.smith</td>
</tr>
<tr>
<td>John</td>
<td>Doe</td>
<td>john.smith</td>
</tr>
<tr>
<td>John</td>
<td>Doe</td>
<td>john.doe</td>
</tr>
</tbody>
</table>

first: "John",
last: "Smith",
display: "john.smith"

first: "John",
last: "Doe",
display: "john.smith"

first: "John",
last: "Doe",
display: "john.doe"
first: "John"
last: "Smith"
username: "john.smith"
Appian SAIL: Functional Reactive Programming

- **Original Context**: first: "John", last: "Smith", username: "john.smith"
- **Evaluate Expression for Context & UI**
- **Client Renders UI**

![Diagram showing the flow of information from original context to UI rendering.](image-url)
Appian SAIL: Functional Reactive Programming

Original Context → Evaluate Expression for Context & UI → Client Renders UI → User Interacts with UI

- first: "John"
- last: "Smith"
- username: "john.smith"
Appian SAIL: Functional Reactive Programming

Original Context

Evaluate Expression for Context & UI

Client Renders UI

User Interacts with UI

Update Context

first: "John"
last: "Smith"
username: "john.smith"

last: "Doe"
Appian SAIL: Functional Reactive Programming

Original Context

Evaluate Expression for Context & UI

Client Renders UI

User Interacts with UI

Update Context

first: "John"
last: "Smith"
username: "john.smith"

last: "Doe"
Appian SAIL: Functional Reactive Programming

Original Context

Evaluate Expression for Context & UI

Client Renders UI

User Interacts with UI

Update Context

first: "John"
last: "Smith"
username: "john.smith"

last: "Doe"
Functional Programming

$f(x)$
Functional Programming

One typically thinks of functions as mathematical operations.
Functional Programming

One typically thinks of functions as mathematical operations.

\[
\text{add}(a, b) = f(a, b) \{ \ a + b; \ }
\]

\[
\text{add}(1, 2) = 3 \\
\text{add}(2, 3) = 5
\]
Functional Programming

How about a function to define a User Interface?
Functional Programming

How about a function to define a User Interface?

a!textField(label: "First", value: first, saveInto: first)
Functional Programming

How about a function to define a User Interface?

```apl
a!textField(label: "First", value: first, saveInto: first)
```

First

John
Functional Programming

How about a function to define a User Interface?

```xml
<HelloWorld name="MIT"/>
```

```xml
<div>
  <h1>Hello {this.props.name}!</h1>
</div>
```

https://facebook.github.io/react/
Why Functional Reactive UIs?

- Easier to test & reason about
  - Given some data (inputs), render a UI
  - Function describes state at any given point in time
  - UI functions only depend on inputs, not hidden state, not global state. i.e. stateless UIs
Why Functional Reactive UIs?

- Easier to test & reason about
  - Given some data (inputs), render a UI
  - Function describes state at any given point in time
  - UI functions only depend on inputs, not hidden state, not global state. i.e. stateless UIs
- Composable
  - UIs are assembled by declaratively gluing functions
Why Functional Reactive UIs?

- Easier to test & reason about
  - Given some data (inputs), render a UI
  - Function describes state at any given point in time
  - UI functions only depend on inputs, not hidden state, not global state. i.e. stateless UIs
- Composable
  - UIs are assembled by declaratively gluing functions
- Powerful
  - Functions encapsulate business and UI logic
  - No separate, underpower, templating language
  - Templates are functions, placeholders are parameters
Appian Engineering & Internship

http://www.appian.com/about/careers/life-at-appian/

http://www.appian.com/about/careers/university-recruiting/

http://www.appian.com/about/careers/engineering/
GOOD LUCK!
thank you!