Resilient React.js

Building apps with a functional approach

MidwestJS 2017

Jeff Barczewski

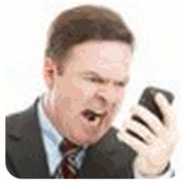
- @jeffbski
- jeff@codewinds.com
- https://codewinds.com/mwjs2017

Jeff Barczewski

- Married, Father, Catholic
- 27 yrs (be nice to the old guy :-)
- JS (since 95, exclusive last 5 years)
- Open Source: redux-logic, pkglink, …
- Work: OCI, MasterCard ApplePay, Elsevier, RGA
- Founded CodeWinds, live/online training (React, Redux, Immutable, RxJS) – I love teaching, contact me

CodeWinds Training

- Live training (in-person or webinar)
- Self-paced video training classes (codewinds.com)
- Need training for your team on any of these?
  - React
  - Redux
  - RxJS
  - JavaScript
  - Node.js
  - Functional approaches
- I'd love to work with you and your team
- I appreciate any help in spreading the word.



PHP CEO
@PHP_CEO

Follow

HIRING PRO TIP: HIPPIES ARE NOT THE SAME AS HIPSTERS

I HAVE MADE THIS MISTAKE AND WE NOW HAVE A FRONTEND MVC FRAMEWORK WRITTEN IN FORTRAN

RETWEETS
1,463

FAVORITES
825

My early career

- Aerospace Engineer – US Air Force, ASD/XR Wright Patterson AFB, Ohio
  - Fortran 77 with extensions
  - C
  - C++ / Interviews
- Consulting
  - C++
  - Java

Inspired

F15 Eagle

F-15E Strike Eagle by Gerry Metzler - IMG_214 Licensed CC BY-SA 2.0

Afghanistan, F-15E 391st" by Staff Sgt. Aaron Allmon (USAF) - Src. Public domain

Life happens

F15 Single Wing Landing

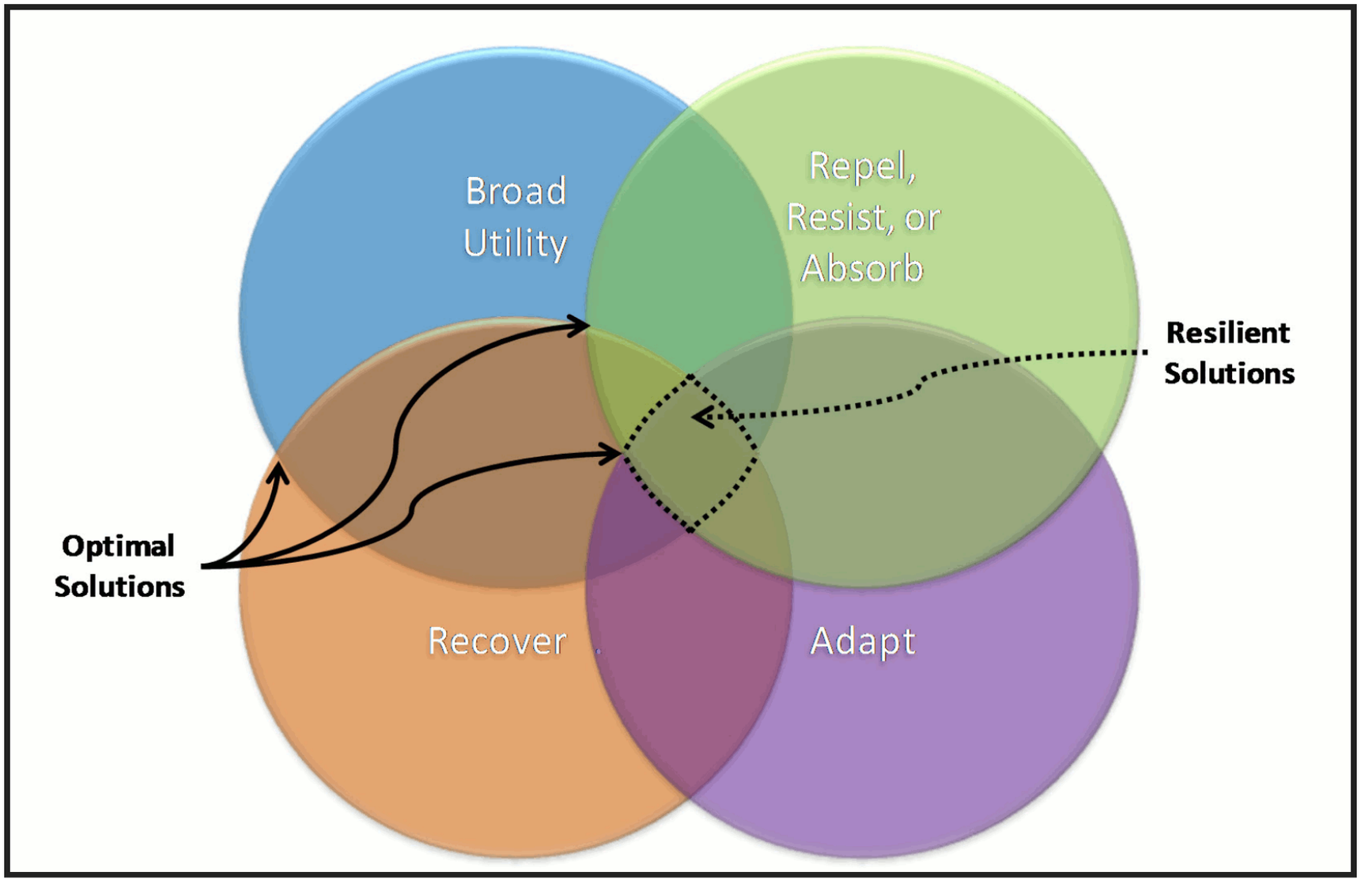
F-15 Single wing by Israeli Defense Force

F-15 Single wing landing by History Channel

Traits that inspire me

- Performant
- Adaptability
- Durability
- Trusted

Resilience


Broad
Utility
Repel,
Resist, or
Absorb
Resilient
Solutions
Optimal
Solutions
Recover
Adapt

Battle Ready

- Highly scalable Node.js microservice
- Fault tolerant
- Benchmarking - will it handle the load?
- Monitoring - how is everything working?
- Debugging - why did my transaction fail?

The plan

- Make sure every piece the architecture is performant
- Run lots of instances on each server and have many servers
- Dashboard with charts to monitor
- Plan for dealing with the log data
- Real benchmark in production
- Kill things to watch fail over and recovery (pagers lit up)

D-Day - The launch

- War room staffed 24 hrs (three shifts)
- Hotline to all partners
- Monitors activated
- All hands on deck

Lessons learned

- stateless processes can generally be scaled nicely
- great suite of automated tests
- functional JavaScript is easy to test, understand, and augment
- benchmarking beyond expected loads and to the breaking point really provides great insights
- testing in production found lots of things that needed correction
- writing reactive dashboard/monitoring was challenging (lots of state to keep track of)

Object Oriented Programming

OO JavaScript (p1)

```
class App extends Component {
  constructor(props) {
    super(props)
    this.state = { todos: [] };
  }
  addTodo = (text) => {
    const todos = [
      {
        id: createUniqueID(),
        completed: false,
        text: text
      },
      ...this.state.todos
    ]
    this.setState({todos});
```

OO JavaScript (p2)

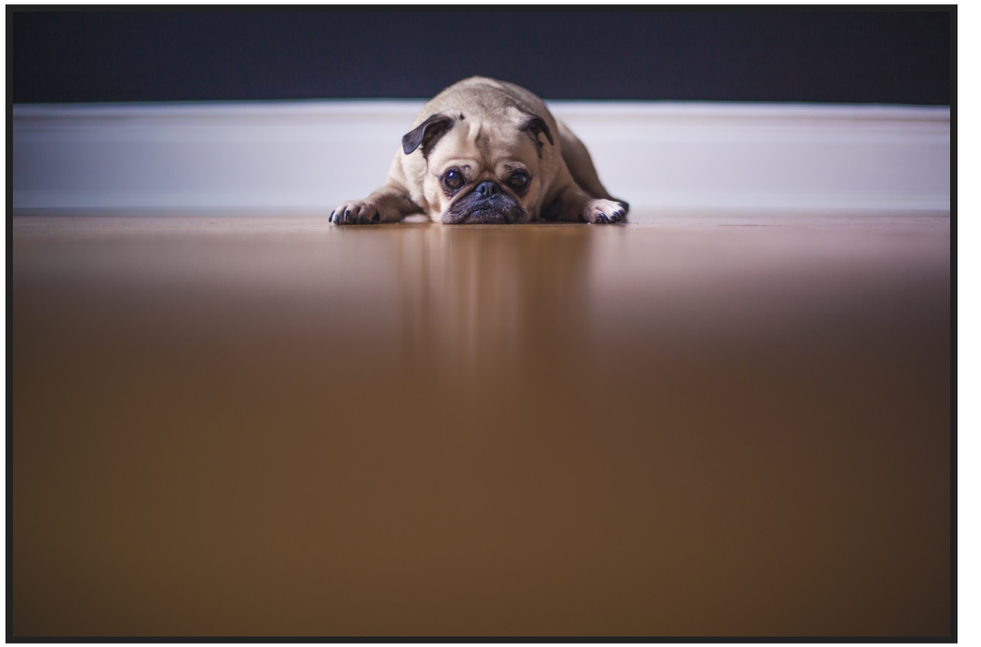
```
  render() {
    return (
      <div>
        <Header addTodo={this.actions.addTodo} />
        <MainSection todos={this.state.todos} actions={this.actions} />
      </div>
    );
  }
}
```

OOP Advantages

- Methods live close to the data they work with
- Reuse through inheritance
- Encapsulation - protect data – changes performed by methods
- Flexibility through Polymorphism – obj.draw()

OOP Shortcomings

- Clutter – classes quickly grow in any real usage, everything is commingled - Hard to reason about
- Inheritance spreads implementation over many classes/files
- OOP favors mutation
- Cross domain sharing is difficult



Functional Programming

Functional JS

```
export const App = ({todos, actions}) => (
  <div>
    <Header addTodo={actions.addTodo} />
    <MainSection todos={todos} actions={actions} />
  </div>
)

export default connect(
  state => ({
    todos: state.todos
  }),
  dispatch => ({
    actions: bindActionCreators(TodoActions, dispatch)
  })
)(App);
```

Functional JS Advantages

- Easy to reason about especially for pure functions
- Composition is simple
- Separation of concerns – do one thing well
- Testing is easy for stateless functions
- Treating data immutably reduces surprises and enables features like undo and auditing
- Event driven system like Redux creates predictable one directional data flow and makes it easy to react in multiple domains

Higher Order

- **Higher Order Functions** – function that takes a function and returns a function

```
const twice = (fn, x) => fn(fn(x));
const fn = x => x + 3;
const result = twice(fn, 7);  // (7+3)+3 = 13
```

- **Higher Order Components** (HOC) – function that takes a component and returns a component

```
const Profile = ({ name }) => <div>{ name }</div>;
const PureProfile = pure(Profile);
ReactDOM.render(<PureProfile name={myName} />, div);
```

Functional Composition

```
const getCurrentDate = () => new Date();
const format = date => date.toLocaleString();
const createTodayMessage = formattedDate => `Today is ${formattedDate}`;

const str = createTodayMessage(format(getCurrentDate()));

const createFormattedTodayMessage = compose(
  createTodayMessage,
  format
);

const str2 = createFormattedTodayMessage(getCurrentDate());
```

lodash/fp

- functional style use of lodash
- immutable
- for most calls, data is last
- curries

https://lodash.com/

lodash/fp (continued)

```
import { get, set, update, compose } from 'lodash/fp';
const state = { a: { aa: { aaa: 123 }}};
const aaa = get('a.aa.aaa', state); // 123

const state2 = set('a.aa.aaa', 234, state); // use dotted path OR
const state2 = set(['a', 'aa', 'aaa'], 234, state); // use array path

const state3 = update('aa.aa.aaa', x => x + 1); // increment aaa

const state4 = compose(
 set('a.aa.aaa', 345),
 set('b.bb.bbb', 456)
)(state3);
```

Selectors

```
const state = {
 items: [
   { id: 1, name: 'Foo', catId: 20 },
   { id: 2, name: 'Bar', catId: 30 }
 ],
 categories: [
   { catId: 20, name: 'Games' },
   { catId: 30, name: 'Business' }
 ]
};
const itemsSelector = state => state.items;
const categoriesSelector = state => state.categories;
```

Combining (Naively)

```
const itemsWithCategoriesSelector = state => {
  const items = itemsSelector(state);
  const categories = categoriesSelector(state);
  const findCategory = catId => find(c => c.catId === catId)(categories);

  const itemsWithCategories = items.map(i => {
    const category = findCategory(i.catId);
    return compose(
      set('category', category),
      unset('catId')
    )(i);
  });

  return itemsWithCategories;
};
```

Reselect - memoized selectors

```
const itemsWithCategoriesSelector = createSelector(
  /* simple light selectors */
  itemsSelector,
  categoriesSelector,
  /* complex expensive computation to memoize */
  (items, categories) => {
    const findCategory = catId => find(c => c.catId === catId)(categories);
    return items.map(i => {
      const category = findCategory(i.catId);
      return compose(
        set('category', category),
        unset('catId')
      )(i);
    });
  }
```

Stateless Function Components

```
function Orders({ orders }) {
  return (
    <ul>
      { orders.map(x => (
        <li key={ x.id }>{ x.name } - { x.date }</li>
      ))}
    </ul>
  );
}
```

Stateless function components

(ES6 arrow functions)

```
const Orders = ({ orders }) => (
  <ul>
    { orders.map(x => (
      <li key={x.id}>{ x.name } - { x.date }</li>
    ))}
  </ul>
);
```

Stateless Function Components

Advantages

- Extremely simple, easy to reason about
- Easy to test, no state, pass props to test modes
- Pure functions – input + output, nothing else
- No need to use “this”, props are local variables
  - Modular and reusable
- FaceBook team will continue to create optimizations. Favors functional style.

Recompose

- Utility library for creating HOC's
  - "The lodash for React"
- npm install recompose
- Compose in common functionality using parameterized HOC functions
- Allows you to stay in the functional world
  - Stateless function components

https://github.com/acdlite/recompose

Solving problems with Recompose

```
const PureProfile = pure(Profile);  // optimized

const OptimProfile = onlyUpdateForKeys(['name', 'age'])(Profile);

const DefaultedComp = defaultProps({
  greeting: 'Hello'
})(Comp);

const EntComp = renameProp('first', 'firstName')(Profile);
```

Recompose (continued)

```
const Comp = withProps(props => ({
  fullName: `${props.first} ${props.last}`,
  mode: 1
}))(EchoProps);
ReactDOM.render(<Comp first="John" last="Smith" />, div);

const Comp = mapProps(props => ({
  firstName: props.first,
  lastName: props.last
}))(EchoProps);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
```

Recompose (continued)

```
const Comp = compose(
  pure,
  withProps(props => ({
    fullName: `${props.first} ${props.last}`,
    mode: 1
  }))
)(EchoProps);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
ReactDOM.render(<Comp first="Amanda" last="Green" />, div);
```

Recompose (continued)

```
const Form1 = ({ values, onChange, onSubmit }) => {
  return (
    <form onSubmit={onSubmit}>
    <input name="first" value={values.first} onChange={onChange} />
    <input name="last" value={values.last} onChange={onChange} />
    <button>Submit</button>
    </form>
  );
};
```

Recompose (continued)

```
import { set } from 'lodash/fp';
const Comp = compose(
  withState('values', 'updateValues', { first: '', last: ''}),
  withHandlers({
    onChange: ({ updateValues }) => ev => {
      const {name, value} = ev.target;
      updateValues(x => set(name, value, x));
    },
    onSubmit: ({ values, updateValues }) => ev => {
      ev.preventDefault();
      console.log('submit', values); // do something with data
      updateValues(x => ({ first: '', last: '' }));
    }
  })
)(Form1);
```

Recompose (continued)

```
const Loading = props => <div>Loading...</div>;
const Loaded = ({ content }) => <div>Loaded { content }</div>;

const Comp = branch(
  props => !props.content,
  renderComponent(Loading)
)(Loaded);

ReactDOM.render(<Comp />, div, () => {
  console.log(div.innerHTML);
  ReactDOM.render(<Comp content="hello" />, div, () => {
    console.log(div.innerHTML);

  });
});
```

Recompose (continued)

```
const Loading = props => <div>Loading...</div>;
const MainContent = ({ items }) => <ul>
  { items.map(x => <li key={x.id}>{x.name}</li> )}
</ul>;

const DynamicContent = compose(
  lifecycle({
    componentDidMount() {
      axios.get('http://yourserver.com')
        .then(res => { this.setState({ items: res.data.result}); });
    }
  }),
  branch(props => !props.content, renderComponent(Loading))
)(MainContent);
```

Summary

- Functional JavaScript is fun!
- Functional JavaScript is easy to reason about
- Composable
- Testable
- Recompose wraps your components allowing you to use stateless function components and to compose in other functionality

Thanks

- https://codewinds.com/mwjs2017 (slides, resources)
- https://codewinds.com/ (newsletter tips/training)
- jeff@codewinds.com
- @jeffbski