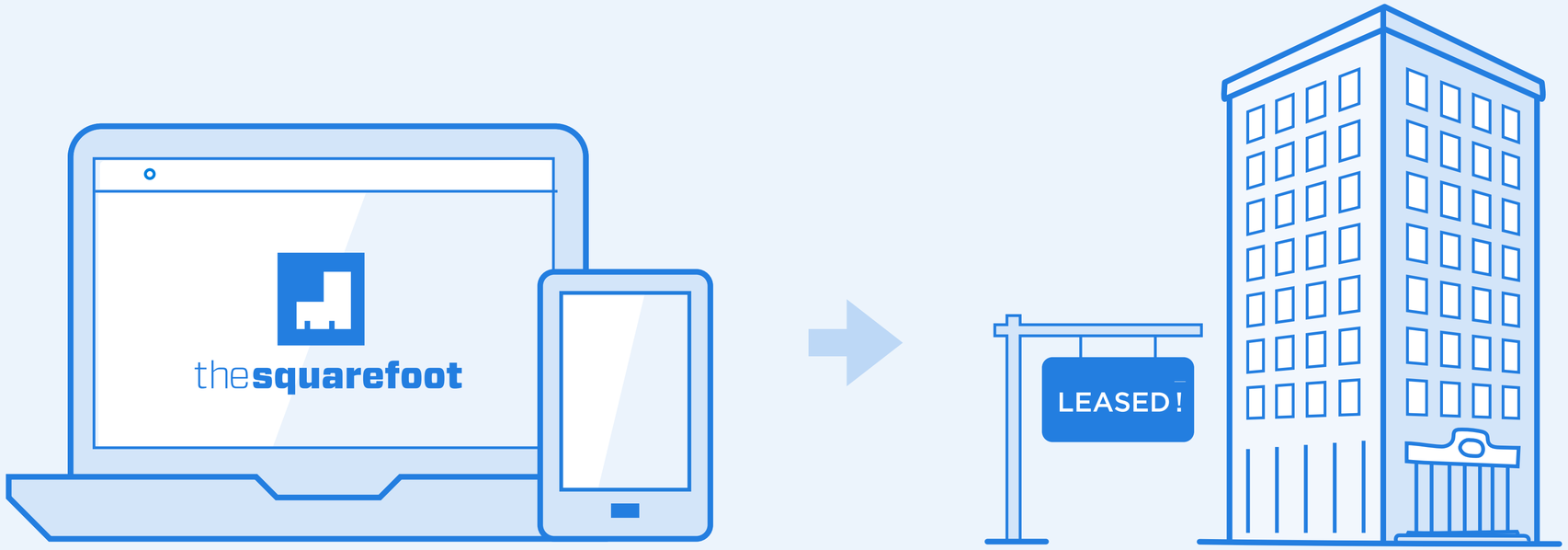
thesquarefoot

JAVASCRIPT ON RAILS

JS STACKUP
JUNE 14, 2016

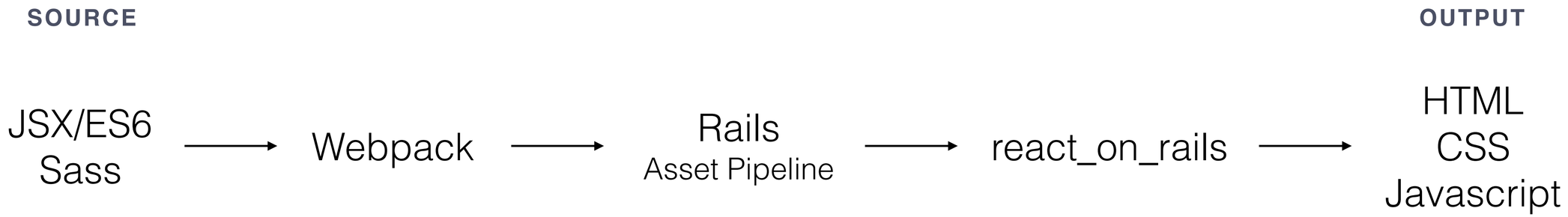
TheSquareFoot is the marketplace that connects businesses with the spaces they deserve.



Javascript on Rails: Previously

- Monolithic Ruby on Rails application
- Markup in Haml
- Client-side logic in Coffeescript
- Rails asset pipeline

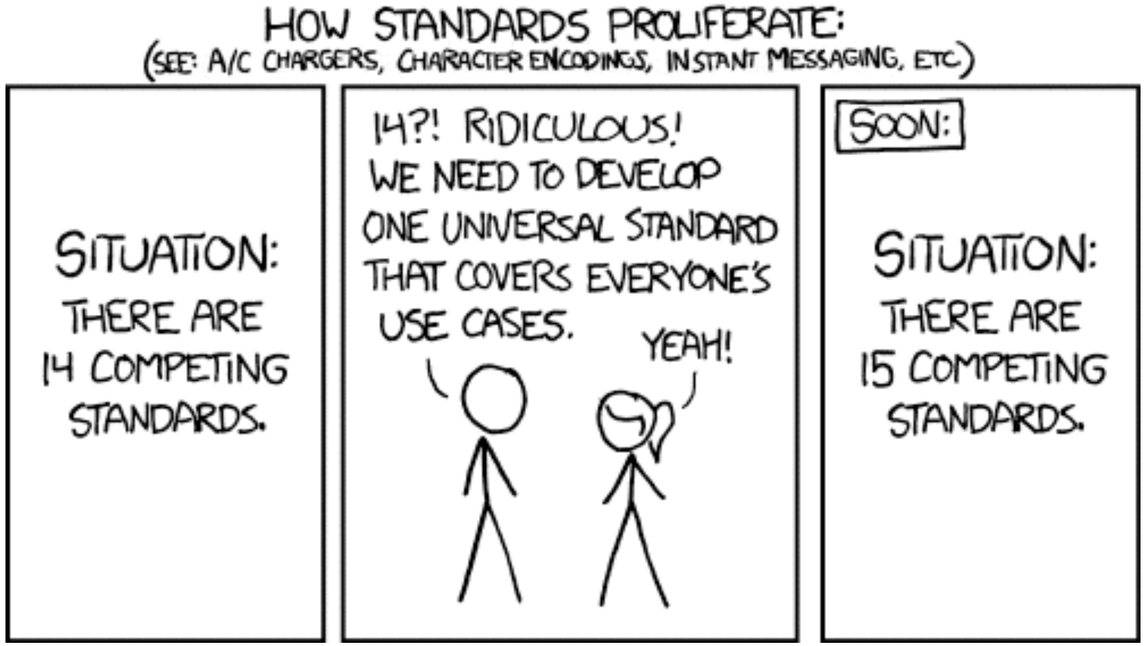
Javascript on Rails: Transition



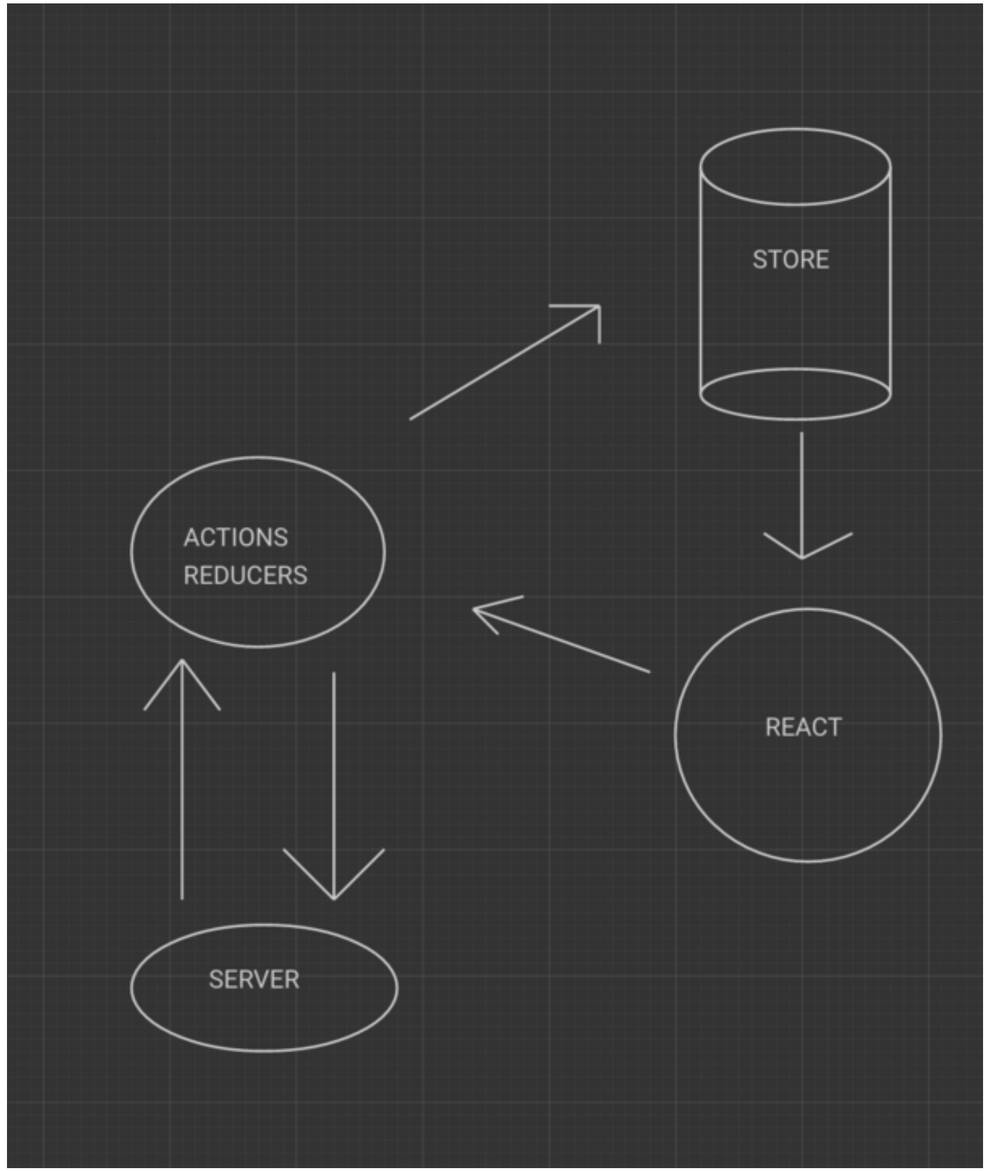
- Thin-server application with rich Javascript presentation layer
- Markup in JSX
- Client-side logic in ES6
- Webpack build system

Javascript on Rails: Transition

- Documentation is generally written for fresh apps, not so helpful for updating and maintaining a legacy codebase
- No wrong answers, but there sure are a lot of answers

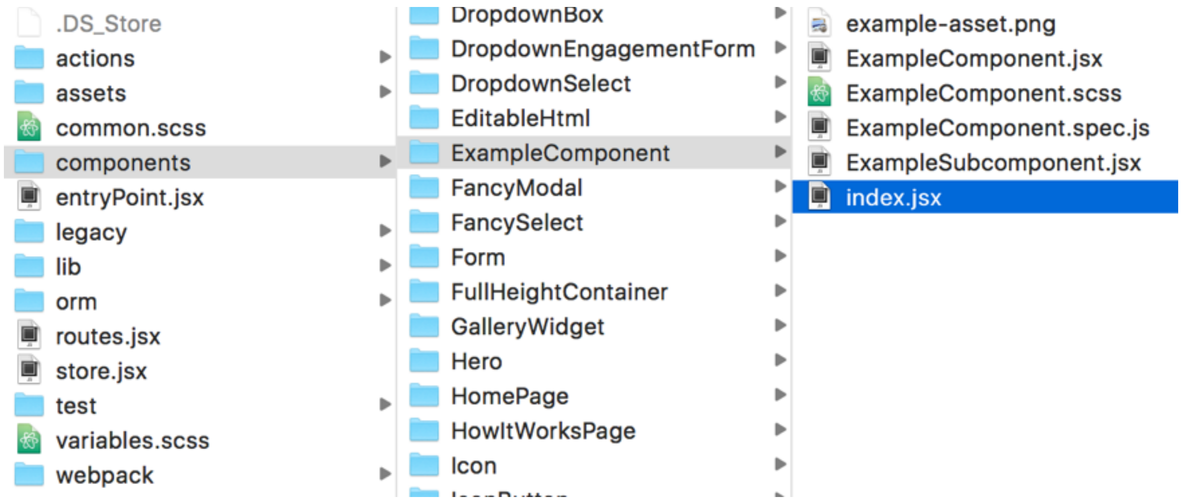


Javascript on Rails: Project Structure



Javascript on Rails: Component Structure

- All shared components live in a single flat directory
- Each component has a directory containing all of its assets
- Components only get their own directory if used in multiple places
- Redux actions and reducers scoped by model, may move to Component-based structure in the future



Aside: Favorite bits of ES6/ES2015+

- Arrow functions + class properties for early-binding React event handlers

```
class MyComponent extends React.Component {
  // ES6
  render() { return <div onClick={this.onClick.bind(null, this.props.someProp)} />; }
  onClick(param) { doSomethingWithProp(param); }

  // ES7
  render() { return <div onClick={this.onClick} />; }
  onClick = () => { doSomethingWithProp(this.props.someProp); };
}
```

- Destructuring and spread operator offer lots of sugar

```
const { foo, bar, baz } = this.props;
const foo = this.props.foo,
      bar = this.props.bar,
      baz = this.props.baz;

function PassThroughComponent({ importantProp, ...otherProps }) {
  if (importantProp !== SOME_VALUE) return;
  return <InnerComponent {...otherProps} />;
}
```

```
<MyComponent {...{ foo, bar, baz }} />
<MyComponent foo={foo} bar={bar} baz={baz} />
```

Javascript on Rails: What went right

- Javascript-based frontend disconnected from Rails backend
- Frontend can use requires/imports, npm
- Server can prerender React
- Modern tooling: Live reload, linting, testing
- CSS Modules makes interacting with legacy styles safer

Javascript on Rails: What went wrong

- Transition was not quick and easy
- Complicated build system to debug
- Can’t communicate with Rails, integration with asset-sync
- Prerendering issues with SVG sprites and browser APIs

Client side data structure

constraints:

- structure of redux store needs to be independent of view
- need to be able to share state between page transitions/views

client side data structure

solution:

- use the relational data model that we have already designed on the server
- use redux-orm - gives us a basic queryable js object database

client/server interaction

traditional options:

- create many endpoints and make multiple requests per view (RESTful)
- create ad hoc endpoints for views - not DRY (message based)

client/server interaction

solution:

- create two endpoints through which client can access entire entire application (read and write)

client/server interaction

1. client makes request to server using structured JSON
2. server knows how to turn request into relational data
3. client knows how to consume relational response and recreate using redux-orm

read endpoint - request/response

```
{
  read: {
    model: 'Listing',
    where_clause: { id: 1234 },
    attributes: [
      'price_per_square_foot',
      'square_feet',
      {
        listing_contacts: {
          contact: [
            'name',
            'email',
            'phone',
          ]
        }
      }
    ]
  }
}
```

```
[
  {
    model: 'Listing',
    attributes: {
      id: 1234,
      price_per_square_foot: 67,
      square_feet: 4500,
    }
  },
  {
    model: 'ListingContact',
    attributes: {
      contact_id: 5678
      listing_id: 1234
    }
  },
  {
    model: 'Contact',
    attributes: {
      id: 5678,
      name: 'Ross',
      email: 'ross@thesquarefoot.com',
      phone: 8675309
    }
  },
]
```

read endpoint - action/reducer

```
function loadRecordsIntoReduxORM(records) {
  return (dispatch) => {
    records.forEach(record => {
      const { model, attributes } = record;

      dispatch({
        type: `create/${model}`,
        attributes: attributes
      });
    });
  };
}

export function apiRead(readReq) {
  return (dispatch) => {
    request
      .post('/api/read')
      .send({ read: readReq })
      .end((err, res) => {
        dispatch(loadRecordsIntoReduxORM(res.body));
      });
  };
}
```

Advantages

- no thinking about data modeling on the client
- allows front end developer to build features with little support from backend developer

Drawbacks

- Front end developer must understand database schema and any idiosyncrasies that exist on the backend
- client data structure tied to data models on the server

Questions?

Questions?