Reusable Component Design

Cory House

@housecor | reactjsconsulting.com

The Plan

Reusable component design tips

Atomic Design

- What?
- Why?

Reusable Component Design Tips

Tip 1

Avoid weak wrapper elements

```
// Avoid
<p>
    {children}
</p>
```

```
// Prefer
<div>
    {children}
</div>
```

Tip 2

Specify PropTypes

Why PropTypes

```
YourMom.propTypes = {
  /** Importance, on a 1 - 10 scale */
  importance: PropTypes.number.isRequired,

  /** Cookie deliciousness, on 1 - 10 scale */
  cookies: PropTypes.number.isRequired,
};
```

Clarify API

Warn consumers

Specify object shape

Enhanced autocompletion

FLOW IS A STATIC TYPE CHECKER FOR JAVASCRIPT.

GET STARTED INSTALL FLOW Star 11,297

Current Version: v0.44.1

CODE FASTER.

Tip 3

Don’t hard code HTML IDs

Don’t Hard Code HTML IDs

May conflict with other markup

Rules out multiple instances

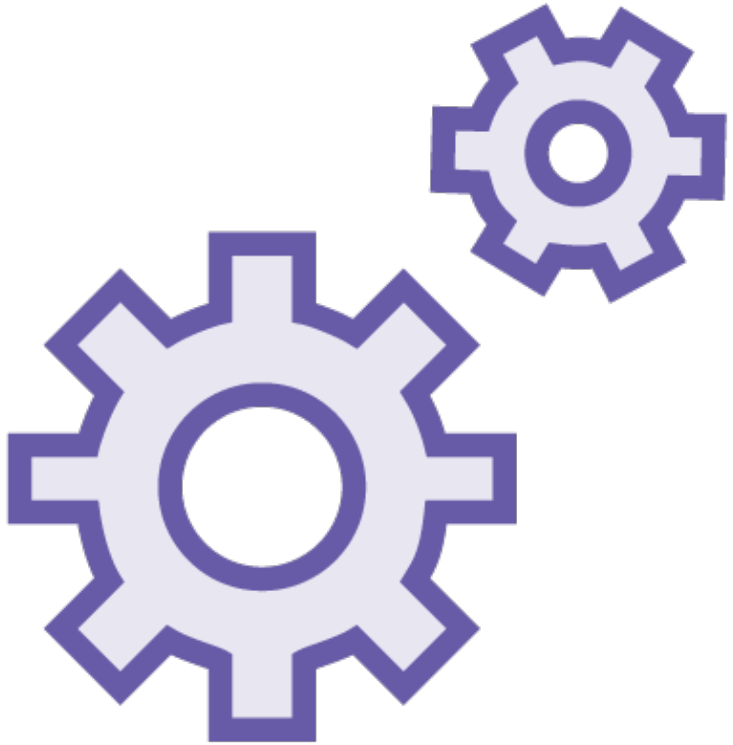
Accept HTML ID via prop

```
MyComponent.propTypes = {
    htmlId: React.PropTypes.string.isRequired
}
```

Tip 4

Set Logical Defaults

Set Logical Defaults



Save users typing

Help assure useful behavior

Convey common use case

Default Props

```
// Below MyCounter class
MyCounter.defaultProps = { initialCount: 0 };


// Using static property
class MyCounter extends React.Component {
  static defaultProps = { initialCount: 0 };


// Stateless component
const myCounter = ({initialCount = 0}) => {
```

Tip 5

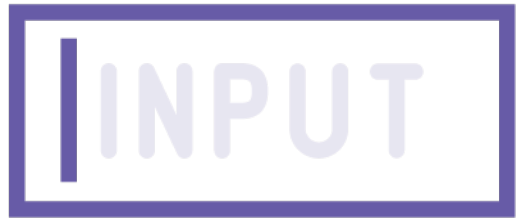
Accessibility Matters

Accessibility Concerns

Keyboard Input

Tab Indexes

Semantic HTML



ARIA

Tip 6

Consider Configuration Objects

```
<CustomerDetail
  firstName="Cory"
  lastName="House"
  title="Developer"
  office="Home"
/>
```

```
<CustomerDetail customer={{
  firstName:"Cory",
  lastName:"House"
  title: "Developer”,
  office: "Home"
}}/>
```

Why a Config Object?

Consistent component API

Less typing

Less error prone

Avoid if your component has few props

Tip 7

Consider Server Rendering

Why Support Server Rendering?

SEO

Static site generation

- Gatsby
- Phenomic

Performance

The public will expect it

To Support Server Rendering

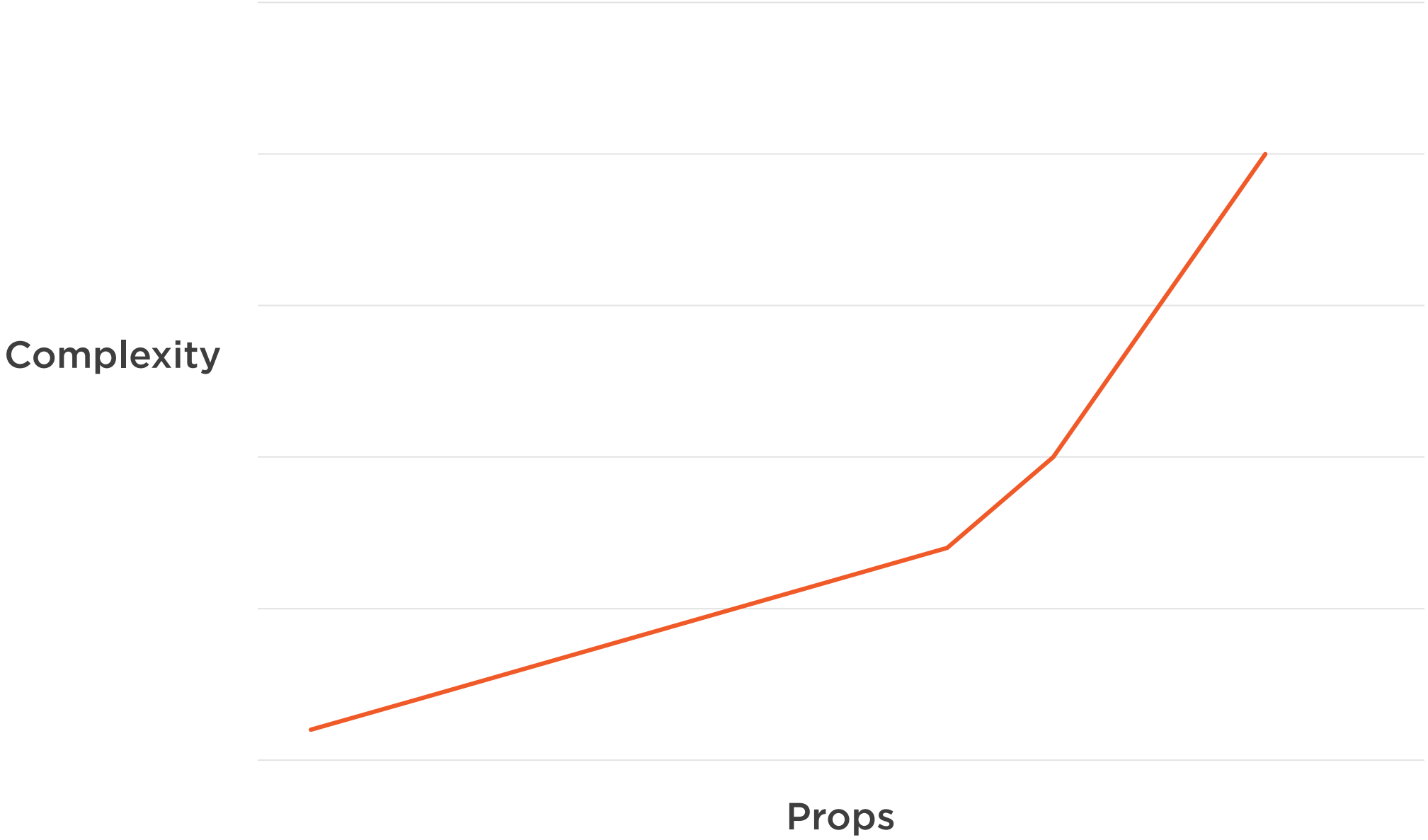
Don’t assume your component is in a browser

- **Avoid document or window calls**
- **Avoid using setTimeout**

Tip 8

Remember the Single Responsibility Principle

The single responsibility principle applies to component design too.


Complexity
Props

Prefer separate, simple components to one complicated and highly customizable component.

8 Quick Tips

1. Avoid weak wrapper elements
2. Specify PropTypes
3. Don’t hard code HTML IDs
4. Set logical defaults
5. Accessibility matters
6. Consider configuration objects
7. Consider server rendering
8. Remember the single responsibility principle

What Is Atomic Design?

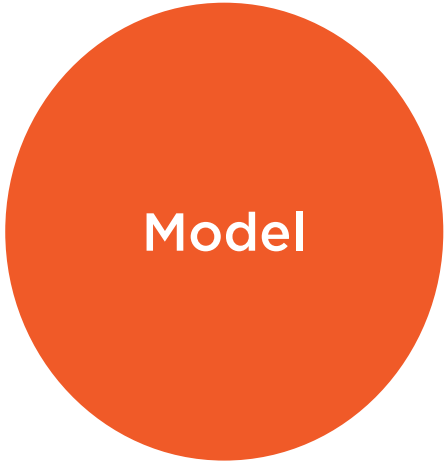
Atomic Design

Atoms	Basic building blocks
Molecules	Groups of two or more atoms
Organisms	Groups of molecules functioning together

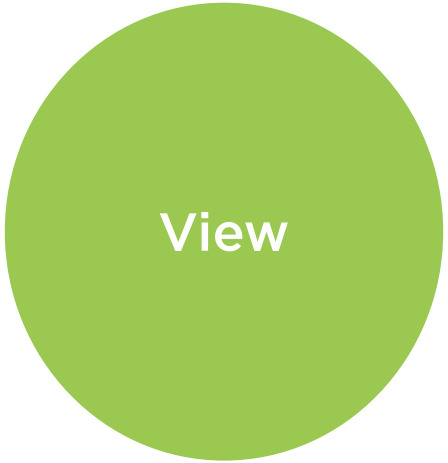

AIRBAG
Mercedes-Benz

Atoms → Molecules → Organisms

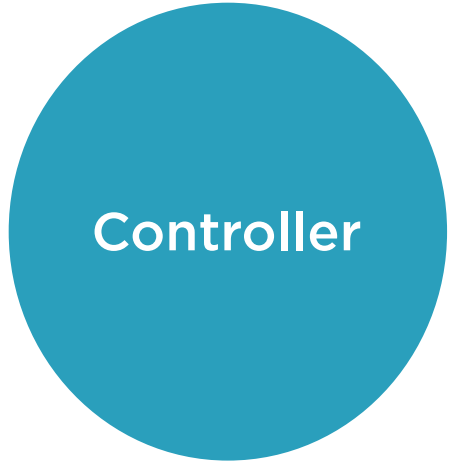
Home Page



Data

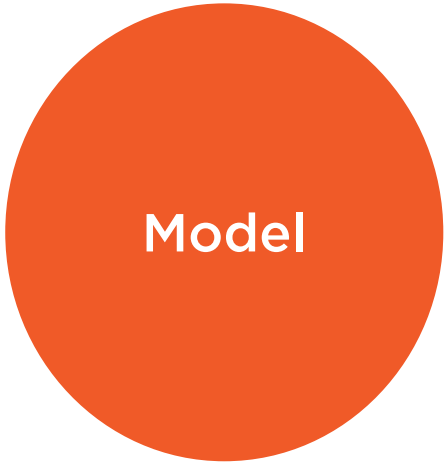


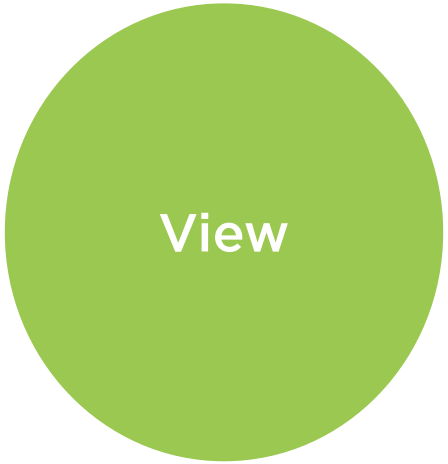
HTML

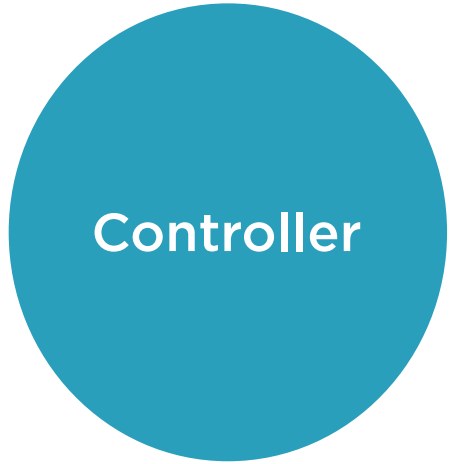


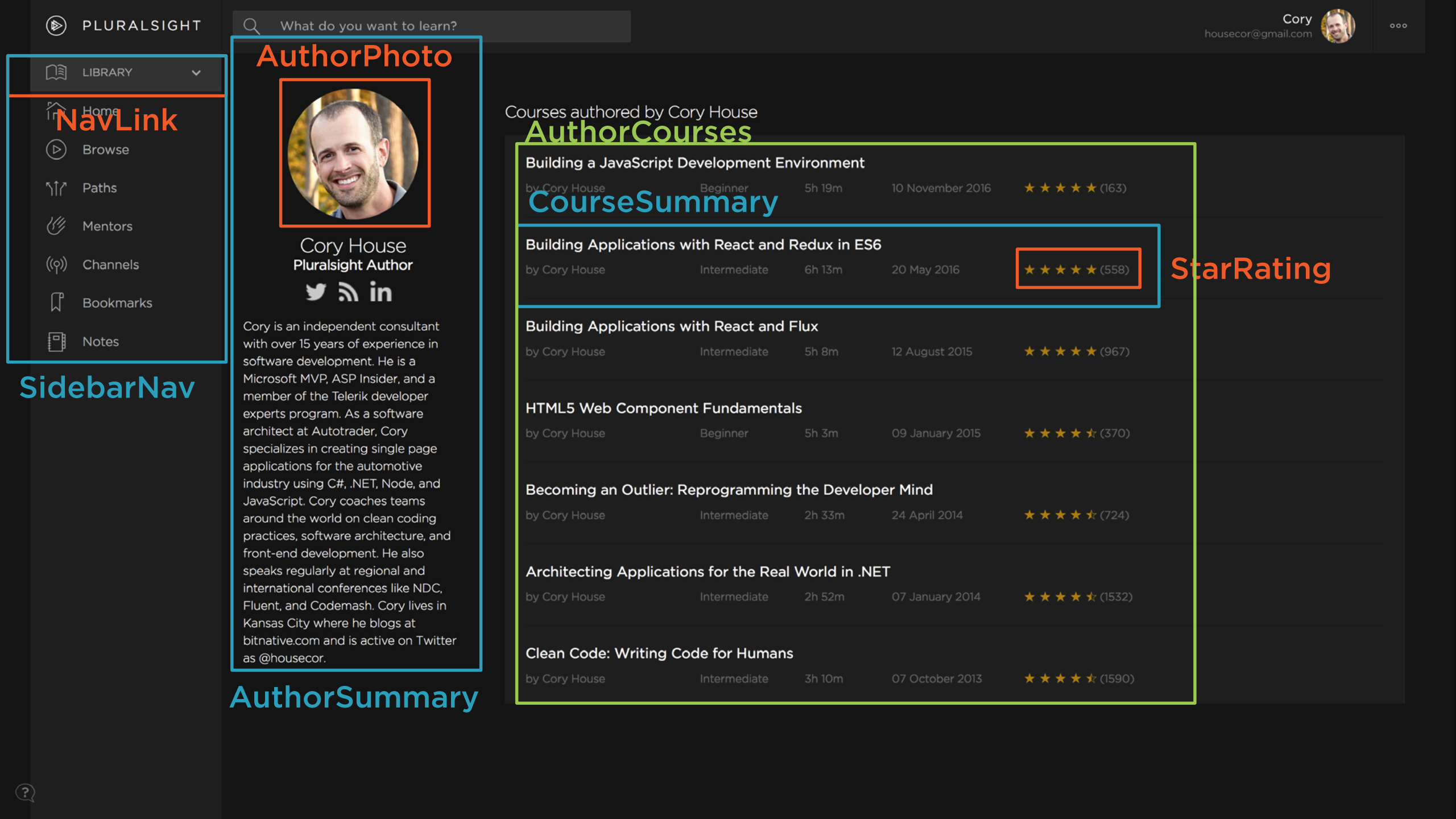
Logic

Product Page








PLURALSIGHT
What do you want to learn?
Cory
housecor@gmail.com
LIBRARY
Home
Browse
Paths
Mentors
Channels
Bookmarks
Notes
NavLink
SidebarNav
AuthorPhoto
Cory House
Pluralsight Author
Cory is an independent consultant with over 15 years of experience in software development. He is a Microsoft MVP, ASP Insider, and a member of the Telerik developer experts program. As a software architect at Autotrader, Cory specializes in creating single page applications for the automotive industry using C#, .NET, Node, and JavaScript. Cory coaches teams around the world on clean coding practices, software architecture, and front-end development. He also speaks regularly at regional and international conferences like NDC, Fluent, and Codemash. Cory lives in Kansas City where he blogs at bitnative.com and is active on Twitter as @housecor.
AuthorSummary
Courses authored by Cory House
AuthorCourses
Building a JavaScript Development Environment
by Cory House Beginner 5h 19m 10 November 2016 (163)
CourseSummary
Building Applications with React and Redux in ES6
by Cory House Intermediate 6h 13m 20 May 2016 (558)
StarRating
Building Applications with React and Flux
by Cory House Intermediate 5h 8m 12 August 2015 (967)
HTML5 Web Component Fundamentals
by Cory House Beginner 5h 3m 09 January 2015 (370)
Becoming an Outlier: Reprogramming the Developer Mind
by Cory House Intermediate 2h 33m 24 April 2014 (724)
Architecting Applications for the Real World in .NET
by Cory House Intermediate 2h 52m 07 January 2014 (1532)
Clean Code: Writing Code for Humans
by Cory House Intermediate 3h 10m 07 October 2013 (1590)

PLURALSIGHT

What do you want to learn?

Cory
housecor@gmail.com

LIBRARY

Home

Browse

Paths

Mentors

Channels

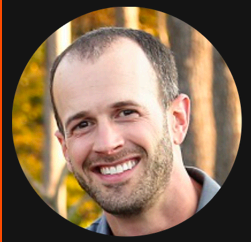
Bookmarks

Notes

NavLink

SidebarNav

AuthorPhoto



Cory House
Pluralsight Author

Cory is an independent consultant with over 15 years of experience in software development. He is a Microsoft MVP, ASP Insider, and a member of the Telerik developer experts program. As a software architect at Autotrader, Cory specializes in creating single page applications for the automotive industry using C#, .NET, Node, and JavaScript. Cory coaches teams around the world on clean coding practices, software architecture, and front-end development. He also speaks regularly at regional and international conferences like NDC, Fluent, and Codemash. Cory lives in Kansas City where he blogs at bitnative.com and is active on Twitter as @housecor.

AuthorSummary

Courses authored by Cory House

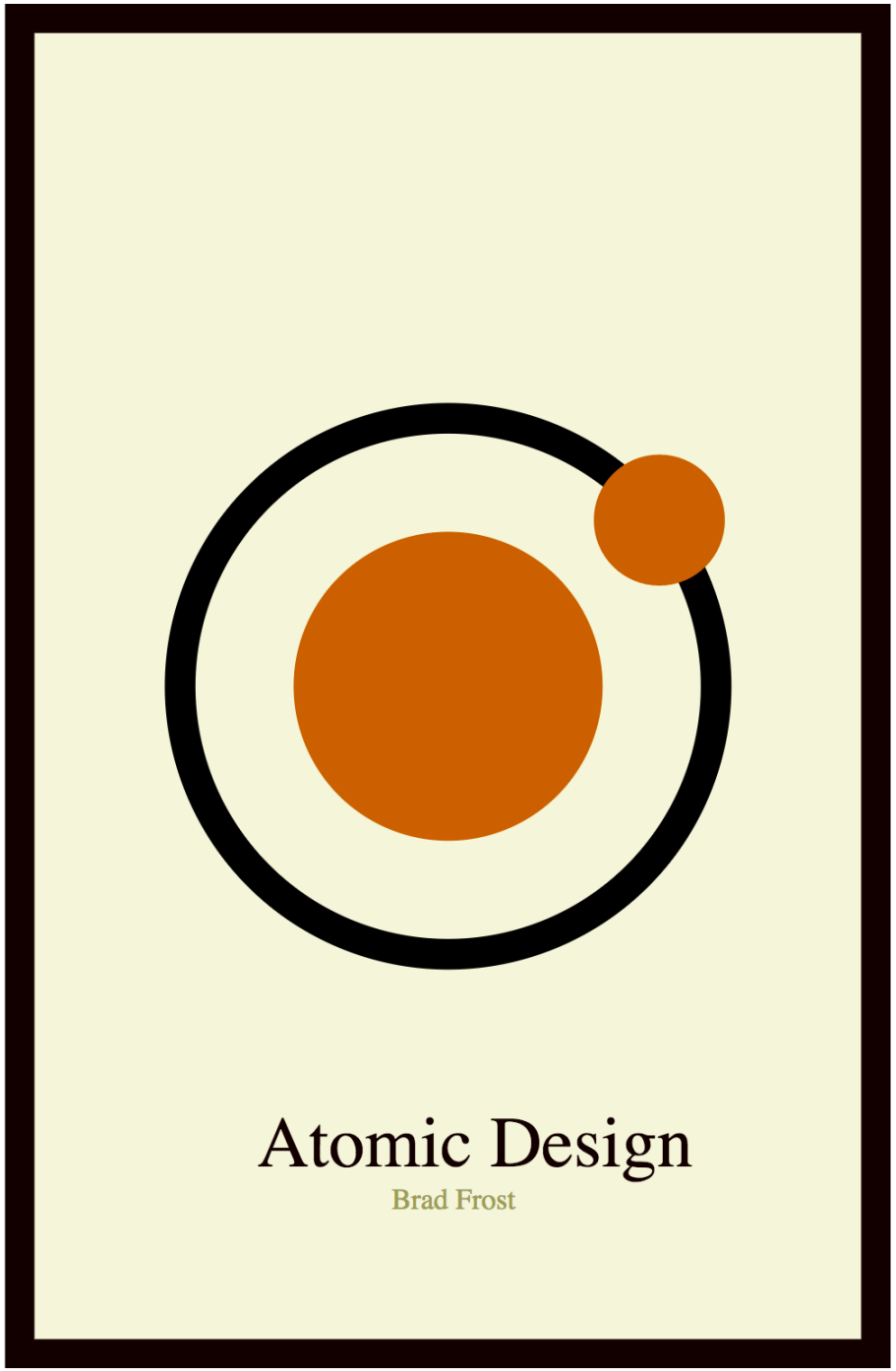
AuthorCourses

CourseSummary

StarRating

Building a JavaScript Development Environment	by Cory House	Beginner	5h 19m	10 November 2016	★★★★★ (163)
Building Applications with React and Redux in ES6	by Cory House	Intermediate	6h 13m	20 May 2016	★★★★★ (558)
Building Applications with React and Flux	by Cory House	Intermediate	5h 8m	12 August 2015	★★★★★ (967)
HTML5 Web Component Fundamentals	by Cory House	Beginner	5h 3m	09 January 2015	★★★★½ (370)
Becoming an Outlier: Reprogramming the Developer Mind	by Cory House	Intermediate	2h 33m	24 April 2014	★★★★½ (724)
Architecting Applications for the Real World in .NET	by Cory House	Intermediate	2h 52m	07 January 2014	★★★★½ (1532)
Clean Code: Writing Code for Humans	by Cory House	Intermediate	3h 10m	07 October 2013	★★★★½ (1590)

React: Nested Components

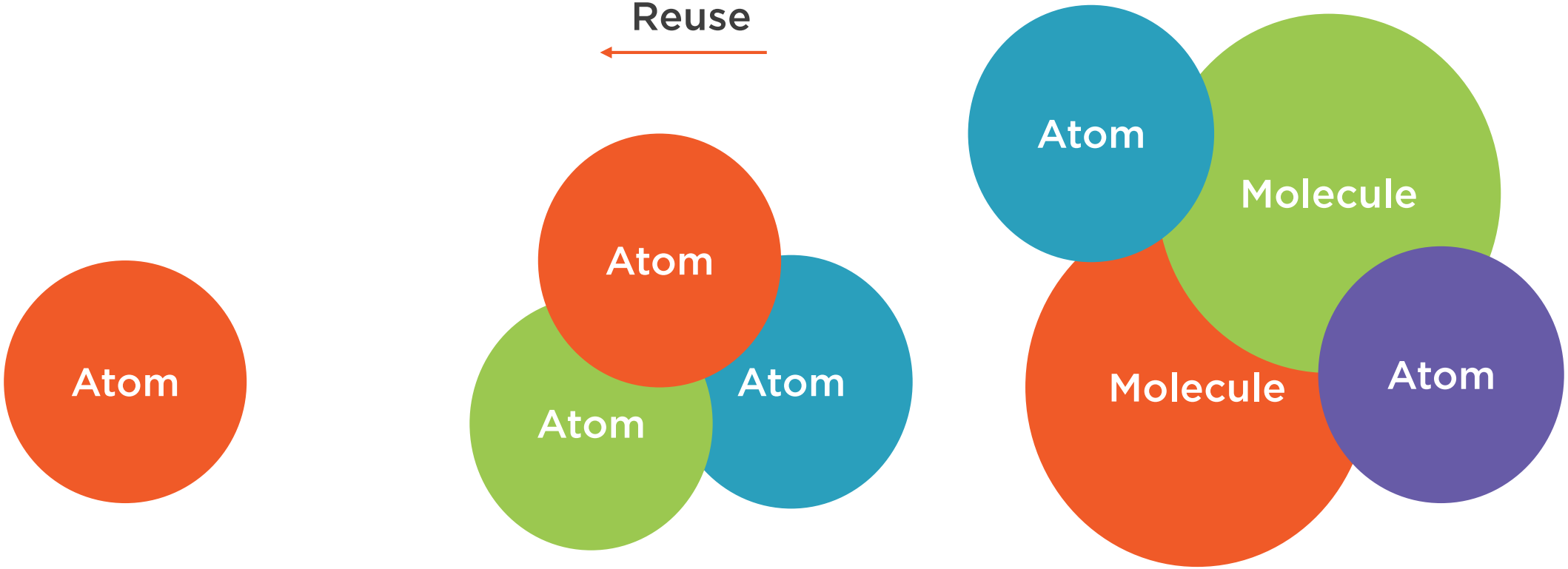


Atomic Design by Brad Frost

atomicdesign.bradfrost.com

Difficulty

Reuse

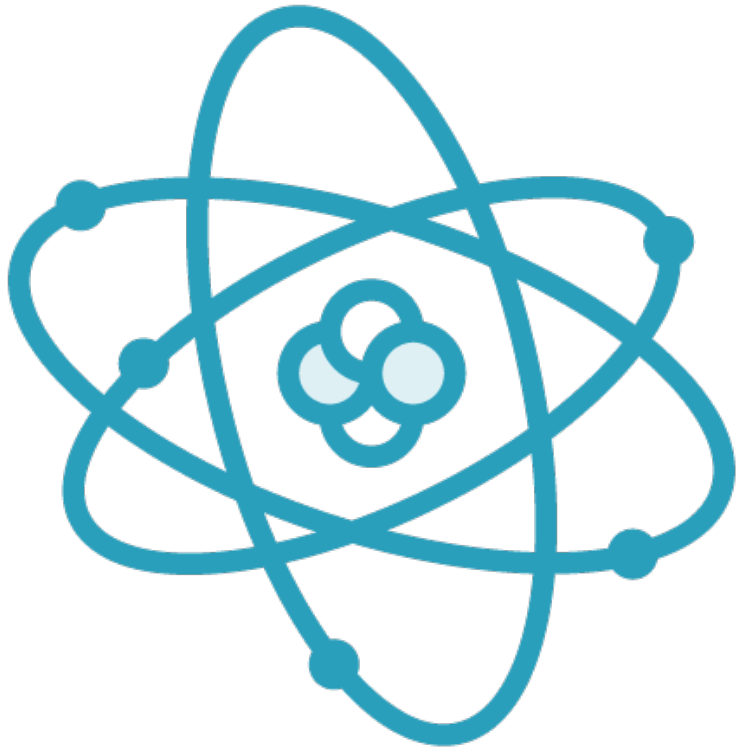


Atom

Molecule

Organism

Atomic Design
=
Modular Design



5 Stages of Atomic Design

1. Atoms
2. Molecules
3. Organisms

Focus for this module

4. Templates

Put components in a layout

5. Pages

Populate templates with content

Why Atomic Design?

Multiple layers of abstraction

Encourages reuse

Convey component relationships

Convey intent

Atomic design = hierarchy.

Wrap Up

Reusable component design:

- Avoid weak wrapper elements
- Specify PropTypes and defaults
- Don’t hard code HTML IDs
- Consider config objects
- Have a single responsibility

Atomic Design: Describe components with scientific terms

- Multiple layers of abstraction
- Encourages reuse
- Convey component relationships
- Convey intent

Next up: Atoms