Reusable & Scalable Components

Zahin Omar Alwa
@zahinize

Components….FTW! 😎🔥

- React JS
- Vue JS
- Angular JS
- Bootstrap
- UIKit
- Web Components
- Polymer
- Semantic UI
- Foundation

Components?

They are much like Legos! 😁

Why?

- Pure UI
- Functional programming
- No side effects
- Reusable, Themable & Scalable
- Easy to change
- Single Responsibility Principle

How?

- Make as few components as possible
- Components should be “just generic enough”
- Should be extracted organically from app pages as you build them
- Two simple components are better than one generic, complicated component

Few components

Less is More! 😊

- Too many layers in component tree?
- A new component for every div or style change?
- Lots of props to send down?
- Props or signature changes at every layer?
- Over optimisation of your app?

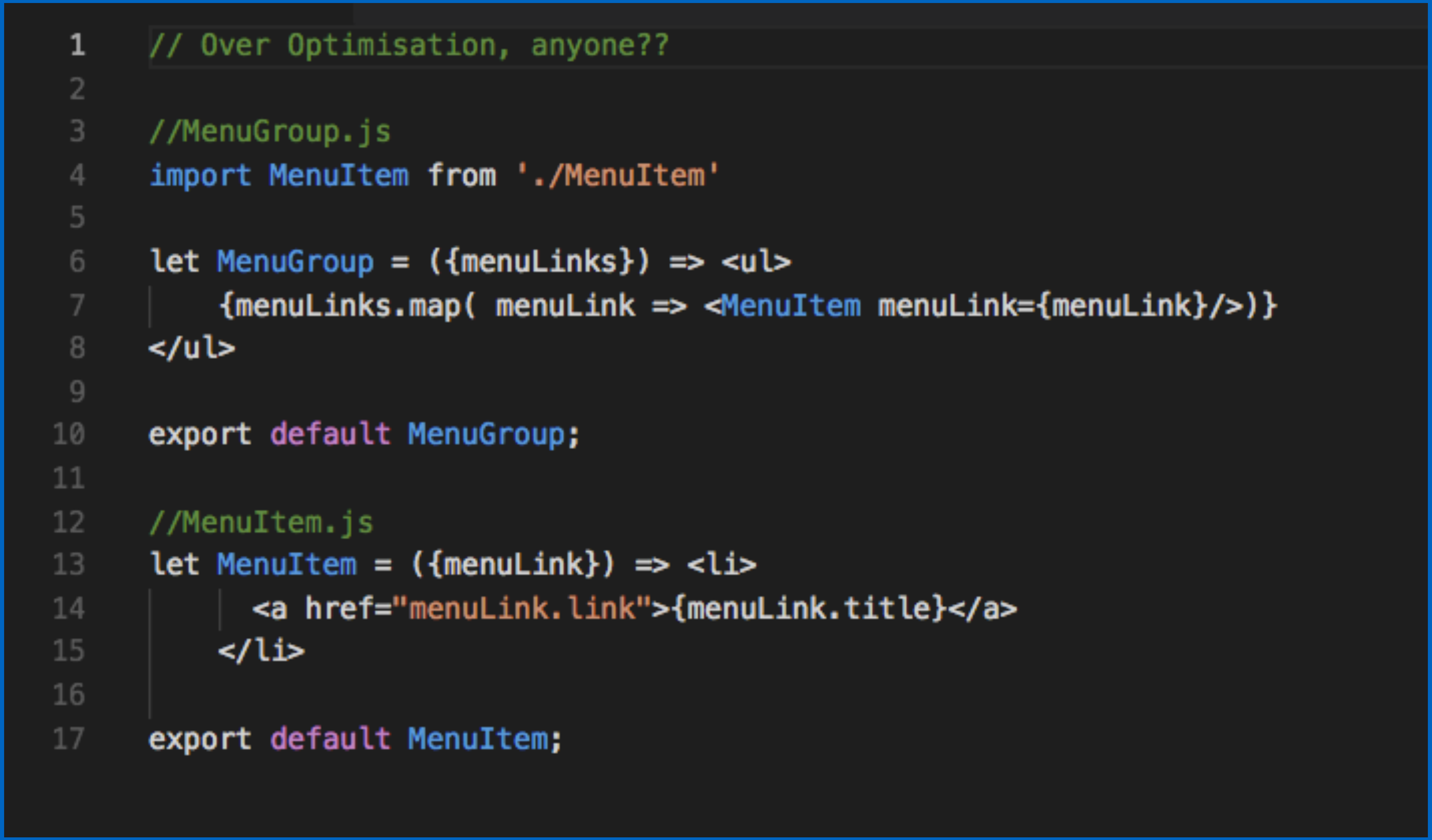
Few components

Less is More! 😊

Only divide a page or a block in multiple components when absolutely necessary

Few components

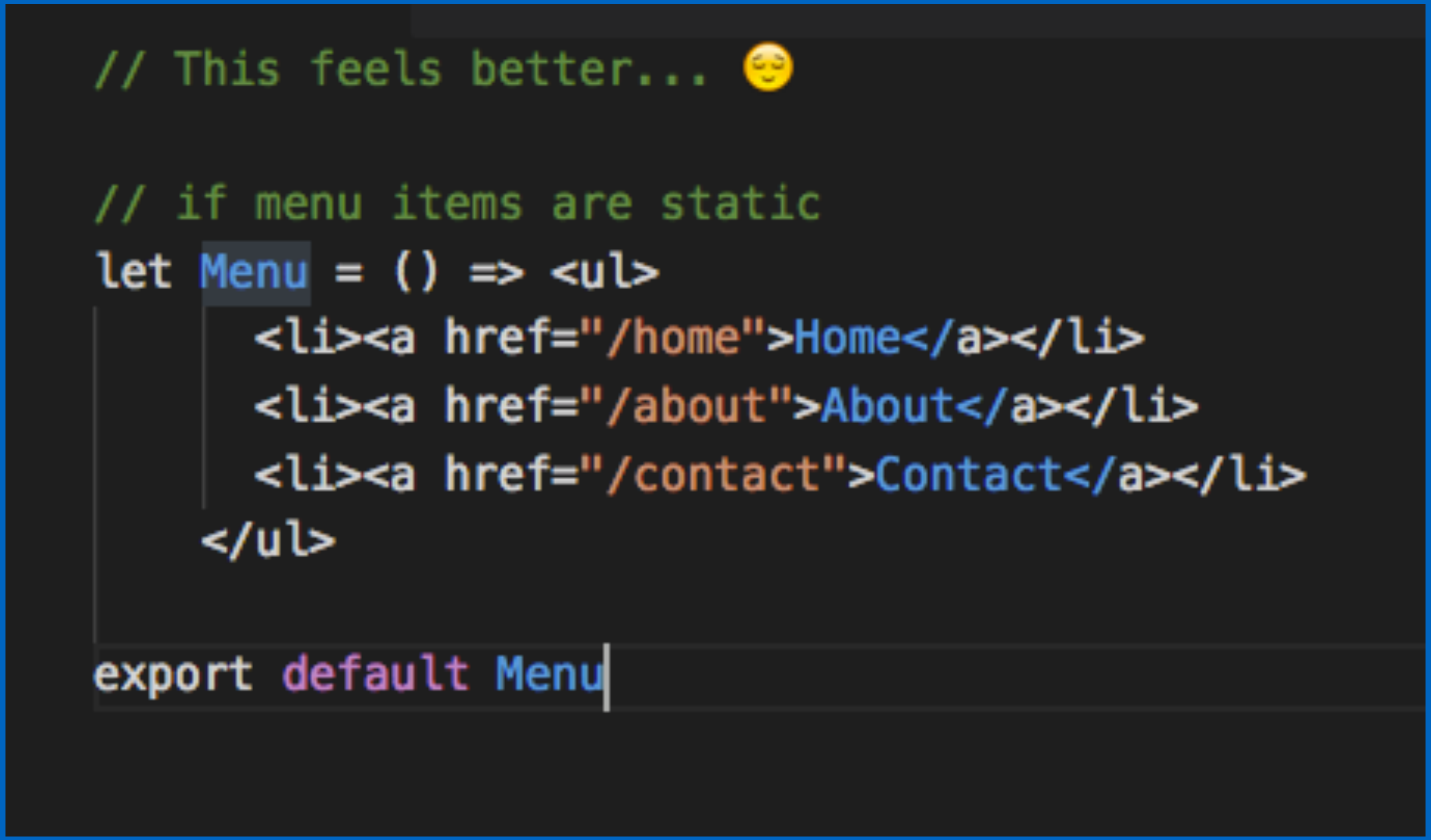
Less is More! 😊



```
// Over Optimisation, anyone??

//MenuGroup.js
import MenuItem from './MenuItem'

let MenuGroup = ({menuLinks}) => <ul>
    {menuLinks.map( menuLink => <MenuItem menuLink={menuLink}/>)}
</ul>

export default MenuGroup;

//MenuItem.js
let MenuItem = ({menuLink}) => <li>
      <a href="menuLink.link">{menuLink.title}</a>
    </li>

export default MenuItem;
```

Few components

Less is More! 😊



```
// This feels better... 😌

// if menu items are static
let Menu = () => <ul>
        <li><a href="/home">Home</a></li>
        <li><a href="/about">About</a></li>
        <li><a href="/contact">Contact</a></li>
    </ul>

export default Menu
```

Just generic enough

Hell Yeah, Generic! 😈

- Simple component with few use cases?
- Generic component with tons of use cases and props?

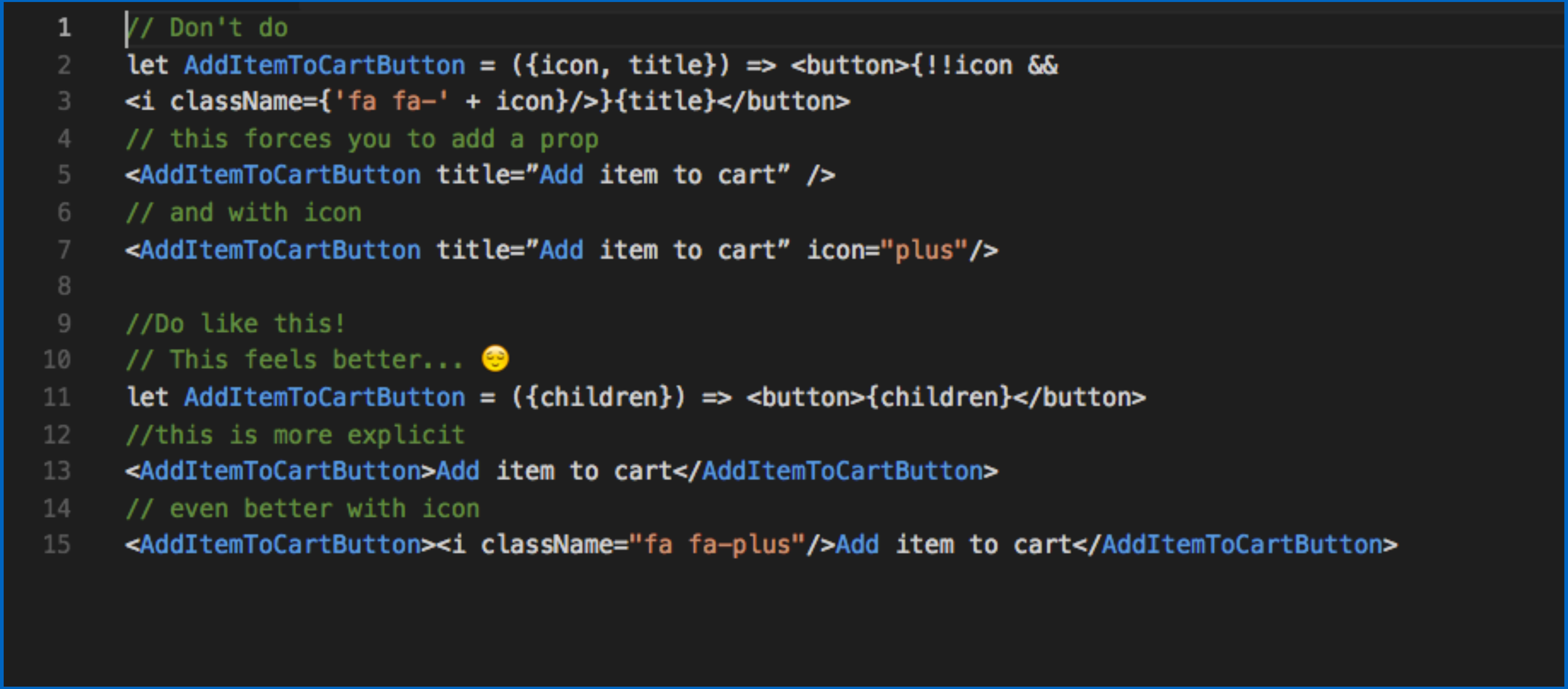
Just generic enough

Hell Yeah, Generic! 😈

Only use external component libraries if you really have to

Just generic enough

Hell Yeah, Generic! 😈



```
// Don't do
let AddItemToCartButton = ({icon, title}) => <button>{!!icon &&
<i className={'fa fa-' + icon}/>}{title}</button>
// this forces you to add a prop
<AddItemToCartButton title="Add item to cart" />
// and with icon
<AddItemToCartButton title="Add item to cart" icon="plus"/>

//Do like this!
// This feels better... 😌
let AddItemToCartButton = ({children}) => <button>{children}</button>
//this is more explicit
<AddItemToCartButton>Add item to cart</AddItemToCartButton>
// even better with icon
<AddItemToCartButton><i className="fa fa-plus"/>Add item to cart</AddItemToCartButton>
```

Football Time ⚽

France 🇫🇷 4 - 3 Argentina 🇦🇷

❤

More is better…Sometimes

Really??? 😮

- Easier to duplicate -> hack, then to reuse?
- Wanna make that reusable component fit into your page?

More is better...Sometimes

Really??? 😮

Top down Approach

Building a library of components

Trying to fit them onto your page

More is better…Sometimes

Really??? 😮

Bottom-up Approach

Build a page first

Extract components only when you need to

Summary

- Start by building the first page in a single component
- Build the second page also in a single component
- If, and only if, there are common components between the two pages, make them generic and extract them to a separate folder.

The Gain…💪

- Code is abstract & DRY
- Reused in multiple apps & projects
- Iteration speed increases
- Less changes of future deprecation
- Appreciation by teammates and developer community
- Can be released as Open Source Software 😍

Quick Links

- 3 Essential Rules Of Reusable Components
- Designing Reusable Components
- How Components Won The Framework Wars

Thank You

Zahin Omar Alwa
@zahinize