Composition

How components in React can be combined to work together.

WE'LL COVER THE FOLLOWING

Why Composition Seems like a Good Idea

One of the biggest benefits of React is composability. I personally don't know of any other framework that offers such an easy way to create and combine components. In this section, we will explore a few composition techniques which proved to work well.

Let's look at a simple example. Let's say that we have an application with a header and we want to place navigation inside. We have three React components - `App`, `Header` and `Navigation`. They have to be nested into each other so we end up with the following dependencies:

```
<App> -> <Header> -> <Navigation>
```

The trivial approach for combining these components is to reference them in the places where we need them as we have done in previous lessons.

```
// app.js
import Header from './Header.js';

export default function App() {
  // 'App' referencing 'Header'
  return <Header />;
```

```
}

// Header.js
import Navigation from './Navigation.js';

export default function Header() {
  // 'Header' referencing 'Navigation'
  return <header> <Navigation /> </header>;
}

// Navigation.js
export default function Navigation() {
  return (<nav> ... </nav>);
}
```

Why it's Not The Best Idea

However, by following this approach, we may encounter a couple of problems:

- **Dependencies:** We may consider the `App` as a place where we do our main composition. The `Header`, however, may have other elements like a logo, search field or a slogan. What if we need the same `Header` component, but without the `Navigation`? We can't easily achieve that because we have the two bound tightly together. It would, therefore, be nice if they could instead somehow be passed from the `App` component so we don't create hard-coded dependencies.
- **It's difficult to test**: We may have some business logic in the `Header` and in order to test it, we have to create an instance of the component. However, because it imports other components we will probably create instances of those components too and it becomes difficult to test. Additionally, we could also end up breaking our `Header` test by doing something wrong in the `Navigation` component, which is misleading. *(Note: to some extent shallow rendering solves this problem by rendering only the `Header` without its nested children.)*

Using React's children API

In React we have the handy children prop. That's how the parent reads/accesses its children. This API will make our Header agnostic and dependency-free:

```
import React from 'react';
```

```
export default function Navigation({ children }) {
  return <p> "Some navigation" </p>;
};
```

Notice also that if we don't use `{ children }` in `Header`, the `Navigation` component will never be rendered.

It now becomes easier to test because we may render the `Header` with an empty `<div>`. This will isolate the component and will let us focus on one piece of our application.

Passing a child as a prop

Every React component receives props. As we mentioned already there is no any strict rule about what these props are. We may even pass other components.

```
import React from 'react';

const Title = function () {
  return <h1>Hello there!</h1>;
}
```

This technique is useful when a component like `Header` needs to take decisions about its children, but don't bother about what they actually are. A simple example is a visibility component that hides its children based on a specific condition.

Quick quiz on React Composition!

1 What is composition in React?

In the following two sections, we will describe in detail two methods of composing components in React.