Babel and React

 (https://github.com/learn-co-curriculum/react-hooks-babel-and-react)  (https://github.com/learn-co-curriculum/react-hooks-babel-and-react/issues/new)

Learning Goals

- Learn what Babel is
- Understand how Babel integrates with React

Overview

In this lesson, we'll unpack what **Babel** brings to the table when developing React applications.

Babel

This is the Tower of Babel. If you don't have time to procrastinate and **read the wiki** (https://en.wikipedia.org/wiki/Tower_of_Babel), and want to get on with learning programming, allow us to provide the **TL;DR** (https://en.wikipedia.org/wiki/TL;DR) and why it is relevant to the Babel tool we use:

The Tower of Babel was a colossal construction project long ago. It was being built by a united humanity speaking the same language, with the intention of reaching such heights that heaven itself could be accessed. While it was being constructed, the God in the story afflicted the united humans by confounding their speech. This ensured the once united humanity could no longer communicate. What made this ambitious project possible was that multiple cultures, languages, idioms, etc. were all using *a common standard*.

As you may already know, JavaScript (based on the ECMAScript [ES] standard) is an evolving language. Over time we have had several iterations that have incorporated additional features and language constructs, such as ES6 arrow functions, class syntax, `let`, and `const`. There are also **new features** (https://github.com/tc39/proposals) being introduced into the language regularly. What was needed was a way to move all the different standards of JavaScript usage to the same standard. **That** is what Babel does — it translates all kinds of new JavaScript features into a common, standard code.

Less metaphorically, Babel gained popularity because it **compiled/transpiled (https://stackoverflow.com/questions/43968748/is-babel-a-compiler-or-transpiler)** newer ES6 syntax and language features into the older (and more widely deployed, at that time) ES5. This was especially important when ES6 came out because many browsers had not yet updated their JavaScript engines to interpret the new language features of ES6.

Nowadays, you are less likely to encounter browsers **not** implementing ES6 syntax. JavaScript is still updated regularly, and most modern browsers do a good job keeping up! If you're ever curious about whether or not a particular feature is supported in a browser, **caniuse.com (https://caniuse.com/)** is a great resource.

Then why is Babel important?

If most popular browsers have moved to integrate modern JavaScript features, then why is Babel "still a thing?"

Babel's competency was in reading in one type of text and making in-place transformations such that another type of text came out. Some developers realized that by processing their code with Babel, they could write code that's terse and convenient and then have Babel turn that code into verbose, compliant JavaScript code.

Let's take as an example how JSX, which is a React specific syntax, can be transformed, via Babel, into valid JavaScript:

```
const profile = (
  <div>
    <img src="avatar.png" className="profile" />
    <h3>{[user.firstName, user.lastName].join(" ")}</h3>
  </div>
);
```

When the above is run through Babel, we receive the following executable code:

```
var profile = React.createElement(
  "div",
  null,
  React.createElement("img", { src: "avatar.png", className: "profile" }),
```

```
  React.createElement("h3", null, [user.firstName, user.lastName].join(" "))
);
```

Don't worry if the syntax above is unfamiliar! What you should take away is that JSX code in the first block (that *HTML-like* syntax) was transformed into valid JavaScript syntax in the second block after Babel had a go at it.

While you don't **strictly** need Babel as a dependency when writing React code, there aren't many (if any) developers who write React applications without JSX (and therefore Babel). My fingers think that typing that first one is better (because they're lazy). My brain also likes that JSX paints an HTML picture in my mind's eye. JSX removes the burden on the programmer of calculating an intermediary picture of the DOM in their brain when reading this code. So, we teach and write using the pre-Babel-compiled (first syntax above) JSX in our React applications.

Not Just For JSX

In addition to the JSX magic it provides, Babel can also compile other features and syntactic sugar that is not yet, or never will be, a part of ECMAScript! One example of this is a Babel plugin that enables the usage of **language features proposed for ECMAScript, but not yet implemented (https://babeljs.io/docs/plugins/preset-stage-2/)** .

Conclusion

Babel enables us to use syntax that browsers won't natively recognize by **pre-compiling** (or *transpiling*) it into syntax that browsers *do* natively recognize. When used with React, this includes (but is not limited to) transpiling JSX.

Resources

- **Babel (http://babeljs.io/)**