

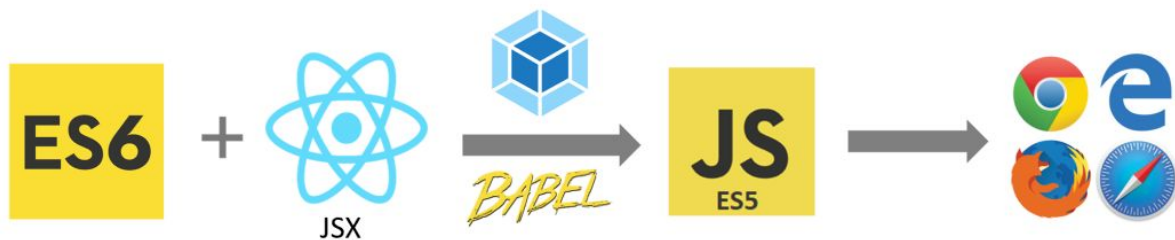


Lesson 3: Transpiling your code

3.1 What does it mean to transpile your code?

Transpiling means “translating” code that’s written in one language (or a version of it) to the equivalent code written in another language (or a different version).

For example, code that’s written in ES6, JSX or in TypeScript can be transpiled to ES5, which is understood and supported by all browsers.




In this tutorial, we’ll use **Babel** as transpiler or source-to-source compiler.

3.2 Why do you need a transpiler?

A logical question at this step is why use a transpiler if you're writing your own code and if it's all written in JavaScript? Why complicate things?

Although technically you're correct and all the code *is* written in JavaScript, not everyone uses the same JS version and syntax.

Here's an example: a function written in ES5 syntax, followed by an ES6 arrow function.

Docs Setup Try it out Videos Blog 🔍 Search

```
1 var greetings_es5 = function greetings (name) {  
2   return 'hello ' + name  
3 }  
4  
5 const greetings_es6 = (name) => {  
6   return `hello ${name}`;  
7 }  
  
1 "use strict";  
2  
3 var greetings_es5 = function greetings(name) {  
4   return 'hello ' + name;  
5 };  
6  
7 var greetings_es6 = function greetings_es6(name) {  
8   return "hello ".concat(name);  
9 };
```

3.2 Why do you need a transpiler?

As a professional developer, you'll rarely start a project from scratch. In most cases you'll join an existing project and you'll work in a mixed team.

Some of your colleagues might be using older syntax or features that aren't yet supported by all browsers. To make sure your code doesn't break when it's pushed to production and to keep it consistent, you'll use a transpiler.

Adding a transpiler to your project offers a couple of advantages:

- It helps browsers understand your code, regardless of the syntax you use
- It allows you to use your preferred language and syntax so that you can work more efficiently
- It gives your team freedom to use the syntax they're comfortable with

Now let's go ahead and install Babel!

3.3 Install Babel for JS transpiling

If we wouldn't have npm installed, we would have to manually go to Babel's website, download the tool, install and configure it.

However, with npm, you can just type in the terminal:

```
npm install --save-dev @babel/cli@7.8.4 @babel/core@7.0.0 @babel/preset-env@7.0.0  
@babel/register@7.0.0 @babel/node@7.0.0
```

What does this all mean?

When you type this command, npm will install on your machine whatever packages you tell it to.

3.3 Install Babel for JS transpiling

In this case, we're installing the following packages:

- `save-dev`: this means that all packages are installed as development dependencies, not as project dependencies
- `babel/cli`: this allows us to run babel directly from the terminal
- `babel/core`: this is the core functionality of Babel
- `babel/preset-env`: this allows you to use the latest JS syntax
- `babel/register`: this is needed for Babel to work properly
- `babel/node`: this compiles presets and plugins

3.3 Install Babel for JS transpiling

If you open now the package.json file, you'll see that all these packages have been added to dependencies.

```
"devDependencies": {  
  "@babel/cli": "^7.8.4",  
  "@babel/core": "^7.0.0",  
  "@babel/node": "^7.0.0",  
  "@babel/preset-env": "^7.0.0",  
  "@babel/register": "^7.0.0",  
}
```

3.4 Configure Babel

In order for Babel to actually do its work, we need to do one more step, and that's creating a configuration file.

In the root folder of your project, create a file called `.babelrc`, and add the following code:

```
{  
  "presets": ["@babel/preset-env"],  
}
```

The presets and plugins that you add in this file will be loaded by Babel when transpiling the code. Plugins aren't new, and presets are nothing but sets of plugins needed for supporting certain language features.

3.4 Configure Babel

If you want to run Babel and transpile the code, you have to type in the terminal the following command:

```
npm run babel-node
```

This is a little impractical though, as you want the transpiling to happen automatically when you start your project or prepare the production build.

In order to avoid the repetitive tasks, we can automate the transpiling process by adding a script in the `package.json` file.

```
"scripts": {  
  "start": "babel-node ./server.js"  
},
```

3.4 Configure Babel

As we go through this tutorial, we'll add more npm scripts to the package.json file.

These scripts will help us run all the processes automatically - transpiling, linting, bundling, minifying - every time we initiate the project with a command like *npm run start*.

For example, for automating the production build and the pre-production linting, we'll add the following scripts:

```
"scripts": {  
  "prebuild": "npm-run-all clean-dist lint",  
  "build": "babel-node ./build.js"  
}
```

We'll discuss these more in detail in the coming lessons!