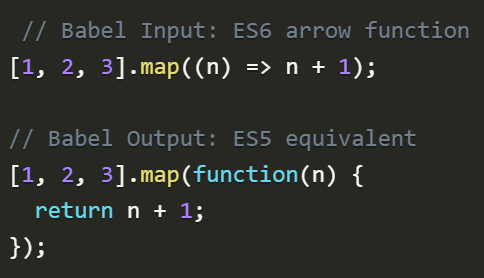
BABEL

Babel es usado principalmente para convertir código ES6+ en una versión compatible con navegadores y ambientes tanto actuales como anteriores.



Podemos empezar instalando dos librerías: @babel/core y @babel/cli. La primera es el core o núcleo, aquí se encuentra la principal funcionalidad de Babel. La segunda librería nos permite usar Babel desde la terminal. Ambas se pueden instalar con npm.

npm install --save-dev @babel/core @babel/cli

npx babel src --out-dir dist

Gracias a @babel/cli el comando anterior funciona y lo que hace es tomar todos los archivos de JavaScript que encuentre en la carpeta src, aplicar las transformaciones que hayamos especificado y colocar los archivos resultantes en la carpeta dist.

En este caso no hemos definido alguna transformación por lo que el código resultante será igual al que se encuentra en src.

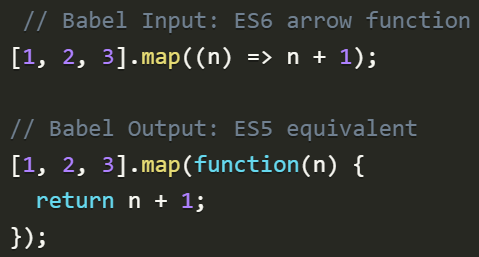
PLUGINS

Las transformaciones de código que se mencionaron anteriormente se realizan a través de plugins. Cada plugin le dice a babel cómo transformar el código. Por ejemplo, el plugin @babel/plugin-transform-arrow-functions permite transformar arrow functions.

npm install --save-dev @babel/plugin-transform-arrow-functions

npx babel src --out-dir dist --plugins=@babel/plugin-transform-arrow-functions

Con la opción --plugins definimos los plugins que queremos usar.



En esta ocasión el compilado en la carpeta dist habrá hecho la transformación del arrow function. Podemos usar varios plugins para transformar distintos features de ES6+.

PRESETS

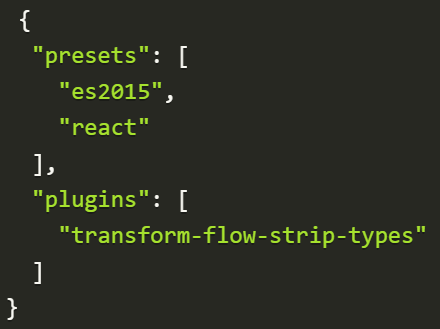
En lugar de agregar todos los plugins que queremos implementar uno por uno podemos usar presets. Estos son conjuntos ya preestablecidos de ciertos plugins. Por ejemplo, el preset @babel/preset-env contiene todos los plugins necesarios para soportar las versiones modernas de JavaScript. La instalación y el uso es igual que los plugins.

npm install --save-dev @babel/preset-env

npx babel src --out-dir dist --presets=@babel/preset-env

Con la opción --presets definimos el preset que queremos usar. Para el ejemplo anterior el resultado es el mismo. La diferencia está en que usando el plugin @babel/plugin-transform-arrow-functions babel solo va a transformar arrow functions, mientras que usando el preset @babel/preset-env babel podrá transformar la mayoría de características que vienen con ES6+.

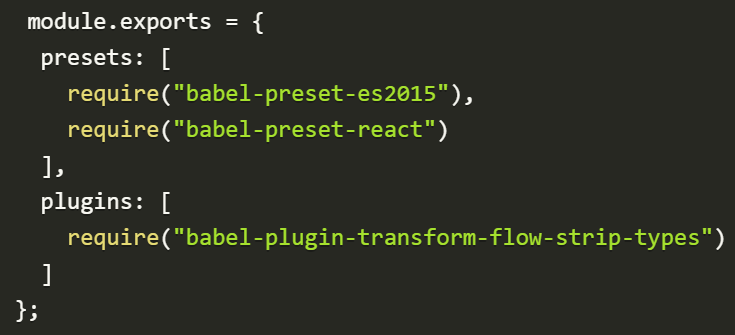
Babel es tan flexible que nos permite crear nuestros propios presets con los plugins que necesitemos e incluso otros presets.



Ahora debemos crear un nuevo proyecto siguiendo la convención de nombres babel-preset-\* y colocar dos archivos. El primero es un nuevo package.json donde vamos a definir las dependencias del preset.



El segundo archivo que necesitamos es un index.js donde vamos a exportar el contenido del archivo de configuración reemplazando los strings del plugin/preset por require.



Después de publicarlo en npm se puede usar como cualquier otro preset.

BABEL POSTWORK

Babel contiene un módulo @babel/polyfill el cual incluye core-js y regenerator-runtime para emular un ambiente ES2015+. Esto significa que se pueden usar nuevos built-ins como Promise o WeakMap, métodos estáticos como Array.from o Object.assign, métodos de instancia como Array.prototype.includes, entre otras características. Este polyfill se agrega tanto al scope global como los prototype nativos (como String) para poder trabajar.

npm install --save-dev core-js regenerator-runtime

En vez de configurar @babel/polyfill sólo debemos importar las librerías de la siguiente forma:



Si intentaste trabajar con generators o funciones async seguramente te habrás dado cuenta que Babel estaba teniendo problemas para compilar el código correctamente, después de instalar y usar tanto core-js como regenerator-runtime no deberías de tener problemas de nuevo.