Adding Webpack

In this lesson, we'll continue to set up our React and TypeScript project by adding a tool that will bundle the code together.

WE'LL COVER THE FOLLOWING

- Installing Webpack
- Installing Webpack Babel and ESLint plugins
- Enabling a TypeScript based Webpack configuration
- Configuring Webpack
- Adding type checking into the Webpack process
- Wrap up

Installing Webpack

Webpack is a popular tool that we can use to bundle all our JavaScript code into the bundle.js file that our index.html is expecting. Let's install the core webpack library as well as its command-line interface:

```
npm install --save-dev webpack webpack-cli @types/webpack
```

Webpack has a web server that we will use during development. Let's install this:

```
npm install --save-dev webpack-dev-server @types/webpack-dev-server
```

Why do you think we installed these Webpack dependencies as development dependencies?



Installing Webpack Babel and ESLint plugins

We need a webpack plugin, babel-loader, to allow Babel to transpile the React and TypeScript code into JavaScript. Let's install this:

```
npm install --save-dev babel-loader
```

Enabling a TypeScript based Webpack configuration

The Webpack configuration file is JavaScript based as standard. However, we can use TypeScript if we install a package called ts-node. Let's install this:

```
npm install --save-dev ts-node
```

Configuring Webpack

Webpack is configured using a file called webpack.config.ts. Let's create this in the project's root directory with the following content:

```
import path from 'path';
import webpack from 'webpack';
const config: webpack.Configuration = {
  entry: './src/index.tsx',
  module: {
    rules: [
        test: /\.(ts|js)x?$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-env', '@babel/preset-react', '@babel/
preset-typescript'],
          },
        },
      },
    ],
  },
  resolve: {
    extensions: ['.tsx', '.ts', '.js'],
```

```
},
output: {

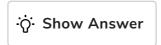
   path: path.resolve(__dirname, 'build'),
   filename: 'bundle.js',
},
devServer: {
   contentBase: path.join(__dirname, 'build'),
   compress: true,
   port: 4000,
},
};
export default config;
```

Here are the critical bits in this configuration file:

- The entry field tells Webpack where to start looking for modules to bundle. In our project, this is index.tsx.
- The module field tells Webpack how different modules will be treated.

 Our project is telling Webpack to use the babel-loader plugin to process files with .js, .ts, and .tsx extensions.
- The resolve.extensions field tells Webpack what file types to look for in which order during module resolution.
- The output field tells Webpack where to bundle our code. In our project, this is the file called bundle.js in the build folder.
- The devServer field configures the Webpack development server. We are telling it that the root of the web server is the build folder, and to serve files on port 4000.

With the above configuration, what address in the browser will run the app?



If we wanted the bundled file to be called dist.js, what would we change in the Webpack configuration?



Will any TypeScript type checking occur with the above configuration?



Adding type checking into the Webpack process

We can use a package called <code>fork-ts-checker-webpack-plugin</code> to enable the Webpack process to type check the code. This means that Webpack will inform us of any type errors as well as our code editor. Let's install this package:

```
npm install --save-dev fork-ts-checker-webpack-plugin @types/fork-ts-check
er-webpack-plugin
```

Let's add this to webpack.config.ts:

```
import ForkTsCheckerWebpackPlugin from 'fork-ts-checker-webpack-plugin';

const config: webpack.Configuration = {
    ...,
    plugins: [
        new ForkTsCheckerWebpackPlugin({
            async: false,
            eslint: true,
        }),
    ],
};
```

Here's an explanation of these ForkTsCheckerWebpackPlugin settings:

- async: We have set this to false so that Webpack waits for the type checking process to finish before it emits any code.
- eslint: We have set this to true so that Webpack informs us of any linting errors.

Wrap up

The project is nearly set up now. The last thing we need to do is add a couple

of npm scripts to do some important tasks. We'll do this in the next lesso	n.