Step 2: Initialize your Application and Configure Webpack

To continue with the setup, you have to have nodejs installed on your machine. If you haven’t installed node, visit nodejs.org, and follow the installation instructions there.

Execute

```
npm init
```

inside the folder of your application. Provide the necessary details about your application by answering the questions in the terminal. Once you are done answering or skipping these questions, a `package.json` file is created in your folder.

Webpack will be our default tool for bundling our packages for distribution, and managing dependencies. Install Webpack as a local dependency:

```
npm install webpack --save-dev
```

Npm package installations can be global or local. In case of global installations, you may have to run the command as admin, using `sudo` in Linux and Mac, or you may have to run your command line as an administrator in Windows.

Note that it is rarely a good idea to install global packages, as you have no control over the runtime environment of other people running your code. If you work in a team, I suggest installing every npm dependency locally. If you prefer a global installation for experimenting, feel free to do so.

Webpack does not support the ES6 syntax by default. We will use a module loader called babel-loader. You can install it using loader called babel-loader. You can install it using `npm`.

```
npm install babel-loader babel-core  --save-dev
npm install babel-preset-es2015 babel-preset-stage-2 --save-dev
```

We have installed the following dev dependencies:

- `babel-loader`: babel module loader,
- `babel-core`: the core transpiler used for transpiling JavaScript,
- `babel-preset-es2015`: ES6/ES2015 support,
- `babel-preset-stage-2`: babel preset for some extra features such as destructuring.

Now that Webpack is available, let's create a configuration file `webpack.config.js`.

```
module.exports = {
    entry  : './src/main.js',
    output : {
        path     : __dirname,
        filename : 'myaccount.dist.js'
    },
    module : {
        loaders: [ {
                test   : /.js$/,
                loader : 'babel-loader',
                exclude: /node_modules/,
                query: {
                    presets: [
                        'es2015',
                        'stage-2'
                    ]
                }
            }
        ]
    }
};
```

Note that in Webpack 4, you have to write `rules` instead of `loaders` in the `module` property:

```
module.exports = {
    entry  : './src/main.js',
    output : {
        path     : __dirname,
        filename : 'myaccount.dist.js'
    },
    module : {
        rules: [ {
                test   : /.js$/,
                loader : 'babel-loader',

                exclude: /node_modules/,
                query: {
                    presets: [
                        'es2015',
                        'stage-2'
                    ]
                }
            }
        ]
    }
};
```

Our entry point `src/main.js` will be created shortly. All dependencies specified in this file will be copied to `myaccount.dist.js` recursively. Babel-loader is also invoked, transpiling all Javascript files it finds.