Chapter 25: React, Webpack & TypeScript installation

Section 25.1: webpack.config.js

```
module.exports = {
    entry: './src/index',
    output: {
        path: __dirname + '/build',
        filename: 'bundle.js'
    },
    module: {
        rules: [{
            test: /\.tsx?$/,
            loader: 'ts-loader',
            exclude: /node_modules/
        }]
    },
    resolve: {
        extensions: ['.ts', '.tsx']
    }
};
```

The main components are (in addition to the standard `entry`, `output` and other webpack properties):

The loader

For this you need to create a rule that tests for the `.ts` and `.tsx` file extensions, specify `ts-loader` as the loader.

Resolve TS extensions

You also need to add the `.ts` and `.tsx` extensions in the `resolve` array, or webpack won't see them.

Section 25.2: tsconfig.json

This is a minimal tsconfig to get you up and running.

```
{
    "include": [
        "src/*"
    ],
    "compilerOptions": {
        "target": "es5",
        "jsx": "react",
        "allowSyntheticDefaultImports": true
    }
}
```

Let's go through the properties one by one:

`include`

This is an array of source code. Here we have only one entry, `src/*`, which specifies that everything in the `src` directory is to be included in compilation.

`compilerOptions.target`

Specifies that we want to compile to ES5 target

`compilerOptions.jsx`

Setting this to **`true`** will make TypeScript automatically compile your tsx syntax from **`<div />`** to `React.createElement("div")`.

`compilerOptions.allowSyntheticDefaultImports`

Handy property which will allow you to import node modules as if they are ES6 modules, so instead of doing

```
import * as React from 'react'
const { Component } = React
```

you can just do

```
import React, { Component } from 'react'
```

without any errors telling you that React has no default export.

Section 25.3: My First Component

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';


interface AppProps {
    name: string;
}
interface AppState {
    words: string[];
}

class App extends Component<AppProps, AppState> {
    constructor() {
        super();
        this.state = {
            words: ['foo', 'bar']
        };
    }

    render() {
        const { name } = this.props;
        return (<h1>Hello {name}!</h1>);
    }
}

const root = document.getElementById('root');
ReactDOM.render(<App name="Foo Bar" />, root);
```

When using TypeScript with React, once you've downloaded the React DefinitelyTyped type definitions (`npm install --save @types/react`), every component will require you to add type annotations.

You do this like so:

```
class App extends Component<AppProps, AppState> { }
```

where `AppProps` and `AppState` are interfaces (or type aliases) for your components' props and state respectively.