**ASP.Net MVC CRUD Operation:** Followed the below link:

<https://www.youtube.com/watch?v=_uSw8sh7xKs&t=1684s>

# Package Manager Console:

1. EntityFrameworkCore\Add-Migration "InitialMigration"
2. EntityFrameworkCore\Update-Database

For Sample React Application: Followed the course in Educative:

<https://www.educative.io/courses/building-web-applications-with-react-and-aspnet-core/creating-a-react-and-typescript-app>

**Creating a React and TypeScript App**

Learn to create a React and TypeScript app.

Previously, we discovered that **create- react-app (CRA)** was leveraged by the Visual Studio template to create the React app. We also learned that CRA did a lot of valuable setup and configuration for us. We are going to leverage CRA in this section to create our React app. CRA is a package in the npm registry that we will execute to scaffold a React and TypeScript project. First, we will take the time to understand the benefits of using TypeScript.

**Understanding the benefits of TypeScript**

TypeScript adds an optional static typing layer on top of JavaScript that we can use during our development. Static types allow us to catch certain problems earlier in the development process. For example, if we make a mistake when referencing a variable, TypeScript will spot this immediately once we’ve mistyped the variable, as shown in the following screenshot:

Press

+

to interact

TypeScript catching an unknown variable

Another example is that, if we forget to pass a required property when referencing a React component, TypeScript informs us of the mistake straight away:

Press

+

to interact

TypeScript catching a missing React component property

This means we get a build-time error rather than a runtime error.

This also helps tools such as Visual Studio Code provide accurate IntelliSense; robust refactoring features, such as renaming a class; and great code navigation.

As we start building our frontend, we’ll quickly experience the types of benefits that make us more productive.

Now that we are starting to understand the benefits of TypeScript, it’s time to create a React project that uses TypeScript in the next subsection.

**Creating the app with CRA**

Let’s create the React and TypeScript app with CRA by carrying out the following steps:

1. Open Visual Studio Code in the “QandA” folder we created earlier. Note that we should be at the same level as the backend folder and not inside it.
2. Open the Terminal in Visual Studio Code, which can be found in the “View” menu by pressing “Ctrl + ' ”. Execute the following command in the Terminal:

*npx create-react-app frontend --template typescript*

The npx tool is part of npm that temporarily installs the create-react-app npm package and uses it to create our project.

We have told the create-react-app npm package to create our project in a folder called frontend.

The --template typescript option has created our React project with TypeScript.

1. If we look in the “src” folder, we’ll see that the App component has a tsx extension. This means that this is a TypeScript component.
2. Let’s verify whether the app runs okay by executing the following commands in the terminal:

*cd frontend*

*npm start*

1. The app will appear in our browser after a few seconds

**Adding Linting to React and TypeScript:** <https://www.educative.io/courses/building-web-applications-with-react-and-aspnet-core/adding-linting-to-react-and-typescript>

Learn to add linting to React and TypeScript in this lesson.

Linting is a series of checks that are used to identify code that is potentially problematic. A linter is a tool that performs linting, and it can be run in our code editor as well as in the **continuous integration (CI)** process. So, linting helps us write consistent and high- quality code as it is being written.

ESLint is the most popular linter in the React community and has already been installed in our project for us by CRA. Due to this, we will be using ESLint as our linting tool for our app.

**Note:** TSLint was a popular alternative to ESLint for linting TypeScript code but is now deprecated.

In the following subsections, we will learn how to configure ESLints rules, as well as how to configure Visual Studio Code to highlight violations.

**Configuring Visual Studio Code to lint TypeScript code**

CRA has already installed ESLint and configured it for us.

**Note:** Note that ESLint doesn’t appear in our package.json file. Instead, it is part of the CRA package. This can be confirmed by opening the package.json file in node\_modules\react-scripts.

We need to tell Visual Studio Code to lint TypeScript code. Let’s carry out the following steps to do this:

1. First, let’s reopen Visual Studio Code in the “frontend” folder. This is required for an extension that we are going to install in a later step.
2. Go to the “Extensions” area in Visual Studio Code (“Ctrl + Shift + X”) and type “eslint” into the search box in the top-left corner. The extension we are looking for is called “ESLint” and is published by Dirk Baeumer:

**Configuring linting rules**

Now that Visual Studio Code is linting our code, let’s carry out the following steps to understand how we can configure the rules that ESLint executes:

1. Let’s create a file called .eslintrc.json in the “frontend” folder with the following code:

*{*

*"extends": "react-app"*

*}*

Implementing code in .eslintrc.json file

This file defines the rules that ESLint executes. We have just told it to execute all the rules that have been configured in CRA.