# **React Basics Lab Instructions**

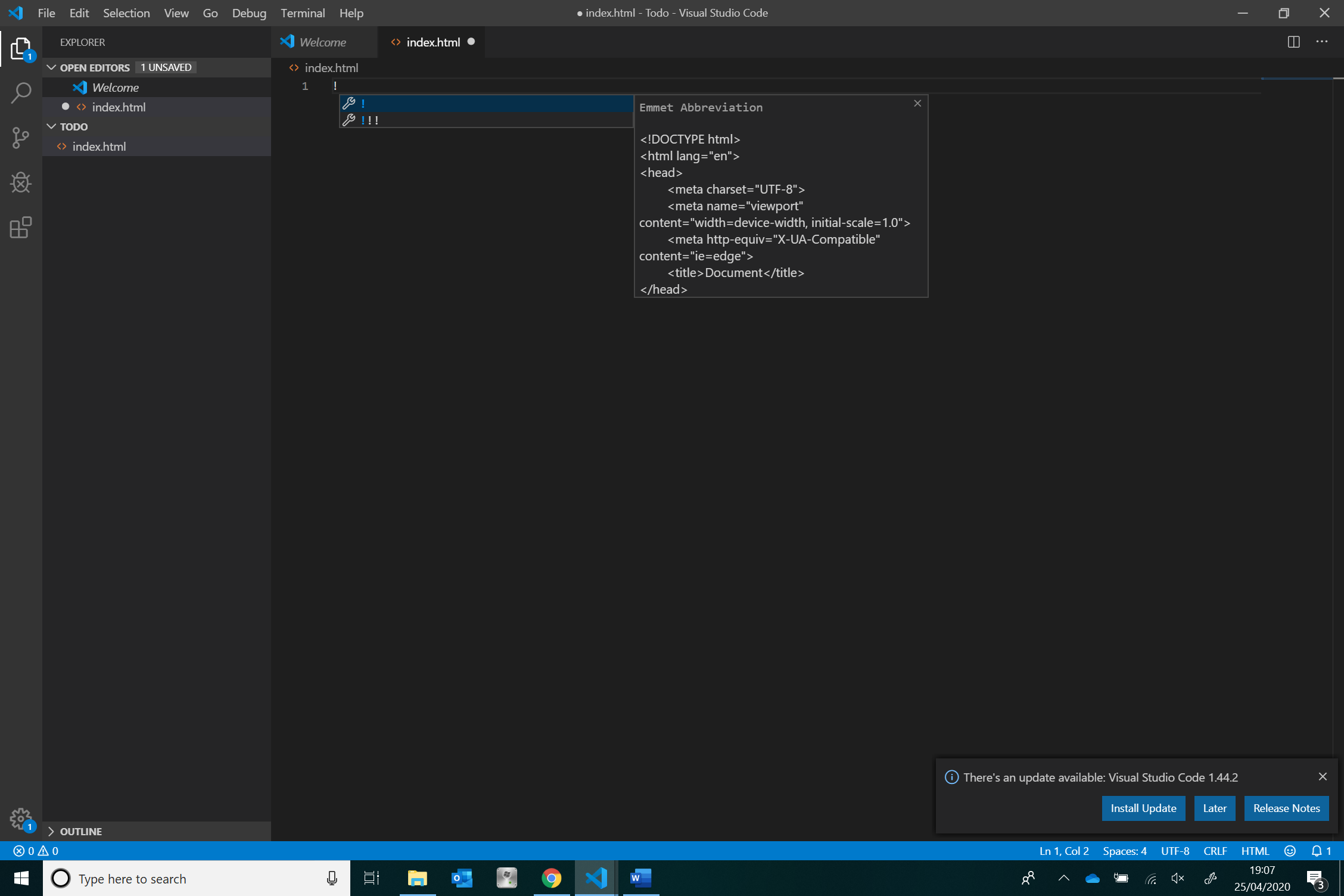
This document provides step by step procedure for quickly setting up a React project and creating a simple Todo List application. The instructions are divided into sections and the end of each section is considered a checkpoint where you can compare your code up to that point with the expected code (which can be downloaded from blackboard). For example, after completing the instructions in Section 2, you can compare your project with the project in section2.zip on blackboard which should contain the result of following all the instructions from the start of the lab instructions up to the end of Section 2.

## **Section 1: Setting Up**

1. Create an empty folder and name it Todo.
2. Open VSCode, and from the ‘File’ menu select ‘Open Folder’. Navigate to the location of your ‘Todo’ folder in the opened file explorer window, select the folder and click ‘Select Folder’.
3. Using the file explorer on the left-hand side of VSCode, create a new file in your root project folder and name it ‘index.html’.



1. Open up ‘index.html’ and add a html document template by typing ‘!’ and pressing enter when VSCode gives you the option on add a template.



1. Change the title of the html document from ‘Document’ to ‘Todo’ by modifying the title tag in the head of the document.
2. Add the two script tags below for loading React and ReactDom in the head section of your html document.

<script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>

<script  src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>

1. Add one final script tag in the head section of your html document for loading Babel, which will let us use JSX.

<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>

1. Everything is now set for you to start wiring React code. To test if its all working, lets create and display a simple component that returns a heading with the text ‘Todo App’
2. Add a div tag in the body of your html document, give it an id of ‘root’. We will be rendering our React app in this div.

<body>

    <div id="root">

    </div>

</body>

1. Below the div element created in step 9, create a script for writing our React app. We will be using JSX, and JSX requires babel (which we imported in step 7), therefore have to specify ‘text/babel’ as the value for the script’s ‘type’ attribute.

<body>

    <div id="root">

    </div>

    <script type='text/babel'>

    </script>

</body>

1. At this point we want our React app to just render the heading ‘Todo App’ i.e. the component <h1> Todo App </h1>. To do this, we use ReactDOM’s ‘render’ method, which takes as its parameters the component to render and the location for it to be rendered which in our case is the div with id ‘root’.

<body>

    <div id="root">

    </div>

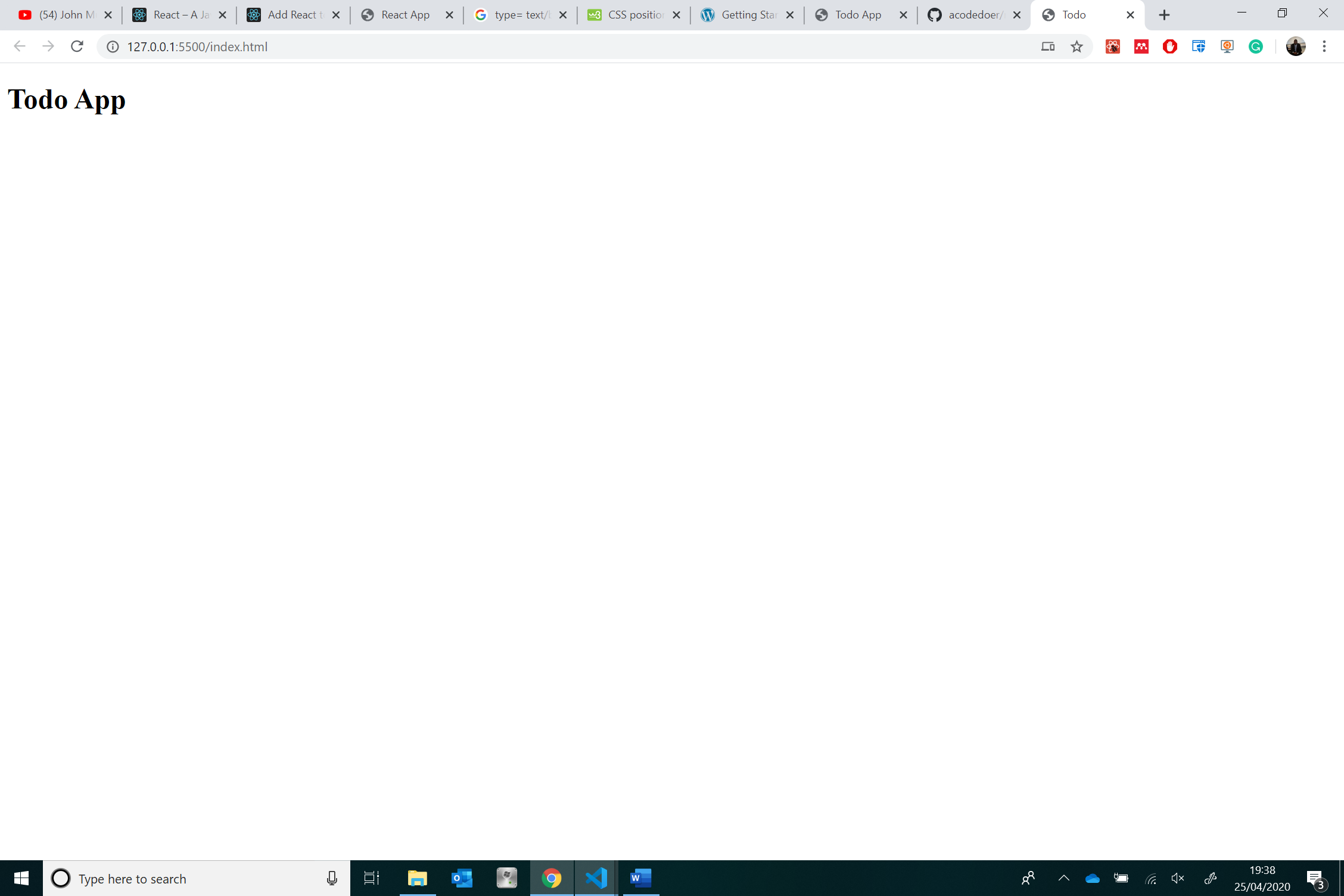
    <script type='text/babel'>

        ReactDOM.render(<h1>Todo App</h1>,document.getElementById("root"))

    </script>

</body>

1. Run your project using VSCodes LiveServer, this should open up a browser window that looks like the image below.



1. Now that we have React setup, we can move to writing more complex components. If you are having difficulties completing this section, you can have a look at this sections checkpoint source code on blackboard.