Building Web Applications using ReactJS

QUICKLABS GUIDE

CONTENTS

QuickLabs Environment Set-Up

Code Editing

1. Open **VSCode** (or download and install if not present).
    - Use the *desktop shortcut* to open the **VSCode** download page:
    - For **Windows** users download the **64-bit System Installer**.
2. Check for *updates* and download and install if necessary:
    - For **Windows** Users click **Help - Check for updates**;
    - For **MacOS** Users click **Code - Check for updates**.
3. Using **File - Open**, navigate to the **QuickLabs** folder and click **Open**. This will give you access to all of the **QuickLab** files and solutions needed to complete the **QuickLabs**.

NodeJS

1. Use the *desktop shortcut* to open the **NodeJS** download page.
2. Download and install the **LTS** version for the operating system you are working in:
    - For **Windows** users, download the **Installer file (.msi)**;
    - For **MacOS** users, download the **Installer file (.pkg)**.

Do This Before Each QuickLab

Unless specifically directed to do otherwise, the following steps should be taken before starting each QuickLab:

1. Point the terminal/command line at the QuickLab **starter** folder that contains the **package.json**.
2. Run the command:

```
npm i
```

3. Compile and output the project by running the command:

```
npm start
```

4. Navigate the browser to:

https://localhost:3000/

if it does not open automatically.

Code Snippets

Whilst the QuickLabs provided are supposed to challenge you, they are not supposed to baffle! Annotated snippets for each instruction are provided at the end of each QuickLab. Try not to use them until you have had a go at writing the code yourself!

Quick Lab 1 – Get a ReactJS App Up and Running

Objectives

- To be able to use the create-react-app node package extractor to quickly scaffold a ReactJS application
- To be able to launch the application in the browser using the command line

Overview

In this Quicklab, you will set up a ReactJS application using a special node package extractor called create-react-app. Once the installation of files has completed, you will launch the application in the browser and see it running.

Activity

Skip the 'Before Each QuickLab' for this Activity.

1. Using **CTRL + '** on the keyboard (**CTRL +** ` on MacOS) or by using click-path **View – Terminal** (or **Terminal – New Terminal** on **MacOS**), open **VSCode's** integrated terminal or click the terminal icon on the bottom bar.

2. Using the `cd` command, navigate to the **QuickLabs/a-react-app/ folder**.
3. Create a *new* ReactJS application using the command:

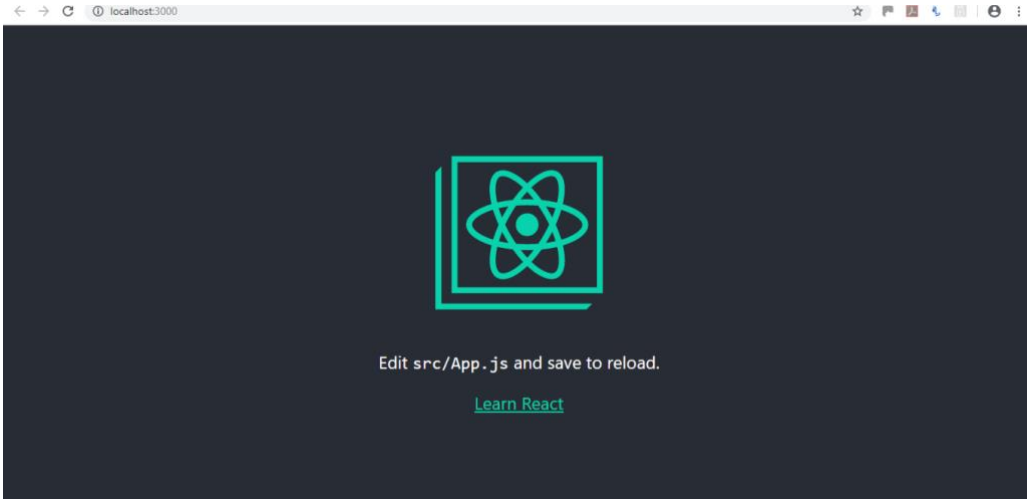
```
npx create-react-app starter
```

Wait for the installation to complete.

4. Use the `cd` command to change into the **starter** folder and then run the application by using the command:

```
cd starter
npm start
```

Your browser should open at *http://localhost:3000* with the following screen:



This is the end of Quick Lab 1

Quick Lab 2 – Build and Run the Application

Objectives

- To be able to build production-ready code using the scripts provided in the application
- To understand what the build process creates and where the files are put

Overview

In this QuickLab, you will produce a production-ready set of code for the skeleton application. This will make bundles of the HTML, CSS and JavaScript needed to efficiently deploy the application. You will explore the files that are created and view the application in the browser.

Activity

Skip the section 'Before Each QuickLab' before continuing using the a-react-app/starter folder.

1. In VSCode, if the server is running on the command line, press **CTRL+C** to stop it.
2. Make sure that the command line is pointing to **QuickLabs/a-react-app/starter**.
3. Make a production ready version of the application by typing:

```
npm run build
```

4. Once the process has finished, install a server to view the application:

```
npm i -g serve
```

5. Once the installation is complete, run the app in the server:

```
serve -s build
```

6. Open the browser to *http://localhost:5000* and view the application.
7. Browse the files created in a new folder called **build** in the application root:
    - Find **index.html** and its reference to the JS files;
    - Find the JavaScript files – view these in **VSCode**.

Building the application optimises the files for the fastest download without affecting functionality.

This is the end of Quick Lab 2

Quick Lab 3 – Create a Function Component

Objectives

- To be able to create Function components
- To be able render components as children of others

Overview

In this QuickLab, you will create Function components in their own files. You will then import these components into parent components, rendering as part of the parent's return.

Activity – Part 1 – MyComponent

Skip the section 'Before Each QuickLab' before continuing using the a-react-app/starter folder.

1. Create a new file called **MyComponent.jsx** in the **a-react-app/starter/src** folder.
2. Add an `import` for `React` from `'react'`.
3. Create a `const` called `MyComponent` as an *arrow function* that takes *no arguments*.
4. Make the function `return` a single `<h1>` with the text `Hello World`.
5. `export` `MyComponent` as a `default`.
6. Open **App.js** from the same folder and *delete EVERYTHING* in its `return`.
7. Put `MyComponent` as an *element* in the `return`, ensuring that it is *imported*.
8. Save all files and run the application – use `npm start` from the command line if required.

The app's display should now have been replaced with the content provided in **MyComponent**.

Activity – Part 2 – AnotherComponent

1. Create a new file called **AnotherComponent.jsx** in the **a-react-app/starter/src** folder.
2. Add an `import` for `React` from `'react'`.
3. Create a `const` called `AnotherComponent` as an *arrow function* that takes *no arguments*.
4. Make the function `return` a **React Fragment** `<></>` with 2 that contain some text – we used 10 'lorem ipsum' words.
5. `export` `MyComponent` as a `default`.
6. Open **MyComponent.jsx** and add `<AnotherComponent />` *under* the `<h1>` and wrap both in a React Fragment `<></>` (ensuring **AnotherComponent** is *imported*).
7. Save all files and run the application (**npm start** from the command line).

You should see the text from **AnotherComponent** seamlessly displayed.

Code Snippets

Part 1 – MyComponent.jsx

```
import React from 'react';                                  // 2

const MyComponent = () => {                                 // 3
  return (                                                  // 4
    <h1>Hello World</h1>
  );
};

export default MyComponent;                                 // 5
```

Part 1 – App.jsx

```
import React from 'react';
import MyComponent from './MyComponent';                    // 7

function App() {
    return (
        <MyComponent />                                     // 7
    );
}

export default App;
```

Part 2 – AnotherComponent.jsx

```
import React from 'react';                                  // 2

const AnotherComponent = () => {                            // 3
  return (                                                  // 4
    <>
      <p>lorem ispum…</p>
      <p>lorem ipsum…</p>
    </>
  );
};

export default MyComponent;                                 // 5
```

MyComponent.jsx

```
import React from 'react';
import AnotherComponent from './AnotherComponent';          // 6

const MyComponent = () => {
  return (                                                  // 6
    <>
      <h1>Hello World</h1>
      <AnotherComponent />
    </>
  );
};

export default MyComponent;
```

This is the end of Quick Lab 3

Quick Lab 4 - Creating a Class Component

Objectives

- To be able to create a Class component
- To be able to nest components in others

Overview

In this QuickLab, you will create a new Angular Component using the CLI, exploring the files that are created and modified as part of the process. You will then nest this new component in the existing App component.

Activity

Skip the section 'Before Each QuickLab' before continuing using the a-react-app/starter folder.

1. Create a new file called **MyClassComponent.jsx** in the **a-react-app/starter/src** folder.
2. Add an `import` for `React` and `{ Component }` from `'react'`.
3. Create a `class` called `MyClassComponent` that `extends` `Component`.
4. Make the `render` function `return` a **React Fragment** `<></>` with a `<h2>` and a `<p>` that contains some text.
5. `export` the `MyClassComponent` as a `default`.
6. Open **MyComponent.jsx** and add `<MyClassComponent />` under the `<AnotherComponent />` (ensuring **MyClassComponent** is imported).
7. Save all files and run the application (**npm start** from the command line if not running already)

You should see the **MyClassComponent** seamlessly displayed with all of the others.

Code Snippets

MyClassComponent.jsx

```
import React, { Component } from 'react';                          // 2

class MyClassComponent extends Component {                         // 3
  render () {                                                      // 4
    return (
      <>
        <h2>Class Components</h2>
        <p>Work in a similar way to Function Components</p>
      </>
    );
  }
}

export default MyClassComponent;                                   // 5
```

MyComponent.jsx

```
import React from 'react';
import AnotherComponent from './AnotherComponent';
import MyClassComponent from './MyClassComponent';                 // 6

const MyComponent = () => {
  return (
    <>
      <h1>Hello World</h1>
      <AnotherComponent />
      <MyClassComponent />                                         // 6
    </>
  );
};

export default MyComponent;
```

This is the end of Quick Lab 4

Quick Lab 6 – Thinking in React Part 1 – Component Hierarchy

Objectives

- To be able to take acceptance criteria, a mock-up and some static data to produce a suitable component hierarchy for an application

Overview

In this QuickLab, you will use acceptance criteria, the provided mock-ups and static data to identify a component hierarchy for a Todo application. A hierarchy is needed for an ‘AllTodos’ UI and an ‘Add/Edit Todo’ UI.

Activity

Skip the 'Before Each QuickLab' for this Activity.

1. Using the information provided below, create a component hierarchy for the ‘AllTodos’ UI and the ‘Add/Edit Todo’ UI.

Acceptance Criteria – ALL UIs

Footer	Header
Acceptance Criteria Delete 0% ☐ The UI should be created using ReactJS and JSX ☐ Every UI should have a <footer> ☐ The <footer> section should use the Bootstrap CSS classes to automatically set the top margin and left and right padding to 1rem ☐ The text in the footer should be centered using the "container" and "align-center" Bootstrap CSS classes You have unsaved edits on this field. View edits - Discard ☐ The 'footer' should contain the text '© QA Ltd 2019-'	Acceptance Criteria Delete 0% ☐ The UI should be created using ReactJS and JSX ☐ Every UI should have a <header> which wraps a <nav> with Bootstrap classes to set it as a navbar that is responsive and stacks the navigation on small screens ☐ The <nav> should contain the QA Logo having alt text of "QA Ltd" and a width of 100 ☐ The logo should be linked to https://www.qa.com, opening in a new window. The link should use Bootstrap class to signify that this is the brand and have rel properties of noopener and noreferrer. ☐ The <nav> section should contain the title 'Todo App' in a <h1> that is linked to / in an <a> that has a Bootstrap class to signify that this is for the brand

Acceptance Criteria – Specific UIs

All Todos	Add/Edit Todos
Acceptance Criteria Delete 0% ☐ The UI should be created using ReactJS and JSX ☐ All Todos UI should display a title of 'Todos List' ☐ Todos List should be presented in a striped table. ☐ Todos List table should have 3 columns: "Description", "Date Created" and "Action". ☐ Each Todo in the list should be presented as a row in the table ☐ Completed Todos should be struck through and an action of "N/A" ☐ Todos not completed should have an action of "Edit" which is a link	**Acceptance Criteria** Delete 0% ☐ The UI should be created using ReactJS and JSX ☐ Add/Edit UI should be wrapped in a with a class of addEditTodo and display a title of a <h3> with text 'Add/Edit Todo'. ☐ Todo to add or edit should be presented in a <form>. ☐ The todoDescription should be wrapped in a with a Bootstrap CSS class of "form-group" ☐ The todoDescription should have a <label> with the text 'Description:' ☐ The todoDescription should be an <input> with a 'type' of "text", a 'name' of "todoDescription" and a Bootstrap CSS class of "form-control". ☐ The todoDateCreated should be wrapped in a with a Bootstrap CSS class of "form-group". ☐ The todoDateCreated should have a <label> with the text 'Created on:' and display the current date and time in a next to it. ☐ The submit button should be wrapped in a with a Bootstrap CSS class of "form-group". ☐ The todoCompleted should be wrapped in a with a Bootstrap CSS class of "form-group". ☐ The todoCompleted should have a <label> with the text 'Completed:' ☐ The todoCompleted should be an <input> with a 'type' of "checkbox", a 'name' of "todoCompleted" and a Bootstrap CSS class of "form-control" ☐ The submit button should be an <input> with a 'type' of "submit", have a 'value' of "Submit" and Bootstrap CSS classes of "btn btn-primary".

Mock Data

A copy of this data is also available in the file **sampleTodos.json** in **b-static-version/starter/src**.

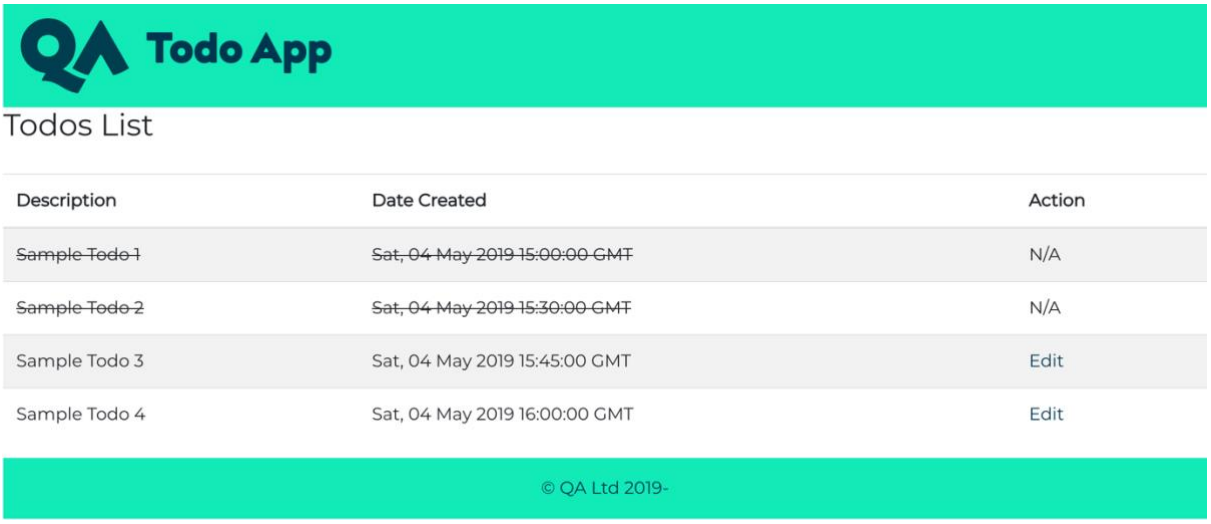
Notes:

- _id is in the format generated when an item is added to MongoDB
- todoDateCreated is in ISO Date format as this is used to store dates/times in MongoDB.

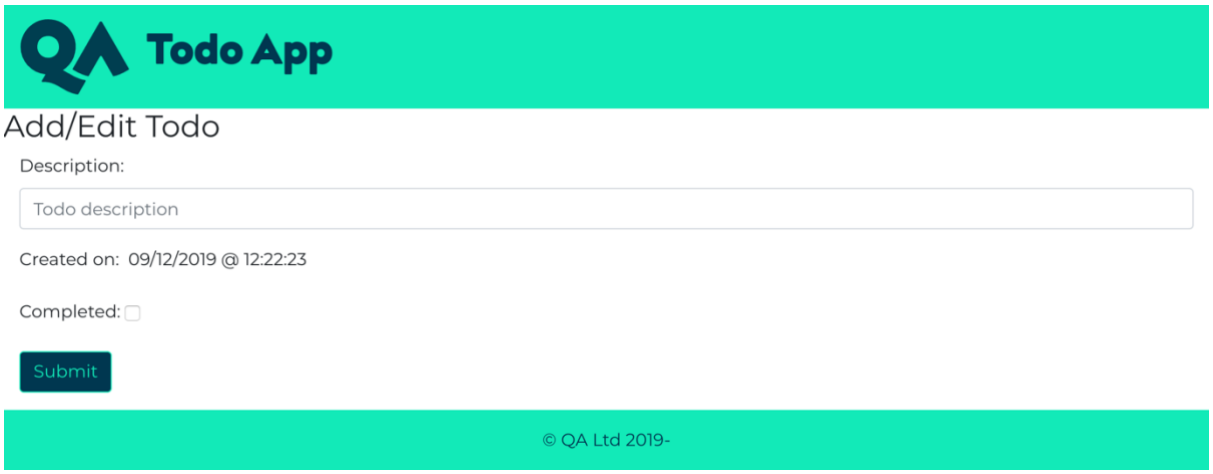
```
[
  {
        "_id": "5cc084952deb33810d2ec464",
        "todoDescription": "Sample Todo 1",
        "todoDateCreated": "2019-05-04T15:00:00.000Z",
        "todoCompleted": true
    },
    {
        "_id": "5cc08495bf3fd62d03f2f4c2",
        "todoDescription": "Sample Todo 2",
        "todoDateCreated": "2019-05-04T15:30:00.000Z",
        "todoCompleted": true
    },
    {
        "_id": "5cc08495bf3fd62d03f2f4c2",
        "todoDescription": "Sample Todo 3",
        "todoDateCreated": "2019-05-04T15:45:00.000Z",
        "todoCompleted": false
    },

    {
        "_id": "5cc08495bf3fd62d03f2f4c2",
        "todoDescription": "Sample Todo 4",
        "todoDateCreated": "2019-05-04T16:00:00.000Z",
        "todoCompleted": false
    }
]
```

Wireframes/Mock Ups

All Todos



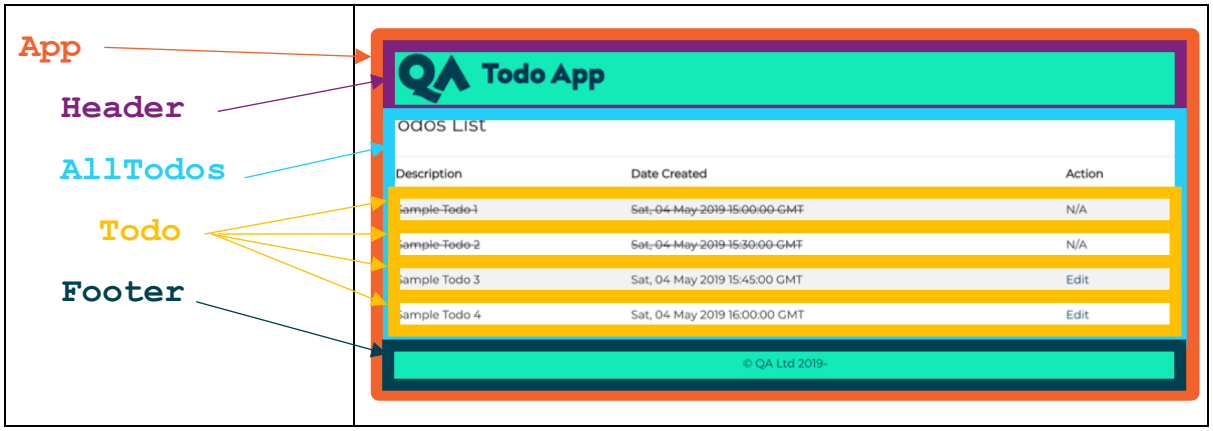
Add/Edit Todo



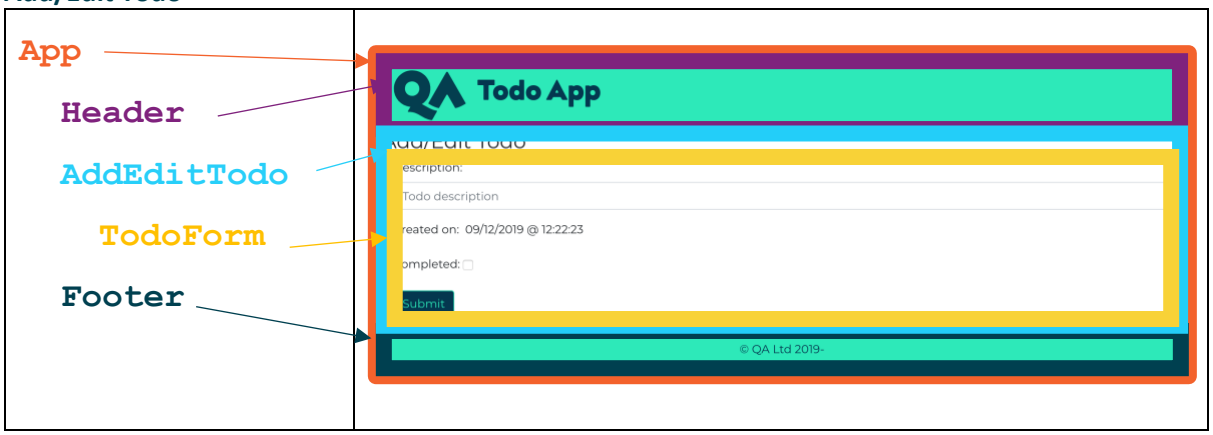
This is the end of Quick Lab 6

QuickLab 6 – Thinking in React – Part 1 – Component Hierarchy – Sample Solution

All Todos



Add/Edit Todo



Quick Lab 7a – Create Common Header and Footer Components

Objectives

- To be able to create static Function components and integrate them into an application

Overview

In this QuickLab, you will create the components for the header and footer sections of the application. These components will be placed in the Components folder of the application and linked to the App component to display them. The acceptance criteria should be used as a guide to create the components in the first instance. A step-by-step guide is provided, as are the code snippets for reference if find that you aren't sure where to start.

Activity – Header and Footer Acceptance Criteria

Complete the section 'Before Each QuickLab' for b-static-version/starter before continuing.

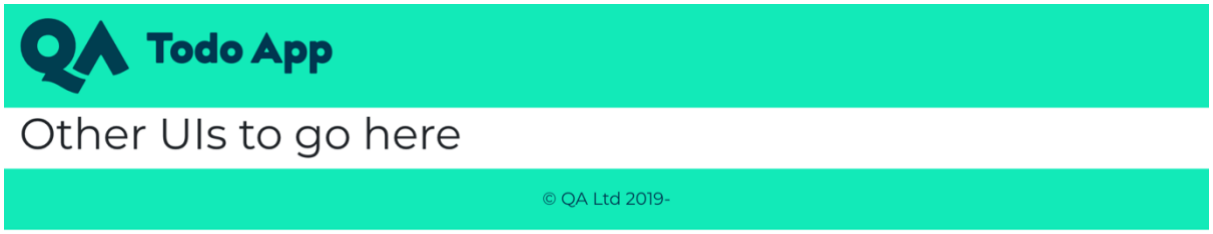
You will find that this project has already been set up and imports Bootstrap (along with popper.js and jQuery) to enable a fully responsive application to be made. The logo can be found in the Components/images folder as an SVG and should be imported into the header. Additional CSS has been provided (along with branding fonts) and is imported into the App component so it is available anywhere in the Component tree.

1. Use the acceptance criteria below to create Header and Footer components.

Footer	Header
Acceptance Criteria Delete 0% ☐ The UI should be created using ReactJS and JSX ☐ Every UI should have a <footer> ☐ The <footer> section should use the Bootstrap CSS classes to automatically set the top margin and left and right padding to 1rem ☐ The text in the footer should be centered using the "container" and "align-center" Bootstrap CSS classes You have unsaved edits on this field. View edits - Discard ☐ The 'footer' should contain the text '© QA Ltd 2019-'	Acceptance Criteria Delete 0% ☐ The UI should be created using ReactJS and JSX ☐ Every UI should have a <header> which wraps a <nav> with Bootstrap classes to set it as a navbar that is responsive and stacks the navigation on small screens ☐ The <nav> should contain the QA Logo having alt text of "QA Ltd" and a width of 100 ☐ The logo should be linked to https://www.qa.com, opening in a new window. The link should use Bootstrap class to signify that this is the brand and have rel properties of noopener and noreferrer. ☐ The <nav> section should contain the title 'Todo App' in a <h1> that is linked to / in an <a> that has a Bootstrap class to signify that this is for the brand

2. Import these into the App component to display them.

Desired Outcome



Activity – Header Stepped Instructions

1. In **b-static-version/starter/src/Components** create a new file called **Header.jsx**.
2. Add an `import` for `React` from `react`.
3. Add an `import` of `logo` from `'./images/qalogo.svg'`.
4. Create a **Function component** called `Header` that has *no parameters*.
5. The `return` of the component should have wrapping `<header>` and `<nav>` elements:
    - `<nav>` should have classes `navbar` `navbar-expand-sm`;
    - A **link** to https://www.qa.com with a **class** of `navbar-brand`, a **target** of `_blank` and a **rel** of `noreferrer`;
    - The **link** should contain an **image** whose **src** is `{logo}`**, alt** is `QA Ltd` and **width** is `100`;
    - A **sibling link** to `/` with a **class** of `navbar-brand` and **text** of `Todo App`.
6. `export` `Header` as `default`.
7. Save the file.
8. Open **App.js** from **b-static-version/src for editing**.
9. Add an `import` for `Header` from `./Components/Header`.
10. Within the outer `<div>`, add a **child** of `<Header />` as an **older sibling** of the inner `<div>`.
11. Save the file.

Activity – Footer Stepped Instructions

1. In **b-static-version/starter/src/Components** create a new file called **Footer.jsx**.
2. Add an `import` for `React` from `react`.
3. Create a **Function component** called **Footer** that has *no parameters*.
4. The `return` of the component should have wrapping `<footer>` element that:
    - Has Bootstrap classes of `mt-auto` (to set the top margins to automatic), `py-3` (to set padding left and right to 1rem), `text-center` and `container`;

- Has text content of `©QA Ltd 2019-`.

5. `export` `Footer` as `default`.
6. Save the file.
7. Open **App.js** from **b-static-version/src** for editing.
8. Add an `import` for `Footer` from `./Components/Footer`.
9. Within the outer `<div>`, add a **child** of `<Footer />` as a **younger sibling** to the inner `<div>`.
10. Save the file.

Use `npm start` to run the application and check that the output is as shown in the desired outcome as above.

Code Snippets

Header.jsx

```
import React from 'react';                                          // 2
import logo from './images/qalogo.svg';                             // 3
const Header = () => {                                              // 4
  return (
    <header>                                                        // 5
      <nav className="navbar navbar-expand-sm">                     // 5.1
        <a                                                          // 5.2
          href="https://www.qa.com"
          className="navbar-brand"
          target="_blank"
          rel="noopener noreferrer"
        >
          <img                                                      // 5.3
            src={logo}
            alt="QA Ltd"
            style={{ width: '100px'}}
          />
        </a>
        <a className="navbar-brand" href="/">                       // 5.4
          <h1>Todo App</h1>
        </a>
      </nav>
    </header>
  );
};
export default Header;                                              // 6
```

Footer.jsx

```
import React from 'react';                                          // 2
const Footer = () => {                                              // 3
  return (                                                          // 4.1
    <footer className="mt-auto py-3 container text-center">
      &copy; QA Ltd 2019-                                           // 4.2
    </footer>
  );
};
export default Footer;                                              // 5
```

App.js

```
import React from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap';
import 'popper.js';
import 'jquery';
import './Components/css/qa.css';
import Header from './Components/Header';                           // H9
import Footer from './Components/Footer';                           // F8
function App() {
  return (
    <div className=container>
      <Header />                                                    // H10
      <div className=container><h1>Other UIs go here</h1></div>
      <Footer />                                                    // F9
    </div>
  );
};
export default App;
```

This is the end of Quick Lab 7a

Quick Lab 8 – Exploring Props

Objectives

- To be able to use props in a component
- To be able to define propTypes for a component's props and ensure that they are present if needed
- To be able to supply defaultProps for a component
- To be able to pass props to a child from its parent

Overview

In this QuickLab, you will create a component called **ComponentWithProps** that uses 4 props (**header**, **content**, **number** and **nonexistent**) to populate a header and 3 paragraphs in its **return**. This will be rendered by the **MyComponent** component. You will observe the output and then add **PropTypes** for **header**, **content** (both strings) and **number** that are all required. The browser output will be observed again. Next, **defaultProps** will be added for header, content and number before providing actual props for **content** and **number** when it is called in the render of **MyComponent**. Finally, you will inspect the browser output to ensure all warnings have been removed.

Activity – ComponentWithProps - Step-by-step

Skip the 'Before Each QuickLab' for this Activity and work in the a-react-app/starter project.

1. Create a new file called **ComponentWithProps.jsx** in **a-react-app/starter/src**.
2. Add an `import` for `React` from `react`.
3. Define a **Function component** called `ComponentWithProps` that has `props` as an argument and a `return` that has:
    - A wrapping **React Fragment**;
    - A `<h1>` that uses `header` from `props` as its content;
    - A `<p>` that uses `content` from `props` as its content;
    - A `<p>` that uses `number` from `props` as its content along with some text;
    - A `<p>` that uses `nonexistent` from `props` as its content along with some text.
4. `export` `ComponentWithProps` as `default`.
5. Save the file.
6. Open **MyComponent.jsx** for editing and `import` the *new component*.
7. *Add it* to the `return` but ***DO NOT*** supply any props at this point.
8. Ensure that you wrap the `return` of **MyComponent** in a **React Fragment**.
9. Save the file and run the application.

You should find that the application runs without errors (check the console), although there are

empty elements where props were not found and spaces where props where included as part of other text.

Activity – Using PropTypes – Step-by-step

1. In **ComponentWithProps.jsx**, add an `import` for `PropTypes` from `prop-types`.
2. Before the `export` statement, add `ComponentWithProps.propTypes` and set it to an **object**.
    - Add **keys** of `header` and `content` with **values** that will **ensure** that **both** are **required strings**;
    - Add a key of `number` with a **value** that will ensure it is a **required number**.
1. Save the file and return to the browser console output.

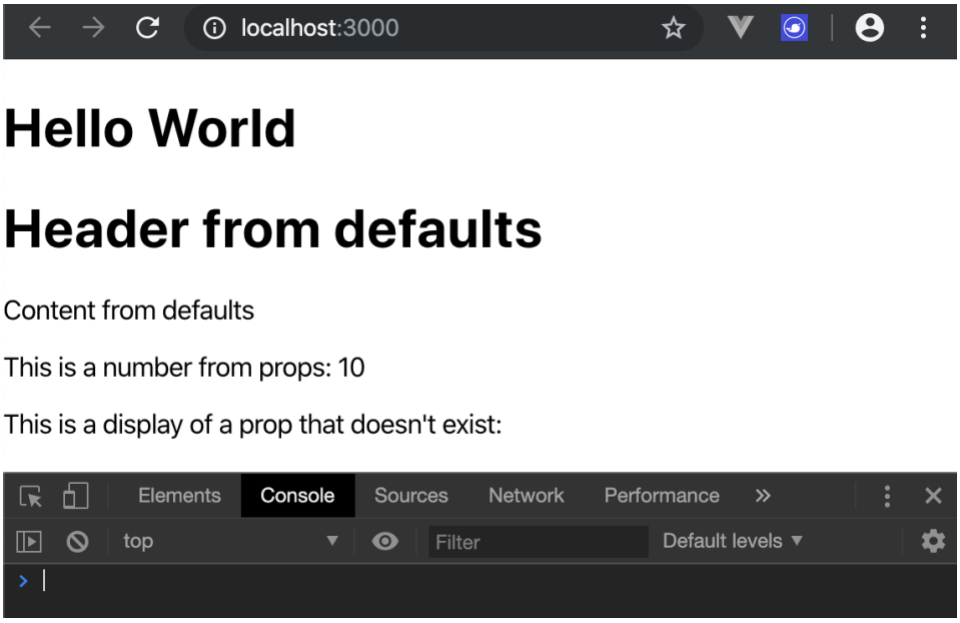
You should see that the application still renders the same but there are 3 warnings displayed on the console.

```
▶Warning: Failed prop type: The prop `header` is marked as        index.js:1375
required in `ComponentWithProps`, but its value is `undefined`.
    in ComponentWithProps (at MyComponent.jsx:8)
    in MyComponent (at App.js:6)
    in App (at src/index.js:7)
▶Warning: Failed prop type: The prop `content` is marked as       index.js:1375
required in `ComponentWithProps`, but its value is `undefined`.
    in ComponentWithProps (at MyComponent.jsx:8)
    in MyComponent (at App.js:6)
    in App (at src/index.js:7)
▶Warning: Failed prop type: The prop `number` is marked as        index.js:1375
required in `ComponentWithProps`, but its value is `undefined`.
    in ComponentWithProps (at MyComponent.jsx:8)
    in MyComponent (at App.js:6)
    in App (at src/index.js:7)
```

Activity – Using defaultProps – Step-by-step

1. Under the **PropTypes** defined in the last part, add a declaration for `ComponentWithProps.defaultProps` and set it to an object.
    - Add a **key** of `header` with **value** `` `Header from defaults` ``;
    - Add a **key** of `content` with **value** `` `Content from defaults` ``;
    - Add a **key** of `number` with a **value** of `100`.
2. Save the file and check the browser console output.

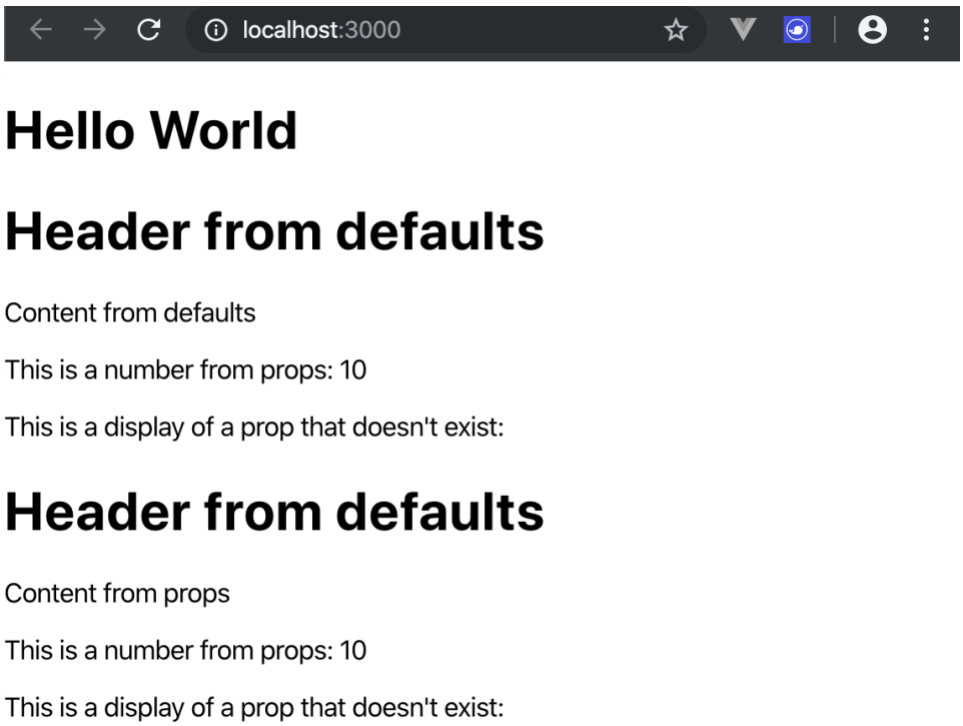
You should see that the warnings have disappeared, with values supplied as defaults displayed on the web page.



Activity – Supplying props from the Parent – Step-by-step

3. Open **MyComponent.jsx** for editing and add another **ComponentWithProps** to the `return`, supplying the following **props** (attributes):
    - `content` with a **value** of `"Content passed from props"`;
    - `number` evaluated in JavaScript to `10`.
4. Save the file and view the browser.

You should notice that the values displayed on the web page in the second rendering of **ComponentWithProps** match those to where they are picked up from in the code.



Code Snippets

ComponentWithProps.jsx

```
import React from 'react';                                          // CWP1
import PropTypes from 'prop-types';                                 // PT1

const ComponentWithProps = props => {                               // CWP2
  return (
    <>                                                              // CWP3.1
      <h1>{props.header}</h1>                                       // CWP3.2
      <p>{props.content}</p>                                        // CWP3.3
      <p>                                                           // CWP3.4
        This is a number from props:
        {props.number}
      </p>
      <p>                                                           // CWP3.5
        This is a display of a prop that
        doesn't exist: {props.nonexistent}
      </p>
    </>
  );
}

ComponentWithProps.propTypes = {                                    // PT2
  header:   PropTypes.string.isRequired,                            // PT2.1
  content:  PropTypes.string.isRequired,                            // PT2.1
  number:   PropTypes.number.isRequired                             // PT2.2
};

ComponentWithProps.defaultProps = {                                 // DP1
  header:   `Header from defaults`,                                 // DP1.2
  content:  `Content from defaults`,                                // DP1.3
  number:   10                                                      // DP1.3
};

export default ComponentWithProps;                                  // CWP4
```

MyComponent.jsx

```
import React from 'react';
import ComponentWithProps                                           // CWP6
from 'ComponentWithProps';

const MyComponent = () => {
  return (
    <>                                                              // CWP8
      <h1>Hello World</h1>
      <ComponentWithProps />                                        // CWP7
      <ComponentWithProps
        content="Content passed from props"                         // SP1.1
        number={10}                                                 // SP1.2
      />
    </>                                                             // CWP8
  );
};
```

This is the end of Quick Lab 8

Quick Lab 10a – Thinking in React Part 2 – A Static Version – Components with static data

Objectives

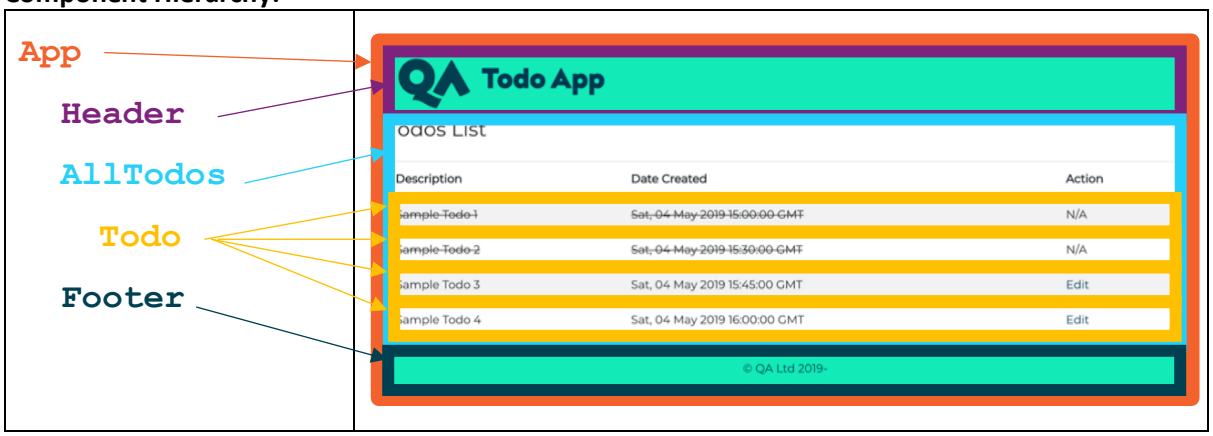
- To be able to use static external data to populate components
- To be able to use the map function to create multiple components
- To be able to conditionally render items dependent on some value

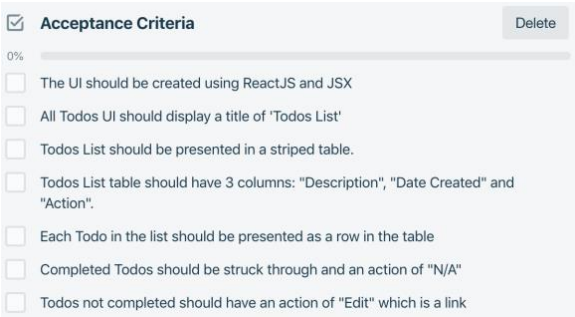
Overview

In this QuickLab, you will use the data supplied in the file **src/sampleTodos.json** to populate the AllTodos view. You should use the component hierarchy identified earlier (and shown below) and the Acceptance Criteria to produce the components needed for the AllTodos UI. A Todo model (basically a JavaScript class to define the shape of a Todo) has been defined in the **./utils** folder for use with the **instanceof PropTypes** check.

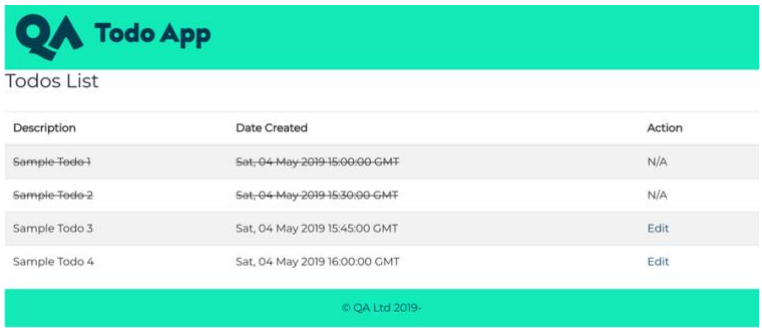
Continue working in the **b-static-version/starter** folder.

Component Hierarchy:





Desired Outcome



Activity 1 – The Todo component

Skip the 'Before Each QuickLab' for this Activity and continue working in the b-static-version/starter folder.

1. Create a new file in the **src/Components** folder called **Todo.jsx**.
2. Insert the *boilerplate code* for an *empty Function component* that receives a **prop** of `{todo}`.
1. Import `PropTypes` from `prop-types`.
2. Import `TodoModel` from `./utils/Todo.model`.
3. Set a `const` `dateCreated` to be a `new` `Date` that **parses** the `todo.todoDateCreated` and converts it to a **UTC string**.
4. Set a `const` `completedClassName` that is *conditionally set* to `` `completed` `` if `todo.todoCompleted` is **true** and an *empty string* if not.
5. Declare a variable `completed`.
6. Use an `if` statement to set `completed` to the **string** `` `N/A` `` if `todo.todoCompleted` is **true** and to the markup `<a href="/">Edit</a>` if not.
7. **Return** a **table row** that has *3 cells* whose *first 2* have a `className` set by `completedClassName` and whose **content** is the `todo.todoDescription` and `dateCreated` respectively. The *final cell* should **render** the `completed` variable.
8. **Before** the `export` statement add `Todo.propTypes` as an **object** that sets a **key** of `todo` to be a **call** to `instanceOf` on `PropTypes`, passing `TodoModel` as the argument.
9. Save the file.

Activity 2 – The AllTodos component

1. Create a new file in the **src/Components** folder called **AllTodos.jsx**.
2. Insert the boilerplate code for an empty Function component that receives *no props*.
3. Import the CSS file for **AllTodos** found in the css folder.
4. Import `sampleTodos` from the **sampleTodos.json** file.
5. Import `Todo` from the **Todo** file.
6. `TodoModel` should be imported from **./utils/Todo.model**;
7. Inside the component function, set a `const` `todos` that **maps** the `sampleTodos` array with an arrow function that:
    - Takes `currentTodo` as an argument;
    - Has a line in the function body that creates a **new** `TodoModel` called `todo` by passing in the properties from `currentTodo` in the order **description**, **date created**, **completed** and **_id** into the **TodoModel** *constructor*;

- **Returns** a `Todo` component with a **property** of `todo` set to the `todo` and a `key` of the ***todo's _id***.

8. Make the component **return** a wrapping `div` with a `className` of `row` with:
    - A `h3` with **text** of `Todo List`;
    - A *sibling table* with `classNames` `table` and `table-striped`;
    - A `thead` that has a *table row* that has the 3 headings **Description**, **Date Created** and **Action**;
    - A `tbody` that renders the array of `todos`.
9. Save the file.

Activity 3 – Render the AllTodos component

1. Open **App.js**.
2. Replace the placeholder text inside the inner `div` with the `className` container with an `AllTodos` component.
3. Save the file and fire up the application.

Code Snippets

Todo.jsx

```
import React from 'react';                                          // 1.2
import PropTypes from 'prop-types';                                 // 1.3
import TodoModel from './utils/Todo.model';                 // 1.4
const Todo = ({ todo }) => {                                        // 1.2
  const dateCreated =                                               // 1.5
  new Date(Date.parse(todo.todoDateCreated)).toUTCString();
  const completedClassName = todo.todoCompleted ? `completed` : ``; // 1.6
    let completed;                                                  // 1.7
    if (todo.todoCompleted) {                                       // 1.8
        completed = `N/A`                                           //
    } else {                                                        //
        completed = <a href="/">Edit</a>                            //
    }                                                               //
    return (                                                        // 1.9
        <tr>                                                        //
            <td className={completedClassName}>                     //
              {todo.todoDescription}                                //
            </td>                                                   //
            <td className={completedClassName}>                     //
              {dateCreated}                                         //
            </td>                                                   //
            <td>{completed}</td>                                    //
        </tr>                                                       //
    );                                                              //
};                                                                  //
Todo.propTypes = {                                                  // 1.10
  todo: PropTypes.instanceOf(TodoModel)
};

export default Todo;
```

AllTodos.jsx

```
import React from 'react';                                          // 2.2
import './css/AllTodos.css';                                        // 2.3
import sampleTodos from '../sampleTodos.json';                      // 2.4
import Todo from './Todo';                                          // 2.5
import TodoModel from './utils/Todo.model';                         // 2.6
const AllTodos = () => {                                            // 2.2
  const todos = sampleTodos.map(currentTodo => {                    // 2.7
    const todo = new TodoModel(currentTodo.todoDscription,          //
      currentTodo.todoDateCreated, currentTodo.todoCompleted,       //
      currentTodo._id);                                             //
    return <Todo todo={todo} key={todo._id} />});                   //
  return (                                                          // 2.8
    <div className="row">                                           //
      <h3>Todos List</h3>                                           //
      <table className="table table-striped">                       //
        <thead>                                                     //
          <tr>                                                      //
            <th>Description</th>                                    //
            <th>Date Created</th>                                   //
            <th>Action</th>                                         //
          </tr>                                                     //
        </thead>                                                    //
        <tbody>{todos}</tbody>                                      //
      </table>                                                      //
    </div>                                                          //
  );                                                                //
};

export default AllTodos;
```

App.jsx

```
import React from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap';
import 'popper.js';
import 'jquery';
import './Components/css/qa.css';

import Header from './Components/Header';
import Footer from './Components/Footer';
import AllTodos from './Components/AllTodos';

function App() {
  return (
    <div className="container">
      <Header />
      <div className="container">
        <AllTodos />                                                // 3.2
      </div>
      <Footer />
    </div>
  );
}

export default App;
```

This is the end of Quick Lab 10a

Quick Lab 10c – Thinking in React Part 2 – A Static Version – Adding a Form

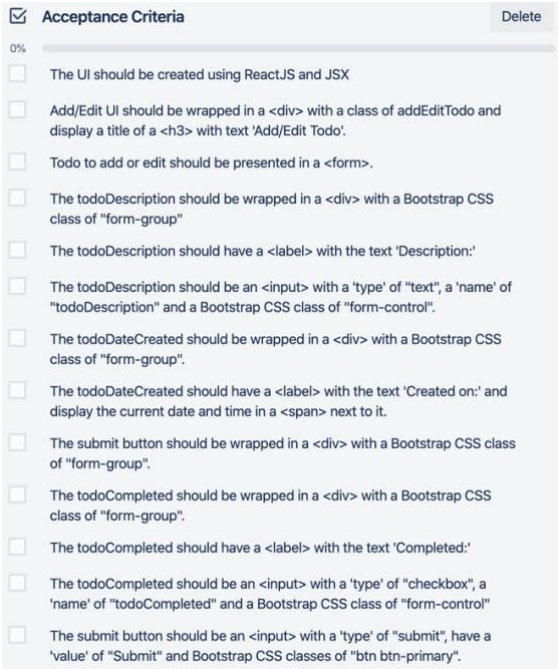
Objectives

- To be able to add a static, non-interactive form to an application

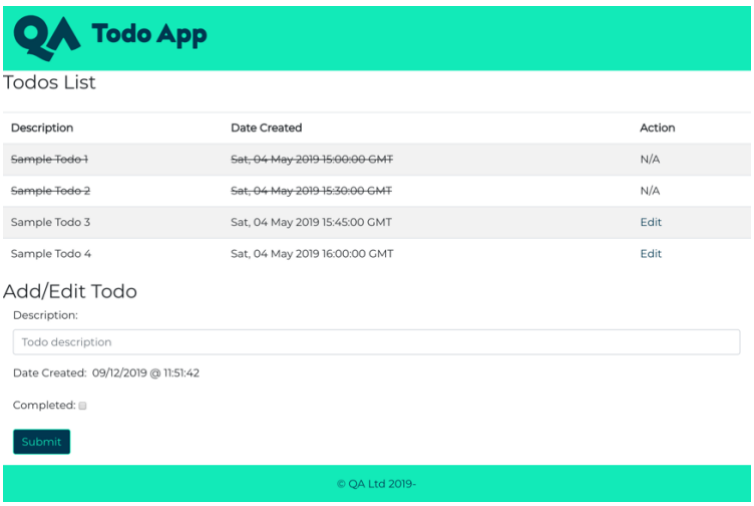
Overview

In this QuickLab, you will create the components needed to put the UI to add or edit a Todo into the application. Use the acceptance criteria and the mock-up provided to help. A **TodoForm** component will be created that allows the input of the todo's description, uses a supplied utility component called **DateCreated** (available in **/Components/utils**), provides a checkbox for the 'completed' status and a submit button. A wrapping **AddEditTodo** component will be created to provide the title and render the form and this will be added under the **AllTodos** component in the **App** component.

Continue working in the **b-static-version/starter** folder.



Desired Outcome



Activity 1 – Create the TodoForm Component

1. In the **Components** folder, create a new file called **TodoForm.jsx**.
2. Add the boilerplate code to create a Functional component that does not receive any props.
3. Import `DateCreated` from `` `./utils/DateCreated` ``.
4. Make the function **return** a wrapping **form** element that encloses:
    - A **div** with a **className** of `form-group` containing:
        - A **label** for `todoDescription` with the **content** of `Description: `
        - A **text input** with a **name** of `todoDescription`, a **placeholder** of `Todo Description` and a **className** of `form-control`.
    - A **div** with a **className** of `form-group` containing:
        - A **label** for `todoDateCreated` with the **content** of `Created on: `
        - A **DateCreated** component.
    - A **div** with a **className** of `form-group` containing:
        - A **label** for `todoCompleted` with the **content** of `Completed: `
        - A **checkbox input** with a **name** of `todoCompleted`.
    - A **div** with a **className** of `form-group` containing:
        - A **submit input** with a **value** of `Submit` and **classNames** `btn` and `btn-primary`.
5. Save the file.

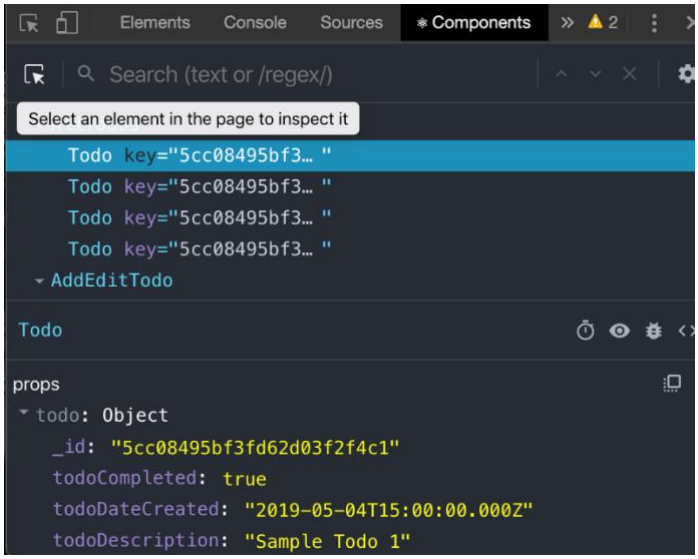
Activity 2 – Create the AddEditTodo Component

1. In the **Components** folder, create a new file called **AddEditTodo.jsx**.
2. Add the boilerplate code to create a Functional component that does not receive any props.
3. **Import AddEditTodo.css** from the appropriate path (**./css/AddEditTodo.css**).
4. The **return** of the function should be a wrapping **React.Fragment** that encloses:
    - A **div** with **classNames** of `addEditTodo` and `row` that wraps a **h3** with the content `Add/Edit Todo`;
    - A **TodoForm** component (imported from **./TodoForm**).
5. Save the File.

Activity 3 – Add the new components to the app

1. Open **App.js** for editing.
2. **Import** and then **add** the **AddEditTodo** component under the **AllTodos** component.
3. Save the file.

Launching the application in the browser should show the UI as shown in the desired outcome. Additionally, check that the 4 rendered Todo components prop values show in the Component section of the React Developer Tools:



Code Snippets

TodoForm.jsx

```
import React from 'react';                                          // 1.2
import DateCreated from './utils/DateCreated';                      // 1.3

const TodoForm = () => {                                            // 1.2
  return (                                                          // 1.4
    <form>                                                          //
      <div className="form-group">                                  //
        <label htmlFor="todoDescription">                           //
          Description:                                         //
        </label>                                                    //
        <input type="text" name="todoDescription"                  //
          placeholder="Todo description"                            //
          className="form-control" />                               //
      </div>                                                        //
      <div className="form-group">                                  //
        <label htmlFor="todoDateCreated">                           //
          Date Created:                                        //
        </label>                                                    //
        <DateCreated />                                             //
      </div>                                                        //
      <div className="form-group">                                  //
        <label htmlFor="todoCompleted">                             //
          Completed:                                           //
        </label>                                                    //
        <input type="checkbox" name="todoCompleted" />              //
      </div>                                                        //
      <div className="form-group">                                  //
        <input type="submit" value="Submit"                         //
         className="btn btn-primary" />                             //
      </div>                                                        //
    </form>                                                         //
  );                                                                //
};                                                                  // 1.2

export default TodoForm;                                            // 1.2
```

AddEditTodo.jsx

```
import React from 'react';                                          // 2.2
import './css/AddEditTodo.css';                                     // 2.3
import TodoForm from './TodoForm';                                  // 2.4

const AddEditTodo = () => {                                         // 2.2
  return (                                                          // 2.4
    <>                                                              //
      <div className="addEditTodo row">                             //
        <h3>Add/Edit Todo</h3>                                      //
      </div>                                                        //
      <TodoForm />                                                  //
    </>                                                             //
  );                                                                //
}                                                                   // 2.2

export default AddEditTodo;                                         // 2.2
```

App.js

```
import React from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap';
import 'popper.js';
import 'jquery';
import './Components/css/qa.css';

import Header from './Components/Header';
import Footer from './Components/Footer';
import AllTodos from './Components/AllTodos';
import AddEditTodo from './Components/AddEditTodo';                // 3.2

function App() {
  return (
    <div className="container">
      <Header />
      <div className="container">
        <AllTodos />
        <AddEditTodo />                                            // 3.2
      </div>
      <Footer />
    </div>
  );
}

export default App;.
```

This is the end of Quick Lab 10c