CST-391 Activity 7 Guide

Contents

[Mini App #3 – Dynamic Components Demo 2](#_Toc126842144)

[Dynamic Components Conclusion 8](#_Toc126842145)

[Part 5 Tracks, Lyrics and Video 8](#_Toc126842146)

[Stopping Point #5 - Dynamic Components Demo 9](#_Toc126842147)

[Part 6: Create New Album 10](#_Toc126842148)

[Stopping Point #6 – New Album 17](#_Toc126842149)

[Part 7: Edit an Album 18](#_Toc126842150)

[Stopping Point #7 – Edit and Album 24](#_Toc126842151)

# Mini App #3 – Dynamic Components Demo

It's time to start a new project to demonstrate new features that you can use to generate the song tracks on the music application. We will return to the music application in a few pages.

**Blog Postings**

This exercise will demonstrate how to dynamically add and remove items from a list. You will create a small application that looks like this. You can add new posts to a blog with the data entry form at the top of the page. Each posting will have a delete button that removes the post.

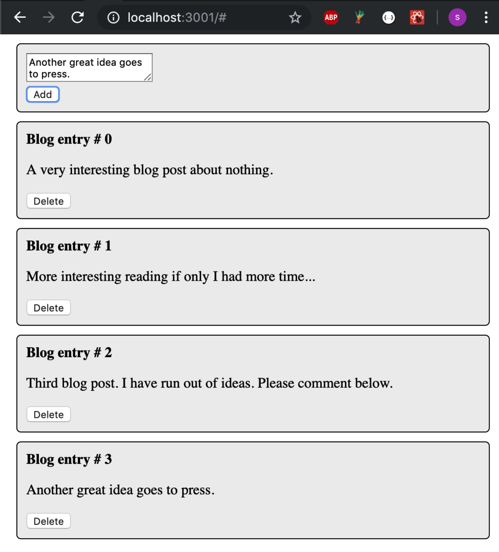


Figure 1 *App Preview*

**Create a New React App**

1. Start a new app by typing

**npx create-react-app blog** at the command prompt.

1. Start the app with **cd blog** and **npm start** commands.
2. Open Visual Studio Code to the folder of the new project.
3. Delete App.css and logo.svg.
4. In App.js start with this code:

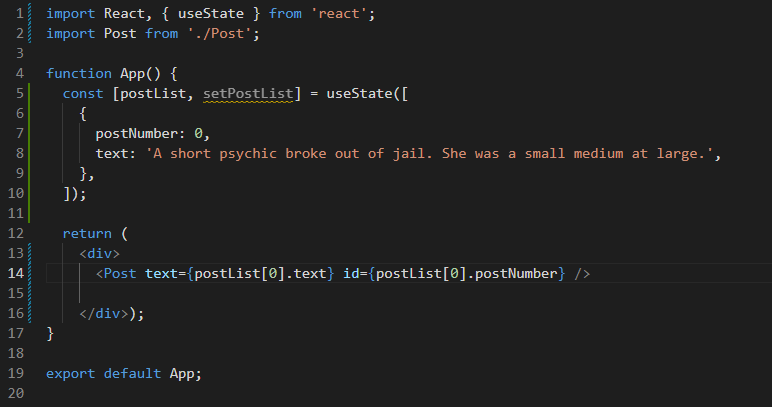


Figure 2 *App.js*

1. Create a Post.js file to display a blog post element.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3 *Post.js*

Notice the use of the props object and the way the object is set up in App.js on line 14.

1. Add a Post.css file.

Text

Description automatically generated

Figure 4 *Post.css*

1. At this point, you should be able to run the application.

Graphical user interface, text, application, email

Description automatically generated

Figure 5 *Current View*

**Implementation Notes**

You might wonder why functional components need to import React. Neither App.js nor Post.js has an obvious reference to React. However, when the JSX code is compiled to JavaScript, the reference to the React code becomes clear:

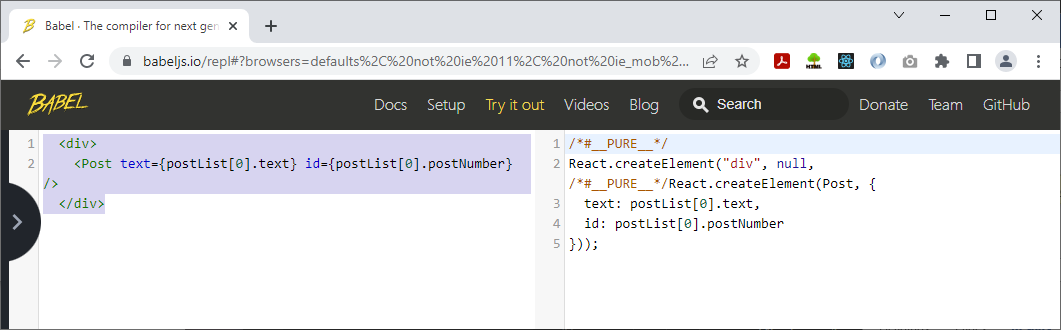
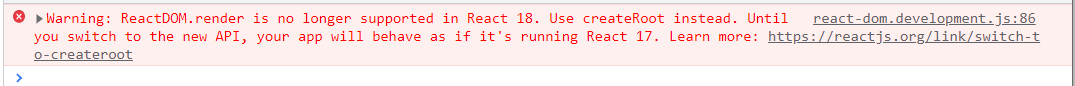


Figure 6 *Use of 'React'*

Also, notice the mechanism for creating the props object that Post will receive.

You might get this warning:



React 18 was just released two days before creating this tutorial. Evidently, create-react-app is not yet updated for React v18. Using it with npx should use the latest release of the tool. In a few days, programmers may no longer see that warning.

Remember that semantic versioning means the switch to React 18.x from 17.x involves breaking changes. This warning does not impede our application, but it can be fixed if you wish. Follow the link included in the warning to read more about React 18. It seems a monumental update in React.

1. Hard code a few more blog post items in the state and display them.



Figure 7 *Blog Posts*

1. If you hard coded your list of posts, modify your code to use the map function:

Text

Description automatically generated

Figure 8 *'postList.map'*

Important points about this code:

Notice the use of the 'posts' collection of JSX. You can build complex JSX code and render it later by enclosing the structure in curly braces.

When you create a list of components, React wants a unique key in each element to help it track and delta changes. We use postNumber here to provide that key. You can read more on this in "Lists and Keys," at <https://reactjs.org/docs/lists-and-keys.html>.

1. Add a Delete button and the associated onClick handler to the Post.js component.

Text

Description automatically generated

Figure 9 *Post.js with onDelete*

The syntax used on line 10 allows parameters to be specified.

1. Since the parent of Post is App and App is managing state, App should also own the onDelete method. Modify App.js to add this method and pass it to Post in the onDelete attribute:

Text

Description automatically generated

Figure 10 *handleDeletePost*

The updatePostList is set to postList with the id of the deleted post filtered out. We update the state with a call to setPostList and the component is rendered. You should now be able to delete your posts.

**Add a New Post**

In this section, we are going to add a data entry that will allow us to add to the list of blog posts.

1. Create a new component file called AddPost.js. Use the code shown here. This is an example of a **controlled component**. That means that the state of the component is updated on each onChange event which makes it possible to send the text area's content back to the App component on the onClick event.

Text

Description automatically generated

Figure 11 *AddPost.js*

1. Modify the App component to display the data entry form, as well as handle the onClick event that is sent by callback from AddPost.js.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 12 *App.js Update*

'handleAddPost' creates a new post from with a postId and the text.

'postId' is a new state variable we create on line 20.

The postList is modified with a callback method that takes current list and appends the newPost object. To read more about this technique, refer to "Update Arrays with React useState Hook Without Push," at <https://www.techiediaries.com/react-usestate-hook-update-array/>.

The '…' in front of the array on line 28 is called the spread syntax. It compiles to an iterable object that creates a comma separated list of the array contents. To read more on this, refer to "Rest Parameters and Spread Syntax," at <https://javascript.info/rest-parameters-spread>.

Finally, postId state is modified, adding one for the next post.

1. At the bottom of App display the <AddPost /> component and, of course, import AddPost.

Text

Description automatically generated

Figure 12 *Use AddPost*

You should now be able to both add and remove posts from the application.

## Dynamic Components Conclusion

This exercise demonstrated how to dynamically add and remove components from a page.

The changes are passed upward to the parent who manages the master list of posts.

**Deliverables**

1. Take screenshots of the application you created. Be sure to show the various features that were illustrated in this lesson. Place the captured images in the provided Microsoft Word document titled "Activity Summary Page." Caption each picture to explain what is being shown.
2. ZIP file.
   1. Delete the npm\_modules folder to remove the 40,000 or so files it contains. This folder can easily be recreated with the npm install command.
   2. Zip the project folder and include it as an attachment to the assignment.
3. Save these deliverables to be turned in as directed by the instructor.

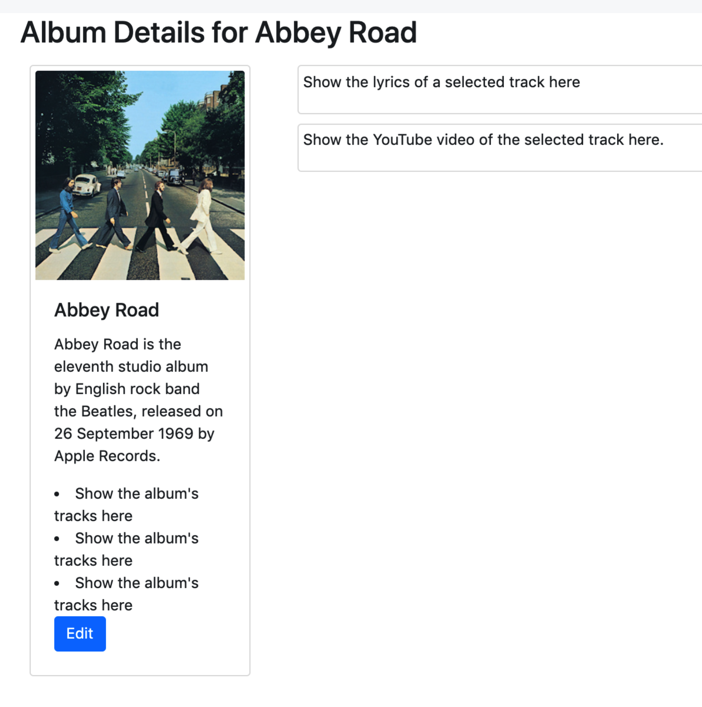
**Returning to the Music Application**

The previous exercise is designed to give you skills for managing a feature in the music application.

# Part 5 Tracks, Lyrics and Video

There is a lot more work to do for the OneAlbum component. This should be an interactive component that displays tracks, lyrics and videos.

* The design drawings of <OneAlbum /> show that there are four more child components needed to make this work.
* The selectedTrack should change value when the user clicks on a song track title.



<TracksList /> - container component that shows a list of tracks.

<TrackTitle /> - single element that shows title of one track. The **onClick** event of TrackTitle triggers a method which loads the <TrackLyrics/> and <TrackVideo/> content.

<TrackLyrics /> - displays the words of the selected song.

<TrackVideo /> - displays the video of the selected song.

Figure 13 *OneAlbum*

This part of the application is left to you as an optional challenge. Use similar techniques (creating a dynamic list) to what was shown in the previous mini-application example.

## Stopping Point #5 - Dynamic Components Demo

Save the project and summarize the progress you have made.

**Deliverables**

1. Take screenshots of the application you created. Be sure to show the various features that were illustrated in this lesson. Place the captured images in the provided Microsoft Word document titled "Activity Summary Page." Caption each picture to explain what is being shown.
2. Write a one-paragraph summary of the new features that have been added. Define new terminology that was used in the lesson.
3. Save this document to be turned in as directed by the instructor.

# Part 6: Create New Album

In this section, we will create a form that will accept data from the user to add an album to the collection. In a previous step, we created a placeholder form in order to make the Navigation bar work. Now we are going to fill in the details on the new album form.

1. Go to [Bootstrap](https://getbootstrap.com/docs/4.6/components/forms/) and get some HTML code for a default form. Choose Components > Forms and use the first example.

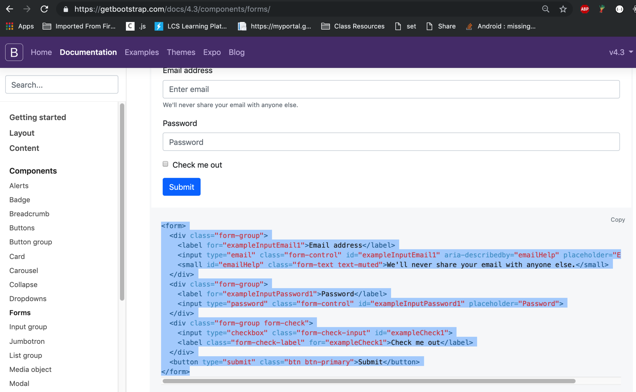


Figure 14 *Bootstrap Form Page*

1. Open the NewAlbum.js file and paste the Bootstrap starting code and modify it to match JSX standards. Change all class references to className and close all open elements.

Text

Description automatically generated

Figure 15 *NewAlbum.js*

You should see the new form in the browser window.

Graphical user interface, text, application, email

Description automatically generated

Figure 16 *Browser View*

We need to close all tags and use 'className' rather than 'class' because 'class' is a keyword in JavaScript. This is JSX, a computer language, not HTML. Recall that it is compiled to JavaScript. Tag ends and avoiding keywords (class) is therefore required:

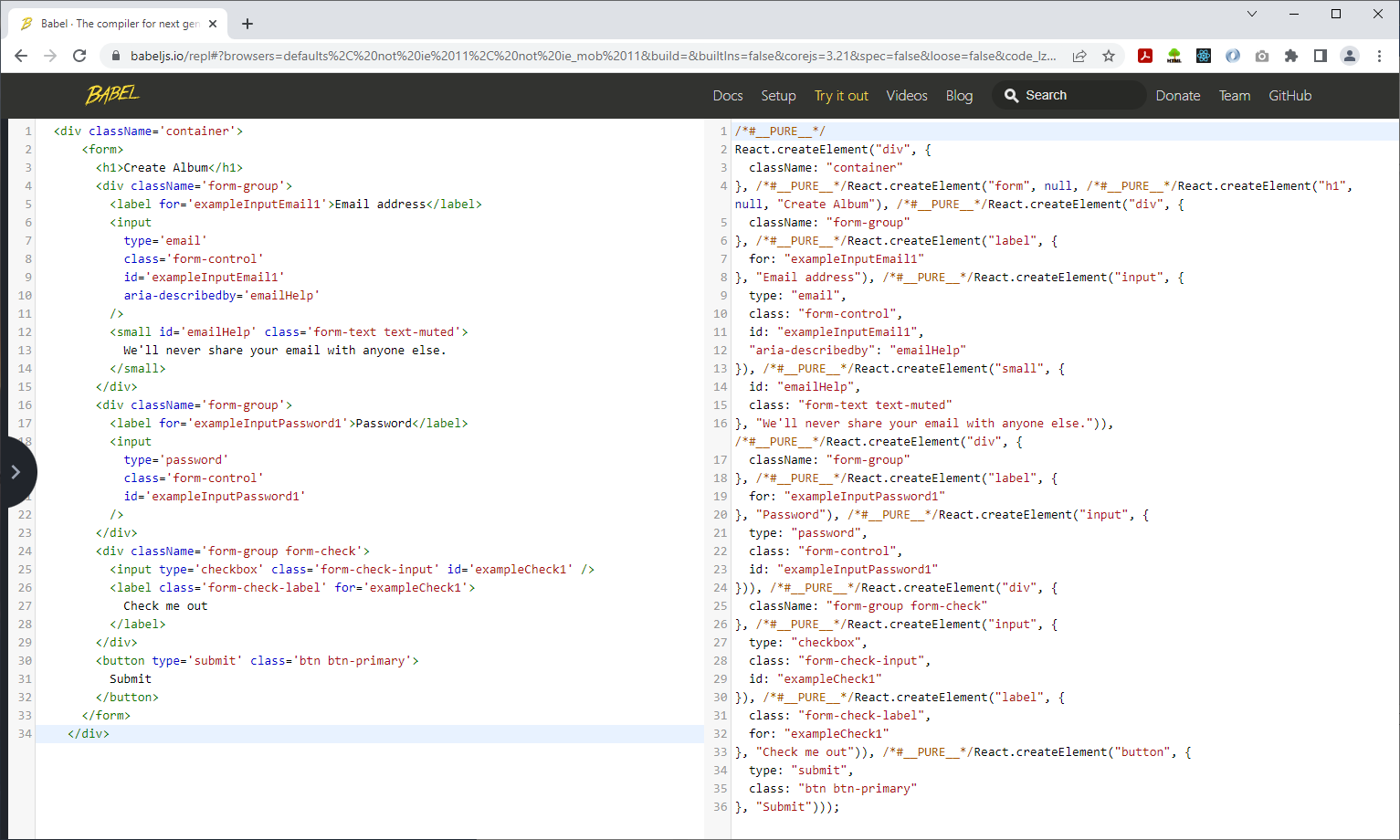


Figure 17 *Why Closing Elements is Important*

1. We copy an example from Bootstrap and modify it to fit our needs. A typical workflow as we have learned! Modify the form to show details about a new album. The following code is not ready to run; we need to provide onChange and handleFormSubmit. We will do this next.

Text

Description automatically generated

Figure 18 *NewAlbum.js*

**Controlled Components in a Data Entry Form**

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. In React, a mutable state is typically kept in the state property of components and only updated by changing state. Refer to "Controlled Components," at <https://reactjs.org/docs/forms.html#controlled-components>, to learn more about controlled components.

To create a controlled component:

* Set the value attribute of an element to a React state-controlled attribute.
* Create an onChange method that modifies the state when the value changes. In a text input control, this means on each keystroke. A controlled component always contains the current content.

A computer screen capture

Description automatically generated with medium confidence

Figure 19 *onSubmit*

1. Call useState for each controlled value. Also shown is the call to useNavigate that will soon employ.

Text

Description automatically generated

Figure 20 *useState and useNavigate*

1. Create the onChange events to set the state for form values:

Text

Description automatically generated

Figure 21 *onChange Events*

We now have controlled components for each control in our form. We are ready to handle the form submit (line 59 and line 75 of the form).

Text

Description automatically generated

Figure 22 *Handle Form Submit*

Modify App.js, adding a onNewAlbum method and passing it as an attribute to the NewAlbum component:

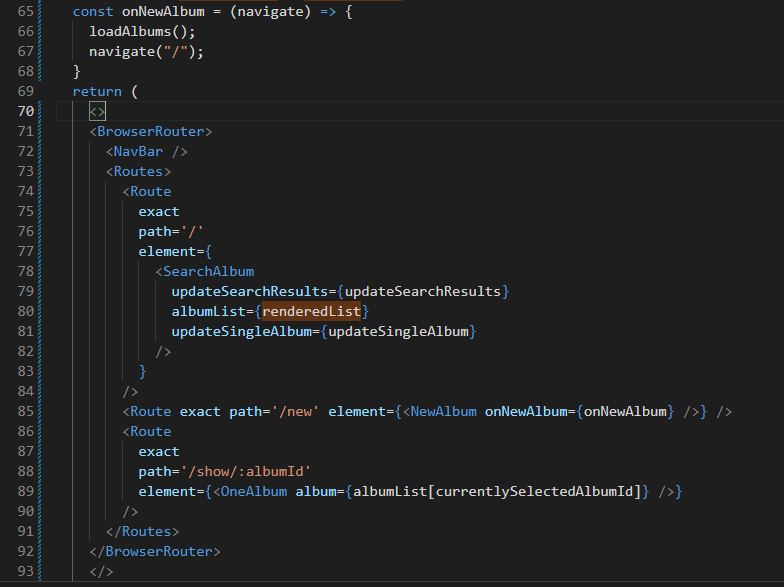


Figure 23 *Updated App.js*

You should now be able to create and view a new album.

Graphical user interface, website

Description automatically generated

Figure 24 *Browser View*

**Full Text**

In case you missed something, here is the full script for the NewAlbum.js file.

Text

Description automatically generated

Figure 25 *NewAlbum.js File Script Part 1*

Text

Description automatically generated

Figure 26 *NewAlbum.js File Script Part 2*

**More to Do (*Optional*)**

The current data entry form only serves to add a new album. None of these features are completed yet. They are left to you to complete.

1. The New Album form does not have the ability to add new music tracks. This part of the application is left to you as a challenge to complete.
2. The New Album form does not give any feedback to the user regarding whether a new album is successfully created or the new album creation fails.
3. The application should automatically navigate back to the main screen and update the albumlist state of the App component.

## Stopping Point #6 – New Album

Save the project and summarize the progress you have made.

**Deliverables**

1. Take screenshots of the application you created. Be sure to show the various features that were illustrated in this lesson. Place the captured images in the provided Microsoft Word document titled "Activity Summary Page." Caption each picture to explain what is being shown.
2. Write a one-paragraph summary of the new features that have been added. Define new terminology that was used in the lesson.
3. Save this document to be turned in as directed by the instructor.

# Part 7: Edit an Album

Now we are going to allow the editing of an existing album. There are two paths we can take.

* Copy NewAlbum, create a new component EditAlbum, then tweak the code in EditAlbum to behave as an editor.
* Modify NewAlbum so that it also edits an existing album.

It can be easier to copy and modify code. However, this approach will make you pay in a production release. Each modification, new feature, or bug fix in NewAlbum will need to be replicated in EditAlbum. We are going to modify NewAlbum so that it also supports edit functionality. It's not the easy path, but it is the correct one.

1. Rename the NewAlbum component to the more general name EditAlbum. You need to change the component name, its file name, import statements, and its use.
2. Next, modify the newly renamed EditAlbum component. Most of this is differentiating 'new' from 'edit'. If an album is present in 'props', then we are in edit album mode; otherwise, we are in new album mode.

Text

Description automatically generated

Figure 27 *EditAlbum.js*

1. When the form is submitted, the new and the edited album code are largely the same. When editing an album, the original album ID must be maintained. Alternatively, when creating a new album, the database assigns a primary key that becomes the album ID.

Text

Description automatically generated

Figure 28 *handleFormSubmit*

1. Saving the album needs to distinguish between 'post' and 'put'. Recall that in a REST API, POST is Create and PUT is Modify.

Text

Description automatically generated

Figure 29 *saveAlbum*

1. A small change for the page title.

Text

Description automatically generated

Figure 30 *Page Title Switch*

That's it for EditAlbum. We have seen two props objects used in EditAlbum: props.album (new: added for editing) and props.onEditAlbum (existing: renamed from onNewAlbum). Since the parent of EditAlbum is App.js, we will next examine that component to see how those props objects are set up for EditAlbum. We will also look at the flow through the application.

Text

Description automatically generated

Figure 31 *App.js*

*Line 60*

The name 'onEditAlbum' is changed from 'onNewAlbum'. Otherwise, no change. It updates the list of albums from the database and navigates to the root of the application.

*Line 76*

'updateSingleAlbum' is specified for SearchAlbum. This element is not changed, but it is important to note. We will discuss updateSingleAlbum in the next section.

*Lines 80 and 81*

Line 80 has the component name change only. Line 81 is new for the edit album. It uses the same component (EditAlbum) and callback (onEditAlbum). It also provides in the album attribute the specific album to be edited. This mechanism is identical to the one previously developed for 'OneAlbum' on line 82.

The 'edit' or 'show' path is determined by the updateSingleAlbum method.

Text

Description automatically generated

Figure 32 *App.js*

1. 'updateSingleAlbum' is updated with a URI in the parameter. It is this URI that determines the 'edit' or the 'show' path. We have seen that updateSingle album is an attribute provided to SearchAlbum. We will look at SearchAlbum next to see how this is used.

Text

Description automatically generated

Figure 33 *SearchAlbum.js*

1. This component is unchanged. The important aspect to note is 'updateSingleAlbum' is passed as onClick to the 'AlbumList' component, shown next.

Text

Description automatically generated

Figure 34 *AlbumList.js*

1. AlbumList has an intermediate callback method on line 6. Intermediate methods (a method that calls its parent and that is also provided to a child component) are not necessary and this one should be factored out. The ultimate child component can callback through many arbitrary ancestor levels. The top-level method just needs to be set in an attribute at each level of the component chain. Regardless of this, you can see the parameter list on line 8 is complete. The final component we need to look at is Card.js.

Text

Description automatically generated

Figure 35 *Card.js*

1. 'handleButtonClick' is another intermediate function that should be factored out. It is an artifact left from development and adds no information.
2. Line 22 adds a new edit button.
3. The URIs that will lead to 'show' or 'edit' are specified on lines 17 and 23, attributes of their respective buttons.

**Unfinished Business (*Optional*)**

The current data edit form is still lacking these features. They are left to you to complete.

1. The Edit Album form does not have the ability to display or modify music tracks. This part of the application is left to you as a challenge to complete.
2. The Edit Album form does not give any feedback to the user regarding whether an album is successfully updated or the album update fails.
3. The application should automatically navigate back to the main screen and update the albumlist state of the App component.
4. Add an event to handle the cancel button.

## Stopping Point #7 – Edit and Album

Save the project and summarize the progress you have made. Submit all deliverables as directed by the instructor.

**Deliverables**

1. Take screenshots of the application you created. Be sure to show the various features that were illustrated in this lesson. Place the captured images in the provided Microsoft Word document titled "Activity Summary Page." Caption each picture to explain what is being shown.
2. Write a one-paragraph summary of the new features that have been added. Define new terminology that was used in the lesson.
3. Two ZIP files. Submit one zip file of the mini application and another zip file of the current state of the music application.
   1. Delete the npm\_modules folder to remove the 40,000 or so files it contains. This folder can easily be recreated with the npm install command.
   2. Zip the project folder and include it as an attachment to the assignment.