How I approached this project:

Although I did not use different files for my several React components, you can see through my code how I broke down the desired goal into different pieces with a structured hierarchy. Here is an image depicting the various pieces of the project, and below the image is a bullet hierarchy of the image (with coordinating colors to the boxes – the colors/boxes may be hard to see, so I recommend zooming in a bit for better analysis).

Graphical user interface, application, website

Description automatically generated

* Red – a filterable grades table webpage
  + Purple – the filtering tool search bar
    - Orange – the search box itself (receives user input)
  + Blue – the table itself, including static headers (no need for separate header category)
    - Green – the table rows based on the filter (displays user input)
      * Black – the color of the class name (based on grade/took discrepancy)
      * Gray – a Yes or No value based on the Boolean value of “took” for the class

With these basic components in mind, I structured the project into two main pieces – the filtering tool search bar, and the table of information.

For the filtering tool search bar, I designated it as heading 4, and inside the heading called a React function to handle user change to the search bar, called “updateGradeValue.” This React function is perhaps the biggest key to the project, as it handles updating the state “grade” to what the user desires, and then filters the classes based on which have matching grades to the grade entered (with the exceptions of “all” which returns all of the classes and the empty string which displays no classes). The filtered classes based on the current grade state are placed in a state of their own separate from the original group of classes so as to not overwrite or delete any data. Below, you can see h4 dedicated to creating the overarching heading section, and the updateGradeValue function.

Text

Description automatically generated

Text

Description automatically generated

The second main piece to the project is the table itself, which I designed in a div section of my render() function. Because the headings are static, I saw no need to create a separate component for their creation, so I included them with the creation of the table directly. Below you will find the code for the table, including the body which will be described below.

Text

Description automatically generated

For the table body/rows, I utilized a React function of my own creation called “createTableRows” in the div section where the table is created inside the “tbody” tag. This function takes the current filtered classes state and creates the table rows based off the data provided. It includes a call to a React function called “getColor” which simply returns the correct color for the class name (black or red) based on if there is a discrepancy between the grade and the took values to properly color the class name in each row. createTableRows also includes a minor logic statement that changes the took value to Yes or No based on if the value for took is true or false. Because this component was so minor, it made sense to include it right in the creation of the table rows. Below you will find the code for createTableRows, as well as getColor.

Text

Description automatically generated

So while I used several React components in the same file, I used “Thinking in React” to plan the hierarchy of my webpage and used React functions to turn the design into reality. Although I didn’t explicitly attempt to start from a static state and work towards a fully functioning page, that’s actually exactly how I ended up working through the project. I started by just having an only-text header and one-color text table with true/false values for took. For representing the UI state, I already had some of the state mapped out of necessity for creation (specifically the classes state where all of the original data resides), but I then added in the grade and filteredClasses states for later use.

The component to hold all the states was an easy choice, simply being the overarching webpage component so that all downstream components could access the necessary data. Piece by piece, I then added in functionality to include coloring the class names based on discrepancies, Yes/No based on the took value, and filtering based on user input, which included adding inverse data flow. For example – updateGradeValue for the search box component edited the data inside the filteredClasses state to be accessed by the createTableRows component to create the data table rows.

Although I personally believe my project is appropriately designed for the size and purpose of the project, I can see how files named according to the specific hierarchal components can be useful for organizational purposes, especially with more complex projects. I will be sure to use such a strategy going forward.

Below you will find the originally submitted images for grading.

*Graphical user interface, text, application, website

Description automatically generated*

Graphical user interface, application, website

Description automatically generated