Home

ECS 162 Slice the Pie Support: David and Akshey Spring 2021

Say you got to run the University. First, learn about your funding sources, with a guessing game. How much comes from taxpayers? Tuition? Philanthropy and endowment? Then, you get to allocate money to things like buildings, undergraduate teaching, and financial aid. Finally, see how well your choices match the ones the real Provost made. Implemented with React and D3.

Thanks to: Benjamin Lee.

An example of a similar Website, but with way more words and less gamification, is the Balance the California state budget page.

Website description

There are three phases: income, expenses, and comparison. The first two both work the same. The user gets a blank pie chart - a dark circle - and a table of categories to fill in. As long as the total dosn't add up to 100, they see corresponding pieces of pie partially covering the circle. If they try to give numbers adding up to more than 100, it automatically makes the last number they entered smaller so that the total is 100, and the pie chart is complete. If they then want to add more pieces, they have to make some of the ones they already have smaller.

When they hit "Next" after the first phase (Revenue), they get to the second phase (Expenditures). They can build up a similar pie chart. Then, when they hit "Compare", they get to the third and fourth screens, comparing their choices with the actual revenues and expenditures.

1. Begin with the common starter project, which gives you React and a simple server. It's under "Final Project" under Teams, on Replit,

just like the other assignments.

2. The real budget information, with pie charts, is in this Budget Overview that the University publishes. We'll make some adjustments to make it more understandable to the casual user.

   For income, we can use the following breakdown, based on the first chart on page one:

   ```
   let revenueData = [ { name: "Medical Center",
   value: 45 }, { name: "Student Fees", value: 4 },
   { name: "State of California", value: 8 }, {
   name: "Tuition", value: 11 }, { name: "Research
   Grants and Contracts", value: 13 }, { name: "Pell
   Grants", value: 1 }, { name: "Non-educational
   Services", value: 11 }, { name: "Gifts,
   Endowments, Interest, Etc.", value: 7 } ]
   ```

3. For our chart of expenditures, we will turn the first table on page 2 into a pie chart. Again, let's rename some categories to be more user friendly, and make them add up to 100.

   ```
   let expenditureData = [ { name: "Medical Center",
   value: 43 }, { name: "Teaching and Teaching
   Support", value: 23 }, { name: "Research", value:
   11 }, { name: "Student Services and Financial
   Aid", value: 8 }, { name: "Operations and
   Maintenance (Buildings, etc)", value: 2 }, {
   name: "Administration", value: 3 }, { name: "Non-
   Educational Services", value: 2 }, { name:
   "Public Service", value: 2 }, { name:
   "Depreciation, Interest, etc.", value: 6 } ]
   ```

4. Here are the designs for how the app should behave.

5. Start by looking at the specs and deciding how to break each picture down into a set of React components (so the picture is composed of several widgets). For instance, Slicethepie-mobile-1-1 looks to me like a header, a progress bar, a pie chart, and a box of entry items.

6. Next, decide which widgets store data. The box of entry items (EntryBox?) and the pie chart have to share data, so let's combine them into a single Revenue component. This should store the user's choices, since they come from EntryBox and are input to the pie chart. The overall state of the app's display (shown in the progress bar - revenue, expenditures, compare) can be stored in the parent App, which could have different children at different

times, eg. a Revenue component, an Expenditures component and a Compare component. The EntryBox itself will have to store the current total of percents the user has entered, since it cannot allow the total to exceed 100 percent.

7. I would start by adding the pie chart, since you have a couple of examples of how to do this, either as a function component or as a class component.
8. Next add the header (should be easy?) and the box of entry items (EntryBox?) on the bottom; you can leave out the "more info" circles for now. To report the data back up to the parent App, you'll need pass a callback function into the EntryBox; each button could return its index and its value to the parent component. See this example of changing a chart with a button.
9. In the callback functions in the App component, store the data returned by the EntryBox elements as a State variable (if you are writing cutting-edge React with all function components, you can set this up with the useState() hook). Changing the State of the parent component should re-render all its children, so you should see the pie chart change.
10. You can build your own progress bar component, with SVG elements called by D3 functions (d3.circle, d3.line). Some people have found acceptable progress bar React components, but I think you'll get more out of building one, and it is not actually more difficult.
11. Add the "more info" circles and pop-outs for the EntryBox items. You can use any unicode "info circle" character, you don't have to exactly match the spec drawing. Each should have a React "onclick" function. Explaination text for each of the categories that can go in the popups is in this document. To add the info text, some people have use the react-tooltip module; an overlay (like we did with the chart in Fitness 2) is another choice. Any reasonable way to get rid of the info text is fine.
12. Next add the pop-out functions for labels for the pie slices (if they are not already there from the example). To add an onclick for an element added by D3, you add to its attributes (such as size, color):

```
  .on("click", function(d) {
  // code you want executed on the click event });
```

Again, you don't have to make it look exactly like the spec drawing; anything clear and visible is fine. Having the pie slice get bigger is good feedback for the user, though - look at the "outerradius" attribute of d3.arc. The pie slice should stay big and labeled until the user clicks it again, or until the user clicks a different pie slice.

13. Once you have revenues going, Expenditures should be very similar. You'll need to change the display state in the App component and have an if statement selecting different return() functions depending on the display state. Also display state should be passed to the progress bar as props.display (or something...it's a variable name).
14. The pie chart example now shows how to display two pie charts.