Solution: Create a basic List component

There are three types of operations you need to apply to the list of desserts: filtering, sorting and mapping.

Although the order of the operations is not that important, it's recommended to leave the final projection (mapping) to the end, since that final projection may skip some of the data needed for the filtering and sorting criteria.

Filtering

The first requirement is to display desserts that have less than 500 calories. That means Cheesecake, which has 600 cal, should be omitted. When you need to eliminate elements from your lists based on a certain condition or set of conditions, you need to use the `filter()` method.

The `filter` method creates a copy of the array, filtered down to just the elements from the original array that pass the test. In order words, it will return a new list with just the elements that fulfil the condition.

Each dessert from the list has a property called `calories`, which is an integer representing the number of calories. Therefore, the condition to be implemented should be as follows:

```
const lowCaloriesDesserts = props.data
 .filter((dessert) => {
   return dessert.calories < 500;
 })
```

`lowCaloriesDessert` variable will then hold a list of three desserts, without Cheesecake.

Sorting

The second requirement you have to implement is sorting the list by calories, from low to high or in ascending order. For that, arrays in JavaScript offer the `sort()` method, which sorts the elements of an array based on a comparison function provided. The `return` value from that comparison function determines how the sorting is performed:

compareFn(a, b) return value	sort order
`> 0`	sort `a` after `b`
`< 0`	sort `a` before `b`
`=== 0`	keep original order of `a` and `b`

You can chain one operation after another. Recall that `filter` returns the new array with the filtered down elements, so sort can be chained right after that, as below:

```
const lowCaloriesDesserts = props.data
 .filter((dessert) => {
   return dessert.calories < 500;
 })
 .sort((a, b) => {
   return a.calories - b.calories;
 })
```

The compare function makes sure the sorting occurs in ascending order, according to the table above.

Mapping

Finally, to apply the desired projection and display the information as requested, you can chain the `map` operator at the end and return a `<li>` item with the dessert name and its calories, both separated by a dash character, and the word "cal" at the end.

The final code should look like below:

```
const lowCaloriesDesserts = props.data
 .filter((dessert) => {
   return dessert.calories < 500;
 })
 .sort((a, b) => {
   return a.calories - b.calories;
 })
 .map((dessert) => {
   return (
     <li>
       {dessert.name} - {dessert.calories} cal
     </li>
   );
 });
```

And the full implementation of the DessertsList component:

```
const DessertsList = (props) => {
 const lowCaloriesDesserts = props.data
  .filter((dessert) => {
    return dessert.calories < 500;
  })
  .sort((a, b) => {
    return a.calories - b.calories;
  })
  .map((dessert) => {
    return (
      <li>
        {dessert.name} - {dessert.calories} cal
      </li>
    );
  });
 return <ul>{lowCaloriesDesserts}</ul>;

}
export default DessertsList;


```

Final result

This is what should be displayed in your browser: