

WEB102 | Intermediate Web Development

Intermediate Web Development Summer 2025 (@ Section 1 | Wednesdays 4PM - 6PM PDT)
Personal Member ID#: 107789

Week 3: Lab 3 - On My Grind

Overview

Have you ever wanted to work at Starbucks? You're about to be one step closer. In this app inspired by Wordle, memorize coffee drink recipes and quiz yourself on a random real (and possibly discontinued ☺️) Starbucks drink's temperature, flavor, milk type, and blendedness. The app will let you know if you're a natural at barista-ing.

View an exemplar of what you'll be creating in this lab [here!](#)

Required Features

Quiz, quiz, quiz! In the required features for this project, you will have a form with multiple choice inputs that can be selected and their values will populate the answer box above them. Then when submitted, the answer boxes will turn red if the answers placed there are incorrect and purple if the answers are correct. Can you handle the Starbucks drink station at an 8 am rush?



On My Grind

So you think you can barista? Let's put that to the test...

Hi, I'd like to order a:

toffee almond cappuccino



Temperature



- ☐ hot
- ☐ lukewarm
- ☐ cold

Syrup



- ☐ mocha
- ☐ vanilla
- ☐ toffee
- ☐ maple
- ☐ caramel
- ☐ other
- ☐ none

Milk



- ☐ cow
- ☐ oat
- ☐ goat
- ☐ almond
- ☐ none

Blended



- ☐ yes
- ☐ turbo
- ☐ no

Check Answer



On My Grind

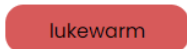
So you think you can barista? Let's put that to the test...

Hi, I'd like to order a:

toffee almond cappuccino

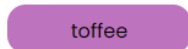


Temperature



- ☐ hot
- ☒ lukewarm
- ☐ cold

Syrup



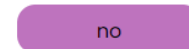
- ☐ mocha
- ☐ vanilla
- ☒ toffee
- ☐ maple
- ☐ caramel
- ☐ other
- ☐ none

Milk



- ☒ cow
- ☐ oat
- ☐ goat
- ☐ almond
- ☐ none

Blended



- ☐ yes
- ☐ turbo
- ☒ no

Check Answer

stretch Features

In the stretch goals, we are making it a little harder by having users have to guess our four ingredients for each drink and type in their answer. Of course this means that they cannot just type in anything, their guesses still have to be in line with what options we have given for the 4 categories. How can we enforce that?



On My Grind

So you think you can barista? Let's put that to the test...

Hi, I'd like to order a:

Temperature

Guess the ingredient...

hot

lukewarm

cold

Syrup

Guess the ingredient...

mocha

vanilla

toffee

maple

caramel

other

none

Milk

Guess the ingredient...

cow

oat

goat

almond

none

Blended

Guess the ingredient...

yes

turbo

no

Check Answer



Goals

By the end of this lab you will be able to...

- ☐ Create React forms from scratch and dynamically populating their answer choices
- ☐ Create a new custom component within a parent component and passing in props and ways to edit state variables
- ☐ Creating and using React forms
- ☐ Creating and setting state variables of all kinds
- ☐ Using state variables to edit information on a page
- ☐ Importing data files into an app
- ☐ Using user input to change a visual aspect of the app

Features

Required Features

- ☐ One drink at a time is displayed

- ☐ The user can click one option for each of the four categories to input their answer
- ☐ The selected option populates into the box above, and if an option is already in the box, it is replaced by the new option
- ☐ Clicking “check answer” submits the answer to be checked for correctness
- ☐ After clicking “check answer”, the user can see which components of a drink they entered correctly and incorrectly by having the box for that answer change visually
- ☐ Clicking on the new drink button shows a new drink and clears the input boxes

Stretch Features

- ☐ The user can type in their response in each box instead of having to click on the option, and typed inputs are properly validated

Resources

- React Forms
- Handling Events

Lab Instructions


Required Features

Step 0: Set up

In this step, we'll create a new React project, install required dependencies, and remove the extra code we won't need for this lab.

- ☐ Create a new React project by running the command `npm create vite@latest barista-app`
- ☐ Follow the prompts and instructions the Terminal gives you. This project will be with the React framework in JavaScript.
- ☐ Then in your terminal, run the following commands:

```
cd barista-app
npm install
npm run dev
```

 **Tip:** Running `npm run dev` will start up a server that stays active and constantly updates your project as you type. If you want to quit this behavior, press `ctrl + c` or delete the Terminal window in VS Code.

You should see this on your screen:



Vite + React

count is 0

Edit src/App.jsx and save to test HMR

Click on the Vite and React logos to learn more

In the `App.jsx` file in your newly created project:

- ☐ Remove everything inside the `return()` statement. When you check your local running version of the app, there should be nothing there.
- ☐ Delete the following line from the top of the file.

```
import reactLogo from './assets/react.svg'
```

Step 1: Create our React form structure

In this step, we will be creating our React form, which will allow users to input their guesses for the drink ingredients. We want to dynamically create this form.

- ☐ In the `src` folder, create a new folder called `Components`, which will hold all of our custom components so that we can use them in our app.
- ☐ In this `Components` folder, create a new file called `BaristaForm.jsx`.

Since all of our form details will be in this `BaristaForm` component, we need to import it into our `App.jsx` file.

In `App.jsx`:

- ☐ At the top of the file, import the `BaristaForm` component from the `Components` directory.

```
import BaristaForm from './Components/baristaForm';
```

Then, inside the `return` statement, create the following:

- ☐ A `div` with the class name `"title-container"`
 - ☐ Inside this `div`, create an `h1` with the class name `"title"` and text `"On My Grind"`
 - ☐ After the `h1`, create a `p` element with the text `"So you think you can barista? Let's put that to the test..."`
- ☐ After the `div`, add a `BaristaForm` component.

✨ AI Opportunity

▼ *Use AI as a pair programming partner* → Finishing the `return` statement


Need some help creating these elements? Tell Copilot what you are trying to do to have it help you fill in the `return` statement. For example:

You:

In the return statement, I need to write code that does the following:

1. A div with the class name "title-container"
2. Inside this div, create an h1 with the class name "title" and text "On My Grind"
3. After the h1, create a p element with the text "So you think you can barista? Let's put that to the test..."
4. After the div, add a BaristaForm component

Help me update my code with these elements.

▼ **Want to double-check your code? Compare what you have to this** 

```
<div>
  <div className="title-container">
    <h1 className="title">On My Grind</h1>
    <p>So you think you can barista? Let's put that to the test...</p>
  </div>
  <BaristaForm />
</div>
```

Now, we need to actually create the `BaristaForm` component.

- ☐ In the `BaristaForm.jsx` file, add this starter code:

```
import React, {Component, useState} from "react";
const BaristaForm = () => {

  return (
    <div>
    </div>
  );

};

export default BaristaForm;
```

Now that we have the bare bones set up, within our `div` tags in the `return` statement, we will add the basic parts of our form -- the `form` tags, the `New Drinks` button, and the `Check Answer` button. We need to use the `onClick` event handler in both buttons so that they will respond properly when we click them, but we will add this functionality later.

Inside the `div` tags in the `return` statement:

- ☐ Add opening and closing `form` tags.
- ☐ After the closing `form` tag, create a `button`:
 - ☐ Set the type to `"submit"`.
 - ☐ Give it the class names `"button"` and `"submit"`.
 - ☐ It should call the `onCheckAnswer()` function when it is clicked.
 - ☐ The button text should be `"Check Answer"`.
- ☐ After the `Check Answer` button, create another `button`:
 - ☐ Set the type to `"new-drink-button"`.
 - ☐ Give it the class names `"button"` and `"submit"`.
 - ☐ It should call the `onNewDrink()` function when it is clicked.
 - ☐ The button text should be `"New Drink"`.

AI Opportunity

▼ Use AI as a pair programming partner → Creating the buttons

Try using Copilot as a pair programmer to help you with this if you get stuck. You could try having it act as the driver or the navigator and interact with it similar to how you would when your driver or navigator is your peer sitting next to you. For example, let's say we want Copilot to act as the driver:

You:

You are an experienced software engineer. You and I will be pair programming on a project. I will be the navigator, and you will be the driver. As a navigator, when I explain the logic to you, I want you (as the driver) to implement it in JSX.

It will then respond to confirm its role and let you know it's willing to help. Then you can give it the task you want it to complete. For example:

You:

I need to create a button element that has a type "submit", class names "button" and "submit", calls the onCheckAnswer function when it is clicked, and sets the button text to "Check Answer". Write the code to create this button element.

Make sure to review the code it gives you. Does it make sense? Put in your project and test it. Does it work? Do you need to change anything to match the requirements?

▼ Want to double-check your code? Compare what you have to this [↗](#)

```
<div>
  <form >

    // we will fill this in soon!

  </form>

  <button type="submit" className="button submit" onClick={onCheckAnswer}>
    Check Answer
  </button>

  <button
    type="new-drink-button"
    className="button newdrink"
    onClick={onNewDrink}
  >
    New Drink
  </button>
</div>
```

And since we mentioned out `onNewDrink()` and `onCheckAnswer()` functions for our button event handlers, let's make some empty functions using arrow notation so that at least the function names exist in our component, even if we don't have any code in them currently. Try it on your own!

- ☐ Use arrow notation to write an empty `onNewDrink()` function.
- ☐ Use arrow notation to write an empty `onCheckAnswer()` function.

🌟 AI Opportunity

▼ Use AI to explain code concepts → Arrow notation examples

If you're not sure how to create these empty functions using arrow notation, ask Copilot for an example. For example:

You:

Can you show me an example of how to write a function using arrow notation in JavaScript?

Use this example to set up your empty `onNewDrink()` and `onCheckAnswer()` functions.

▼ Want to double-check your code? Compare what you have to this ↴

```
const onNewDrink = () => {  
  
};  
  
const onCheckAnswer = () => {  
  
};
```

Lastly, let's add the title to our form.

- ☐ Inside of the `return` statement and inside of our containing `div` tags, but before our `form` tags, let's add an `h2` title:

```
<h2>Hi, I'd like to order a:</h2>
```

And now we have the basic structure of our form!

📌 Checkpoint 1:

At the end of this step, you should be able to see your page title and two buttons to check the answer and reset to a new drink, but not the actual inputs for our form yet. This will be done in the next step.

Hi, I'd like to order a:

Check Answer

New Drink

Step 2: Dynamically create our form inputs with our different ingredient choices and place the user inputs into state variables

In this step, we will be creating the radio buttons that our users will be able to choose from in order to submit guesses for our drink ingredients. We will also need some state variables to keep track of the user choices so that we can compare them against the correct answers.

- First, right under the `const BaristaForm = () => {` that we use to create this component, create a state variable using `useState()` to handle all of the controlled inputs for our four basic ingredient categories -- `temperature`, `milk`, `syrup`, and `blended`.

```
const [inputs, setInputs] = useState({
  'temperature': '',
  'milk': '',
  'syrup': '',
  'blended': ''
});
```

Now that we can store what choices the user makes, we need to show them the list of options.

- Create a variable called `ingredients` underneath the new state variables.

```
const ingredients = {
  'temperature' : ['hot', 'lukewarm', 'cold'],
  'syrup': ['mocha', 'vanilla', 'toffee', 'maple', 'caramel', 'other', 'none'],
  'milk': ['cow', 'oat', 'goat', 'almond', 'none'],
  'blended': ['yes', 'turbo', 'no']
}
```

Note: There are some values in the lists above that do not appear in our real drinks list JSON and serve as fakeouts for users. You can add or change out the fakeout choices if you would like, but be sure to review the list of real drinks first to make sure that you are not taking out an answer choice that will actually be used in one of the drinks.

Next, we are going to create a new component to take in our `ingredients` and set up different blocks of answer choices for each list in `ingredients`.

- ☐ In the `Components` folder, create a new file called `RecipeChoices.jsx`.
- ☐ Add this starter code for the `RecipeChoices` component:

```
import React, { Component, useEffect, useState } from "react";

const RecipeChoices = () => {
  return (
    <div>

    </div>
  );
};

export default RecipeChoices;
```

We know that this component will be used to make different inputs via radio buttons, so we need to pass in the different answer choices to the radio buttons as props. We also need to save the user's selection from this nested component (the radio buttons) back in our parent component `BaristaForm` (as state). Because we are passing in a list of choices to be displayed as radio buttons, we can use `.map()` to match each choice with a radio button.

The `RecipeChoices` component uses the following four props:

- How we will handle change when different inputs are selected so that the selections get saved within the `BaristaForm` component's state
- A label describing the corresponding ingredient (e.g., `"milk"`)
- A list of answer choices
- A checked variable that deselects the user's choice when they request a new drink

- ☐ Add props between the `()` in the `const RecipeChoices` line.

```
const RecipeChoices({ handleChange, label, choices, checked })
```

- ❑ Inside the `div` tags in the `return` statement, loop through each of the `choices` (if the list is populated), and then create a list item that represents one radio button input.

```
<div className="radio-buttons">
  {choices &&
    choices.map((choice) => (
      <li key={choice}>
        <input
          id={choice}
          value={choice}
          name={label}
          type="radio"
          onChange={handleChange}
          checked={checked == choice}
        />
        {choice}
      </li>
    ))}
</div>
```

▼ 💡 Need more explanation on the input categories?

- `id` and `value` keep track of what our form is recognizing as our choice.
- `name` groups input buttons together so that the app recognizes they are all answers to the same question. The `name` attribute is what enforces radio buttons as mutually exclusive, so that when one is selected, any others with the same `name` will be deselected.
- `onChange` will let the form know what to do when the user selects a choice.
- `checked` keeps track of whether the radio button will be selected or deselected.

Everything in the `{ }` is what we will pass in from our parent component, `BaristaForm`.

Back in `BaristaForm.jsx`, we will use our `RecipeChoices` component to add actual questions and answers to our form.

For each of our four ingredients, `temperature`, `milk`, `syrup`, and `blended`, we will create a small title for that ingredient using `h3` tags, add an answer space `div` with a reference to the state variable of that ingredient so that the user selection is displayed above the form choices, and then a `RecipeChoices` component with real values for all of the props that we pass into that component.

- ☐ Add the following code for the `temperature` ingredient.

```
<h3>Temperature</h3>
<div className="answer-space" >
  {inputs["temperature"]}
</div>
<RecipeChoices
  handleChange={(e) => setInputs((prevState) => ({
    ...prevState,
    [e.target.name]: e.target.value,
  })))}
  label="temperature"
  choices={ingredients["temperature"]}
  checked={inputs["temperature"]}
/>
```

- ☐ Do this same thing for the rest of the four ingredients.

✨ AI Opportunity

▼ Use AI as a pair programming partner → Getting unstuck

AI tools like Copilot and ChatGPT work really well when they are given an example to base its answer on. If you get stuck doing the same for the rest of the ingredients, try asking Copilot for help:

You:

I wrote code to create a RecipeChoice component for the temperature ingredient. I need to do the same for the milk ingredient. Help me add the milk ingredient using the same structure as the temperature ingredient.

Make sure you have the `RecipeChoice.jsx` and `BaristaForm.jsx` files open when you ask Copilot for help. Copilot attempts to use the files you have open for context when answering questions, so you should have both of these open so it can provide more accurate responses.

📌 Checkpoint 2:

Once you have this, you should now have a form with each section and a list of answer choices to pick from in the form of radio buttons. When you select each radio button, its value will also appear above the answer choices, changing as you select different buttons.

The choices you have listed for each ingredient directly correspond to the `ingredients` dictionary you made above. By changing the dictionary, you can change what answer choices are listed.

Hi, I'd like to order a:

Temperature

- ☐ hot
- ☒ lukewarm
- ☐ cold

Syrup

- ☐ mocha
- ☐ vanilla
- ☐ toffee
- ☐ maple
- ☐ caramel
- ☐ other
- ☐ none

Milk

- ☐ cow
- ☐ oat
- ☐ goat
- ☐ almond
- ☐ none

Blended

- ☐ yes
- ☐ turbo
- ☐ no

Hi, I'd like to order a:

Temperature

lukewarm

- ☐ hot
- ☒ lukewarm
- ☐ cold

Syrup

mocha

- ☒ mocha
- ☐ vanilla
- ☐ toffee
- ☐ maple
- ☐ caramel
- ☐ other
- ☐ none

Milk

cow

- ☒ cow
- ☐ oat
- ☐ goat
- ☐ almond
- ☐ none

Blended

yes

- ☒ yes
- ☐ turbo
- ☐ no

Step 3: Importing our JSON drinks list to the form and adding the ability to grab new drinks from the list with the 'New Drink' button

In this step, we will be adding the list of drinks that users will be trying to guess the ingredients to. We have provided a `drinks.json`, with the real Starbucks drinks options that you will need to use in this lab. You can download the file from this [Unit 3 GitHub Repo](#).

First, we need to import our file with our different drinks and their ingredients.

- ☐ At the top of `BaristaForm.jsx`, import the `drinks.json` file.

```
import drinksJson from "../drinks.json"
```

- ☐ Create state variables to keep track of what drink we have currently and what the true recipe is behind that drink. How can we create them using `useState()`?

✨ AI Opportunity

▼ Use AI as a pair programming partner → Creating state variables

If you're stuck, try asking Copilot how we should do this. For example:

You:

I need to create state variables to keep track of the current drink and what the true recipe is behind that drink. Can you show me how I should create these?

- ▼ 💡 **Hint:** Look at the data types that the `drink.json` file uses for the `name` key and the `ingredients` key.

```
const [currentDrink, setCurrentDrink] = useState('');  
  
const [trueRecipe, setTrueRecipe] = useState({});
```

Now we are going to add some logic to the `onNewDrink()` function that we created earlier. When we request a new drink, we want to clear out the current values of our state variables since they apply only to the previous drink and that will deselect whatever radio buttons the user has chosen. Then, we want to get another randomly chosen drink from our `drinksJson` file.

- ☐ First, create a new function to select another random drink called `getNextDrink()`.

```
const getNextDrink = () => {  
  
}
```

- ☐ Within this function, we'll use `Math` to get a random number from 0 to the numerical length of the `drinks` list in our `drinksJson`.

```
let randomDrinkIndex = Math.floor(Math.random() * drinksJson.drinks.length);
```

- ☐ Set the `currentDrink` state variable to the name at the random index of the `drinks` list in our `drinksJson`.

- ☐ Set the `trueRecipe` state variable to the `ingredients` associated with that drink at the random index of the `drinks` list in our `drinksJson`.

```
setCurrentDrink(drinksJson.drinks[randomDrinkIndex].name);
setTrueRecipe(drinksJson.drinks[randomDrinkIndex].ingredients);
```

- ☐ Back in our `onNewDrink()` function, we want to set our state variable for our ingredients to be empty, and then call `getNextDrink()`

```
setInputs({
  'temperature': '',
  'milk': '',
  'syrup': '',
  'blended': '' });

getNextDrink();
```

The last thing we want to do now is render our `currentDrink` onto our page so that users can actually see it.

- ☐ Create a small `div` right under the `"Hi, I'd like to order a:"` title to do this with the class name `"drink-container"`. Inside this `div`, add the following elements:
 - ☐ An `h2` with the class name `"mini-header"` with the `currentDrink` state variable as its text.
 - ☐ A button with the type `"new-drink-button"` and class names `"button"` and `"newdrink"` that calls the `onNewDrink()` function when the button is clicked. Set the text for the button to `"🔄"`


✨ AI Opportunity

▼ Use AI as a pair programming partner → Getting unstuck

Try using ChatGPT as a pair programmer to help you with this if you get stuck. Make sure to review the code it gives you. Does it make sense? Put it in your project and test it. Does it work? Do you need to change anything to match the requirements?

▼ Want to double-check your code? Compare what you have to this [🔗](#)


```

<div className="drink-container">
  <h2 className="mini-header">{currentDrink}</h2>
  <button
    type="new-drink-button"
    className="button newdrink"
    onClick={onNewDrink}
  >
    
  </button>
</div>

```

📌 Checkpoint 3:

After this step, if you press , you should now be able to see a new drink from the drinks json pop up for users to guess what the ingredients are for it.

Note: Since there is no logic in this app so far to make a drink pop up by default, you will always have to click to have a drink be displayed and start the quiz. Feel free to explore ways to have a drink by default if you would like!



Hi, I'd like to order a:

unicorn frappuccino

Temperature

- ☐ hot
- ☒ lukewarm
- ☐ cold

Syrup

- ☐ mocha
- ☐ vanilla
- ☐ toffee
- ☐ maple
- ☐ caramel
- ☐ other
- ☐ none

Milk

- ☐ cow
- ☐ oat
- ☐ goat
- ☐ almond
- ☐ none

Blended

- ☐ yes
- ☒ turbo
- ☐ no

Step 4: Adding functionality to check user inputs against the true result on form submission

In this step, we will be adding functionality that checks the user choices in our form and compares them with the real recipe when we click the button. We will also create a way for an ID to be placed on the answer-space when a user has the wrong answer for an ingredient.

First, we need some more state variables to represent whether we have a correct ingredient, and we can do this with just simple strings. To make this clearer, we can make a different state variable for the correctness of each ingredient.

- ☐ Create state variables to keep track of whether we have the correct temperature, correct syrup, correct milk, and correct blended.

✨ AI Opportunity

▼ Use AI to explain code concepts → State variables

Don't forget, you can use Copilot to help you if you forget how to create a state variable. Try asking for an example and use that to help you create these.

▼ Want to double-check your code? Compare what you have to this ↴

```
const [correct_temp, setCheckedTemperature] = useState('');
const [correct_syrup, setCheckedSyrup] = useState('');
const [correct_milk, setCheckedMilk] = useState('');
const [correct_blended, setCheckedBlended] = useState('');
```

We also need to reset these state variables when we order a new drink so that a user's errors on one drink do not carry over to others.

- ☐ In your `onNewDrink()` function, set each state variable to empty strings.

▼ Want to double-check your code? Compare what you have to this ↴

```
setCheckedTemperature('');
setCheckedSyrup('');
setCheckedMilk('');
setCheckedBlended('');
```

With this, we need to create some logic in our `onCheckAnswer()` function to go through each state variable we have and check it against the `trueRecipe` that we got from our `drinksJson`.

- ☐ Add this code to do this for `temperature` :

```
if (trueRecipe.temp !== inputs['temperature']){
  setCheckedTemperature('wrong');
}
else {
  setCheckedTemperature("correct");
}
```

- ☐ Finish this pattern for the other three ingredients.

✨ AI Opportunity

▼ Use AI as a pair programming partner → Repeating a pattern

Since you have an example, you can use this with Copilot to help you follow the same structure for the other ingredients. For example:

You:

I have written code to set `correct_temp` for the temperature ingredient. I need to do the same for the milk ingredient. Help me add the milk ingredient using the same structure as the temperature ingredient.

And the last step, is we need to make it possible to visually change our answer spaces when we check the answers by adding our `correct_*` state variables as IDs in the `answer-space` divs. This will allow us to make different CSS selectors for the `#correct` and `#wrong` IDs to trigger a visual change.

- ☐ To do this, add the following code to each `<div className="answer-space" >` for each ingredient. Here is an example for this with the state variable for the `temperature` .

```
id={correct_temp}
```

▼ 💡 Here's what it should look like for the `temperature` category!

```
<h3>Temperature</h3>

<div className="answer-space" id={correct_temp}>

  {inputs['temperature']}
</div>
```

Do something similar for the rest of the ingredients, of course replacing `correct_temp` with `correct_milk`, `correct_syrup`, and `correct_blended` where relevant.

📌 Checkpoint 4:

After this step, you should now have functionality to check the correct answers for each ingredient, and added variable class names that we can then use to affect the color of that answer space using CSS.

You won't see any visual updates to your app at this point corresponding to whether an answer is correct or incorrect.

Step 5: Add CSS styling to make our form organized and visibly appealing

In this step, we will be organizing our app using Flexbox and adding other visual changes to make it unique!

In order to use Flexbox, we will need to create a container class for everything in our form and also create mini-containers around each set of ingredient answer spaces and answer choices so they are treated each as one unit to be moved. We would not want our answer choices for an ingredient to not be aligned properly under the title.

- ☐ Change the following code snippet below:

```
<form>
```

to

```
<form className="container">
```

- ☐ Also change the following code snippet below:

```
<h3>Temperature</h3>
<div className="answer-space" id={correct_temp}>
  {inputs["temperature"]}
</div>
<RecipeChoices
  handleChange={(e) =>
    setInputs((prevState) => ({
      ...prevState,
      [e.target.name]: e.target.value,
    }))
  }
  label="temperature"
  choices={ingredients["temperature"]}
  currentVal={inputs["temperature"]}
/>
```

to

```
<div className="mini-container">
  <h3>Temperature</h3>
  <div className="answer-space" id={correct_temp}>
    {inputs["temperature"]}
  </div>
  <RecipeChoices
    handleChange={(e) =>
      setInputs((prevState) => ({
        ...prevState,
        [e.target.name]: e.target.value,
      }))
    }
    label="temperature"
    choices={ingredients["temperature"]}
    currentVal={inputs["temperature"]}
  />
</div>
```

- ☐ After adding in these new classes, you can now create two CSS selectors for `.container` and `.mini-container` and give them both these rules:

- ☐ Set `display` to `flex`
- ☐ Set `justify-content` to `space-evenly`

- ☐ For the `.container` selector only, add:

- ☐ Set `align-items` to `flex-start`
- ☐ Set `margin` to `auto`
- ☐ Set `position` to `relative`

☐ And for the `.mini-container` selector only, add:

☐ Set `flex-direction` to `column`

☐ Set `margin` to `50px;`

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Flexbox styling examples

Need some help creating these CSS rules? Don't forget, you can let Copilot know what you're trying to do and ask for some examples to get an idea of the code you need.

After this, you can make other stylistic changes, like:

- ☐ Giving your `wrong` and `correct` id selectors new background colors to visually indicate when a user gets an ingredient guess wrong or right.
- ☐ Making your `answer-space` class have a background color and set width and height so that it is always visible.
- ☐ For the `li` elements, setting their `list-style` to `none` so there are no weird bullet points next to your radio buttons

But other than that, you have full creative freedom to make your app your own!

✨ AI Opportunity

▼ *Use AI to brainstorm ideas for your code*

If you want to get some ideas, try giving Copilot some of your code and asking for styling suggestions. Be sure to ask questions about its suggestions and test them in your app to see how they look.

▼ 💡 If you want some ideas of what to do, the example project that you have been seeing screenshots of throughout these instructions incorporates these rules...

```
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600&displ
```

```
#root {  
  font-family: 'Poppins', sans-serif;  
}
```

```
body{  
  background-color: white;  
  color: black;  
}
```

```
li {  
  list-style: none;  
}
```

```
.container {  
  display: flex;  
  justify-content: space-evenly;  
  align-items: flex-start;  
  margin: auto;  
  position: relative;  
}
```

```
.mini-container {  
  display: flex;  
  justify-content: space-evenly;  
  flex-direction: column;  
  margin: 25px;  
  padding: 15px;  
  border-radius: 15px;  
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);  
}
```

```
.mini-header{  
  height: 25px;  
  padding-top: 10px;  
  padding-bottom: 10px;  
  background-color: rgb(226, 195, 147);  
  text-align: center;  
  border-radius: 10px;  
  width: 93%;  
  margin: 0;  
}
```

```
#wrong {  
  background-color:rgb(219, 93, 93);  
}
```

```
#correct {  
  background-color:rgb(191, 116, 191);  
}
```

```

.answer-space {
  background-color: rgb(226, 195, 147);
  height: 20px;
  padding-top: 10px;
  padding-bottom: 10px;
  width: auto;
  margin-bottom: 20px;
  border-radius: 15px;
  margin-left: auto;
  clear: both;
}

li {
  margin-top: 3px;
  margin-bottom: 3px;
  margin-right: 10%;
  margin-left: 10%;
  background-color: #FFDD00;
  border-radius: 15px;
}

button.newdrink {
  width: auto;
  text-align: center;
}

.button {
  background-color: rgba(138, 98, 12, 0.942);
  font-weight: bold;
  color: white;
}

.title {
  display: inline-block;
  margin-bottom: 0;
}

.title-container {
  text-align: left;
  margin-bottom: 50px;
}

.drink-container {
  display: flex;
  justify-content: space-between;
}

```

📌 Checkpoint 5:

After this, you should now have a beautiful app that allows you to quiz yourself on popular Starbucks drinks!



On My Grind

So you think you can barista? Let's put that to the test...

Hi, I'd like to order a:

toffee almond cappuccino



Temperature



- ☐ hot
- ☐ lukewarm
- ☐ cold

Syrup



- ☐ mocha
- ☐ vanilla
- ☐ toffee
- ☐ maple
- ☐ caramel
- ☐ other
- ☐ none

Milk



- ☐ cow
- ☐ oat
- ☐ goat
- ☐ almond
- ☐ none

Blended



- ☐ yes
- ☐ turbo
- ☐ no

Check Answer

🎉 Congratulations, you've completed this lab! 🎉

If you have time left over, continue on to the stretch features to customize and improve your app!

Stretch Features

Step 6: Switch out input radio buttons for input text box with extra user input validation

In this step, we'll allow users to type in their answers instead of selecting the appropriate answer choice as a radio button. We will be making our input changes for this step in `recipeChoices.jsx`.

- ❑ Since we now only require one input text box instead of a different input radio button for each choice, we can comment out/get rid of the `input` elements in our `.map()` function for all of our choices. All we need to do in that `.map()` function is list out our choices.

```
{choices &&
  choices.map((choice) => (
    <li key={choice}>

      {choice}

    </li>
  ))}
```

- ❑ Next, above where we list out our choices, we need to make another input tag that is for text, and many of the arguments needed for this input tag are the same as the ones needed for the radio button.

```
<input
  type="text"
  name={label}
  value={currentVal}
  placeholder="Guess the ingredient..."
  onChange={handleChange}
  className = "textbox"
/>
```

- ❑ Next, we want to add some extra validation if a user happens to make a spelling error or generally input some text that doesn't match any of the potential choices. In our `onCheckAnswer()` function, we want to add and `alert()` if the input doesn't match anything in our original list of ingredient choices. Try this on your own!

✨ AI Opportunity

- ▼ *Use AI to explain code concepts* → `Array.prototype.includes()`

Try asking Copilot for examples using `Array.prototype.includes()`. Try to modify the examples to fit what you're trying to do here.

If you're still stuck, try breaking this step down to tell Copilot what you're trying to do. Use it like a pair programmer as the driver to generate solutions and test them in your code.

- ▼ 💡 **Need a bigger hint?**

Here is the example for temperature:

```
if (!ingredients['temperature'].includes(inputs['temperature'])) {  
  alert("For temperature, that isn't even an option!")  
}
```

Feel free to change this from an alert to some other behavior, like highlighting the text box or some other cool feature!

- ☐ Lastly, feel free to go back to your `App.css` and change up some selectors if you want the text box to look or be placed differently. It already has a `.textbox` class so you can edit the look or position of the text input using that class name as a selector.

📌 **Checkpoint 6:** At the end of this step, you should have your list of answer choices with a text box above them, that is able to take in user input and still respond as previously by checking the answers and alerting the user if their input doesn't match what is present in the list of choices.



On My Grind

So you think you can barista? Let's put that to the test...

Hi, I'd like to order a:

Temperature

Guess the ingredient...

hot

lukewarm

cold

Syrup

Guess the ingredient...

mocha

vanilla

toffee

maple

caramel

other

none

Milk

Guess the ingredient...

cow

oat

goat

almond

none

Blended

Guess the ingredient...

yes

turbo

no

Check Answer

127.0.0.1:5173 says
For temperature, that isn't even an option!

OK



On My Grind

So you think you can barista? Let's put that to the test...

Hi, I'd like to order a:

salted caramel frappuccino



Temperature

narwhals

narwhals

hot

lukewarm

cold

Syrup

mocha

mocha

mocha

vanilla

toffee

maple

caramel

other

none

Milk

cow

cow

cow

oat

goat

almond

none

Blended

yes

yes

yes

turbo

no

Check Answer

🎉 Congratulations 🎉

You've completed the lab AND stretch goals! 🚀

💡 **Tip:** Remember to come back and reference this lab when you need to do similar things in your project!