

Front End

Class 7

August 25, 2021

Agenda

- Studio review
- Kahoot
- Code walkthrough with slides
- Studio

Concepts

- <https://css-tricks.com/whats-the-difference/>
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors#basic_selectors
- <https://web.dev/learn/css/selectors/>
- <https://www.codecademy.com/courses/learn-css/lessons/learn-css-selectors/exercises/pseudo-class>

To make styles easy to edit, it's best to style with a type selector, if possible. If not, add a class selector. If that is not specific enough, then consider using an ID selector.

Design System is made of

- Design Standards
- Code Library
- Usage Guidance
- Design Toolkits, Assets, and Templates
- Themes
- Governance Model

Why create a Design System

- With the continued rapid growth of devices and with the blurring of “Web” and “app” lines, it’s as complex as ever to design and build good products in a sustainable, cost-efficient way. Design systems have sprung up to help teams do this, and some like Material Design by Google have even been open-sourced.
- Why not use one of these systems like Material or Bootstrap as-is? There are both higher-level and more practical reasons for going custom. At the most fundamental level, having Element stand on its own as Bayer’s corporate design system gives us the independence to set a design and code direction that we believe in and that best matches our business.
- Element builds on Material, incorporating a lot of its out-of-the-box usability and quality code that is becoming a standard across the industry, but it isn’t beholden to that system. Where we disagree with a decision Google made for its business or where we want to deviate from what Google decided to either introduce or deprecate, we can take a different route. This may happen because our own research and experience differs from Google’s or because our product needs are simply different. Like many other companies that have created their own systems, we’re able to do what makes most sense for us.
- Lyft had similar reasons for building on top of Material Design rather than just using it as it is.
- "A design system isn’t like a code of ethics. It’s more like a set of company values. It’s good for everyone to share the same ethics. But it’s not necessarily appropriate for every company to share the same values,"
- ALEX MACDUFF, UX INTERACTION DESIGNER
- [Cookie Cutter Design System](#)
- Beyond this independence, there are more basic reasons for a custom system:
- We can create design and code kits and that fit our own workflows and technologies
- We can build new components where they don’t exist
- We can own our roadmap and remain generally responsive to our users
- We can create themes to express our many brands
- We set guidelines that matter most to us and our users

Benefits of a Design System

- Design systems enable teams to make higher-quality products more quickly by standardizing and scaling good design and development. This is especially important in large organizations where teams often work in silos on the same kinds of routine things that may not create a lot of value.
- By automating common elements and focusing teams on the kinds of things that are unique to their products, design systems:
 - Improve usability and accessibility
 - Increase efficiency
 - Reduce costs
 - Enable more rapid prototyping and innovation
 - Improve collaboration and cross-organizational knowledge sharing

Benefits of a Design System

- <https://medium.com/@zeolearn/6-best-reactjs-based-ui-frameworks-9c780b96236c>
- <https://medium.com/@gavizoid16/choosing-your-css-library-with-react-fdb9a02f9aac>
- <https://www.codeinwp.com/blog/react-ui-component-libraries-frameworks/>

Conditional & Dynamic Styles

Styled Components

CSS Modules

Dynamic inline styles

Course Goal

Add Goal

Do all exercises!

Finish the course!

Dynamic inline styles

```
const formSubmitHandler = (event) => {  
  event.preventDefault();  
  if (enteredValue.trim().length === 0) {  
    return;   
  }  
  props.onAddGoal(enteredValue);  
};
```

Need to provide some input to user.

Use isValid to store whether input is valid or not and display red validation error in the case input isn't valid.

Dynamic inline styles

```
return (  
  <form onSubmit={formSubmitHandler}>  
    <div className="form-control">  
      <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</  
      <input type="text" onChange={goalInputChangeHandler} />  
    </div> (JSX attribute) type: string  
    <Button type="submit">Add Goal</Button>  
  </form>  
);  
};
```

Dynamic inline styles

Course Goal

Add Goal

Do all exercises!

Finish the course!

Dynamic inline styles

Course Goal

Add Goal

Do all exercises!

Finish the course!

Dynamic inline styles

```
return (  
  <form onSubmit={handleSubmit}>  
    <div className="form-control">  
      <label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>  
      <input  
        style={{  
          borderColor: !isValid ? "red" : "black",  
          background: !isValid ? "salmon" : "transparent",  
        }}  
        type="text"  
        onChange={goalInputChangeHandler}  
      />  
    </div>  
  </form>  
)
```

Course Goal



Add Goal

Do all exercises!

Finish the course!

Dynamic inline styles

Inline css has highest priority and it overrides a bunch of css.

We can work on reset functionality and then change from inline styles.

```
const goalInputChangeHandler = (event) => {  
  if (event.target.value.trim() > 0) {  
    setIsValid(true);  
  }  
  setEnteredValue(event.target.value);  
};
```


Dynamic inline styles

Course Goal

Add Goal

Do all exercises!

Finish the course!

Setting css styles dynamically

We could create and apply a css class dynamically (invalid)

```
.form-control.invalid input {  
  border-color:  red;  
  background:  #fad0ec;  
}
```

```
.form-control.invalid label {  
  color: red;  You, 11 minutes  
}
```

```
return (
  <form onSubmit={formSubmitHandler}>
    <div className={`form-control ${!isValid ? "invalid" : ""}`}>
      <label>Course Goal</label>
      <input type="text" onChange={goalInputChangeHandler} />
    </div>
    <Button type="submit">Add Goal</Button>
  </form>
);
```

Styled Components

It's a package. Build components with styles attached to it.

```
npm i --save styled-components
```

<https://styled-components.com/docs>

Styled Components

<https://www.freecodecamp.org/news/a-quick-introduction-to-tagged-template-literals-2a07fd54bc1d/>

Tagged Template Literals

Executed as a method behind the scene. And will be available as a new component (button in our example).

Multi-line syntax

No need for selector for element `.button` and for pseudo selector use `&`.

Styles are unique for those components.

Styled Components

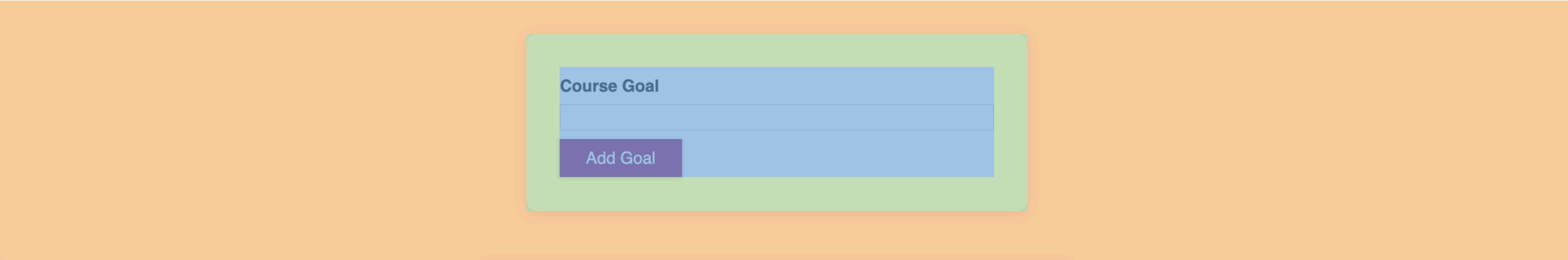
```
import styled from "styled-components";

const Button = styled.button`
  font: inherit;
  padding: 0.5rem 1.5rem;
  border: 1px solid #8b005d;
  color: white;
  background: #8b005d;
  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
  cursor: pointer;
}

&:focus {
  outline: none;
}

&:hover,
&:active {
  background: #ac0e77;
  border-color: #ac0e77;
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
};

export default Button;
```



ection#goal-form 480 × 169.76

ml body div#root div section#goal-form form button.sc-bdnxRM.dpKYyZ

Elements Sources Console Network Performance Memory Security Lighthouse Application Augury

```
<!DOCTYPE html>
<html lang="en">
<head>...</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root">
    <div>
      <section id="goal-form">
        <form>
          <div class="form-control ">...</div>
          <button type="submit" class="sc-bdnxRM dpKYyZ">Add Goal</button> == $0
        </form>
      </section>
    </div>
  </div>
<!--
  This HTML file is a template.
  If you open it directly in the browser, you will see an empty page.
-->
```

Styles Computed Layout Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

element.style {

.dpKYyZ {

font: inherit;

padding: 0.5rem 1.5rem;

border: 1px solid #8b005d;

color: white;

background: #8b005d;

box-shadow: 0 0 4px rgb(0 0 0 / 26%);

cursor: pointer;

}

* {

box-sizing: border-box;

}

button {

appearance: auto;

-webkit-writing-mode: horizontal-tb !important;

index.css:1

user agent stylesheet

Styled Components and Dynamic Props

Using multiple styled component in different places. Same kind of thing for CourseInput.

```
import Button from "../../UI/Button/Button";
import "./CourseInput.css";

const FormControl = styled.div`
  margin: 0.5rem 0;

  & label {
    font-weight: bold;
    display: block;
    margin-bottom: 0.5rem;
  }

  & input {
    display: block;
    width: 100%;
    border: 1px solid #ccc;
    font: inherit;
    line-height: 1.5rem;
    padding: 0 0.25rem;
  }

  & input:focus {
    outline: none;
    background: #fad0ec;
    border-color: #8b005d;
  }

  & .invalid input {
    border-color: red;
    background: #fad0ec;
  }

  & .invalid label {
    color: red;
  }
`;
```

Styled Components and Dynamic Props

Using multiple styled component in different places.

Same kind of thing for CourseInput.

```
return (  
  <form onSubmit={formSubmitHandler}>  
    <FormControl className={!isValid && "invalid"}>  
      <label>Course Goal</label>  
      <input type="text" onChange={goalInputChangeHandler} />  
    </FormControl>  
    <Button type="submit">Add Goal</Button>  
  </form>  
);
```

You, a day ago • new version

Styled Components and Dynamic Props

Can also use props

```
import React, { useState } from "react";
import styled from "styled-components";

import Button from "../../UI/Button/Button";
import "../CourseInput.css";

const FormControl = styled.div`
  margin: 0.5rem 0;

  & label {
    font-weight: bold;
    display: block;
    margin-bottom: 0.5rem;
    color: ${(props) => (props.invalid ? "red" : "black")};
  }

  & input {
    display: block;
    width: 100%;
    border: 1px solid ${(props) => (props.invalid ? "red" : "#ccc")};
    background: ${(props) => (props.invalid ? "fadedec" : "transparent")};
    font: inherit;
    line-height: 1.5rem;
    padding: 0 0.25rem;
  }

  & input:focus {
    outline: none;
    background: #fadedec;
    border-color: #8b005d;
  }
`;
```

```
return (
  <form onSubmit={formSubmitHandler}>
    <FormControl invalid={!isValid}>
      <label>Course Goal</label>
      <input type="text" onChange={goalInputChangeHandler} />
    </FormControl>
    <Button type="submit">Add Goal</Button>
  </form>
);
```

Styled Components and Media queries

Media query for switching between big and smaller devices.

Using media query!
@media.

```
You, seconds ago | 1 author (You)
import styled from "styled-components";

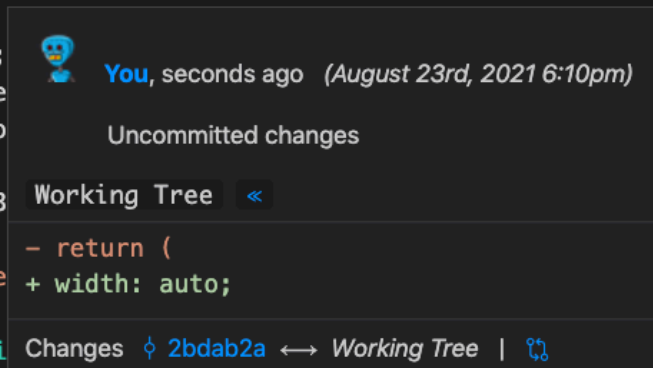
const Button = styled.button`
  width: 100%;
  font: inherit;
  padding: 0.5rem;
  border: 1px solid #ccc;
  color: white;
  background: #000;
  box-shadow: 0 0 0px #000;
  cursor: pointer;

  @media (min-width: 768px) {
    width: auto;
  }

  &:focus {
    outline: none;
  }

  &:hover,
  &:active {
    background: #ac0e77;
    border-color: #ac0e77;
    box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
  }
`;

export default Button;
```



Course Goal

dddf

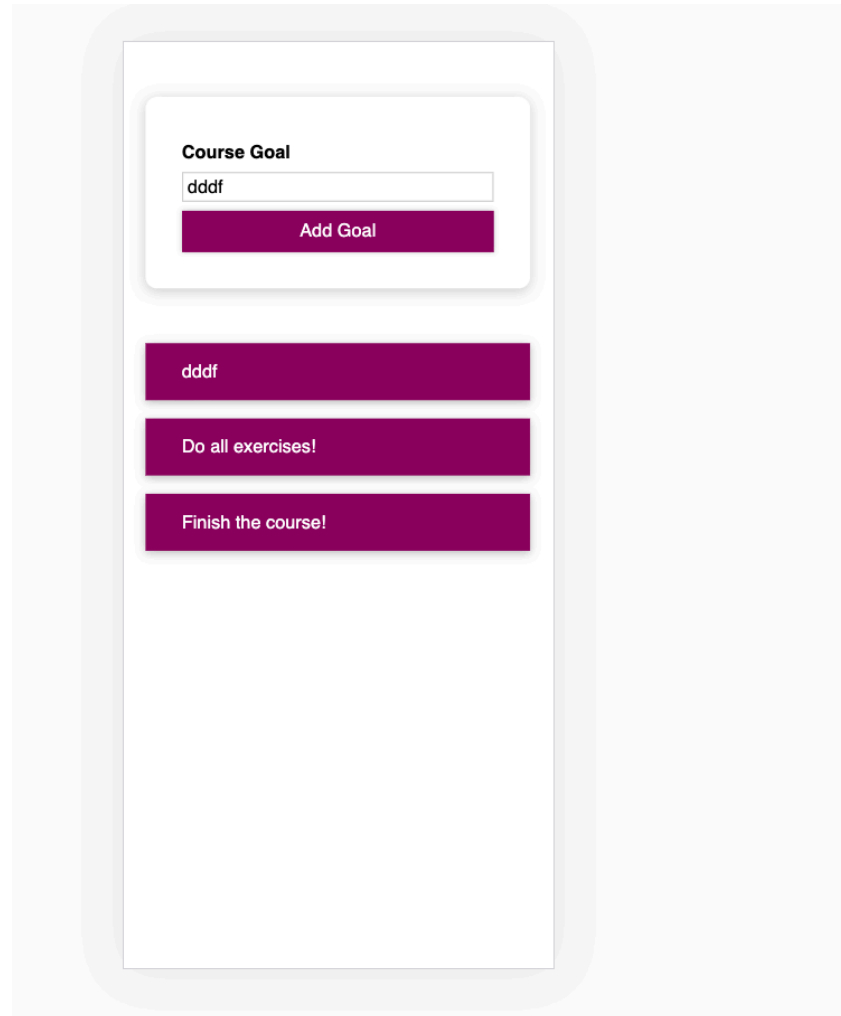
Add Goal

dddf

Do all exercises!

Finish the course!

Styled Components and Media queries

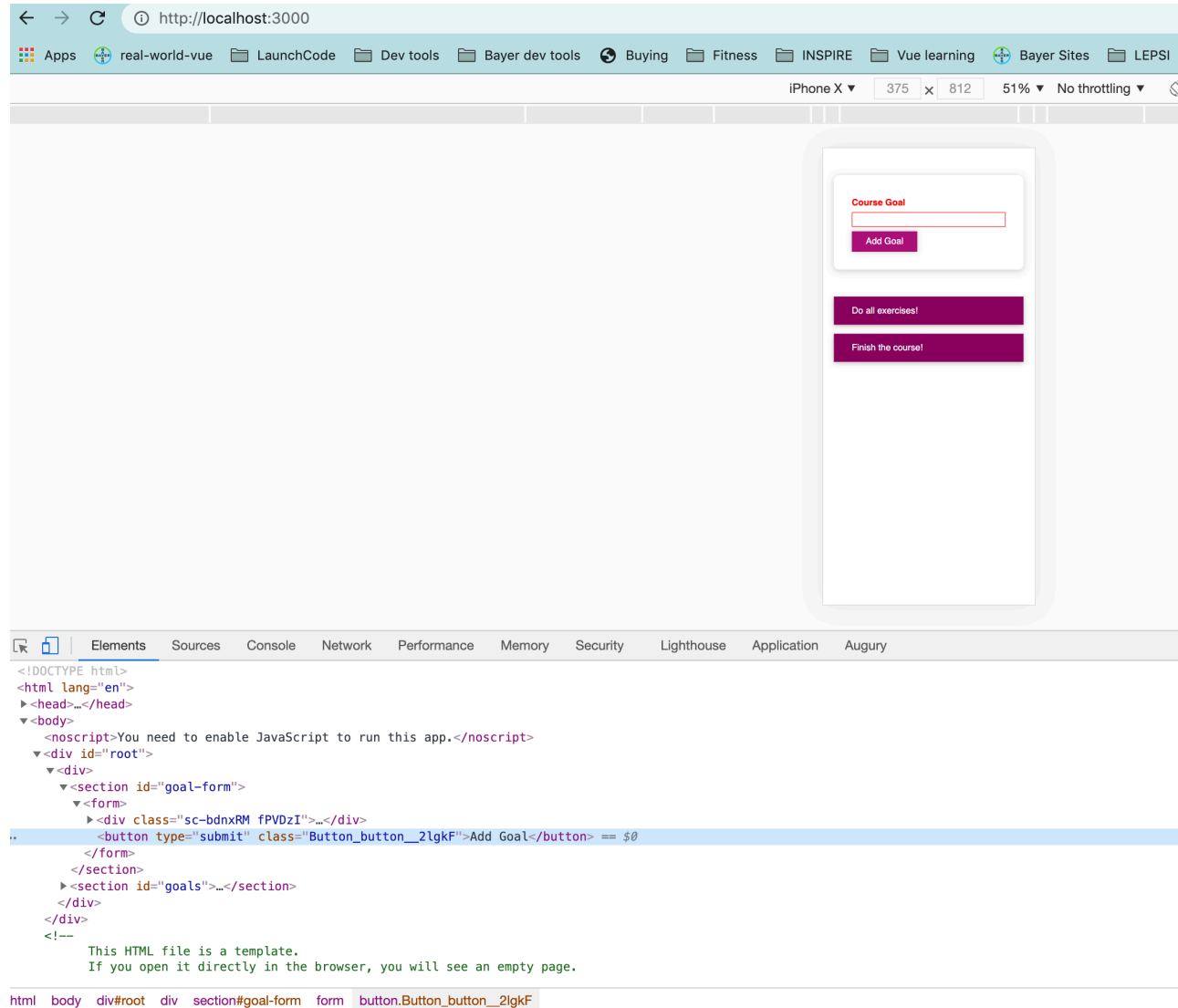


The image shows a mobile application interface with a light gray background. A white rounded rectangle represents the app screen. At the top, a section titled 'Course Goal' contains a text input field with the placeholder text 'dddf' and a purple button labeled 'Add Goal'. Below this, there is a list of three purple rectangular items, each containing white text: 'dddf', 'Do all exercises!', and 'Finish the course!'. The bottom half of the screen is empty white space.

CSS Modules.

```
components > src > Button > Button.module.css > Button
You, a minute ago | 1 author (You)
1 import React from "react";
2 import styles from "./Button.module.css";
3 // import styled from "styled-components";
4
5 // const Button = styled.button`
6 //   width: 100%;
7 //   font: inherit;
8 //   padding: 0.5rem 1.5rem;
9 //   border: 1px solid #8b005d;
10 //   color: white;
11 //   background: #8b005d;
12 //   box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
13 //   cursor: pointer;
14
15 //   @media (min-width: 768px) {
16 //     width: auto;
17 //   }
18 // }
19
20 // &:focus {
21 //   outline: none;
22 // }
23
24 // &:hover,
25 // &:active {
26 //   background: #ac0e77;
27 //   border-color: #ac0e77;
28 //   box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
29 // `;
30
31 const Button = (props) => {
32   return (
33     <button type={props.type} className={styles.button} onClick={props.onClick}>
34       {props.children}
35     </button>
36   );
37 };
38
39 export default Button;
```

CSS Modules.



Dynamic Styles with CSS Modules.

You, a minute ago | 1 author (You)

```
import React, { useState } from "react";
// import styled from "styled-components";

import Button from "../../UI/Button/Button";
// import "./CourseInput.css";
import styles from "./CourseInput.module.css";

// const FormControl = styled.div`
//   margin: 0.5rem 0;

//   & label {
//     font-weight: bold;
//     display: block;
//     margin-bottom: 0.5rem;
//     color: ${(props) => (props.invalid ? "red" : "black")};
//   }

//   & input {
//     display: block;
//     width: 100%;
//     border: 1px solid ${(props) => (props.invalid ? "red" : "#ccc")};
//     background: ${(props) => (props.invalid ? "fad0ec" : "transparent")};
//     font: inherit;
//     line-height: 1.5rem;
//     padding: 0 0.25rem;
//   }

//   & input:focus {
//     outline: none;
//     background: #fad0ec;
//     border-color: #8b005d;
//   }
// `;
```

Dynamic Styles with CSS Modules.

```
const CourseInput = (props) => {
  const [enteredValue, setEnteredValue] = useState("");

  const [isValid, setIsValid] = useState(true);

  const goalInputChangeHandler = (event) => {
    if (event.target.value.trim().length > 0) {
      setIsValid(true);
    }
    setEnteredValue(event.target.value);
  };

  const formSubmitHandler = (event) => {
    event.preventDefault();
    if (enteredValue.trim().length === 0) {
      setIsValid(false);
      return;
    }
    props.onAddGoal(enteredValue);
  };

  return (
    <form onSubmit={formSubmitHandler}>
      <div
        className={` ${styles["form-control"]} ${!isValid && styles.invalid}`}
      >
        <label>Course Goal</label>
        <input type="text" onChange={goalInputChangeHandler} />
      </div>
      <Button type="submit">Add Goal</Button>
    </form>
  );
};

export default CourseInput;
```


Dynamic Styles with CSS Modules. Media.

```
1  .button {
2    width: 100%;
3    font: inherit;
4    padding: 0.5rem 1.5rem;
5    border: 1px solid #8b005d;
6    color: white;
7    background: #8b005d;
8    box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
9    cursor: pointer;
10 }
11
12 .button:focus {
13   outline: none;
14 }
15
16 .button:hover,
17 .button:active {
18   background: #ac0e77;
19   border-color: #ac0e77;
20   box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
21 }
22
23 @media (min-width: 768px) {
24   .button {
25     width: auto;
26   }
27 }
```

Studio