Getting Started with Emotion

Learn how to apply CSS-in-JS to your React app with the Emotion library.

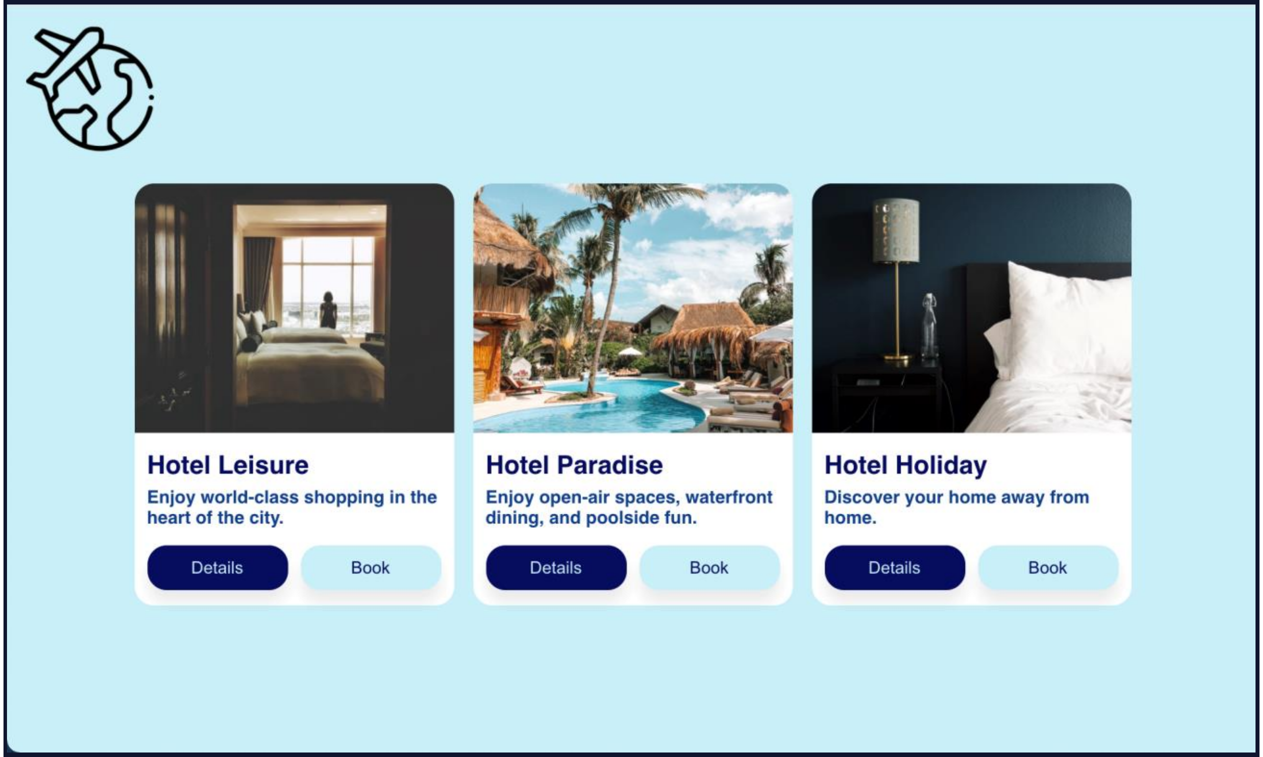
- Language: JavaScript ES6
- Frameworks: React v17.01.1, @emotion/react v11.5.0, @emotion/styled v11.3.0
- Duration: 30 minutes
- Author: Rob Merrill
- Publish Date: Nov 17, 2021

Introduction

In this tutorial, you will style a React app with the Emotion library as an example of using CSS-in-JS. You will be styling a travel app that features three hotel cards. Each card contains a main image, title, description, and buttons prompting the user to view more details about the hotel or book a room.

You will move step-by-step through the process of setting up Emotion and use the CSS-in-JS syntax to apply styles to an existing React app. You will go over the `css` prop, styled components, composition, theming, and animations.

Here is a screenshot of what the styled React app will look like by the end of this tutorial:


Hotel Leisure
Enjoy world-class shopping in the heart of the city.
Details
Book
Hotel Paradise
Enjoy open-air spaces, waterfront dining, and poolside fun.
Details
Book
Hotel Holiday
Discover your home away from home.
Details
Book

For now, the React app will just be a static website, as the focus is to learn how to apply CSS-in-JS to a React app.

Let's get started!

Download the Starter and Solution Code

Download the folder containing the starter and solution code of the project.

The **starter-code** folder contains code to help you get started. Follow the instructions below to complete the project.

We've also included a solution code in the **solution-code** folder to help guide you if you get stuck. You can also compare your solution to our solution after you complete the project.

Project Setup

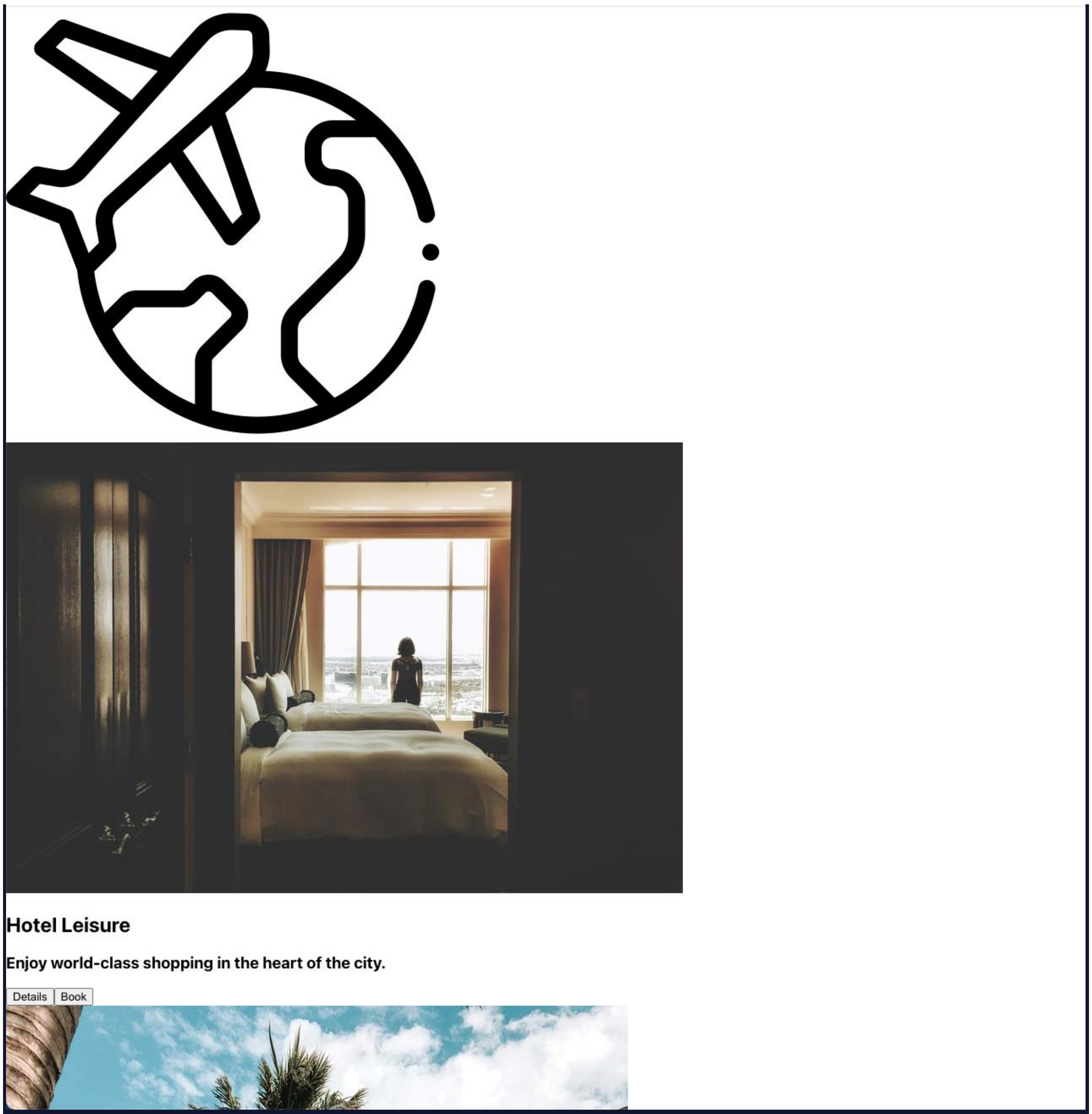
From the root of the **starter-code** folder, run the following command to install the necessary dependencies:

```
npm install
```

Next, start the app with the following command:

```
npm start
```

You should see the following un-styled React app in the browser at **localhost:3000**.



Hotel Leisure

Enjoy world-class shopping in the heart of the city.

There are some glaringly obvious issues, such as that the logo is oversized and the cards are sitting vertically on top of each other. You can now begin to style the React app with CSS-in-JS with the Emotion library.

Installing the Packages

There are two primary ways of using Emotion: framework-agnostic or specifically with React.

The `@emotion/css` package is framework agnostic and the simplest way to use Emotion. It is the recommended way to use Emotion with a non React app.

It is best to use the "@emotion/react" package when using React with a build environment that can be configured.

To install the Emotion library for a React app, stop the server (Ctrl + C), and run the following command:

```
npm i @emotion/react
```

When the package has finished installing, start the server again with the following command:

```
npm start
```

You are ready to use CSS-in-JS to style the app with the Emotion library!

The `css` Prop

The `css` prop allows you to style elements. It can support an object or a tagged template literal which can be demonstrated by styling the `<main>`, logo `<img>` and the `<div>` that contains all of the hotel cards. Each card consists of the main image, title, and description from our `hotels` array along with the buttons a user can click on to learn more or book.

At the top of the **starter-code/src/App.js** file, import the `css` prop from `emotion`. Include the comment `/** @jsxImportSource @emotion/react */` at the top of the file when using Create React App 4. This comment informs Babel to customize the automatic runtime import.

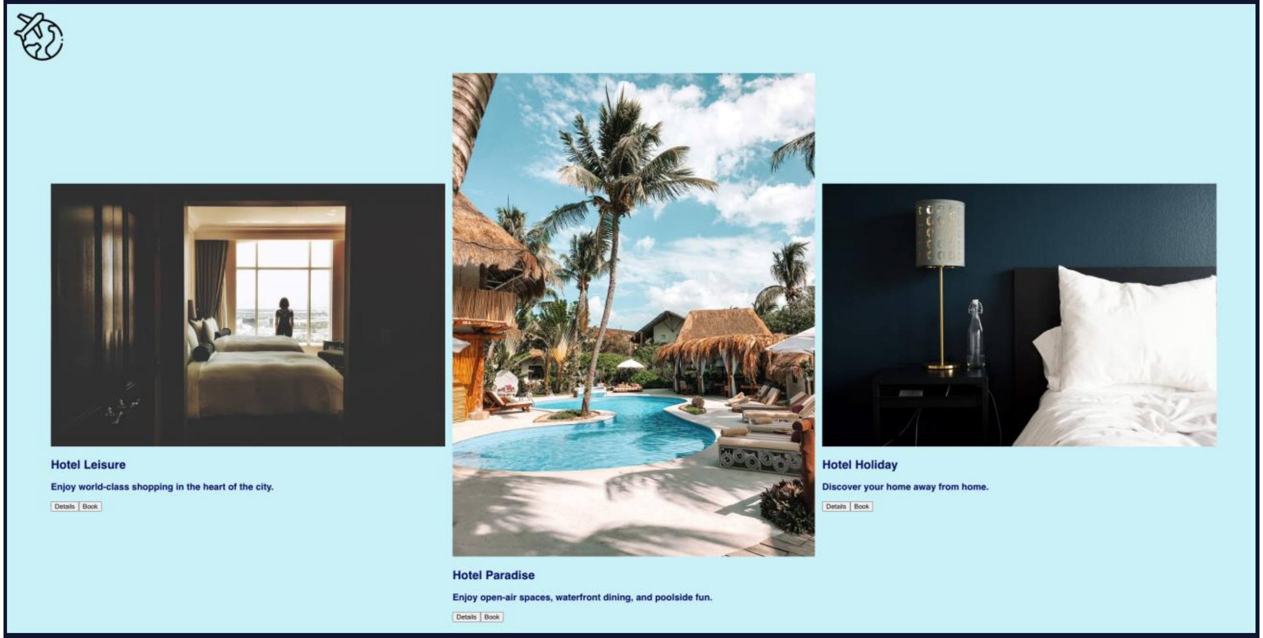
```
/** @jsxImportSource @emotion/react */
import { css } from "@emotion/react";
```

You can now use the `css` prop to style your tags. Copy the `<main>`, logo `<img>` and the container `<div>` in the `return` statement below and paste it over the current `<main>`, `<img>` and `<div>` elements in **App.js**.

```
...
function App() {
  ...
  return (
    <main
      css={{
        color: "#03045e",
        background: "#caf0f8",
        height: "1200px",
        fontFamily: "helvetica",
      }}
    >
      <img
        src={logo}
        alt="logo"
        css={css`
          display: absolute;
          margin-top: 15px;
          margin-left: 15px;
          height: 100px;
          width: 100px;
        `}
      />
      <div
        css={css`
          display: flex;
          justify-content: center;
          align-items: center;
          gap: 15px;
          padding: 20px;
          @media (max-width: 900px) {
            display: grid;
          }
        `}
      >
        ...
      </div>
    </main>
  );
}

export default App;
```

If you copy-paste the above code, you should see the following in the browser:



Hotel Leisure

Enjoy world-class shopping in the heart of the city.

Details | Book

Hotel Paradise

Enjoy open-air spaces, waterfront dining, and poolside fun.

Details | Book

Hotel Holiday

Discover your home away from home.

Details | Book

Object Styles

The `<main>` element uses an object with style object styles and values. Instead of writing CSS properties in `kebab-case`, like regular CSS, you write them in `camelCase`. Notice properties are separated by a comma. Object styles are especially helpful because you don't need a CSS call like with string styles but object styles can also be used with styled components.

```
<main
  css={{
    color: "#03045e",
    background: "#caf0f8",
    height: "1200px",
    fontFamily: "helvetica",
  }}
>
  ...
</main>
```

String Styles

The `css` prop also used tagged template literals to style the `<img>` and `<div>` elements. Template literals are enclosed by the backtick (` `). Notice the css properties are written in `kebab-case` and separated by a semicolon.

```
<img
  src={logo}
  alt="logo"
  css={css`
    display: absolute;
    margin-top: 15px;
    margin-left: 15px;
    height: 100px;
    width: 100px;
  `}
/>
```

Styled Components

The Emotion library has a package called `@emotion/styled`, which gives us access to `styled`, that allows you to create components that have styles attached to them. It is similar to `css` prop except it is called with an HTML tag or React component.

To install the `styled` package from the Emotion library, stop the server, and run the following command:

```
npm i @emotion/styled
```

When the package has finished installing, start the server again.

In the empty **starter-code/src/styles.js** file, Create a `CardWrapper` component that will style each individual hotel card in the hotels array found in **App.js**.

At the top of the **styles.js** file, import `styled` from the `@emotion/styled` package. Beneath that, create a `CardWrapper` component that will be used to style each hotel's container `<div>`.

```
import styled from "@emotion/styled";

export const CardWrapper = styled.div`
  width: 250px;
  height: 325px;
  background: #fff;
  border-radius: 15px;
  padding-bottom: 5px;
  @media (max-width: 900px) {
    width: 400px;
  }
`;
```

Note that you are exporting the `CardWrapper` component so that it can be used it in the **App.js** file.

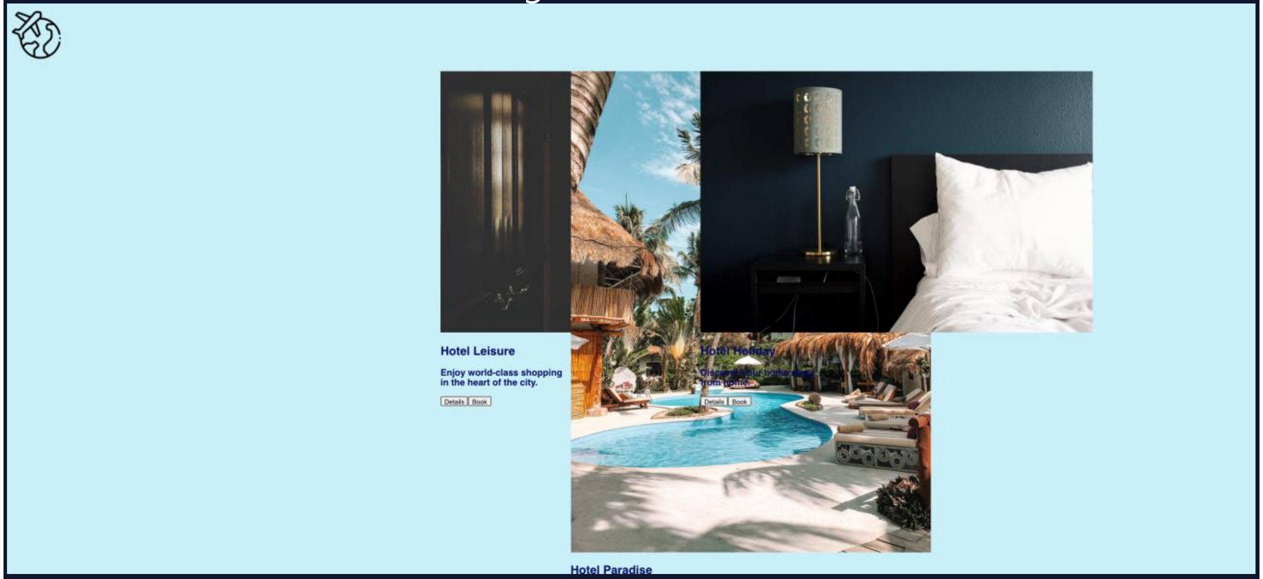
At the top of the **App.js** file, import `CardWrapper` from **styles.js**. Next, change the container `<div>` for each hotel to a `CardWrapper` component.

```
...
import { CardWrapper } from "./styles.js";

...
{hotels.map((hotel) => {
    return (
    <CardWrapper key={hotel.id}>
        <img src={hotel.src} alt={hotel.alt} />
        <div>
            <h2>{hotel.title}</h2>
            <h3>{hotel.description}</h3>
        </div>
        <div>
            <button>Details</button>
            <button>Book</button>
        </div>
    </CardWrapper>
    );
})}

...
```

You should now see the following:



Now, build out styled components to be used for each hotel's image, text container, title, description, buttons container, and buttons in **styles.js**. This is also an excellent time to experiment with your own styling.

```
// styles.js
...

export const CardWrapper = styled.div`
  ...
`;

export const ImageWrapper = styled.img`
  object-fit: cover;
  width: 100%;
  height: 60%;
  border-radius: 15px 15px 0 0;
`;

export const TextWrapper = styled.div`
  padding: 10px;
  height: 50px;
`;

export const TitleWrapper = styled.h2`
  margin: 0;
  font-size: 20px;
`;

export const DescriptionWrapper = styled.h3`
  margin-top: 5px;
  font-size: 14px;
  color: #023e8a;
`;

export const ActionsWrapper = styled.div`
  margin-left: 10px;
  padding: 10px 0;
  display: flex;
`;

export const Button = styled.button`
  width: 100%;
  margin-right: 10px;
  margin-top: 4px;
  border: 0;
  border-radius: 15px;
  box-shadow: 0 10px 10px rgba(0, 0, 0, 0.08);
  padding: 10px 0;
  cursor: pointer;
  transition: all 0.25s cubic-bezier(0.02, 0.01, 0.47, 1);

  &:hover {
    box-shadow: 0 15px 15px rgba(0, 0, 0, 0.16);
  }
`;
```

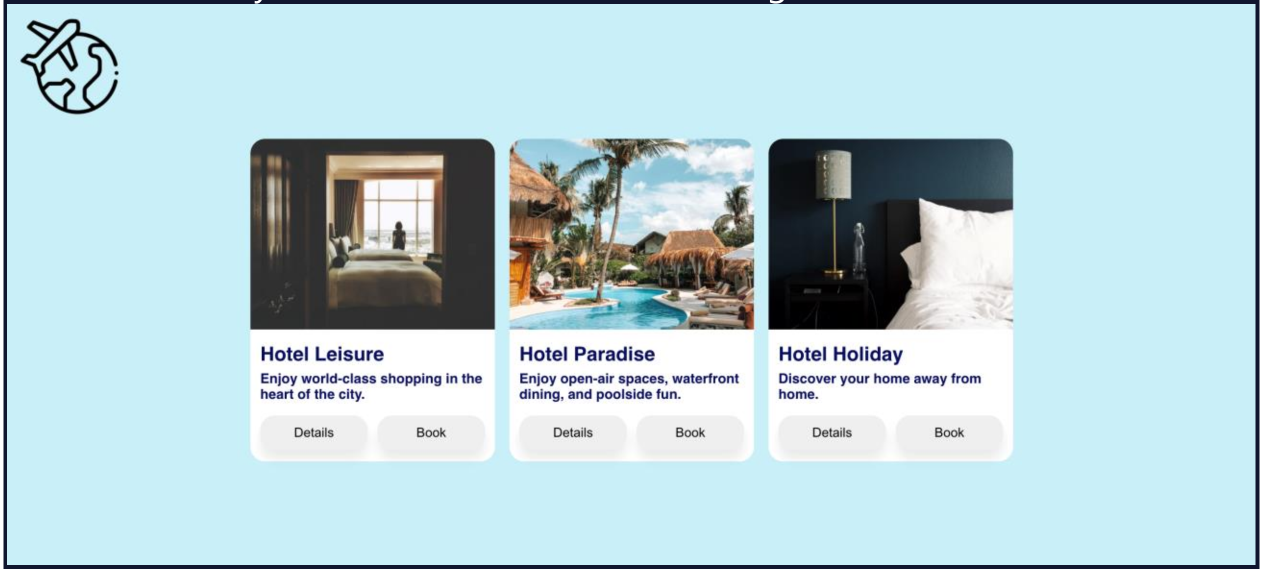
You will now need to import these at the top of **App.js** file and then change the JSX to use these styled components.

```
...
import {
  CardWrapper,
  ImageWrapper,
  TextWrapper,
  TitleWrapper,
  DescriptionWrapper,
  ActionsWrapper,
  Button,
} from "./styles";

...

{hotels.map((hotel) => {
    return (
        <CardWrapper key={hotel.id}>
            <ImageWrapper src={hotel.src} alt={hotel.alt} />
            <TextWrapper>
                <TitleWrapper>{hotel.title}</TitleWrapper>
                <DescriptionWrapper>{hotel.description}</DescriptionWrapper>
            </TextWrapper>
            <ActionsWrapper>
                <Button>Details</Button>
                <Button>Book</Button>
            </ActionsWrapper>
        </CardWrapper>
    );
})}
```

In the browser, you should now see the following:


Hotel Leisure
Enjoy world-class shopping in the heart of the city.
Details
Book
Hotel Paradise
Enjoy open-air spaces, waterfront dining, and poolside fun.
Details
Book
Hotel Holiday
Discover your home away from home.
Details
Book

Composition

Great, the app styling is nearing completion! You probably want to distinguish between the buttons prompting the user to learn more or book a hotel. You can use composition to create variants.

At the bottom of **styles.js** file, create a `PrimaryButton` component and a `SecondaryButton` component that will style the `Button` component. To do this we just wrap the `Button` component in the `styled()` constructor.

You no longer need to export the `Button` component so you can remove the `export` keyword.

```
...
const Button = styled.button`
  ...
`;

export const PrimaryButton = styled(Button)`
  background-color: #03045e;
  color: #caf0f8;
`;

export const SecondaryButton = styled(Button)`
  background-color: #caf0f8;
  color: #03045e;
`;
```

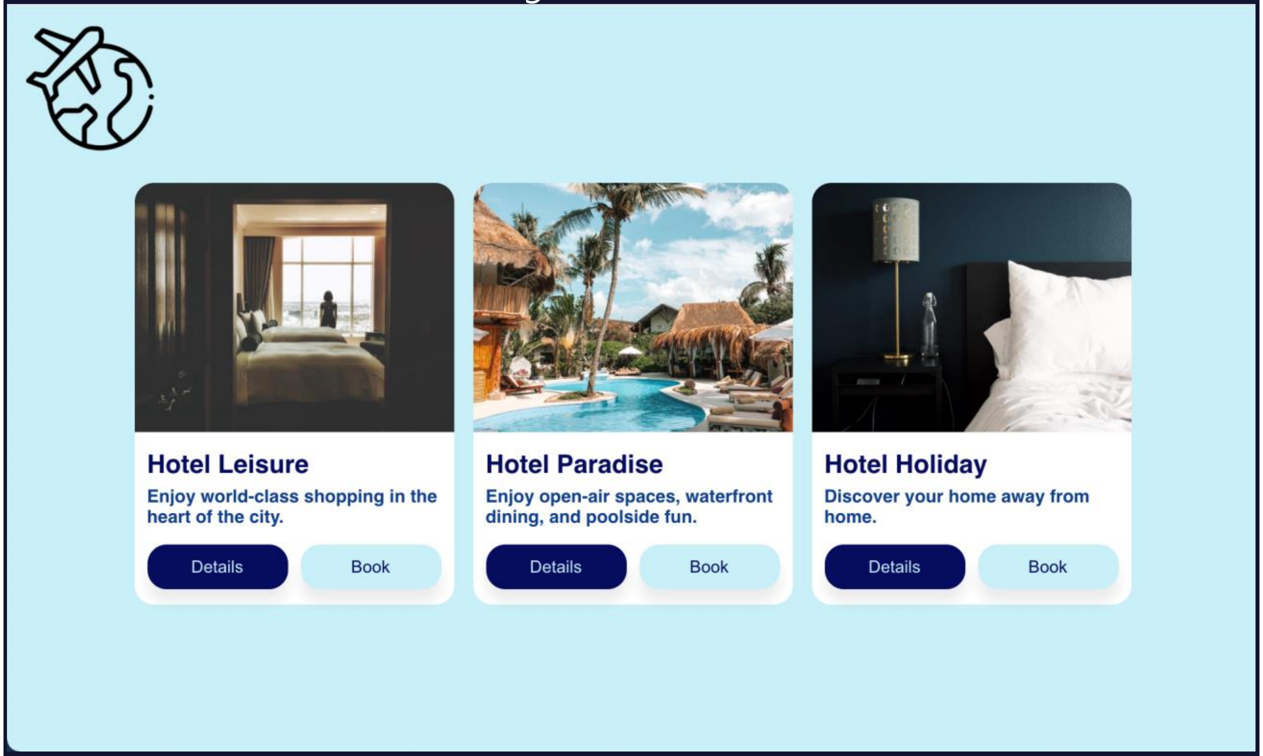
Finally, you need to import these at the top of **App.js** file and then change the JSX to use these components. Notice that we remove Button from the import statement and added in `PrimaryButton` and `SecondaryButton`.

```
...
import {
  CardWrapper,
  ImageWrapper,
  TextWrapper,
  TitleWrapper,
  DescriptionWrapper,
  ActionsWrapper,
  PrimaryButton,
  SecondaryButton,
} from "./styles";

...

                <ActionsWrapper>
                  <PrimaryButton>Details</PrimaryButton>
                  <SecondaryButton>Book</SecondaryButton>
                </ActionsWrapper>
...
```

You should now see the following:


Hotel Leisure
Enjoy world-class shopping in the heart of the city.
Details
Book
Hotel Paradise
Enjoy open-air spaces, waterfront dining, and poolside fun.
Details
Book
Hotel Holiday
Discover your home away from home.
Details
Book

Theming

Great! The design is nearing completion. Now, suppose the design team decides that the primary color `#03045e` used in several locations throughout your app is too dark and they want to change it. This leads you to have to hunt through your app and change each occurrence, which might grow daunting as your app size grows. Theming to the rescue!

```
import styled from "@emotion/styled";

export const theme = {
  colors: {
    primary: "#03045e",
    secondary: "#caf0f8",
    tertiary: "#023e8a",
    quaternary: "#fff",
  },
  fonts: {
    primary: "helvetica",
  },
  fontSize: {
    primary: "20px",
    secondary: "14px",
  },
};

export const CardWrapper = styled.div`
  ...
  background: ${(props) => props.theme.colors.quaternary};
  ...
```

```
`;
...
```

The `<ThemeProvider>` you are going to implement at the top level of your app will give you access to `props.theme` in your styled components. Go ahead and apply the rest of the theming to your styled components.

```
export const CardWrapper = styled.div`
  ...
  background: ${(props) => props.theme.colors.quaternary};
  ...
`;

...
export const TitleWrapper = styled.h2`
  ...
  font-size: ${(props) => props.theme.fontSize.primary};
`;

export const DescriptionWrapper = styled.h3`
  ...
  font-size: ${(props) => props.theme.fontSize.secondary};
  color: ${(props) => props.theme.colors.tertiary};
`;

...
export const PrimaryButton = styled(Button)`
  background-color: ${(props) => props.theme.colors.primary};
  color: ${(props) => props.theme.colors.secondary};
`;

export const SecondaryButton = styled(Button)`
  background-color: ${(props) => props.theme.colors.secondary};
  color: ${(props) => props.theme.colors.primary};
`;
```

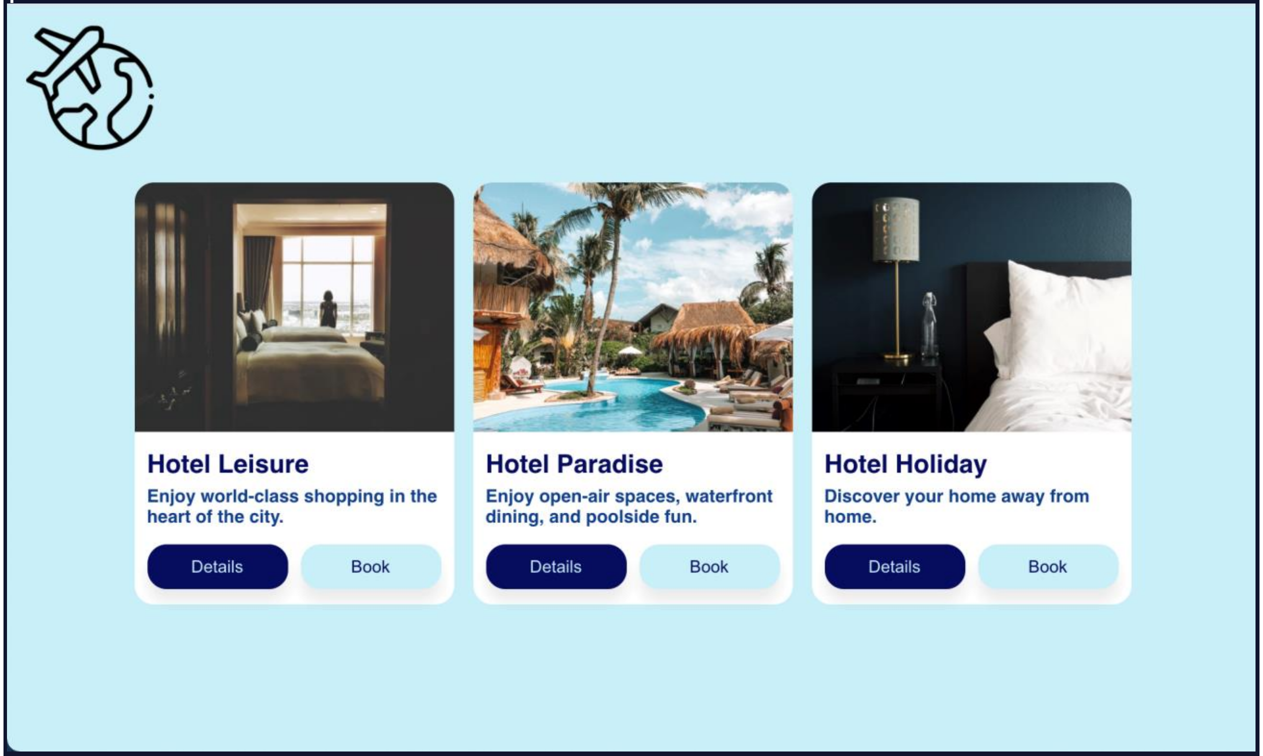
At the top of our **App.js** file, import the `ThemeProvider` object from `@emotion/react` as well as import `theme` from **styles.js** file. Add the `<ThemeProvider>` at the top level of the application which accesses the `theme` object. This will make the `theme` property available to all components in the React app.

You can also update the `css` prop in the `<main>` tag to use the theme object properties and values.

```
import { css, ThemeProvider } from "@emotion/react";
import logo from "./logo.png";
import {
  theme, ...
} from "./styles"

...
function App() {
  {/* Wrap the entire content in a <ThemeProvider> */}
  return <ThemeProvider theme={theme}>
      <main
        css={(theme) => ({
          color: theme.colors.primary,
          background: theme.colors.secondary,
          height: "1200px",
          fontFamily: theme.fonts.primary,
        })}
      >
        ...
      </main>
    </ThemeProvider>
}
```

If you have done things correctly the app should look identical to your previous work:



Hotel Leisure
Enjoy world-class shopping in the heart of the city.
Details
Book
Hotel Paradise
Enjoy open-air spaces, waterfront dining, and poolside fun.
Details
Book
Hotel Holiday
Discover your home away from home.
Details
Book

You can test this by making changes in the `theme` object and seeing the changes reflected in multiple locations in your app.

Keyframes / Animation

The last step is to add some animation to the app. You can define animations using the `keyframes` helper from `@emotion/react`. `keyframes` takes in a CSS `keyframe` definition and returns an object you can use in styles. You can use strings or objects just like `css`.

At the top of **styles.js** file, import `keyframes` from `@emotion/react` and then define a `LogoSpin` keyframe below the `theme` object.

```
...
import styled from "@emotion/styled";
import { keyframes } from "@emotion/react";

export const theme = {
  ...
};

export const LogoSpin = keyframes`
from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
`;
...
```

Finally, import the `LogoSpin` keyframe in `App.js` and update the JSX.

```
...
import {
  LogoSpin,
  ...
} from "./styles";

...
<img
    src={logo}
    alt=""
    css={css`
        ...
        animation: ${LogoSpin} 10s linear infinite;
    `}
/>
```

That’s it! The logo should be rotating 360 degrees every 10 seconds.

Putting It All Together

In this tutorial, you learned how to use the Emotion library as an example of CSS-in-JS. You were able to create styled components, add styles to them, and use them in a React app. You learned how to create a theme object and apply styles to styled components. You also learned how to use the `keyframes` helper to add animation to your app.

Consider using Emotion in your next project to utilize CSS-in-JS. Happy coding!