DIGA4015A:
Interactive Media 4A | React Exercises 1

Exercise 1:

Create a *WeatherDisplay* component that accepts *props* for temperature, condition (sunny, cloudy, rainy, etc.), and a boolean for whether it is day or night. The component should conditionally render messages based on the weather condition and time of day.

```jsx
// WeatherDisplay.jsx
//This is a possible solution, there are all manner of ways in which you, individually, can accomplish it.

import React from "react";

function WeatherDisplay({
  temperature = 25,
  condition = "sunny",
  isDay = true,
}) {
  // These are dynamic styles based on day/night
  const containerStyle = {
    backgroundColor: isDay ? "#E0F7FA" : "#263238",
    color: isDay ? "#000" : "#FFF",
    padding: "20px",
    borderRadius: "10px",
    textAlign: "center",
    width: "300px",
    margin: "20px auto",
    boxShadow: "0 4px 10px rgba(0, 0, 0, 0.2)",
  };

  // Fubction to generate a message based on weather condition and time of day
  const getWeatherMessage = () => {
    if (condition === "sunny") {
      return isDay
        ? "It's a bright and sunny day!"
        : "It's a clear and peaceful night.";
    } else if (condition === "cloudy") {
```

```
    return isDay
      ? "The sky is cloudy today."
      : "The clouds are covering the night sky.";
  } else if (condition === "rainy") {
    return isDay
      ? "It's raining during the day."
      : "Rain is falling through the night.";
  } else if (condition === "snowy") {
    return isDay
      ? "Snow is falling during the day!"
      : "Snow is covering the ground at night.";
  } else {
    return "The weather is unpredictable today.";
  }
 };

 return (
  <div style={containerStyle}>
    <h2>Weather Display</h2>
    <p>{getWeatherMessage()}</p>
    <p>{`Condition: ${condition}`}</p>
    <p>{`Temperature: ${temperature}°C`}</p>
    <p>{isDay ? "It is daytime." : "It is nighttime."}</p>
  </div>
 );
}

export default WeatherDisplay;
```

Exercise 2:

Create an *ImageDisplay* component that conditionally renders alternate text when the image fails to load. Add a CSS class for styling.

```
//ImageDisplay.jsx
//This is a possible solution, there are all manner of ways in which you, individually, can accomplish it.

import React from "react";
import "./ImageDisplay.css";

function ImageDisplay({ imageUrl, altText }) {
  function handleImageError(e) {
    e.target.classList.add("hidden"); // Hide the failed image
    e.target.nextSibling.classList.remove("hidden"); // Show the alternate text
  }

  return (
    <div className="image-container">
      <img src={imageUrl} alt={altText} onError={handleImageError} />
      <div className="error-text hidden">{altText}</div>
    </div>
  );
}

export default ImageDisplay;
```

Further question: why and how did I not include the *props* keyword as a parameter to the *ImageDisplay* component, yet was still able to use the *imageUrl* and *altText* props?

```
/* ImageDisplay.css */
.hidden {
    display: none;
 }

 .image-container {
    text-align: center;
 }

 .error-text {
    /* Styles for the alternate text if the image fails to load */
    color: red;
    font-size: 16px;
 }
```

Exercise 3:

Create a *NestedComment* component that can render comments with replies in a nested format. The component should accept a "comments" object with properties for the *author*, *content*, and an array of *replies*. Each reply should be structured similarly to the comment object, allowing for multiple levels of nesting.

```
// NestedComment.jsx
//This is a possible solution, there are all manner of ways in which you, individually, can accomplish it.

import React from "react";
import "./NestedComment.css";

function NestedComment({ comment }) {
  return (
    <div className="comment">
      <p className="comment-author">{comment.author}</p>
      <p className="comment-content">{comment.content}</p>
```

```
      {/* Now we render nested replies recursively */}
      {comment.replies && comment.replies.length > 0 && (
        <div className="comment-replies">
          {comment.replies.map((reply, index) => (
            <NestedComment key={index} comment={reply} />
          ))}
        </div>
      )}
    </div>
  );
}

export default NestedComment;
```

```
/* NestedComment.css */

.comment {
  border: 1px solid #ddd;
  padding: 10px;
  margin: 10px 0;
  border-radius: 5px;
  background-color: #f9f9f9;
}

.comment-author {
  font-weight: bold;
  color: #333;
  margin-bottom: 5px;
}

.comment-content {
  margin-bottom: 5px;
}

.comment-replies {
  margin-left: 20px; /* Indents replies */
  border-left: 2px solid #ddd;
  padding-left: 10px;
```

```
}
```

```
//App.js

import React from "react";
import NestedComment from "./NestedComment";

function App() {
  const commentsData = {
    author: "Alice",
    content: "This is the main comment.",
    replies: [
      {
        author: "Bob",
        content: "This is a reply to Alice.",
        replies: [
          {
            author: "Charlie",
            content: "This is a reply to Bob.",
            replies: [
              {
                author: "David",
                content: "This is a reply to Charlie.",
                replies: [],
              },
            ],
          },
        ],
      },
      {
        author: "Eve",
        content: "Another direct reply to Alice.",
        replies: [],
      },
    ],
  };

  return (
```

```
    <div>
      <h1>Nested Comments</h1>
      <NestedComment comment={commentsData} />
    </div>
  );
}

export default App;
```