Mobile Application Development Internship Assignment

Section A: Multiple Choice Questions

1. What is React Native?
Ans: a) A cross-platform framework for building mobile apps using JavaScript

2. What is the purpose of the Flexbox layout in React Native?
Ans:a) To create a responsive design that adapts to different screen sizes

3. What is the difference between props and state in React Native?
Ans:a) Props are used to pass data between components, while state is used to manage data within a
component
4. Which of the following is NOT a valid way to style a React Native component?
Ans:d) CSS styles
5. What is the purpose of the fetch() method in React Native?
Ans:a) To make HTTP requests to a server
6. What is the difference between setState() and forceUpdate() in React Native?
Ans:a) setState() updates the state of a component and triggers a re-render, while forceUpdate() re-renders a component without updating its state
7. Which of the following is used to handle user input in React Native?
Ans:d) All of the above
8. What is the purpose of the AsyncStorage module in React Native?
Ans:b) To store data temporarily on the device
9. What is the purpose of the Animated API in React Native?
Ans:a) To create complex animations using JavaScript
10. Which of the following is used to navigate between screens in a React Native app?
Ans:d) All of the above

Section B: Programming Questions

11. Create a new React Native app called "MyApp".

a) What command(s) would you use to create this app?

Ans:To create a new React Native app called "MyApp", you can use the following command: npx react-native init MyApp

b) What is the directory structure of the app?

Ans:

```
MyApp/
├── __tests__/
├── android/
├── ios/
├── node_modules/
├── src/
├── .buckconfig
├── .editorconfig
├── .eslintrc.js
├── .flowconfig
├── .gitattributes
├── .gitignore
├── .watchmanconfig
├── App.js
├── app.json
├── babel.config.js
├── index.js
├── metro.config.js
└── package.json
```

c) What file(s) would you modify to change the app's appearance?

Ans: Change the app's appearance, you would modify the `App.js` file located at the root of the app directory. This file contains the main entry point of the app and defines the layout and styling of the app's components. You can modify the layout and styling using React Native's built-in components and styles or by importing and using third-party libraries.

12. Create a new component called "MyButton" that displays a button with the text "Click me".
a) What props would you pass to this component to change the button's appearance?
Ans:The props that you could pass to the "MyButton" component to change the button's appearance are:

- className: to add a CSS class to the button and apply custom styles.
- style: to add inline styles to the button.
- disabled: to disable the button.
- type: to change the type of the button (e.g., "submit", "reset", "button").
- onClick: to handle clicks on the button.

example implementation of the "MyButton"

```
import React from 'react';



const MyButton = ({ className, style, disabled, type, onClick }) => {
 return (
  <button
   className={className}
   style={style}
   disabled={disabled}
   type={type}
   onClick={onClick}
  >
   Click me
  </button>
 );
};
export default MyButton;
```

b) What state would you use to handle clicks on the button?

Ans:You could use the state to handle clicks on the button and update the component's behavior accordingly. For example, you could use the useState hook to keep track of the button's click count and display it on the button's text:

```
import React, { useState } from 'react';



const MyButton = ({ className, style, disabled, type }) => {

 const [clickCount, setClickCount] = useState(0);



 const handleClick = () => {

  setClickCount(clickCount + 1);

 };



 return (

  <button

   className={className}

   style={style}

   disabled={disabled}

   type={type}

   onClick={handleClick}

  >

   Clicked {clickCount} times
```

```
  </button>

 );

};
```

```
export default MyButton;
```

. **13. Create a new component called "MyList" that displays a list of items.**

a) What props would you pass to this component to change the list's appearance?

Ans:Here are some props that can be passed to the MyList component to change its appearance:

- `items`: An array of objects representing the items to be displayed in the list.
- `itemRenderer`: A function that takes an item from the `items` array as input and returns the JSX that should be used to render that item in the list.
- `className`: A string of class names to be applied to the outermost element of the component.
- `style`: An object representing the inline style to be applied to the outermost element of the component

b) What data structure would you use to store the list items?

Ans:A simple array can be used to store the list items.

c) How would you render the list items?

Ans:import React from "react";

```
function MyList({ items, itemRenderer, className, style }) {

 return (

  <ul className={className} style={style}>

   {items.map((item) => (

    <li key={item.id}>{itemRenderer(item)}</li>))}
```

```
</ul>

  );

}
```

Section C

14. Write a function called "getWeather" that makes an HTTP GET request to the

OpenWeatherMap API and returns the temperature for a given city.

a) What parameters would you pass to this function?

Ans:The parameters to be passed to this function would be the city name and the API key to make a successful request to the OpenWeatherMap API.

b) What is the URL for the OpenWeatherMap API?

Ans:The URL for the OpenWeatherMap API is "api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}".

c) How would you handle errors or exceptions in this function?

Ans:To handle errors or exceptions in this function, we can use try and except blocks. We can use the HTTP response status codes to determine if the request was successful or not. If the status code is 200, we can extract the temperature from the response and return it. If the status code is not 200, we can raise an exception with an appropriate error message to indicate that the request was unsuccessful. Here's an example implementation of the "getWeather" function:

Python:

```
def getWeather(city, api_key):

    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}"

    response = requests.get(url)

    if response.status_code == 200:
```

```
    data = response.json()

    temperature = data["main"]["temp"]

    return temperature

  else:

    raise Exception(f"Failed to retrieve weather data for {city}. Error:
{response.status_code}")
```

15.**Create a new screen in your app called "ProfileScreen" that displays the user's profile information**

a) What navigation method would you use to navigate to this screen?

Ans: The navigation method that we can use to navigate to the ProfileScreen is stack navigation. We can add ProfileScreen as a new screen to our app's navigation stack, and use the navigation prop to navigate to this screen when the user clicks on a button or performs some action.

b) What props would you pass to this screen to display the user's information?

Ans: To display the user's information on the ProfileScreen, we would pass the user object as props. This object would contain the user's name, profile picture, email address, phone number, and any other relevant information that we want to display on the screen**.**

c) What component(s) would you use to display the user's information?

Ans:Text component: We can use this component to display the user's name, email address, phone number, and any other text-based information.

- **Image component: We can use this component to display the user's profile picture.**
- **List component: We can use this component to display the user's information in a list format, with each item representing a different piece of information (e.g., name, email, phone number).**

- **Card component: We can use this component to display the user's information in a visually appealing way, with a card-like layout containing the user's name, profile picture, and other details.**

15.Create a custom hook called "useLocalStorage" that allows you to store and retrieve data from the device's local storage. The hook should take a key and a value as arguments and return a tuple containing the current value and a setter function

a) How would you use this hook to store and retrieve data from local storage?

Ans:

```
import { useState, useEffect } from 'react';



function useLocalStorage(key, initialValue) {
 const [storedValue, setStoredValue] = useState(() => {
  try {
   const item = window.localStorage.getItem(key);
   return item ? JSON.parse(item) : initialValue;
  } catch (error) {
   console.error(error);
   return initialValue;
  }
 });


 useEffect(() => {
  try {
```

```
      const serializedValue = JSON.stringify(storedValue);
      window.localStorage.setItem(key, serializedValue);
    } catch (error) {
      console.error(error);
    }
  }, [key, storedValue]);

  return [storedValue, setStoredValue];
}
```

b) How would you handle errors or exceptions in this hook?

Ans:

```
import { useLocalStorage } from './useLocalStorage';

function MyComponent() {
  const [myValue, setMyValue] = useLocalStorage('myKey', 'initialValue');
 // ...
}
```

16. Create a custom hook called "useLocalStorage" that allows you to store and retrieve data

from the device's local storage. The hook should take a key and a value as arguments and

return a tuple containing the current value and a setter function.

a) How would you use this hook to store and retrieve data from local storage?

Ans:

```
import { useState } from 'react';
```

```
function useLocalStorage(key, initialValue) {
 const [storedValue, setStoredValue] = useState(() => {
  try {
   const item = window.localStorage.getItem(key);
   return item ? JSON.parse(item) : initialValue;
  } catch (error) {
   console.error(error);
   return initialValue;
  }
 });

 const setValue = (value) => {
  try {
   const valueToStore = value instanceof Function ? value(storedValue) : value;
   setStoredValue(valueToStore);
   window.localStorage.setItem(key, JSON.stringify(valueToStore));
  } catch (error) {
   console.error(error);
  }
 };
```

```
  return [storedValue, setValue];
}

function MyComponent() {
  const [name, setName] = useLocalStorage('name', '');

  return (
    <div>
      <input
        type="text"
        value={name}
        onChange={(event) => setName(event.target.value)}
      />
      <p>Your name is: {name}</p>
    </div>
  );
}
```

b) How would you handle errors or exceptions in this hook?

Ans:**To handle errors or exceptions in the `useLocalStorage` hook, we wrap the code that retrieves or sets the stored value in a `try...catch` block. If an error occurs, we log it to the console and return the initial value instead. This ensures that the hook continues**

to function even if local storage is unavailable or if there's an issue with the data stored in it.

17. Create a new component called "MyImagePicker" that allows the user to select an image from their device's photo gallery.

a) What external library or module would you use to implement this component?

Ans: To implement the "MyImagePicker" component, we can use the `react-native-image-picker` library, which provides a cross-platform solution for picking images from a device's photo gallery or camera.

b) What props would you pass to this component to customize its appearance?

Ans:`buttonStyle`**: An object containing styles to apply to the button.**

- `buttonTextStyle`**: An object containing styles to apply to the button text.**
- `imageStyle`**: An object containing styles to apply to the selected image.**
- `placeholderImage`**: The source of an image to display if no image has been selected yet.**
- `placeholderImageStyle`**: An object containing styles to apply to the placeholder image.**

c. How would you handle errors or exceptions when selecting an image?

Ans:import React, { useState } from 'react';

import { Button, Image, View } from 'react-native';

import ImagePicker from 'react-native-image-picker';

function MyImagePicker(props) {

const [imageSource, setImageSource] = useState(props.placeholderImage);

const handleImagePick = () => {

```
  ImagePicker.launchImageLibrary({}, (response) => {
    if (response.uri) {
      setImageSource({ uri: response.uri });
    } else {
      props.onError(response.error);
    }
  });
};

return (
  <View>
    <Button
      title={props.buttonText}
      onPress={handleImagePick}
      style={props.buttonStyle}
      textStyle={props.buttonTextStyle}
    />
    <Image
      source={imageSource}
      style={[{ width: 200, height: 200 }, props.imageStyle]}
    />
```

```
  </View>
 );
}

export default function App() {
 const [error, setError] = useState(null);

 const handleError = (errorMessage) => {
  setError(errorMessage);
 };

 return (
  <View>
   <MyImagePicker
    buttonText="Select an image"
    placeholderImage={require('./placeholder.png')}
    placeholderImageStyle={{ width: 200, height: 200 }}
    onError={handleError}
   />
   {error && <Text style={{ color: 'red' }}>{error}</Text>}
  </View>
```

```
  );

}
```

Section D: Short Answer Questions

1.Explain the concept of "props drilling" in React Native.

Ans:"Props drilling" in React Native refers to the process of passing props down through several levels of nested components. This can become cumbersome as the app grows, as the props may need to be passed through many intermediate components, even if they are not used by those components. It can also make it difficult to manage and update the props, as any changes may require updating many different components.

2.What is the difference between a "controlled" and "uncontrolled" component in React Native?

Ans:In React Native, a "controlled" component is one where the component's behavior and state are controlled by the parent component, which passes all necessary props and handles all necessary event callbacks. An "uncontrolled" component, on the other hand, manages its own behavior and state, and may not require props or event callbacks from a parent component.

3.What is the difference between "component state" and "application state" in React Native?

Ans:"Component state" refers to the internal state of a single component in React Native, which is managed by the component itself and can be updated using setState(). "Application state" refers to the overall state of the entire app, which can include data that needs to be shared between different components. This state can be managed using a state management library like Redux

4.What is Redux and how does it relate to React Native?

Ans:Redux is a state management library for JavaScript applications, including React Native apps. It provides a centralized store for the app's state, which can be updated by dispatching actions. Components can then subscribe to changes in the store and

update themselves accordingly. Redux can simplify the process of managing and sharing application state between different components, making it easier to build complex apps.

5.How would you optimize the performance of a React Native app that has a large number of components?

Ans:There are several ways to optimize the performance of a React Native app with a large number of components. One approach is to use shouldComponentUpdate() or React.memo() to prevent unnecessary updates and re-renders of components. Another approach is to use virtualization techniques like FlatList or SectionList to efficiently render large lists of data. Additionally, optimizing images and using asynchronous rendering can help improve the overall performance of the app.