1. What is React Native?

a) A cross-platform framework for building mobile apps using JavaScript

b) A platform-specific framework for building mobile apps using Java or Swift

c) A desktop application development framework

d) A web development framework

Ans: a) A cross-platform framework for building mobile apps using JavaScript

2. What is the purpose of the Flexbox layout in React Native?

a) To create a responsive design that adapts to different screen sizes

b) To create a fixed layout that remains the same regardless of screen size

c) To create a layout that is optimized for performance

d) To create a layout that is optimized for accessibility

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?

a) Props are used to pass data between components, while state is used to manage data within a component

b) Props and state are the same thing

c) Props are used to manage data within a component, while state is used to pass data between

components

d) None of the above

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?

a) Inline styles

b) External stylesheets

c) JavaScript styles

d) CSS styles

Ans d) CSS styles

5. What is the purpose of the fetch() method in React Native?

a) To make HTTP requests to a server

b) To retrieve data from a local database

c) To update the state of a component

d) None of the above

Ans: a) To make HTTP requests to a server

6. What is the difference between setState() and forceUpdate() in React Native?

a) setState() updates the state of a component and triggers a re-render, while forceUpdate() re-renders a component without updating its state

b) setState() and forceUpdate() are the same thing

c) forceUpdate() updates the state of a component and triggers a re-render, while setState() re-renders a component without updating its state

d) None of the above

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?

a) TouchableOpacity

b) TouchableHighlight

c) TouchableWithoutFeedback

d) All of the above

Ans: d) All of the above

8. What is the purpose of the AsyncStorage module in React Native?

a) To store data permanently on the device

b) To store data temporarily on the device

c) To store data in the cloud

d) None of the above

Ans a) To store data permanently on the device

9. What is the purpose of the Animated API in React Native?

a) To create complex animations using JavaScript

b) To create basic animations using CSS

c) To create complex animations using CSS

d) None of the above

Ans: a) To create complex animations using JavaScript

10. Which of the following is used to navigate between screens in a React Native app?

a) StackNavigator

b) DrawerNavigator

c) TabNavigator

d) All of the above

Ans: d) All of the above

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

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

Ans: react-native init MyApp

b) What is the directory structure of the app?

Ans: The directory structure of the react native app is

MyApp/

├── __tests__/

├── android/

├── ios/

├── node_modules/

├── .gitignore

├── .prettierrc.js

├── .watchmanconfig

├── App.js

├── app.json

├── babel.config.js

```
├── index.js
├── metro.config.js
├── package.json
├── README.md
```

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

Ans: To change the app's appearance, we need to modify the App.js file. This file contains the root component of your app and defines its behaviour and appearance.

12. Create a new component called "MyButton" that displays a button with the text "Click me".

```
import React, { useState } from 'react';
import { View, Button } from 'react-native';

const MyButton = (props) => {
  const [clickCount, setClickCount] = useState(0);

  const handleButtonClick = () => {
    setClickCount(clickCount + 1);
  };

  return (
    <View>
      <Button
        title={`Click me (${clickCount})`}
        backgroundColor={'props.backgroundColor'}
        color={'red'}
        fontSize={'props.fontSize'}
        borderRadius={props.borderRadius}
        onPress={handleButtonClick}
      />
    </View>
  );
};

export default MyButton;
```

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

Ans: To change the button's appearance, we could pass the following props to the MyButton component:

- **title:** A string that sets the text of the button.
- **onPress:** A function that gets called when the button is pressed.
- **color:** A string that sets the color of the button
- **disabled:** A boolean that disables the button if set to true.

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

Ans: To handle clicks on the button in the "MyButton" component, we can use the "useState" hook to create a state variable that keeps track of the number of clicks. We can then update this state variable when the button is clicked by passing a callback function to the "onPress" prop of the button element.

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

```
import React from 'react';
import { View, Text } from 'react-native';
const MyList = (props) => {
  const listItems = [
    { id: 1, title: 'Item 1', description: 'Test Item 1' },
    { id: 2, title: 'Item 2', description: 'Test Item 2' },
    { id: 3, title: 'Item 3', description: 'Test Item 3' },
  ];
  const renderItem = (item) => {
    return (
      <View key={item.id}>
        <Text>{item.title}</Text>
        <Text>{item.description}</Text>
      </View>
    );
  };
  return (
    <View style={{ backgroundColor: props.backgroundColor }}>
      {listItems.map((item) => renderItem(item))}
    </View>
  );
};
export default MyList;
```

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

Ans: in order to change the appearance of the list in the "MyList" component, we can pass props such as "backgroundColor", "color", "fontSize", "padding", "margin", etc. to the container element that holds the list items.

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

Ans: To store the list items in the "MyList" component, we can use an array of objects, where each object represents a list item and contains properties such as "id", "title", "description", etc.

c) **How would you render the list items?**

Ans: To render the list items in the "MyList" component, we can use the "map" method to iterate over the array of list items and render each item as a separate component.

14. Write a function called "getWeather" that makes an HTTP GET request to the OpenWeatherMap API and returns the temperature for a given city.

```
async function getWeather() {
  const city = "Bangalore";
  const apiKey = "YOUR_API_KEY";
  const url = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`;

  try {
    const response = await fetch(url);
    const data = await response.json();
    const temperature = data.main.temp;
    return temperature;
  } catch (error) {
    console.error(error);
  }
}
```

a) **What parameters would you pass to this function?**
Ans: The parameters that we would pass to the function are:
City name (string) for which we want to get the temperature.
API key (string) to authenticate and access the OpenWeatherMap API.

b) **What is the URL for the OpenWeatherMap API?**
Ans: The URL for the OpenWeatherMap API is a combination of URL, city name and api key: http://api.openweathermap.org/data/2.5/weather?q={city_name}&appid={api_key}&units=metric.

This is a REST API that returns weather data for a specified city in JSON format.

c) **How would you handle errors or exceptions in this function?**
Ans: In react native, errors and exceptions can be handled using try-catch blocks. In the case of the getWeather function, we would wrap the HTTP request in a try block and catch any errors that occur. We can then handle the error by displaying a message to the user or logging the error for debugging purposes.

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

```
import React from 'react';

import { View, Text } from 'react-native';
function ProfileScreen() {
   const user = {
     name: 'Test Name',
     email: 'Test@example.com',
     phone: '555-555-5555'
   };

   return (
     <View>
       <Text>Name: {user.name}</Text>
       <Text>Email: {user.email}</Text>
       <Text>Phone: {user.phone}</Text>
     </View>
   );
 }
export default ProfileScreen;
```

a) **What navigation method would you use to navigate to this screen?**
Ans: To navigate to the ProfileScreen in a React Native app, we can use a StackNavigator from the React Navigation library. We can define a stack navigator with two screens: the first screen is the main screen of our app, and the second screen is the ProfileScreen. We can then use the

navigation.navigate method to navigate to the ProfileScreen from the main screen on the user actions such as, clicks on a button.

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 can pass a prop to the screen that contains the user's information. This prop could be an object with properties such as name, email, profile picture,

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

Ans: To display the user's information on the ProfileScreen, we could use a combination of Text and Image components.

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.**

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

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

  useEffect(() => {
    try {
      window.localStorage.setItem(key, JSON.stringify(value));
    } catch (error) {
      console.error(error);
    }
  }, [key, value]);

  return [value, setValue];
}

export default useLocalStorage;
```

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

Ans: To use this hook, you would call it in a functional component and pass in a key and initial value

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

Ans: To handle errors or exceptions in the hook, we can use a try-catch block to wrap the local storage getItem and setItem methods. If an error occurs, we log the error to the console and return the initial value instead. This way, the hook still returns a value and doesn't crash the application if there's an issue with local storage.

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 a component that allows the user to select an image from their device's photo gallery, we would use the react-native-image-picker library. This library provides a simple and customizable way to access and use the device's photo gallery and camera.

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

Ans: Some props we might pass to customize the appearance and behavior of the MyImagePicker component could include:

- **buttonText:** The text to display on the button that triggers the image picker.
- **buttonStyle**: An object containing styles to apply to the button.
- **imageStyle:** An object containing styles to apply to the selected image.
- **onImageSelect:** A callback function that gets called when an image is selected.

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

Ans: we can use the onError prop to specify a function that gets called if there is an error when accessing the photo gallery. We might display an error message to the user, or provide them with alternative options for selecting an image. Apart from that, we can wrap the image picker functionality in a try/catch block to catch any unexpected errors and handle them appropriately

18. **Create a new screen in your app called "SettingsScreen" that allows the user to change their app settings, such as language or theme.**

```
import React, { useState } from 'react';
import { StyleSheet, View, Text, Switch } from 'react-native';

const SettingsScreen = () => {
  const [isDarkModeEnabled, setIsDarkModeEnabled] = useState(false);
  const [isSpanishSelected, setIsSpanishSelected] = useState(false);

  const handleDarkModeToggle = (value) => {
    // Update the isDarkModeEnabled state variable based on the new
value of the Switch component
    setIsDarkModeEnabled(value);
  };

  const handleLanguageToggle = (value) => {
    // Update the isSpanishSelected state variable based on the new value
of the Switch component
    setIsSpanishSelected(value);
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Settings</Text>
      <View style={styles.setting}>
        <Text style={styles.settingText}>Dark mode</Text>
        <Switch
          value={isDarkModeEnabled}
          onValueChange={handleDarkModeToggle}
        />
      </View>
      <View style={styles.setting}>
        <Text style={styles.settingText}>Spanish</Text>
        <Switch
          value={isSpanishSelected}
          onValueChange={handleLanguageToggle}
```

```
        />
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 20,
  },
  setting: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    marginBottom: 10,
  },
  settingText: {
    fontSize: 18,
  },
});

export default SettingsScreen;
```

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

Ans: To navigate to the SettingsScreen, I would use a navigation method such as StackNavigator from react-navigation. This would allow the user to navigate to the SettingsScreen from other screens in the app, and to navigate back to the previous screen using the back button or swipe gesture.

b) **What data structure would you use to store the user's settings?**

Ans: For storing the user's settings, we can use a simple JavaScript object with key-value pairs. Each setting can be stored as a key-value pair in the object, with the key being the setting name and the value being the user's chosen setting.

c) **How would you save the user's settings so they persist across app sessions?**

Ans: To save the user's settings so they persist across app sessions, we can use the AsyncStorage API provided by React Native, which allows you to store and retrieve key-value pairs in the device's local storage we can store the settings object as a JSON string in AsyncStorage when the user makes changes to their settings, and then retrieve it when the app starts up again to restore the user's previous settings.

19. Write a function called "generatePassword" that generates a random password with a given length and complexity.

```
import React, { useState } from 'react';
import { View, Text, Button, TextInput } from 'react-native';

const PasswordGenerator = () => {
  const [password, setPassword] = useState('');
  const [length, setLength] = useState('8');
  const [difficulty, setDifficulty] = useState('medium');

  const generatePassword = () => {
    let result = '';
    const characters = {
      lowercase: 'abcdefghijklmnopqrstuvwxyz',
      uppercase: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
      numbers: '0123456789',
      symbols: '!@#$%^&*()_+-=[]{}|;:,.<>?'
    };
    let chars = '';
    switch (difficulty) {
      case 'easy':
        chars = characters.lowercase + characters.uppercase;
        break;
      case 'medium':
        chars = characters.lowercase + characters.uppercase +
characters.numbers;
        break;
      case 'hard':
        chars = characters.lowercase + characters.uppercase +
characters.numbers + characters.symbols;
        break;
      default:
        chars = characters.lowercase + characters.uppercase +
characters.numbers;
    }
```

```
    for (let i = 0; i < parseInt(length); i++) {
      result += chars.charAt(Math.floor(Math.random() * chars.length));
    }
    return result;
  };

  const handleGeneratePassword = () => {
    const password = generatePassword();
    setPassword(password);
  };

  return (
    <View>
      <View>
        <Text>Length:</Text>
        <TextInput value={length} onChangeText={setLength} />
      </View>
      <View>
        <Text>Difficulty:</Text>
        <TextInput value={difficulty} onChangeText={setDifficulty} />
      </View>
      <Button title="Generate Password" onPress={handleGeneratePassword} />
      <Text>{password}</Text>
    </View>
  );
};

export default PasswordGenerator;
```

a) **What parameters would you pass to this function?**

Ans: The function "generatePassword" would take two parameters: "length" and "complexity". The "length" parameter specifies the length of the password that needs to be generated, and the "complexity" parameter specifies the level of complexity for the password.

b) **What algorithm or library would you use to generate the password?**

Ans: To generate a password, we can use a combination of random characters, numbers, and symbols. One way to generate a password is by using the Math.random() method to generate random numbers and selecting random characters from a pre-defined set of characters.

c) **How would you ensure that the password meets the required complexity criteria?**

Ans: In order to ensure that the password meets the required complexity criteria, we can use conditional statements to add specific types of characters based on the level of complexity selected. For example, if the "complexity" parameter is set to "medium", we can add a combination of lowercase and uppercase letters, along with numbers. Similarly, for a "hard" level of complexity, we can add special characters such as !,@,#,$,% to the password.

20. **Create a new component called "MyMap" that displays a map of a given location using the Google Maps API.**

```
import React from 'react';

import { View } from 'react-native';
import { withScriptjs, withGoogleMap, GoogleMap, Marker } from "react-
google-maps";

const MyMap = withScriptjs(withGoogleMap((props) => {
  const { latitude, longitude } = props;
  return (
    <GoogleMap
      defaultZoom={12}
      defaultCenter={{ lat: latitude, lng: longitude }}
    >
      <Marker position={{ lat: latitude, lng: longitude }} />
    </GoogleMap>
  );
}));

const MyMapComponent = (props) => {
  const { latitude, longitude } = props;

  return (
    <View style={{ height: '400px', width: '100%' }}>
      <MyMap
        googleMapURL={`https://maps.googleapis.com/maps/api/js?sensor=false
&callback=myMap`}
        loadingElement={<div style={{ height: '100%' }} />}
        containerElement={<div style={{padding: '18px' , marginTop: '30px',
alignSelf: 'center', width:'400px', height: '100%' }} />}
        mapElement={<div style={{ height: '100%' }} />}
        latitude={latitude}
```

```
        longitude={longitude}
      />
    </View>
  );
};

export default MyMapComponent;
```

a) **What props would you pass to this component to specify the location?**

Ans: in order To specify the location, you could pass the following props to the MyMap component:

- **center:** An object that contains the latitude and longitude of the center point of the map.
- **zoom:** An integer that specifies the zoom level of the map.
- **markers:** An array of objects that represent markers on the map.

  Each object should contain a position property that specifies the latitude and longitude of the marker.

b) **How would you handle errors or exceptions when displaying the map?**

Ans: To handle errors or exceptions when displaying the map, we can wrap the map rendering code in a try-catch block or can use the onError prop of the GoogleMap component to display an error message.

c) **How would you optimize the performance of this component when displaying large maps?**

Ans: To optimize the performance of the MyMap component when displaying large maps, we can use techniques such as lazy loading or code splitting to only load the necessary parts of the Google Maps API when needed. We can also implement caching or memoization to avoid unnecessary re-renders of the map component.we can use the shouldComponentUpdate lifecycle method or the React.memo higher-order component to prevent unnecessary re-renders of the map component when its props haven't changed. This can help improve performance, especially when the component is used in a large and complex application.

21. **Explain the concept of "props drilling" in React Native.**

Ans: Props drilling is a term used in React Native to describe a situation where data is passed down through multiple levels of components via "props," which are essentially just properties or parameters that are passed from a parent component to its child components. In a typical React Native application, there can be many nested components, each with their own set of props. When a parent component needs to pass data down to a deeply nested child component, it must pass the data through all the intermediate components, even if those components don't actually use the data themselves. This is called props drilling

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

Ans: In React Native, a controlled component is a component whose state is managed by a parent component. This means that the parent component passes down all the necessary props to the child component, including the current value of the component and a function to handle changes to that value. The child component is then responsible for updating its value only when the parent tells it to do so.

On the other hand, an uncontrolled component is a component whose state is managed internally, without the help of a parent component. This means that the component is responsible for keeping track of its own state and updating it as needed.

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

Ans: In React Native, there are two types of state: component state and application state.

Component state refers to the internal state of a single component. This state is used to store data that is used only within the component, such as user input or other component-specific data. Component state is managed using the

setState() method, and is updated using the setState() method or by using a functional update if the new state depends on the previous state.

Application state, on the other hand, refers to the state of the entire application. This state is shared across multiple components and can be used to store data that is used across the entire application, such as user authentication status or other global data. Application state is typically managed using a state management library, such as Redux or MobX.

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

Ans: Redux is a state management library for JavaScript applications, including React Native. It provides a predictable state container, which helps in managing the state of an application in a predictable and efficient way. Redux is based on three principles: a single source of truth, state is read-only, and changes are made through pure functions.

In a React Native application, Redux can be used to manage the global state of the application, making it easier to share data and state across multiple components. With Redux, the state of the entire application is stored in a single object tree called the "store". This makes it easy to read and update the state from any component in the application, without having to pass props down the component tree.

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

Ans: Optimizing the performance of a React Native app that has a large number of components can be a challenging task, but there are several strategies that can help improve the performance:

- Use virtualization: Virtualization is a technique used to render only the visible portion of the component on the screen, which can help reduce the number of components that need to be rendered. React Native provides two virtualization components, FlatList and SectionList, which can be used to render long lists of data efficiently.

- Avoid unnecessary re-renders: React's default behavior is to re-render a component whenever its state or props change. However, if a component doesn't need to be re-rendered, we can prevent it from doing so by implementing the shouldComponentUpdate lifecycle method or by using the React.memo higher-order component.

- Use the useMemo and useCallback hooks: The useMemo hook can be used to memoize expensive computations, while the useCallback hook can be used to memoize event handlers. This can help improve the performance of components that are re-rendered frequently.

- Optimize images: Images can have a significant impact on the performance of a React Native app. You can optimize images by compressing them, reducing their resolution, and using the resizeMode prop to control how they are displayed.

- Use the InteractionManager API: The InteractionManager API can be used to defer expensive tasks until the user has finished interacting with the app. This can help improve the perceived performance of the app by reducing lag and jank.

- Use the shouldRasterizeIOS prop: The shouldRasterizeIOS prop can be used to improve the performance of components that have complex or animated content by rasterizing them as a bitmap, which can be rendered more efficiently.

- Avoid using inline styles: Inline styles can be expensive to compute, especially if they change frequently. Instead, consider using stylesheets or the StyleSheet.create method to define styles.

https://github.com/Rishu20/HappyMonk_Assignment

<END OF FILE>