Mobile Application Development Internship Assignment Solution

Section A: Multiple Choice Questions (20 marks)

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) c) A desktop application development framework
d) d) A web development framework

Answer:-a) A cross-platform framework for building mobile apps using JavaScript
d) A web development framework

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

Answer:-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

Answer:-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
Answer:- 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

Answer:-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() rerenders 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() rerenders a component without updating its state

Answer:-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

Answer:-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

Answer:-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

Answer:-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

Answer:-a) StackNavigator

Section B: Programming Questions (80 marks)

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

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

Answer:- you would use the following commands in your terminal: npx react-native init MyApp

b) **What is the directory structure of the app**?

Answer:- The directory structure of the app would look like this:

```
        MyApp/
├── _tests_/
├── android/
├── ios/
├── node_modules/
├── src/
│   ├── App.js
│   └── index.js
├── .buckconfig
├── .editorconfig
├── .eslintrc.js
├── .flowconfig
├── .gitattributes
├── .gitignore
├── .prettierrc.js
├── .watchmanconfig
├── app.json
├── babel.config.js
├── index.js
├── metro.config.js
├── package.json
├── README.md
└── yarn.lock
```

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

Answer:-To change the appearance of the app, you would modify the App.js file, which contains the code for the app's user interface. You can also add styles to the components using the StyleSheet API provided by React Native. Additionally, you can add new components or modify the existing ones in the src/ directory to customize the app's appearance.

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?**

Answer:-The props that could be passed to the MyButton component to change the button's

appearance are:

onClick: A function that will be called when the button is clicked. disabled: A boolean value that,

when true, disables the button.

className: A string that can be used to add custom styles to the button.

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

Answer:-To handle clicks on the MyButton component, you would need to create a function that is passed to the onClick prop.

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?**

Answer: -To change the list's appearance, we could pass the following props to the component:

items: an array of items to display in the list

ordered: a boolean indicating whether the list should be ordered (<ol>) or unordered (<ul>)

className: a string representing any additional CSS class names to apply to the list element

style: an object representing any additional CSS styles to apply to the list element

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

Answer: -To store the list items, we could use an array. For example:

```
const items = [

 "Item 1",
 "Item 2",

 "Item 3"

];
```

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

Answer:-To render the list items, we could use the map method to create an array of <li> elements:

```
            function MyList(props) {
            const { items, ordered, className, style } = props;

             const ListComponent = ordered ? "ol" : "ul";

             return (
    <ListComponent className={className} style={style}>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ListComponent>
  );
}
```

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?**

Answer:- The function "getWeather" would require the name of the city as a parameter, which would be used to retrieve the temperature for that particular location.

b)**What is the URL for the OpenWeatherMap API?**

Answer: -The URL for the OpenWeatherMap API is "https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}". Here, you would need to replace "{city name}" with the actual name of the city for which you want to retrieve the weather information, and "{API key}" with the API key that you have obtained from OpenWeatherMap.

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

Answer:-To handle errors or exceptions in this function, we can make use of try and except blocks. For example, if the API request fails due to an invalid API key or incorrect city name, we can catch the exception and return an error message to the user.

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?**

Answer:-The navigation method that we would use to navigate to the "ProfileScreen" would depend on the app's navigation stack. If the app is using React Navigation, we could use the "StackNavigation" method to push the "ProfileScreen" onto the stack and display it.

b) **What props would you pass to this screen to display the user's information?**
Answer:-To display the user's information on the "ProfileScreen", we would need to pass the user's data as props to the component. The props would contain the user's name, profile picture, email address, 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?** **Answer:-** To display the user's information on the "ProfileScreen", we could use several components, such as Text, Image, and View. For example, we could use a Text component to display the user's name, an Image component to display their profile picture, and a View component to wrap these components and style them.

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?**

Answer:-To use the "useLocalStorage" hook to store and retrieve data from local storage, we would first import the hook and call it with the desired key and initial value. The hook would return a tuple containing the current value and a setter function that we can use to update the value.

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

Answer:-To handle errors or exceptions in the "useLocalStorage" hook, we can make use of try and catch blocks. For example, if there is an error when attempting to retrieve or set an item in local storage, we can catch the exception and log it to the console. This will prevent the application from crashing and allow us to gracefully handle errors.

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?**

Answer:-To implement the "MyImagePicker" component and allow the user to select an image from their device's photo gallery, we could use the "react-native-image-picker" library. This library provides a crossplatform way to select an image from the device's photo gallery or take a new photo using the camera.

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

Answer:-We can pass several props to the "MyImagePicker" component to customize its appearance and behavior. For example, we could pass a "buttonText" prop to specify the text to display on the button that launches the photo gallery, a "containerStyle" prop to customize the style of the component's container, and a "onImageSelected" prop to handle the selected image URI.

c) **How would you handle errors or exceptions when selecting an image?**
Answer:-To handle errors or exceptions when selecting an image, we can wrap the image picker code in a try and catch block. If there is an error when launching the photo gallery or selecting an image, we can catch the exception and log it to the console. Additionally, we can use a conditional statement to check if the selected image URI is not null before attempting to display it in our app, to prevent our app from crashing if there is no image selected.

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

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

Answer:-To navigate to the "SettingsScreen" in our app, we could use the Stack Navigation method, which allows us to push and pop screens onto a navigation stack. When the user clicks on a "Settings" button or icon, we can push the "SettingsScreen" onto the navigation stack, and when the user is done making changes, we can pop the screen to return to the previous screen.

b) **What data structure would you use to store the user's settings?**
Answer:-To store the user's settings, we could use a simple JavaScript object or a more complex data structure like Redux or Context API. The choice of data structure would depend on the complexity of the app and the amount of data that needs to be store

c) **How would you save the user's settings so they persist across app sessions**?
Answer:-To save the user's settings so they persist across app sessions, we could use AsyncStorage in React Native. AsyncStorage is a simple key-value store that allows us to store data locally on the device. We could create a function in our "SettingsScreen" component that saves the user's settings to AsyncStorage when they make changes, and another function that retrieves the settings when the screen is loaded.

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

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

Answer:-To generate a random password with a given length and complexity, we would pass the following parameters to the "generatePassword" function: length: an integer representing the desired length of the password

complexity: a string or an array of strings representing the desired complexity of the password. For example, "lowercase" for lowercase letters, "uppercase" for uppercase letters, "digits" for digits, and "special" for special characters.

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

Answer:-To generate the random password, we can use the built-in Math.random() function in JavaScript to generate a random index for each character in the password. We can then use a combination of conditional statements and string concatenation to generate the password based on the desired complexity.

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

Answer:- To ensure that the password meets the required complexity criteria, we can use conditional statements to check if the password contains the required characters. For example, if the complexity parameter includes "lowercase", we can check if the password contains at least one lowercase letter. Similarly, we can check for uppercase letters, digits, and special characters.

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

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

Answer:-To specify the location, the following props could be passed to the "MyMap" component:

latitude: the latitude of the location to display on the map longitude: the longitude of the location to

display on the map zoom: the initial zoom level for the map display

apiKey: the API key for the Google Maps API

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

Answer:-To handle errors or exceptions when displaying the map, the following approach could be taken:
The component could render an error message or fallback content if an error occurs while loading the map
The component could also use a try-catch block to handle any exceptions that occur during the map loading process, and log the error to the console or display an error message to the user.

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

Answer:-To optimize the performance of the component when displaying large maps, the following approaches could be taken:

Use lazy-loading or dynamic loading techniques to only load the map when it's needed or when it becomes visible to the user.
Use a caching mechanism to store the map data for subsequent displays, so that it doesn't need to be reloaded each time the component is rendered.
Use a smaller map size or lower resolution, if possible, to reduce the amount of data that needs to be loaded and displayed.

Section D: Short Answer Questions (20 marks)

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

Answer:-"Props drilling" is a term used in React Native (and in React) to describe the situation where props need to be passed down through multiple levels of components in order to reach a child component that needs them. This can happen when a parent component needs to pass some data or functionality down to a deeply nested child component, but the intermediate components in the hierarchy don't actually use that data or functionality themselves. In this case, the data or functionality needs to be passed through each of these intermediate components in the form of props, even though they don't use them, hence the term "props drilling".

Props drilling can make the code more complex and harder to maintain, as the props have to be passed through multiple levels, making it difficult to trace where they are used or modified. Additionally, this can also reduce the performance of the app, as unnecessary props are passed down and can cause unnecessary re-renders.To avoid props drilling, other solutions such as using React Context, Redux, or other state management libraries can be used to manage the data in a centralized location and make it accessible to the necessary components without having to pass them through each intermediate component.

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

Answer:-In React Native, a "controlled" component is a component that is fully controlled by the React state, while an "uncontrolled" component is a component that manages its own state internally.

A controlled component relies on the parent component to manage its state and passes down the state values as props, which means that any changes made to the component's state must be handled by the parent component. When a user interacts with a controlled component, the component's state is updated through callbacks passed down from the parent component. This makes it easier to manage the component's state and data flow, but can result in more code and complexity.

An uncontrolled component, on the other hand, manages its own state internally and does not rely on the parent component to manage its state. Instead, it manages its state through its own internal event handlers and can update its state without involving the parent component. This makes the component simpler and easier to use, but can make it harder to manage the state and data flow.

Examples of controlled components include form inputs, where the input value is managed by the parent component and updated through callbacks, while examples of uncontrolled components include image views, where the image is loaded and managed by the component itself.

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

Answer:- In React Native, "component state" and "application state" refer to two different levels of state management.

Component state refers to the state that is managed within a specific component. This state is local to the component and is not shared with other components in the application. Component state can be managed using the useState hook or by extending the React. Component class and using its state property. Component state is typically used for managing data that is specific to the component, such as form input values or component visibility.
Application state, on the other hand, refers to the state that is shared across multiple components in the application. This state is typically managed at a higher level, such as in the top-level App component or using a state management library like Redux or MobX. Application state can be passed down to child components as props, or accessed using context or other state management techniques. Application state is typically used for managing data that is shared across multiple components, such as user authentication status or global app settings.In summary, component state is used to manage local state within a specific component, while application state is used to manage state that is shared across multiple components in the application.

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

Answer:-Redux is a state management library that is commonly used in React Native (and React) applications to manage application-level state in a centralized and predictable manner. It provides a set of principles and tools for managing state that can help improve the scalability and maintainability of large applications.

In Redux, the state of the entire application is stored in a single object called the "store". The store is read-only, and state updates are performed by dispatching "actions", which are plain JavaScript objects that describe what happened in the application. The store then uses "reducers", which are pure functions that take the current state and an action as input, and return a new state as output. Reducers are responsible for updating the state of the application based on the actions that are dispatched.

In React Native, Redux can be used to manage the state of the application in a centralized and predictable way, allowing multiple components to access and modify the same state without the need for complex data flow management between them. It can also help improve performance by minimizing unnecessary re-renders and by reducing the amount of state that needs to be passed down through props.

Redux is often used in conjunction with other libraries and frameworks in React Native, such as React Navigation or React Native Elements, to manage state related to navigation or UI elements. Overall, Redux is a powerful tool that can help simplify state management in complex React Native applications.

25. **How would you optimize the performance of a React Native app that has a large number of components?**
Answer:-There are several ways to optimize the performance of a React Native app with a large number of components. Here are a few suggestions:

1.Implement shouldComponentUpdate or React.memo to prevent unnecessary re-renders: By default, React re-renders a component every time its state or props change, even if the changes do not affect the visual appearance of the component. You can optimize this by implementing shouldComponentUpdate or React.memo to prevent unnecessary re-renders.

2.Use FlatList or SectionList instead of ScrollView: If you have a large number of items that need to be displayed in a list, consider using FlatList or SectionList instead of ScrollView. These components are optimized for displaying large lists and can improve the performance of your app.

3.Use PureComponent instead of Component: PureComponent is a subclass of Component that implements shouldComponentUpdate with a shallow prop and state comparison. If your component does not have any side effects and only depends on its props and state, using PureComponent can improve the performance of your app.

4.Optimize images and other assets: Large images and other assets can slow down your app's performance. Consider compressing images and other assets or lazy-loading them to improve the performance of your app.

5.Use the Chrome Performance tab to identify performance bottlenecks: Use the Chrome Performance tab to identify performance bottlenecks in your app. The Performance tab can help you identify slow components, memory leaks, and other performance issues.

6.Use a tool like Reactotron or Flipper to debug and optimize your app: Reactotron and Flipper are tools that can help you debug and optimize your React Native app. These tools can help you identify performance issues and optimize your app's performance.