1. a) A cross-platform framework for building mobile apps using JavaScript
2. a) To create a responsive design that adapts to different screen sizes
3. a) Props are used to pass data between components, while state is used to manage data within a component
4. d) CSS styles
5. a) To make HTTP requests to a server
6. a) setState() updates the state of a component and triggers a re-render, while forceUpdate() re-renders a component without updating its state
7. b) TouchableHighlight
8. b) To store data temporarily on the device
9. a) To create complex animations using JavaScript
10. d) All of the above

Section B: Programming Questions (80 marks)

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

Ans) For creating the app named MyApp you would use te command - npx react-native init MyApp

b) What is the directory structure of the app?

The directory structure of the app is as follows-

1. Tests
2. Android
3. Ios
4. Node modules
5. Gitigrone
6. Prettierrc.js
7. Watchmanconfig
8. App.js
9. App.json
10. Babel.config.js
11. Index.js
12. Package.json
13. readme.md

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

If the programmer wants to change the appearance then the programer must modify the app.js file. Which contains the root component of the app responsible for rendering app’s UI.

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

```
import React from 'react';
import { TouchableOpacity, Text } from 'react-native';

const MyButton = () => {
  return (
    <TouchableOpacity>
      <Text>Click me</Text>
    </TouchableOpacity>
  );
};

export default MyButton;
```

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

Various props to the touchableopacity component can be applied on myButton. Few of them are

1. Style: to apply custom components
2. Active Opacity :opacity when clicked
3. Disabled :it can be set to true or false

Example

```
<TouchableOpacity Style= {{backgorund:blue,
            Borderradius:8,
            padding:16,
}}
activeOpacity ={0.8}
onPress={onPress}
Disabled = {disabled}>
```

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

Usestate hook can used to handle clicks

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

```
import React from 'react';
import { FlatList, View, Text } from 'react-native';

const MyList = () => {
  const data = [
    { id: '1', title: 'Item 1' },
    { id: '2', title: 'Item 2' },
    { id: '3', title: 'Item 3' },
    { id: '4', title: 'Item 4' },
    { id: '5', title: 'Item 5' },
  ];

  const renderItem = ({ item }) => (
    <View>
```

```
      <Text>{item.title}</Text>
    </View>
  );

  return (
    <FlatList
      data={data}
      renderItem={renderItem}
      keyExtractor={(item) => item.id}
    />
  );
};

export default MyList
```

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

There are various props few of them are style,Contentcontianerstyle,numColumns,horizontal,itemseparatorComponent,ListhHeaderComponent

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

Using an array of objects to store the list of items. Where each object would represent single item where each item might contain various property.

Example

```
Const data = {
Id:'1',
title:'Item 1'}
```

c) How would you render the list items?

Its quite simple to render a list use define Flatlist using renderItem function which takes object as a input and returns a component that represents the item.

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

```
const getWeather = async (city) => {
  const apiKey = 'your-api-key-here';
  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();

    if (response.ok) {
      const temperature = data.main.temp;
      return temperature;
    } else {
```

```
      throw new Error(data.message);
    }
  } catch (error) {
    console.error(error);
  }
};
```

a) What parameters would you pass to this function?
Parameter would name of the city ex:- getWeather(“Hyderbad”);
b) What is the URL for the OpenWeatherMap API?
`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`;

c) How would you handle errors or exceptions in this function
In the simplest manner it would be by using try catch blocks

```
Try{
Const response = await fetch(URL);
Const data = await responser.json();

Const temperature = data.main.temp;
Return termperature
}
catch(error){
console.error(error);
}
```

15)
Create a new screen in your app called "ProfileScreen" that displays the user's profile information. import React from 'react';

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

const ProfileScreen = () => {
  const user = {
    name: 'Deepak Ram',
    phone: '123-456-7890',
    address: '123 side St',
    email: 'hehehe@example.com',
    institution: 'PQR Corporation',
  };

  return (
    <View style={styles.container}>
      <Text style={styles.label}>Name:</Text>
      <Text style={styles.value}>{user.name}</Text>
```

```
      <Text style={styles.label}>Phone:</Text>
      <Text style={styles.value}>{user.phone}</Text>

      <Text style={styles.label}>Address:</Text>
      <Text style={styles.value}>{user.address}</Text>

      <Text style={styles.label}>Email:</Text>
      <Text style={styles.value}>{user.email}</Text>

      <Text style={styles.label}>Institution:</Text>
      <Text style={styles.value}>{user.institution}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  label: {
    fontSize: 18,
    fontWeight: 'bold',
    marginTop: 10,
  },
  value: {
    fontSize: 16,
    marginTop: 5,
  },
});

export default ProfileScreen;
```

a) What navigation method would you use to navigate to this screen?
There is many navigation tool like stack,tab and drawer. But i Have used stack navigation because its easy to use
b) What props would you pass to this screen to display the user's information?
User’s name, phone number address, email and institution are stirred in state variables, and they would be passed them as props to the profile screen

```
<ProfileScreen
  name={this.state.name}
```

```
  phone={this.state.phone}
  address={this.state.address}
  email={this.state.email}
  institution={this.state.institution}
/>
```

c) What component(s) would you use to display the user's information?
Few of the components which could be sed are text flatlist and card. I have used card since it is more attractive

```
<Card>
  <Card.Title>User Information</Card.Title>
  <Card.Divider />
  <View style={{ marginVertical: 10 }}>
    <Text style={{ fontWeight: 'bold' }}>Name:</Text>
    <Text>Deepak Ram</Text>
  </View>
  <View style={{ marginVertical: 10 }}>
    <Text style={{ fontWeight: 'bold' }}>Phone:</Text>
    <Text>123-456-7890</Text>
  </View>
  <View style={{ marginVertical: 10 }}>
    <Text style={{ fontWeight: 'bold' }}>Address:</Text>
    <Text>123 side St.</Text>
  </View>
  <View style={{ marginVertical: 10 }}>
    <Text style={{ fontWeight: 'bold' }}>Email:</Text>
    <Text>hehehehe@email.com</Text>
  </View>
</Card>
```

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(() => {
    const storedValue = window.localStorage.getItem(key);
    return storedValue !== null ? JSON.parse(storedValue) : initialValue;
  });

  useEffect(() => {
    window.localStorage.setItem(key, JSON.stringify(value));
```

```
  }, [key, value]);

  return [value, setValue];
}
```

a) How would you use this hook to store and retrieve data from local storage?
It can accomplished by using the custom made hook useloaclstorage by calling it the functional component and passing a key and value as a argument. Then the hook will return a tuple containing the current value and a setter function where this setter function can be used to update the value in the local storage
The code would as follows

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

function App() {
  const [name, setName] = useLocalStorage('name', 'John Doe');

  const handleInputChange = (event) => {
    setName(event.target.value);
  }

  return (
    <div>
      <input type="text" value={name} onChange={handleInputChange} />
      <p>Hello, {name}!</p>
    </div>
  );
}

export default App;
```

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

To handle errors in the hook we just use the try catch block a modified version of uselocal storage with try catch would look as follows

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

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

```
    } catch (error) {
      console.error(error);
      return initialValue;
    }
  });

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

  return [value, setValue];
}
```

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

```
import React, { useState } from 'react';
import { Button, Image, StyleSheet, View } from 'react-native';
import ImagePicker from 'react-native-image-picker';

const MyImagePicker = () => {
  const [image, setImage] = useState(null);

  const handleChoosePhoto = () => {
    ImagePicker.launchImageLibrary({ mediaType: 'photo' }, response => {
      if (response.uri) {
        setImage(response);
      }
    });
  };

  return (
    <View style={styles.container}>
      {image && <Image source={{ uri: image.uri }} style={styles.image} />}
      <Button title="Choose Photo" onPress={handleChoosePhoto} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
```

```
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  image: {
    width: 200,
    height: 200,
    resizeMode: 'contain',
  },
});

export default MyImagePicker;
```

) What external library or module would you use to implement this component?
I would use react-native-image-picker library

b) What props would you pass to this component to customize its appearance?
The following components can be added

1. buttonText
2. buttonStyle
3. Imagestyle
4. defaulImage
5. imageQuality
6. image Width
7. imageHeight

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

```
import { ImagePicker } from 'react-native-image-picker';

const handleImageSelection = () => {
  try {
    ImagePicker.launchImageLibrary({}, response => {
      if (response.didCancel) {
        console.log('User cancelled image picker');
      } else if (response.error) {
        console.log('ImagePicker Error: ', response.error);
      } else if (response.customButton) {
        console.log('User tapped custom button: ', response.customButton);
      } else {
        const source = { uri: response.uri };
        // Do something with the selected image
      }
    });
  } catch (error) {
    console.log('Error selecting image: ', error);
  }
};
```

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

```
const generatePassword = (length, complexity) => {
  let password = '';
  const lowercaseChars = 'abcdefghijklmnopqrstuvwxyz';
  const uppercaseChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  const numericChars = '0123456789';
  const specialChars = '!@#$%^&*()_+~`|}{[]\:;?><,./-=';
  let charSet = '';

  if (complexity === 'weak') {
    charSet = lowercaseChars;
  } else if (complexity === 'moderate') {
    charSet = lowercaseChars + uppercaseChars + numericChars;
  } else if (complexity === 'strong') {
    charSet = lowercaseChars + uppercaseChars + numericChars + specialChars;
  }

  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * charSet.length);
    password += charSet[randomIndex];
  }

  return password;
};
```

what algorithm or library would you use to generate the password?
In the program i Have used i have used simple Math.random function in javascript to generate a random password. It could be useful only generate simple passwords
) How would you ensure that the password meets the required complexity criteria?
To achieve the a required criteria we can use the following code

```
const generatePassword = (length, complexity) => {
  let password = '';
  const lowercaseChars = 'abcdefghijklmnopqrstuvwxyz';
  const uppercaseChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  const numericChars = '0123456789';
  const specialChars = '!@#$%^&*()_+~`|}{[]\:;?><,./-=';
  let charSet = '';
  let hasLowercase = false;
  let hasUppercase = false;
  let hasNumeric = false;
  let hasSpecial = false;
```

```
  if (complexity === 'weak') {
    charSet = lowercaseChars;
  } else if (complexity === 'moderate') {
    charSet = lowercaseChars + uppercaseChars + numericChars;
  } else if (complexity === 'strong') {
    charSet = lowercaseChars + uppercaseChars + numericChars + specialChars;
  }

  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * charSet.length);
    const randomChar = charSet[randomIndex];
    password += randomChar;
    if (lowercaseChars.includes(randomChar)) {
      hasLowercase = true;
    } else if (uppercaseChars.includes(randomChar)) {
      hasUppercase = true;
    } else if (numericChars.includes(randomChar)) {
      hasNumeric = true;
    } else if (specialChars.includes(randomChar)) {
      hasSpecial = true;
    }
  }

  if (
    (complexity === 'weak' && hasLowercase) ||
    (complexity === 'moderate' && hasLowercase && hasUppercase && hasNumeric) ||
    (complexity === 'strong' && hasLowercase && hasUppercase && hasNumeric &&
hasSpecial)
  ) {
    return password;
  } else {
    return generatePassword(length, complexity);
  }
};
```

Section D: Short Answer Questions (20 marks)

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

Props drilling refers to the practise of handing down properties (short for properties) across numerous levels of nested components in order to access them at a deeper level in React and React Native.

When a parent component has to provide data down to a child component, it can do so by passing props to the child component as an argument. These props can also be passed

down as an argument if the child component has its own child component, and so forth. This may lead to a situation where the same props are being passed down through numerous tiers of components, which can be annoying and challenging to handle.
Props drilling's biggest drawback is that it can make the code more difficult to comprehend, maintain, and refactor. When the same props are sent down through numerous tiers of components—even when some of those components don't require them—it can occasionally have a detrimental effect on performance.

State management, which entails using a centralised state management solution like Redux, MobX, or the React Context API to store and manage the application state, is a strategy that developers can employ to avoid props drilling. This eliminates the requirement for components to pass down props across numerous levels of nested components in order to access the state they require.

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

An "uncontrolled component" in React Native is one whose value is determined by the user's input, as opposed to a "controlled component" whose value is controlled by React.

A controlled component is commonly created by assigning the value attribute of the component to a state variable, which is subsequently changed anytime the user interacts with the component. The state then always determines the value of the component, and any modifications to the value are immediately reflected in the state.
An uncontrolled component, on the other hand, is one whose value is not controlled by React but rather by the user's input. Uncontrolled components are frequently used when the value of the component can be determined by the component's default behaviour or when you don't need to explicitly access the component's value in your code.

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

The term "component state" describes the internal state of a single component that is controlled by the useState or useReducer hooks. The state of a component is unique to that component and cannot be accessed by other components of the application. Data relevant to a given component, such input values, loading states, or error messages, are stored there.

On the other hand, application state describes the state that various application components share. Usually, a state management library like Redux, MobX, or the React Context API is used to manage it. Data that is shared throughout many components, such as user authentication state or data acquired from an API, is stored in application state.

24. What is Redux and how does it relate to React Native?
Redux is a state management library for JavaScript applications, including React Native. It provides a predictable way to manage application state by creating a centralized store that holds the entire state tree of the application. Redux allows you to write more maintainable,

scalable, and testable code, by separating the concerns of state management from the UI components.
In a Redux architecture, the state is immutable, which means that the state can only be changed by dispatching an action. An action is a plain JavaScript object that describes a change in the state, including the type of the action and any relevant data. A reducer is a pure function that takes the current state and an action, and returns a new state. Reducers are used to update the state based on the dispatched actions.
Redux also provides middleware, which can be used to modify the behavior of the store or to add additional functionality, such as logging or asynchronous actions.
In React Native, Redux is often used to manage the application state, especially for complex applications with multiple components and screens. Redux can help to simplify the management of state across multiple components and screens, and to ensure consistency and predictability of the state.
To use Redux in a React Native application, you need to install the redux and react-redux packages. The redux package provides the core Redux functionality, while the react-redux package provides the bindings between Redux and React Native components. You can create a store using the createStore function from the redux package, and then wrap your root component with the Provider component from the react-redux package, which makes the store available to all components in the application

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

1. Use the VirtualizedList component: If your app has a large list of items, consider using the VirtualizedList component instead of the regular FlatList or ScrollView. VirtualizedList only renders the visible items on the screen, which can improve the app's performance by reducing the number of components that need to be rendered and updated.
2. Implement shouldComponentUpdate or React.memo: You can improve the performance of components by implementing the shouldComponentUpdate lifecycle method or using the React.memo higher-order component. These approaches can prevent unnecessary re-renders of components by checking whether the component's props or state have changed. This can be particularly useful for complex components or components that are frequently updated.
3. Use PureComponent: PureComponent is a built-in React component that automatically implements shouldComponentUpdate based on shallow comparisons of props and state. Using PureComponent instead of a regular component can help to reduce the number of unnecessary re-renders.