Getting Started Guide

React Native Guide

The following guide provides a short introduction into the core principles of React Native. It covers creating a simple layout, handling basic callback interaction, and navigating between screens.
At the end the usage of the created SDK components for the final project is outlined.

1. React Native

React Native is a popular framework for building cross-platform mobile apps using JavaScript and the React library. Developed by Facebook, it bridges the gap between web development and native platforms, enabling developers to utilize familiar React concepts to create high-performance iOS and Android, as well as web applications with a single codebase.
This is possible through a communication layer that translates JavaScript code into native APIs, ensuring near-native performance. Additionally, React Native utilizes a component-based architecture, which makes it easy to reuse and share components across different platforms. Integration with third-party libraries and native modules is streamlined over package managers like NPM , giving developers direct access to platform-specific features and hardware.
Convenience features like hot reloading reduce development time, allowing developers to view changes on real devices or simulators instantly.

2. Set Up Your Environment

1. **Install Node.js**: React Native uses Node.js and npm (or yarn) for package management.
2. **Install the Expo CLI:** Expo simplifies project setup and testing.

```
npm install expo-cli
```

3. **Create a new project:** Use the blank(Typescript) template and name your app

```
npx create-expo-app --template
// Follow wizard
cd your-app-name
```

4. **Install additional packages**

```
npm install react react-native typescript @types/react @types/react-native react-native-web
```

5. **Run the project:** And choose the platform to run on.

```
npx expo start
```

3. Basic Project Structure

React Native provides components that replace the usage of HTML elements.

- **View:** Acts as a content container that wraps all child components.
- **Text:** Basic component for text elements.

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

export default function App() {

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Hello, React Native!</Text>
      <Text>This is a simple layout.</Text>
    </View>
  );
}
```

The appearance of the React Native components is defined in a stylesheet and replaces the usage of a classic CSS stylesheet.
The defined style properties are then passed to the components as seen above, typically through the style prop.

```
const styles = StyleSheet.create({

  container: { flex: 1, justifyContent: 'center', alignItems: 'center' },
  title: { fontSize: 24, fontWeight: 'bold' },

  }
);
```

4. Basic Interaction with Callbacks

Button and Pressable
In order to detect clicks or taps built-in components such as **Button** or **Pressable** can be used:

- **useState:** React Hook that lets you manage state.

- **Button** provides a basic button. The **onPress** prop is your callback function.
- **Pressable** for more control over different press states

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

export default function App() {

  const [count, setCount] = useState(0);

  const handlePress = () => {
    setCount(count + 1);
  };


  return (
    <View style={{ marginTop: 50, padding: 20 }}>
      <Text>Clicked {count} times</Text>
      <Button title="Click me" onPress={handlePress} />
          <Pressable onPress={handlePress}>
                  <Text>Press Me</Text>
          </Pressable>
    </View>

  );

}
```

5. Navigation Between Screens

As most apps have multiple screens, the possibility to navigate between them is required. In order to manage navigation easily, libraries like **React Navigation** can be used.

Installing React Navigation

```
npm install @react-navigation/native
npm install @react-navigation/native-stack
```

In addition, install the required dependencies (React Navigation docs for details).

Setting Up a Stack Navigator
Create a file (App.js) and define two screens (HomeScreen and DetailsScreen).

- **createNativeStackNavigator():** Initiates the navigation object.

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { View, Text, Button } from 'react-native';

const Stack = createNativeStackNavigator();
```

The HomeScreen acts as the initial starting point of the app navigation. On the press of a button a detail page is opened.

- **onPress:** Registers a callback that executes the registered function on the button press.
- **Navigation.navigate('Screen_name'):** Function to move from one screen to another.

```
function HomeScreen({ navigation }) {
  return (
    <View>
      <Text>Welcome to Home Screen!</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}
```

This screen represents a simple new page with further detail. The higher level navigation header can be used to navigate back to the HomeScreen

```
function DetailsScreen() {

  return (
    <View>
      <Text>More details.</Text>
    </View>
  );
}
```

And bring it all together in the main *App()* function.

- **NavigationContainer:** Wraps your whole navigation tree.
- **Stack.Navigator:** Holds the Stack of Screens.
- **Stack.Screen:** Adds individual screens to the stack.

```
export default function App() {

  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

6. Next Topics

To learn more about React Native you can follow the React Native Docs and have a closer look at the following concepts:
• **Styling**: Learn more about React Native's styling based on Flexbox.
• **State Management**: Pass and modify data between screens.
• **API Calls**: Use fetch or libraries for dynamic data fetching.
• **Native Modules**: Enable lower-level device features or platform-specific styling.

Final Project - SDK Setup

For the Final project, two UI patterns were implemented. The components can be separately installed as a package over npm.

General guide for uploading to npm

Make sure all required properties are set in the package.json file

```
"name": "react-native-name-component",
"version": "1.0.0",
"main": "index.ts"
...
```

Publish to npm

```
npm login

// Follow web wizard

npm publish --access public
```

Testimonials Component

The Testimonial UI pattern Link focuses on the visual customization of the provided component. In general, the testimonial component can be utilized to increase trust in a product or brand by positive attestations of previous users or customers.
The visual customization is crucial in order to be adapted to a design brief. The ability to describe the author of the testimonial is crucial as well in order to switch between revealed information about clients or the differentiation between users, clients, brands, or products.

Setup

Install package

```
npm install react-native-testimonials
```

Import components

```
import { TestimonySection, TestimonyItem } from 'react-native-testimonials';
```

Initialize components

```
export default function App() {
        const profilePicture1 = require('./profile_pictures/profil_picture_1.png');
        const profilePicture2 = require('./profile_pictures/profil_picture_2.png');

        const sampleTestimonies: TestimonyItem[] = [
                {
                        author_name: 'John Doe',
                        author_description: 'Engineer',
                        picture: profilePicture1,
                        text_comment: 'This product exceeded my expectations! Easy to use, high quality.',
                },
                {
                        author_name: 'Jane Smith',
                        author_description: 'Designer',
                        picture: profilePicture2,
                        text_comment: 'Excellent performance and reliable results. Highly recommend it!',
                },
        ];


        return (
                <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
                        <TestimonySection
                                // Color Styling
                                color_background="#F0F0F0"
                                color_title="#000000"
                                color_subtitle="#555555"
                                color_testemony_background="#464CF6"
                                color_testemony_text="#D9D9D9"
                                color_testemony_author_name="#000000"
                                color_testemony_author_description="#777777"

                                // Font used for all text
                                font="Helvetica"

                                // Text values
                                text_title="What Our Customers Say"
                                text_subtitle="Real feedback from real users"

                                // Testimonies
                                testimonies={sampleTestimonies}
                        />
                </View>
        );
        }
```

Password Strength Meter Component

The Password Strength Meter UI pattern Link focuses on the functional customization of the provided component. The strength meter helps users to evaluate the security level of their chosen password and can ensure that security risks are reduced and user data remains save.

The functional customization is key in this case as the requirements on what counts as a secure password depend on the estimated risk level and technical security requirements. The visual customization is provided in order to be adapted to the design of the application.

In more detail:

- **PasswordForm** sums up each QualityAttribute's *current_sec_value* to produce *current_sec_score*.
- The **"Continue Button** is only enabled once *current_sec_score* surpasses the *threshold_sec_score*.
- The security level is displayed over the **shadow color** for the "card" and the **progress bar**. Both change at predefined ratio values (weak → medium → strong) based on *current_sec_score* / *max_sec_score*
- Each QualityAttribute defines its specific security values and an evaluate() function which is called every time the input of the password field changes. This allows the integration of a custom behavior. Multiple QualityAttributes can be defined.
- A QualityAttribute is checked as soon as its *current_sec_value* surpasses the minimal requirements posted by *sec_threshhold*.

Setup

Install package

```
npm install react-native-password-meter-component
```

Import components

```
import { PasswordForm, PasswordFormData, QualityAttribute,} from 'react-native-password-meter-component';
```

Initialize components

```
export default function App() {

// Visual Customization
const passwordFormData: PasswordFormData = {

        // Color Styling
        color_background: '#eeeeee',
        color_background_from: '#888888',
        color_background_password_field: '#ffffff',
        color_title: '#ffffff',
        color_password_text: '#000000',
        color_continue_button_active: '#6200EE',

        text_font: 'System',

        // Colors to show password strength
        current_color_shadow: '#808080', // default grey

        color_shadow_weak: 'red',
        color_shadow_medium: 'orange',
        color_shadow_strong: 'lightgreen',

        image_quality_attribute_checked: require('./assets/checked.png'),   // Icon for satisfied Quality Attribute
        image_quality_attribute_unchecked: require('./assets/unchecked.png'), // Icon for unsatisfied Quality Attribute

        image_lock: require('./assets/lock.png'),

};

// functional Customization
const QualityAttributes: QualityAttribute[] = [{
        name: 'At least 8 chars',
        checked: false,
        max_sec_value: 20,
        current_sec_value: 0,
        sec_threshhold: 8,

        evaluate: (passwordText: string) => {
        // Example Logic – Implements the modular functionality to evaluate the strength of a password based on defined
properties.
                return Math.min(passwordText.length, 20)
        },

},
];

// Build component
return (
        <SafeAreaView style={{ flex: 1 }}>
                <PasswordForm
                        passwordForm={passwordFormData}
                        qualityAttributes={QualityAttributes}
                        threshold_sec_score={20}
                        title="Set Your Password"
                />
        </SafeAreaView>
);
```