React Native Forms & Input Handling: Beginner to Advanced

Subtitle: Learn how to create interactive forms, capture user input, and validate data in React Native apps.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Forms are essential for collecting user input, from login screens to feedback forms. This guide covers **TextInput, state management, validation, and submission** in React Native, with practical examples for beginners.

Step 1: Using TextInput

`TextInput` allows users to enter text.

Example:

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

export default function App() {
  const [name, setName] = useState('');

  return (
    <View style={{ padding: 20 }}>
      <TextInput
        placeholder="Enter your name"
        value={name}
        onChangeText={text => setName(text)}
        style={{ borderWidth: 1, padding: 10, marginBottom: 10 }}
      />
      <Text>Your name is: {name}</Text>
    </View>
  );
}
```

Exercise: Create a TextInput for **email** and display it dynamically below the input field.

Step 2: Handling Multiple Inputs

Use **multiple state variables** or a single state object for multiple inputs.

Example:

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

export default function App() {
  const [form, setForm] = useState({ email: '', password: '' });

  return (
    <View style={{ padding: 20 }}>
      <TextInput
        placeholder="Email"
        value={form.email}
        onChangeText={text => setForm({ ...form, email: text })}
        style={{ borderWidth: 1, padding: 10, marginBottom: 10 }}
      />
      <TextInput
        placeholder="Password"
        value={form.password}
        secureTextEntry
        onChangeText={text => setForm({ ...form, password: text })}
        style={{ borderWidth: 1, padding: 10, marginBottom: 10 }}
      />
      <Text>Email: {form.email}</Text>
      <Text>Password: {form.password}</Text>
    </View>
  );
}
```

Exercise: Add **username** to the form and display all three inputs dynamically.

Step 3: Form Submission

Handle submission using **Buttons** and event handlers:

Example:

```
import { Button, Alert } from 'react-native';
```

```
<Button
  title="Submit"
  onPress={() => Alert.alert(`Email: ${form.email}, Password: ${form.password}`)}
/>
```

Exercise: Create a **login form** that validates email and password and shows an alert on submission.

Step 4: Input Validation

- Check for empty fields.
- Validate email format using regex.
- Password length should meet minimum requirements.

Example:

```
const handleSubmit = () => {
  if (!form.email || !form.password) {
    alert('All fields are required');
  } else if (!/\S+@\S+\.\S+/.test(form.email)) {
    alert('Invalid email format');
  } else {
    alert('Form submitted successfully!');
  }
};
```

Exercise: Add **password confirmation** input and validate it matches the password.

Step 5: Best Practices

- Use **controlled components** (`value` + `onChangeText`) for reliable state.
- Validate inputs **before submission**.
- Give users **instant feedback** for invalid inputs.
- Use **keyboardType** (`email-address`, `numeric`) for better UX.
- Keep forms **simple and responsive** for all screen sizes.

Forms and input handling are crucial for interactive apps. Mastering these concepts allows you

Forms and input handling are crucial for interactive apps. Mastering these concepts allows you to create **login screens, registration forms, and data-driven interfaces** effectively.

Visit **haas.dev** for more React Native guides, tutorials, and project examples.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
