|  |
| --- |
| export default function App() {  const [activeTab, setActiveTab] = useState('Calculatrice');  const [display, setDisplay] = useState('');  // ... (reste du code)  } |

Composant App (Logique principale de la calculatrice)

 export default function App(): Ceci définit ton composant fonctionnel principal nommé App. export default en fait le composant principal exporté depuis ce fichier.

 **Variables d'état (useState)**:

* const [activeTab, setActiveTab] = useState('Calculatrice');:
  + activeTab: Une variable d'état qui contient l'onglet actuellement actif (bien que dans ce code, il n'y ait qu'un seul onglet : "Calculatrice"). Sa valeur initiale est 'Calculatrice'.
  + setActiveTab: La fonction utilisée pour mettre à jour l'état activeTab.
* const [display, setDisplay] = useState('');:
  + display: Cette variable d'état stocke la chaîne de caractères qui est affichée sur l'écran de la calculatrice (l'entrée et le résultat). Sa valeur initiale est une chaîne vide.
  + setDisplay: La fonction utilisée pour mettre à jour l'état display.

Fonction

|  |
| --- |
| const handlePress = (value) => {  if (value === 'C') return setDisplay(''); // Effacer  if (value === '=') { // Évaluer  try {  const result = eval(display.replace('×', '\*').replace('÷', '/'));  setDisplay(result.toString());  } catch (e) {  setDisplay('Erreur');  }  } else if (value === '⌫') { // Retour arrière  setDisplay(display.slice(0, -1));  } else if (value === ',') { // Point décimal  if (!display.includes('.')) {  setDisplay(display + '.');  }  } else { // Ajouter d'autres boutons  setDisplay(display + value);  }  }; |

C'est la logique principale de la façon dont la calculatrice réagit lorsqu'un bouton est pressé.

* const handlePress = (value) => { ... };: Cette fonction est appelée chaque fois qu'un bouton de la calculatrice est pressé. Elle prend la **value** du bouton pressé comme argument (par exemple, '7', '+', '=', 'C').
* **Logique des boutons**:
  + if (value === 'C') return setDisplay('');: Si 'C' (Effacer) est pressé, l'affichage est réinitialisé à une chaîne vide.
  + if (value === '=') { ... }: Si '=' est pressé :
    - try { ... } catch (e) { ... }: Un bloc try-catch est utilisé pour gérer les erreurs potentielles pendant le calcul (par exemple, division par zéro, expressions invalides).
    - const result = eval(display.replace('×', '\*').replace('÷', '/'));:
      * display.replace('×', '\*').replace('÷', '/'): Remplace les symboles "×" et "÷" par les opérateurs de multiplication (\*) et de division (/) JavaScript standard, respectivement, afin que eval() puisse les comprendre.
      * **eval()**: C'est une fonction JavaScript qui évalue une chaîne de caractères comme du code JavaScript. **Bien que pratique pour les calculatrices simples, eval() est généralement déconseillé dans les applications de production en raison des risques de sécurité (il peut exécuter du code arbitraire). Pour une calculatrice réelle, tu utiliserais généralement une bibliothèque d'analyse d'expressions mathématiques plus sûre.**
      * setDisplay(result.toString());: Définit l'état display sur le result calculé, converti à nouveau en chaîne.
    - catch (e) { setDisplay('Erreur'); }: Si une erreur se produit pendant eval(), l'affichage montre "Erreur".
  + else if (value === '⌫') { setDisplay(display.slice(0, -1)); }: Si '⌫' (retour arrière) est pressé, il supprime le dernier caractère de la chaîne display en utilisant slice(0, -1).
  + else if (value === ',') { if (!display.includes('.')) { setDisplay(display + '.'); } }: Si ',' (virgule, traitée comme un point décimal) est pressée, elle ajoute un '.' à l'affichage, mais **seulement si un point décimal n'est pas déjà présent** dans la chaîne display actuelle.
  + else { setDisplay(display + value); }: Pour tout autre bouton (chiffres, opérateurs comme '+', '-', etc.), sa value est simplement ajoutée à la chaîne display actuelle.

Tableaux tabs et buttons

|  |
| --- |
| const tabs = ['Calculatrice'];  const buttons = [  ['C', '%', '⌫', '÷'],  ['7', '8', '9', '×'],  ['4', '5', '6', '-'],  ['1', '2', '3', '+'],  ['00', '0', ',', '='],  ]; |

 const tabs = ['Calculatrice'];: Un tableau définissant les onglets disponibles. Actuellement, il n'y en a qu'un seul.

 const buttons = [...]: Un tableau imbriqué représentant la disposition des boutons de la calculatrice. Chaque tableau interne est une rangée de boutons. Cette structure facilite le rendu dynamique du clavier.

5. Instruction return (JSX - Structure de l'interface utilisateur)

|  |
| --- |
| return (  <SafeAreaView style={styles.safeArea}>  <StatusBar barStyle="light-content" backgroundColor="#000" />  <View style={styles.container}>  {/\* Onglets \*/}  <View style={styles.tabs}>  {tabs.map(tab => (  <TouchableOpacity key={tab} onPress={() => setActiveTab(tab)}>  <Text style={[styles.tabText, activeTab === tab && styles.activeTab]}>{tab}</Text>  </TouchableOpacity>  ))}  </View>  {/\* Affichage \*/}  <View style={styles.display}>  <Text style={styles.displayText}>{display}</Text>  </View>  {/\* Clavier \*/}  <View style={styles.keyboard}>  {buttons.map((row, rowIndex) => (  <View key={rowIndex} style={styles.row}>  {row.map((btn, index) => (  <TouchableOpacity  key={index}  style={[styles.button, btn === '=' && styles.equalButton]}  onPress={() => handlePress(btn)}  >  <Text style={styles.buttonText}>{btn}</Text>  </TouchableOpacity>  ))}  </View>  ))}  </View>  </View>  </SafeAreaView>  ); |

C'est le JSX (JavaScript XML) qui décrit l'interface utilisateur de ta calculatrice.

* **<SafeAreaView style={styles.safeArea}>**: S'assure que le contenu est affiché dans la zone visible "sûre" de l'écran de l'appareil, évitant les découpes comme les encoches.
  + <StatusBar barStyle="light-content" backgroundColor="#000" />: Configure la barre d'état :
    - barStyle="light-content": Définit le texte et les icônes de la barre d'état sur une couleur claire (adaptée à un fond sombre).
    - backgroundColor="#000": Définit la couleur de fond de la barre d'état en noir.
* **<View style={styles.container}>**: Le conteneur principal pour tous les éléments de la calculatrice.
* **Section des onglets (<View style={styles.tabs}>)**:
  + {tabs.map(tab => (...))}: Itère sur le tableau tabs. Pour chaque tab (qui est juste 'Calculatrice' dans ce cas) :
    - <TouchableOpacity key={tab} onPress={() => setActiveTab(tab)}>: Une zone tactile (bouton) pour chaque onglet. onPress ne fait actuellement que définir l'état activeTab, mais dans une application à plusieurs onglets, cela changerait les vues.
    - <Text style={[styles.tabText, activeTab === tab && styles.activeTab]}>{tab}</Text>: Affiche le nom de l'onglet. Le tableau style applique toujours styles.tabText, et styles.activeTab uniquement si l'onglet est l'activeTab actuel.
* **Section d'affichage (<View style={styles.display}>)**:
  + <Text style={styles.displayText}>{display}</Text>: Affiche la valeur actuelle de la variable d'état display (les nombres et les résultats).
* **Section du clavier (<View style={styles.keyboard}>)**:
  + {buttons.map((row, rowIndex) => (...))}: Itère sur le tableau buttons (qui contient des rangées de boutons). Pour chaque row :
    - <View key={rowIndex} style={styles.row}>: Un composant View représentant une seule rangée de boutons.
    - {row.map((btn, index) => (...))}: Itère sur les boutons de cette row. Pour chaque btn (par exemple, '7', '+') :
      * <TouchableOpacity key={index} style={[styles.button, btn === '=' && styles.equalButton]} onPress={() => handlePress(btn)}>: Un bouton tactile pour chaque touche de la calculatrice.
        + style={[styles.button, btn === '=' && styles.equalButton]}: Applique le styles.button par défaut et, si le bouton est '=', il applique également styles.equalButton (pour une couleur d'arrière-plan différente).
        + onPress={() => handlePress(btn)}: Lorsque ce bouton est pressé, il appelle la fonction handlePress, en passant la valeur btn du bouton.
      * <Text style={styles.buttonText}>{btn}</Text>: Affiche le texte du bouton (par exemple, '7', 'C', '=').