

Le Hook `useState` en React Native

1. Introduction aux Hooks

- En React (et React Native), les *Hooks* permettent d'utiliser l'état et d'autres fonctionnalités de React sans avoir besoin de classes.
 - `useState` est le **Hook le plus utilisé** pour gérer l'état dans un composant fonctionnel.
-

2. Importation de `useState`

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

3. Syntaxe de `useState`

```
const [state, setState] = useState(valeurInitiale);
```

- `state` : la variable d'état.
 - `setState` : fonction pour mettre à jour cette variable.
 - `valeurInitiale` : valeur de départ (string, number, booléen, objet, etc.).
-

4. Exemple simple : compteur

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

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

  return (
    <View style={{ padding: 20 }}>
      <Text>Vous avez cliqué {count} fois</Text>
      <Button title="Incrémenter" onPress={() => setCount(count + 1)} />
    </View>
  );
};

export default Compteur;
```

5. Utiliser `useState` avec des types différents

Voici des exemples pratiques pour chaque type d'état (`useState`) en React Native :

✓ a. Texte

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

export default function ExempleTexte() {
  const [nom, setNom] = useState('');

  return (
    <View style={{ padding: 20 }}>
      <TextInput
        placeholder="Entrez votre nom"
        value={nom}
        onChangeText={setNom}
        style={{ borderWidth: 1, padding: 10 }}
      />
      <Text>Bonjour, {nom} !</Text>
    </View>
  );
}
```

✓ b. Booléen (toggle)

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

export default function ExempleToggle() {
  const [isVisible, setIsVisible] = useState(false);

  return (
    <View style={{ padding: 20 }}>
      <Button
        title={isVisible ? "Cacher le message" : "Afficher le message"}
        onPress={() => setIsVisible(!isVisible)}
      />
      {isVisible && <Text>Message visible </Text>}
    </View>
  );
}
```

✓ c. Objet

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

export default function ExempleObjet() {
  const [user, setUser] = useState({ nom: '', age: 0 });

  return (
    <View style={{ padding: 20 }}>
      <Button title="Mettre à jour le nom" onPress={() => setUser({
        ...user, nom: 'Ali' })} />
      <Button title="Augmenter l'âge" onPress={() => setUser({ ...user,
        age: user.age + 1 })} />
      <Text>Nom : {user.nom}</Text>
      <Text>Âge : {user.age}</Text>
    </View>
  );
}
```

```
        </View>
    );
}
```

✓ d. Liste

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

export default function ExempleListe() {
  const [tasks, setTasks] = useState([]);

  const ajouterTache = () => {
    const nouvelleTache = { id: tasks.length + 1, text: 'Lire un livre' };
    setTasks([...tasks, nouvelleTache]);
  };

  return (
    <View style={{ padding: 20 }}>
      <Button title="Ajouter une tâche" onPress={ajouterTache} />
      <FlatList
        data={tasks}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => <Text>{item.text}</Text>}
      />
    </View>
  );
}
```

6. Cas d'usage courants en React Native

- Gestion des champs de formulaire (TextInput)
 - Afficher/masquer un composant (Modal, View, ScrollView)
 - Sélection de boutons (RadioButton, Checkbox)
 - État de chargement (loading)
-

7. Bonnes pratiques

- Ne jamais modifier directement une variable d'état :

```
// ✗ Mauvais
state = 'nouvelle valeur';
// ✓ Bon
setState('nouvelle valeur');
```

- Toujours utiliser des copies avec les objets/lists (immutabilité).
-



TP – Mini Application “Gestion Utilisateur & Tâches”

🎯 Objectif général :

Développer une application React Native simple en utilisant le hook `useState` pour gérer différents types de données (texte, booléen, objet, tableau). L'application permettra de :

- saisir le nom d'un utilisateur,
 - afficher ou masquer ses informations,
 - modifier ses données,
 - ajouter dynamiquement des tâches à une liste.
-



Compétences visées :

- Manipuler le **state** avec le hook `useState`.
 - Utiliser des types de données variés (texte, booléen, objet, liste).
 - Créer une interface utilisateur réactive.
 - Gérer les événements (`onChangeText`, `onPress`).
 - Afficher dynamiquement une liste avec `FlatList`.
-

📝 Consignes :

1. Initialisation du projet

- Crée un projet React Native avec `Expo` ou `React Native CLI`.
 - Mets en place un composant `App()` principal avec une structure de base.
-

2. Saisie du nom (Texte)

- Déclare une variable d'état `nom` avec `useState`.
 - Affiche un champ `TextInput` pour permettre à l'utilisateur de saisir son nom.
 - Le texte saisi doit être stocké dynamiquement dans l'état `nom`.
-

3. Stockage de l'utilisateur (Objet)

- Crée un objet `user` avec deux propriétés : `nom` et `age`.
- Ajoute un bouton “Mettre à jour l'utilisateur” :

- Lorsqu'on clique dessus, la propriété `nom` de `user` doit être mise à jour avec la valeur saisie précédemment.
 - L'âge doit être initialisé à 0.

4. Affichage conditionnel (Booléen)

- Crée un état `isVisible` de type booléen.
 - Ajoute un bouton “Afficher les infos” / “Cacher les infos” :
 - Ce bouton doit inverser la valeur de `isVisible`.
 - Si `isVisible` est `true`, afficher les données de `user` (nom et âge).
 - Ajouter un bouton “Augmenter l’âge” dans cette section visible.

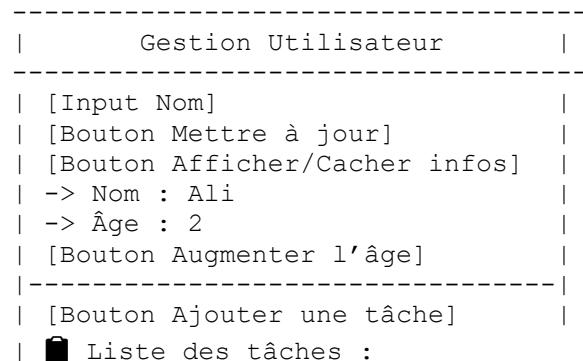
5. Liste de tâches

- Crée un tableau `tasks` dans le state.
 - Ajoute un bouton “Ajouter une tâche” :
 - Lors du clic, ajoute une nouvelle tâche dans la liste.
 - Chaque tâche doit contenir un `id` unique et un `text` du type : “Tâche de ”.
 - Utilise `FlatList` ou une autre méthode pour afficher dynamiquement la liste des tâches.

Astuces :

- Utilise la fonction `setUser({ ...user, ... })` pour mettre à jour partiellement l'objet.
 - Pour ajouter une tâche dans un tableau existant, utilise la syntaxe :
`setTasks([...tasks, nouvelleTache]);`
 - `FlatList` nécessite une `keyExtractor` basée sur l'`id` de chaque tâche.

► Structure suggérée de l'interface :



- Tâche de Ali	
- Tâche de Ali	

Livrables :

- Fichier App.js fonctionnel.
- Interface responsive.
- Fonctionnalités respectées selon les consignes ci-dessus.