Séance 2 : Composants, State et Props

Bienvenue dans cette deuxième séance de formation ! Dans cette session, nous allons explorer en profondeur les concepts essentiels des **composants**, du **state** (état) et des **props** dans React Native. Vous apprendrez à :

- Créer et utiliser des composants réutilisables (fonctionnels et de classe)
- Gérer l'état local avec le hook `useState`
- Transmettre des données entre composants via les props

Ce cours très détaillé est riche en exemples, explications pas-à-pas, liens utiles et comprend un TP complet pour mettre en pratique ces concepts.

1. Introduction aux composants

1.1 Qu'est-ce qu'un composant ?

Un **composant** est une brique de l'interface utilisateur qui peut être réutilisée et combinée pour construire des applications. En React Native, tout est un composant : les vues, textes, images, boutons, etc.

Exemple simple :

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

export default function Bonjour() {
  return <Text>Bonjour, bienvenue dans l'application !</Text>;
}
```

Ce composant Bonjour retourne simplement un texte à afficher.

1.2 Composants fonctionnels vs Composants de classe

Il existe deux manières principales de créer des composants :

Composants fonctionnels

Ils sont définis sous forme de fonctions JavaScript et, grâce aux hooks, ils sont désormais privilégiés.

Exemple de composant fonctionnel :

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

const WelcomeMessage = () => {
  return (
    <View>
      <Text>Bienvenue sur l'application!</Text>
    </View>
  );
};

export default WelcomeMessage;
```

Composants de classe

Historiquement, les composants de classe étaient utilisés pour gérer l'état et le cycle de vie. Aujourd'hui, ils sont moins courants, mais il est utile de les connaître.

Exemple de composant de classe :

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

class WelcomeMessageClass extends Component {
  render() {
    return (
      <View>
        <Text>Bienvenue sur l'application (composant de classe)!</Text>
      </View>
    );
  }
}

export default WelcomeMessageClass;
```

Liens utiles :

- Introduction aux composants React
- Hooks en React

---

2. Le State dans React Native

2.1 Introduction à useState

Le state permet de stocker et de gérer des données dynamiques dans un composant. Dans les composants fonctionnels, le hook `useState` est utilisé pour créer cet état.

Syntaxe :

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

- `state` : La valeur actuelle.
- `setState` : Fonction pour mettre à jour l'état.
- `initialState` : La valeur initiale de l'état.

2.2 Exemple d'utilisation de useState

Créons un composant de compteur qui incrémente une valeur à chaque clic.

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

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

  return (
    <View style={{ alignItems: 'center', marginTop: 50 }}>
      <Text style={{ fontSize: 24 }}>Compteur : {count}</Text>
      <Button title="Incrémenter" onPress={() => setCount(count + 1)} />
    </View>
  );
```

```
};

export default Counter;
```

Explications :

- Le compteur `count` est initialisé à 0.
- `setCount` met à jour la valeur de `count` lorsque le bouton est pressé.
- Chaque modification du state force le re-render du composant.

Liens utiles :

- Documentation de useState

---

3. Les Props : Passage de données entre composants

3.1 Définition et utilisation des props

Les props (propriétés) sont des paramètres passés d'un composant parent à un composant enfant. Elles permettent de rendre les composants dynamiques et réutilisables.

Exemple simple :

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

const Greeting = (props) => {
  return (
    <View>
      <Text>Bonjour, {props.name}!</Text>
    </View>
  );
};

export default Greeting;
```

Dans cet exemple, le composant `Greeting` affiche un message personnalisé grâce à la prop `name` .

3.2 Exemple d’un composant parent et enfant

Créons un exemple où un composant parent transmet des données à un composant enfant.

Composant enfant (`UserCard`) :

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

const UserCard = ({ name, age }) => {
  return (
    <View style={styles.card}>
      <Text style={styles.name}>Nom : {name}</Text>
      <Text>Âge : {age}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  card: {
    padding: 15,
    margin: 10,
    backgroundColor: '#e0f7fa',
    borderRadius: 8,
  },
  name: {
    fontWeight: 'bold',
    fontSize: 18,
  },
});

export default UserCard;
```

Composant parent (`UserList`) :

```jsx
import React from 'react';
import { ScrollView } from 'react-native';
import UserCard from './UserCard';

const UserList = () => {
  const users = [
    { id: 1, name: 'Alice', age: 25 },
```

```
      { id: 2, name: 'Bob', age: 30 },
      { id: 3, name: 'Charlie', age: 28 },
    ];

    return (
      <ScrollView>
        {users.map(user => (
          <UserCard key={user.id} name={user.name} age={user.age} />
        ))}
      </ScrollView>
    );
  };

  export default UserList;
```

Explications :

- Le composant parent `UserList` contient une liste d'utilisateurs.
- Pour chaque utilisateur, il rend un composant `UserCard` en lui passant les props `name` et `age`.

Liens utiles :

- Comprendre les props en React

---

4. Exemples pratiques et cas d'utilisation

4.1 Création d'un composant TodoItem

Dans notre projet "The Todo List App", nous créons un composant `TodoItem` qui affiche une tâche et offre une option pour la supprimer.

jsx

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

  const TodoItem = ({ task, onDelete }) => {
    return (
      <View style={styles.itemContainer}>
        <Text style={styles.itemText}>{task}</Text>
```

```
        <TouchableOpacity style={styles.deleteButton} onPress={onDelete}>
          <Text style={styles.deleteButtonText}>Supprimer</Text>
        </TouchableOpacity>
      </View>
    );
  };

  const styles = StyleSheet.create({
    itemContainer: {
      flexDirection: 'row',
      justifyContent: 'space-between',
      alignItems: 'center',
      padding: 10,
      marginVertical: 5,
      backgroundColor: '#fff',
      borderRadius: 5,
      shadowColor: '#000',
      shadowOffset: { width: 0, height: 2 },
      shadowOpacity: 0.1,
      shadowRadius: 2,
      elevation: 2,
    },
    itemText: {
      fontSize: 16,
    },
    deleteButton: {
      backgroundColor: '#ff5252',
      padding: 5,
      borderRadius: 3,
    },
    deleteButtonText: {
      color: '#fff',
      fontSize: 14,
    },
  });

  export default TodoItem;
```

4.2 Gestion d’état dans une Todo List

Nous allons créer un composant qui permet d’ajouter et de supprimer des tâches dans une liste, en utilisant `useState` .

jsx

```jsx
import React, { useState } from 'react';
import { View, TextInput, Button, FlatList, StyleSheet } from 'react-native';
import TodoItem from './TodoItem';

const TodoList = () => {
  const [task, setTask] = useState('');
  const [tasks, setTasks] = useState([]);

  const addTask = () => {
    if (task.trim()) {
      setTasks([...tasks, { id: Date.now().toString(), text: task }]);
      setTask('');
    }
  };

  const deleteTask = (id) => {
    setTasks(tasks.filter(task => task.id !== id));
  };

  return (
    <View style={styles.container}>
      <TextInput
        style={styles.input}
        placeholder="Ajouter une tâche"
        value={task}
        onChangeText={setTask}
      />
      <Button title="Ajouter" onPress={addTask} />
      <FlatList
        data={tasks}
        keyExtractor={item => item.id}
        renderItem={({ item }) => (
          <TodoItem
            task={item.text}
            onDelete={() => deleteTask(item.id)}
          />
        )}
      />
    </View>
  );
};

const styles = StyleSheet.create({
```

```
    container: {
      padding: 20,
      flex: 1,
    },
    input: {
      borderColor: '#ccc',
      borderWidth: 1,
      padding: 10,
      marginBottom: 10,
    },
  });

  export default TodoList;
```

Explications :

- Un champ de saisie permet d'ajouter une nouvelle tâche.
- Le bouton "Ajouter" appelle `addTask` pour mettre à jour le state.
- `FlatList` affiche les tâches via le composant `TodoItem`.
- Chaque tâche est identifiée par un id unique et peut être supprimée grâce à la fonction `deleteTask`.

Liens utiles :

- Documentation FlatList
- Utilisation de useState

---

5. Travaux Pratiques (TP)

Objectif du TP

Vous devez créer une application Todo List complète en utilisant React Native et Expo. L'application devra permettre :

- L'ajout et la suppression de tâches.
- L'affichage dynamique des tâches avec l'état et les props.

Étapes détaillées du TP

1. Créer un projet React Native avec Expo.
2. Créer des composants pour l'interface utilisateur : champ de saisie, bouton, liste des tâches.
3. Gérer l'état de la liste des tâches avec `useState`.
4. Passer des props entre les composants pour afficher et supprimer les tâches.
5. Tester l'application sur un émulateur ou un appareil mobile.

---

6. Conclusion et ressources complémentaires

Félicitations ! Vous avez couvert les bases essentielles des composants, du state et des props dans React Native. Voici quelques ressources pour aller plus loin :

- Documentation officielle React Native
- React Native - Guide des composants
- Expo - React Native Tools