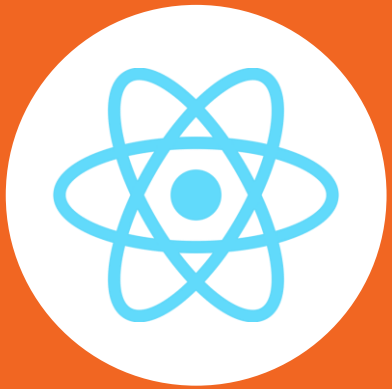
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 2: GIỚI THIỆU VỀ HOOKS

PHẦN 1: GIỚI THIỆU VỀ HOOKS, useState, useRef, userEffect

Mục tiêu

- Giới thiệu chung về hooks trong React Native
- Biết cách sử dụng useState
- Biết cách sử dụng useRef
- Biết cách sử dụng useEffect
- Ở các phiên bản mới từ RN 0.7.0 trở lên thì typescript đã được sử dụng làm ngôn ngữ mặc định khi các bạn khởi tạo dự án. Trước đây thì bạn có thể lựa chọn giữa javascript và typescript. Nhưng vì những ưu điểm vượt trội của typescript nên ở bài học sắp tới chúng ta sẽ làm bằng typescript nhé

Giới thiệu hooks

- Giới thiệu chung về Hooks trong React Native:

  Các Hooks cho phép bạn sử dụng các tính năng khác nhau của React từ các thành phần của bạn. Bạn có thể sử dụng các Hooks có sẵn hoặc kết hợp chúng để xây dựng các Hooks của riêng bạn.

- Có 5 loại Hooks chính thường được sử dụng:

  - 1. State Hooks

    Dùng để lưu trữ giá trị cho component, bạn có thể gán giá trị ban đầu cho nó. Bạn cũng có thể thay đổi giá trị cho biến state này, mỗi lần biến state này thay đổi thì sẽ làm re-render component sử dụng nó.

    Để thêm state vào một component, sử dụng một trong các Hooks sau:

- useState khai báo một biến state mà bạn có thể cập nhật trực tiếp.
- useReducer khai báo một biến state với logic cập nhật nằm trong một hàm reducer."

```
function ImageGallery() {
  const [index, setIndex] = useState(0);
  // ...
```

2.Context Hooks

Context cho phép một component nhận thông tin từ các thành phần cha ở xa mà không cần chuyển nó qua props. Ví dụ, thành phần cấp cao nhất của ứng dụng của bạn có thể

truyền giá trị nó đến tất cả các thành phần bên dưới, bất kể độ sâu của chúng.

useContext đọc và đăng ký theo dõi một context.

```
function Button() {
  const theme = useContext(ThemeContext);
  // ...
```

- 3.Ref Hooks

  Ref dùng để lưu trữ lại giá trị của biến nào nó, việc cập nhật lại giá trị của ref sẽ không gây re-render component.

useRef khai báo một ref. Bạn có thể giữ bất kỳ giá trị nào trong đó, nhưng thường được sử dụng để giữ một nút DOM.

useImperativeHandle cho phép bạn tùy chỉnh ref được khai báo bởi component của bạn. Điều này hiếm khi được sử dụng."

- 4.Effect Hooks

Effects dùng để lắng nghe việc thay đổi giá trị và re-render của component, hook này được sử dụng để thay thế cho việc quản lý life-cycle của ứng dụng

Giới thiệu hooks

```
function ChatRoom({ roomId }) {
  useEffect(() => {
    const connection = createConnection(roomId);
    connection.connect();
    return () => connection.disconnect();
  }, [roomId]);
  // ...
```

Effects là một "escape hatch" từ mô hình React. Đừng sử dụng Effects để điều khiển luồng dữ liệu của ứng dụng của bạn. Nếu bạn không tương tác với một hệ thống bên ngoài, bạn có thể không cần một Effect.

Có hai biến thể của useEffect được sử dụng hiếm khi với sự khác biệt về thời gian:

- useLayoutEffect kích hoạt trước khi trình duyệt khởi tạo lại màn hình. Bạn có thể đo đạc bố cục ở đây.

- useInsertionEffect kích hoạt trước khi React thực hiện các thay đổi trên DOM. Thư viện có thể chèn CSS động ở đây."

5.Performance Hooks

Một cách thông thường để tối ưu hiệu suất tránh việc re-render lại component không cần thiết. Ví dụ, bạn có thể cho React biết để sử dụng lại một phép tính được lưu vào bộ nhớ cache hoặc để bỏ qua việc tính toán lại function hoặc componnent không cần thiết.

Giới thiệu hooks

Để bỏ qua các phép tính và việc tái vẽ không cần thiết, hãy sử dụng một trong những Hooks sau đây:

useMemo cho phép bạn lưu vào bộ nhớ cache kết quả của một phép tính tốn kém.

useCallback cho phép bạn lưu vào bộ nhớ cache định nghĩa của một hàm trước khi chuyển nó xuống một thành phần được tối ưu hóa.

```
function TodoList({ todos, tab, theme }) {
  const visibleTodos = useMemo(() => filterTodos(todos, tab), [todos, tab]);
  // ...
}
```

Đôi khi, bạn không thể bỏ qua việc re-render lại vì màn hình thực sự cần cập nhật. Trong trường hợp đó, bạn có thể cải thiện hiệu suất bằng cách tách các bản cập nhật chặn phải đồng bộ khỏi các bản cập nhật không chặn không cần chặn giao diện người dùng. Để ưu tiên render, hãy sử dụng một trong các Hook sau:

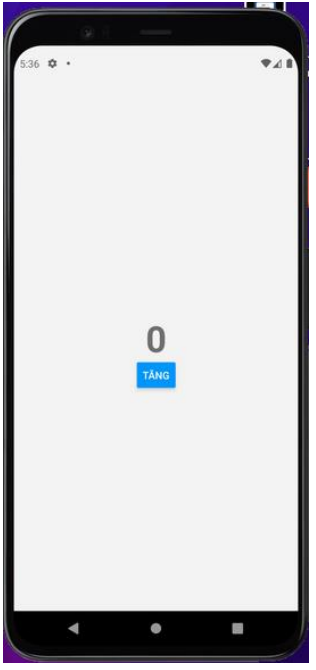
useTransition giúp xác định một phần của ứng dụng là đang trong quá trình chuyển đổi, và React sẽ tạo ra một hiệu ứng chuyển đổi mượt mà khi cập nhật phần đó. Bằng cách sử dụng Hook này, bạn có thể làm cho ứng dụng của mình trông tốt hơn và đáp ứng tốt hơn với hành động của người dùng.

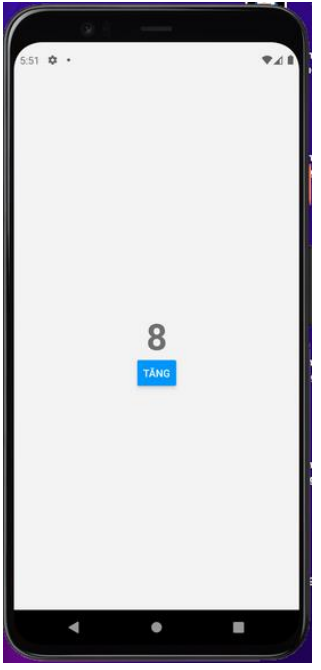
useDeferredValue cho phép trì hoãn việc cập nhật một giá trị đến khi tài nguyên có sẵn và chưa cần thiết để hiển thị ngay lập tức lên giao diện người dùng.

Giới thiệu hooks

- Đến đây thì chúng ta đã năm sơ qua và biết được các hooks phổ biến trong React Native
- Bây giờ chúng ta sẽ đi sâu vô từng ví dụ mỗi hook để hiểu rõ hơn về cách sử dụng nó nhé.

- Xây dựng một ứng dụng đếm số, mỗi lần bấm nút Tăng, thì giá trị sẽ tăng lên một, và giá trị ban đầu bằng 0





- Code mẫu ở bên dưới

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

export const UseStateScreen = () => {
  const [count, setCount] = useState<number>(0);

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

  return (
    <View style={styles.container}>
      <Text style={styles.textCount}>{count}</Text>
      <Button title="Tăng" onPress={handleIncrease} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  textCount: {
    fontSize: 50,
    fontWeight: 'bold',
  },
});
```

- Giải thích

  Ở dòng số 5, chung ta khai báo một biến useState, gồm count là giá trị của state và setCount là một callback để gán lại giá trị cho biến state đó (2 tên này bạn có thể đặt tên tuỳ ý của bạn)

  Các bạn sẽ thấy trong useState<number>(0), <number> ở đây được sử dụng để khai báo kiểu dữ liệu cho state thuộc kiểu number, sau đó chúng ta gán giá trị ban đầu cho nó là 0.

  Hàm handleIncrease() dùng để gán giá trị lại cho state, muốn gán lại giá trị cho state thì chỉ cần gọi setCount.

- Bạn cũng có thể lưu bất cứ dữ liệu nào trong useState, dưới đây là một ví dụ về lưu biến object bằng cách như sau:

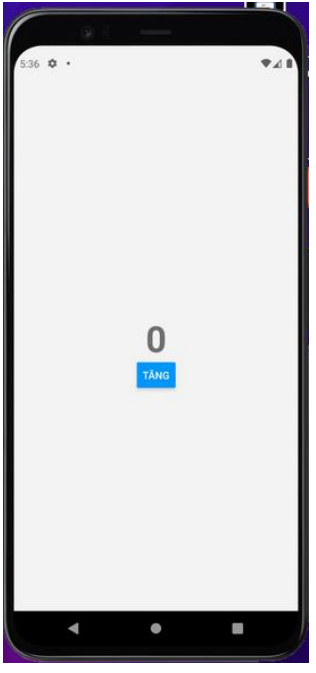
```tsx
type InforUserType = {
  name: string;
  age: number;
};

export const UseStateScreen = () => {
  const [inforUser, setInforUser] = useState<InforUserType>({
    name: 'Nguyen Van A',
    age: 20,
  });

  const updateInforUser = () => {
    // Cập nhật tất cả thông tin trong inforUser
    setInforUser({
      name: 'Nguyen Van B',
      age: 21,
    });

    // Chỉ cập nhật age
    setInforUser({
      ...inforUser,
      age: 21,
    });
  };
```

- Chúng ta sẽ tiếp tục làm ở phần code phía trước demo một số tính chất của useEffect cho mọi người xem nhé.



useEffect

- useEffect có 3 loại, không có dependencies, có dependencies nhưng rỗng, có dependencies

```
export const UseStateScreen = () => {
  const [inforUser, setInforUser] = useState<InforUserType>({…
  });
  const [count, setCount] = useState<number>(0);

  const updateInforUser = () => {…
  };

  const handleIncrease = () => {…
  };

  useEffect(() => {
    console.log('useEffect này chạy mỗi lần component render');
  });

  useEffect(() => {
    console.log('useEffect chỉ chạy lần đầu tiên khi component render');
  }, []);

  useEffect(() => {
    console.log('useEffect chạy mỗi lần state count thay đổi giá trị');
  }, [count]);
```

- Chạy lại ứng dụng, sau đó xem log nhé

```
LOG  Running "DaNenTang2" with {"rootTag":11}
LOG  useEffect này chạy mỗi lần component render
LOG  useEffect chỉ chạy lần đầu tiên khi component render
LOG  useEffect chạy mỗi lần state count thay đổi giá trị, count =  0
```

- Và đây là log sau khi nhấn Tăng lên 1

```
LOG  Running "DaNenTang2" with {"rootTag":11}
LOG  useEffect này chạy mỗi lần component render
LOG  useEffect chỉ chạy lần đầu tiên khi component render
LOG  useEffect chạy mỗi lần state count thay đổi giá trị, count =  0
LOG  useEffect này chạy mỗi lần component render
LOG  useEffect chạy mỗi lần state count thay đổi giá trị, count =  1
```

useRef

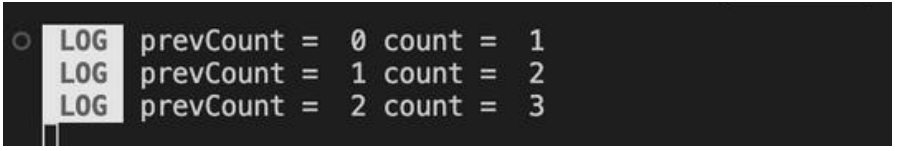
- Chúng ta sẽ tiếp tục làm ở phần code phía trước demo một số tính chất của useRef cho mọi người xem nhé.

```
export const UseStateScreen = () => {
  const [inforUser, setInforUser] = useState<InforUserType>({…
  });
  const [count, setCount] = useState<number>(0);
  const prevCount = useRef<number>();

  useEffect(() => {
    prevCount.current = count;
  }, [count]);

  console.log('prevCount = ', prevCount.current, 'count = ', count);
```

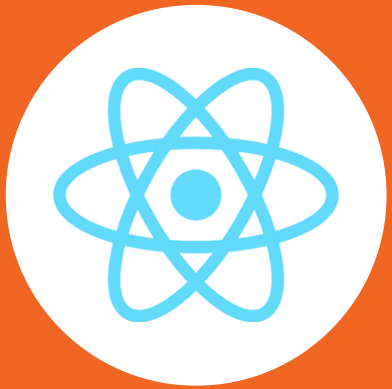
- Sau khi nhấn nút Tăng lần thứ 3, và đây là log của phần code phía trên



- Ở đây bạn có thể tự hỏi tại sao giá trị của preCount và count lại không bằng nhau. Đây là bởi vì sau khi giá trị count được cập nhật, thì nó sẽ gây render lại component, sau nó log sẽ xuất ra và sau khi DOM đã được render xong thì useEffect mới được chạy vào.

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 2: GIỚI THIỆU VỀ HOOKS

PHẦN 2: GIỚI THIỆU VỀ useMemo, useMemo, memo, useContext

Mục tiêu

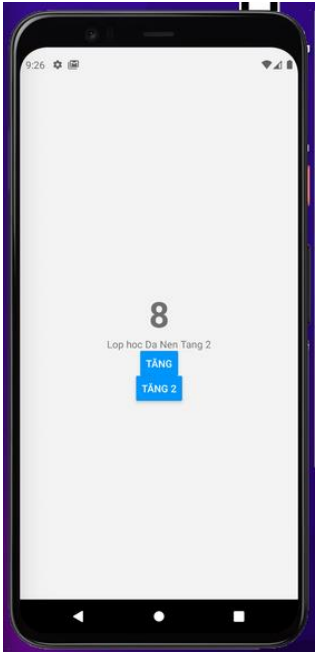
- Tối ưu hoá hiệu năng ứng dụng bằng useMemo, memo, useCallback
- Chia sẽ dữ liệu bằng useContext

memo

- Trong React Native, memo là một React Hook dùng để tối ưu hiệu suất của component bằng cách tránh việc render lại không cần thiết khi không có sự thay đổi về props hoặc state của component.

  Khi một component được wrap bởi memo, React Native sẽ lưu lại phiên bản hiện tại của component và so sánh với phiên bản trước đó khi component được render lại. Nếu props hoặc state không thay đổi, component sẽ không được render lại mà sử dụng phiên bản đã lưu trữ để hiển thị.

- Kết thúc bài này chúng ta sẽ có ứng dụng có giao diện như sau:



- Xây dựng giao diện như bên dưới

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

export const MemoScreen = () => {
  const [count, setCount] = useState<number>(0);
  const [count2, setCount2] = useState<number>(0);

  const handleIncrease1 = () => {
    setCount(count + 1);
  };

  const handleIncrease2 = () => {
    setCount2(count + 1);
  };

  return (
    <View style={styles.container}>
      <Text style={styles.textCount}>{count}</Text>
      <Content count={count} />
      <Button title="Tăng" onPress={handleIncrease1} />
      <Button title="Tăng 2" onPress={handleIncrease2} />
    </View>
  );
};
```

- Tạo component con Content sẽ được bọc bằng memo

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

export const Content = memo(({count}: {count: number}) => {
  console.log('re-render in Content, count =  ', count);
  return (
    <View>
      <Text>Lop hoc Da Nen Tang 2</Text>
    </View>
  );
});
```

- Ở dòng số 4, bạn sẽ thấy prop tên count, đây là cách chúng ta truyền giá trị từ component cha sang component con, bạn có thể đặt bất kỳ tên nào cho prop này, nhưng phải phù hợp với chuẩn đặt tên biến của javascipt

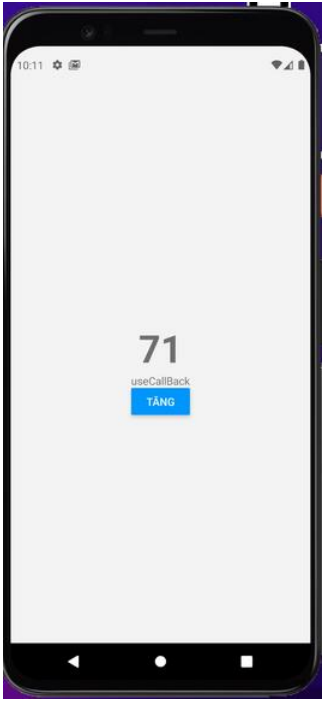
- Giải thích code ở trên
  - Sau khi chạy demo ứng dụng lên, bạn nhấn nút Tăng thì console.log trong component Content sẽ được log ra. Còn khi nhận nhấn Tăng 2 thì sẽ không có console.log nào hiện ra cả!!
  - Tại sao vậy? Bởi vì bạn đang truyền một prop vào component Content, memo wrap component Content sẽ so sánh prop count xem nó có thay đổi giá trị không, nếu giá trị count không thay đổi, nó sẽ không re-render lại component Content
- Đây là một tính năng khá tuyệt vời của memo khi bạn sẽ không cần re-render lại cả một component khi thực sự nó không mang lại kết quả nào cả. Hiệu năng luôn là vấn đề lớn của ứng dụng phát triển bằng React Native, thế nên bạn phải luôn tối ưu ứng dụng của mình tốt nhất có thể.

useCallback

- Trong React Native, useCallback là một hook được sử dụng để tối ưu hóa performance của các component bằng cách tránh việc rendering lại một hàm hoặc callback function mỗi khi component được render lại.

  Khi một component được render lại, các hàm và callback functions trong component cũng sẽ được tạo lại, dẫn đến việc tốn tài nguyên và giảm performance của ứng dụng. useCallback giúp giải quyết vấn đề này bằng cách "nhớ lại" hàm hoặc callback function và chỉ tạo lại chúng khi các dependencies của nó thay đổi.

- Kết thúc bài này chúng ta sẽ có ứng dụng giao diện như sau:



useCallback

- Chúng ta sẽ áp dụng useCallback vào ứng dụng bằng code ở đây

```
export const UseCallBackScreen = () => {
  const [count, setCount] = useState<number>(0);

  const handleIncrease1 = useCallback(() => {
    setCount(prevCount => prevCount + 1);
  }, []);

  return (
    <View style={styles.container}>
      <Text style={styles.textCount}>{count}</Text>
      <ContentUseCallBack onIncrease={handleIncrease1} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  textCount: {
    fontSize: 50,
    fontWeight: 'bold',
  },
});
```

useCallback

- Tạo component con ContentUseCallBack được bọc bằng memo

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

export const ContentUseCallBack = memo(
  ({onIncrease}: {onIncrease: () => void}) => {
    console.log('re-render');
    return (
      <View>
        <Text>useCallBack</Text>
        <Button title="Tăng" onPress={onIncrease} />
      </View>
    );
  },
);
```

Giải thích code ở trên

- Bạn hãy thử bỏ useCallBack ở hàm handleIncrease1, rồi nhấn nút "Tăng" bạn sẽ thấy console log ra chữ re-render, tại sao vậy?

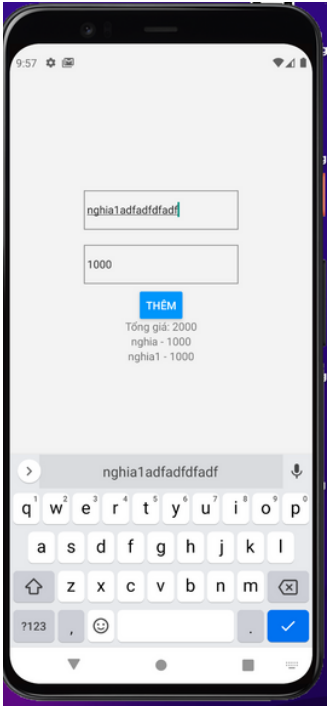
  Bởi vì, hàm là giá trị kiểu tham chiếu, một hàm bình thường sẽ được tạo và cấp một bộ nhớ mới khi component bị re-render. memo bọc? ContentUseCallBack sẽ so sánh tham chiếu đến prop onIncrease, và thấy được hàm onIncrease đã thay đổi vị trí bộ nhớ mới khi component cha re-render. Do đó component ContentUseCallBack bị đã bị render lại và log ra màn hình như những gì chúng ta thấy.

useMemo

- useMemo là một hook trong React để tối ưu hóa việc tính toán các giá trị phức tạp và tránh việc tính toán lặp đi lặp lại không cần thiết trong quá trình render của components.

  Khi sử dụng useMemo, bạn có thể chỉ định một hàm và một danh sách các dependencies (phụ thuộc) và RN sẽ gọi hàm này và trả về kết quả của nó. Kết quả này được lưu trữ trong bộ nhớ đến khi các dependencies thay đổi. Nếu dependencies không thay đổi, kết quả được lưu trữ sẽ được tái sử dụng cho các render tiếp theo, giúp giảm thiểu việc tính toán không cần thiết.

- Demo sẽ có giao diện như sau:



useMemo

- Chúng ta sẽ viết một vài function để các bạn hiểu rõ hơn về cách sử dụng useMemo

```
export default function UseMemoScreen() {
  const [name, setName] = useState('');
  const [price, setPrice] = useState('');
  const [products, setProducts] = useState<ProductType[]>([]);

  const handleSubmit = () => {
    setProducts([
      ...products,
      {
        name,
        price: +price,
      },
    ]);
  };

  const total = useMemo(() => {
    const result = products?.reduce((result, prod) => {
      console.log('Tính toán lại ...');

      return result + prod.price;
    }, 0);

    return result;
  }, [products]);
```

useMemo

- Giao diện ở ví dụ bên trên

```
<View style={styles.container}>
  <TextInput
    value={name}
    onChangeText={setName}
    style={styles.tipContainer}
    placeholder="Nhập tên"
  />
  <TextInput
    value={price}
    onChangeText={setPrice}
    style={styles.tipContainer}
    placeholder="Nhập giá"
  />
  <Button title="Thêm" onPress={handleSubmit} />
  <Text>Tổng giá: {total}</Text>

  {products?.map((product, index) => (
    <Text key={index}>
      {product.name} - {product.price}
    </Text>
  ))}
</View>
```

Giải thích code ở trên

- Có thể bạn sẽ thấy hơi khó hiểu code ở trên, bây giờ bạn thử bỏ useMemo của total function, rồi nhập thông tin từng vào TextInput xem chuyện gì sẽ xảy ra!

  Bạn sẽ thấy log "Tính toán lại ...", log ra mỗi lần khi component re-render, hàm total bị render có cần thiết ?KHÔNG, câu trả lời chắc chắc là không. Hàm total chỉ nên được tính toán lại khi giá trị trong state products thay đổi.

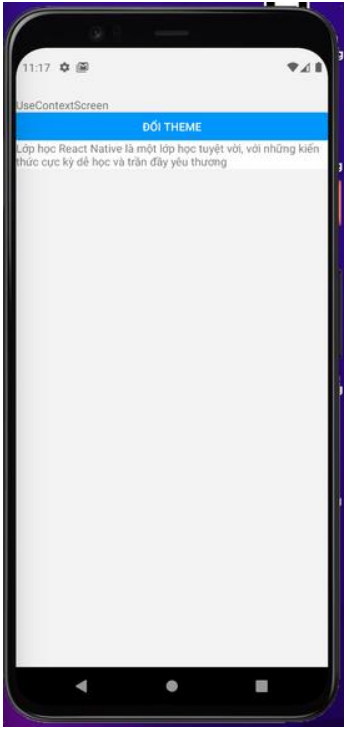
  Bạn có lẽ lại nghĩ, hàm này tính toán đâu mất thời gian đâu, mà cần dùng useMemo làm gì? Vâng, đúng vậy. Bài này của chúng ta cũng không thực sự cần dùng useMemo, nhưng đối với bài toán, cần tính toán phức tạp, hay xử lý bất đồng bộ. Thì useMemo sẽ là cứu tinh cho hiệu suất ứng dụng của bạn.

useContext

- useContext là một hook trong React giúp truy cập vào các giá trị được chia sẻ trên toàn bộ ứng dụng thông qua context, mà không cần truyền qua các components con.

  Context là một cơ chế cho phép chia sẻ dữ liệu giữa các component trong cây component mà không cần truyền props qua từng component. Các giá trị context có thể được cung cấp bởi một component ở mức cao nhất trong cây component và các component con có thể truy cập vào chúng thông qua useContext.

- Demo sẽ có giao diện như sau:



useContext

- Chúng ta bọc Provider vào component cha, nơi chúng ta muốn sử dụng useContext

```
export const ThemeContext = createContext('light');

export default function UseContextScreen() {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme(theme === 'dark' ? 'light' : 'dark');
  };

  return (
    <ThemeContext.Provider value={theme}>
      <View>
        <Text>UseContextScreen</Text>
        <Button title="Đổi theme" onPress={toggleTheme} />

        <Paragraph />
      </View>
    </ThemeContext.Provider>
  );
}
```

useContext

- Paragraph là component con bên trong component UseContextScreen cho nên bây giờ bạn có thể lấy giá trị bên trong useContext

```
export default function Paragraph() {
  const theme = useContext(ThemeContext);
  return (
    <View style={{backgroundColor: theme === 'light' ? 'white' : 'gray'}}>
      <Text>
        Lớp học React Native là một lớp học tuyệt vời, với những kiến thức
        cực
        kỳ dễ học và trần đầy yêu thương
      </Text>
    </View>
  );
}
```

- Giải thích code ở trên
  - Bạn khởi tạo một useContext bằng cách gọi createContext từ React, bạn cũng nên khai báo giá trị khởi tạo cho nó, ở ví dụ trên tôi để gán giá trị light
  - Ở ví dụ trên thì chúng tôi sử dụng ThemeContext.Provider để bọc lại component, nơi giá trị của nó sẽ được phân phối bên trong
  - ?Muốn lấy giá trị của context thì bạn chỉ cần gọi useContext và truyền vào trong là tên của context mà bạn đã khởi tạo, ở trên tên context của tôi là ThemeContext

Tổng kết

- Hiểu và sử dụng các hook trong React Native
- Kết hợp sử dụng các hook với nhau
- Tối ưu hoá ứng dụng khi sử dụng hook
- Cải thiện performance bằng các hook
- Chia sẻ dữ liệu giữa các component bằng useContext

Kết thúc