

TRƯỜNG CAO ĐẲNG CÔNG NGHỆ THỦ ĐỨC
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO KẾT THÚC MÔN HỌC
Lập trình di động 3

DICTIONARY MOBILE APPLICATION

Giảng viên hướng dẫn: **TRƯỜNG BÁ THÁI**

Sinh viên thực hiện:

1. NGUYỄN ANH TOÀN
2. LÊ MINH ĐẠT
3. NGUYỄN ĐÔNG DUY
4. NGUYỄN PHƯƠNG LINH

Ngành: Công nghệ thông tin Lớp: Lập trình di động 3 - Khoá: 16

Tp. Hồ Chí Minh, ngày 20 tháng 12 năm 2018

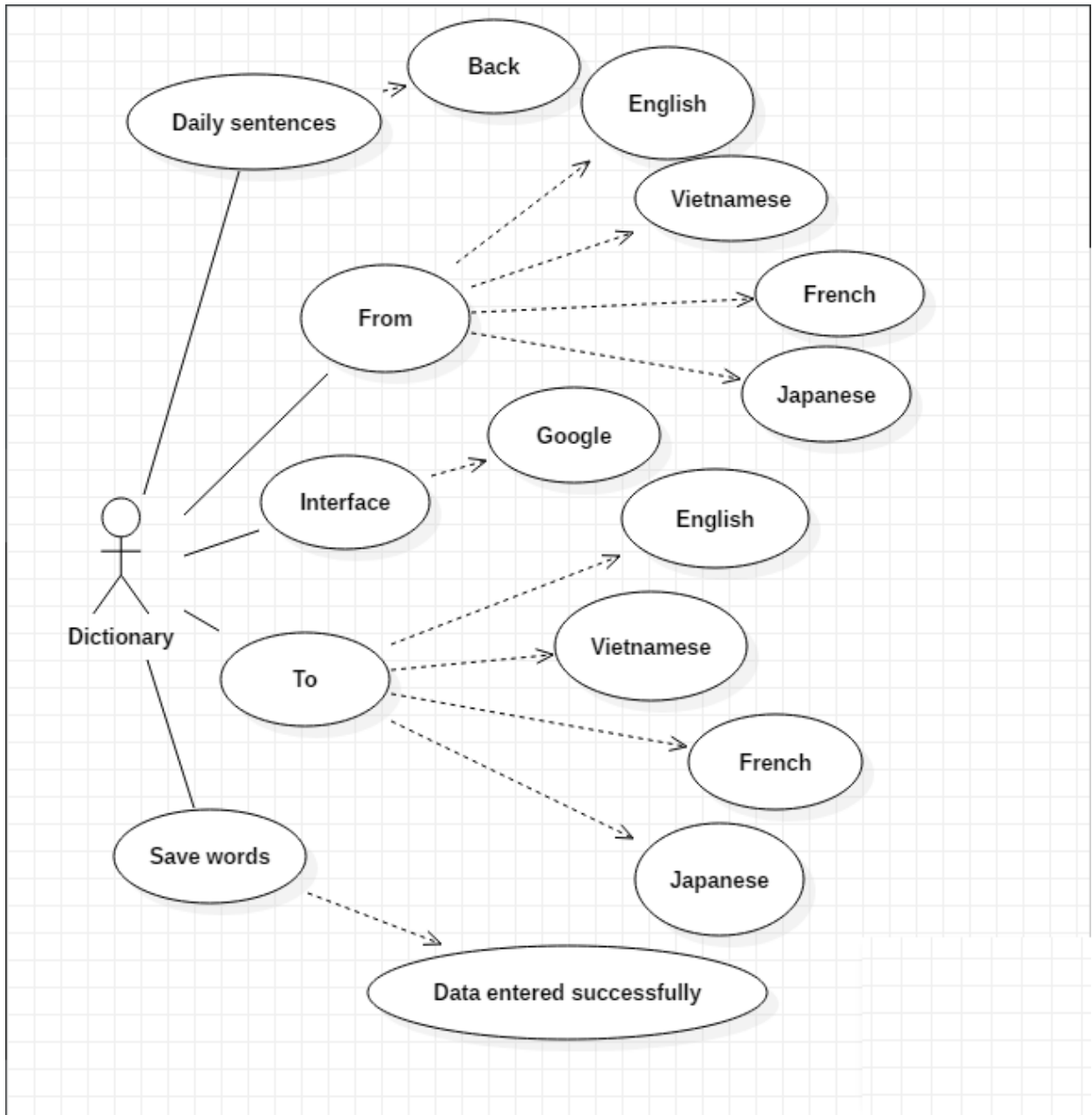
NHẬT KÝ HOẠT ĐỘNG NHÓM

Stt	Họ và tên	Công việc thực hiện	Tự đánh giá	Nhóm đánh giá	Chữ ký
1	Nguyễn Anh Toàn	1. Thiết kế giao diện 2. Cài đặt, sử dụng react-native 3. Cài đặt, thiết lập, sử dụng google-translate-api	8	9	
2	Nguyễn Đông Duy	1. Thiết kế giao diện: Daily Sentences 2. Sử dụng API để lấy tin tức theo ngày	8	9	
3	Nguyễn Phương Linh	1. Đọc và lưu từ vựng lại sau khi search 2. Thiết kế json để khi lưu từ vựng đúng định dạng	8	9	
4	Lê Minh Đạt	1. Chỉnh sửa, thiết kế các components 2. Trang trí giao diện (các file style, css..) 3. Tìm hiểu các components	8	9	

DANH MỤC

DANH MỤC BẢNG BIỂU, HÌNH VẼ, SƠ ĐỒ	1
CHƯƠNG 1. TỔNG QUAN VỀ REACT NATIVE	2
1. Giới thiệu tổng quan về React native.....	2
2. Kiến thức cơ bản về ES6 trong React native	3
Khai báo biến với var, let, và const.....	4
Module import / export	7
Function Parameter: default và rest	8
Object/Array Matching, Short Hand Notation.....	8
Spread Operator	9
String interpolation	9
Classes.....	10
Promise và parallel promise.....	11
Async và Await.....	13
3. Component trong react native.....	13
A. The basic component	13
B. User interface	17
4. API Trong React Native.....	19
CHƯƠNG 2. PHÂN TÍCH THIẾT KẾ HỆ THỐNG.....	22
2.1 Phân tích hệ thống.....	22
2.1.1. Feature/Component #1: Dictionary screen	22
2.1.1.2. Functional Requirements	25
2.2 Thiết kế hệ thống	27
2.2.1. Dictionary Mobile Application Main Screen.....	27
CHƯƠNG 3. CÀI ĐẶT VÀ SỬ DỤNG GOOGLE TRANSLATE API	30
3.1. Cài đặt Google Translate Api:	30
3.2. Khai báo và sử dụng hàm xử lý dịch	31
3.3. Test chương trình khi dùng Google-translate-API:	34
CHƯƠNG 4. CÀI ĐẶT VÀ KIỂM THỬ.....	36
4.1 Cài đặt	36
4.2 Kiểm thử	36
CHƯƠNG 5. KẾT QUẢ ĐẠT ĐƯỢC	38
5.1 Kết quả đạt được	38
5.2 Các kết luận và kiến nghị.....	39
TÀI LIỆU THAM KHẢO	40

DANH MỤC BẢNG BIỂU, HÌNH VẼ, SƠ ĐỒ



CHƯƠNG 1. TỔNG QUAN VỀ REACT NATIVE

1. Giới thiệu tổng quan về React native

Sự phát triển về CNTT ngày càng khủng khiếp. Những công nghệ mới đã và đang làm thay đổi thế giới một cách nhanh chóng. Từ lúc mới ra đời cho đến nay smartphone đã có những bước tiến mạnh mẽ vì vậy mà những công nghệ kèm theo cũng đòi hỏi những nhà phát triển phần mềm viết ra nhiều phần mềm hơn để có thể sử dụng được trên smartphone chứ không còn đơn thuần là những ứng dụng nhắn tin gọi điện thông thường.

Kèm theo đó là số lượng người dùng di động (smartphone) tăng lên chóng mặt. Cụ thể theo những báo cáo mới đây mà các nhà khoa học đã thống kê thì thời gian dành cho smartphone trung bình qua khảo sát hàng tỉ người thì rơi vào khoảng 3 tiếng một ngày.

Đồng thời giá smartphone đang ngày càng rẻ qua từng năm, vì các dòng smartphone giá rẻ đến từ các quốc gia phát triển và đông dân như Ấn Độ, Trung Quốc. Cùng với đó công nghệ sản xuất càng dần hoàn thiện hơn kéo giá smartphone xuống tới các tầng lớp phổ thông của xã hội. Từ đó smartphone trở thành một thiết bị có thể được sở hữu dễ dàng.

Nhiều công ty sản xuất phần mềm trên di động hoặc những công ty lớn nhìn thấy đây là mảnh đất màu mỡ và tiềm năng phát triển của nó rất lớn. Đồng thời dựa theo số liệu thống kê như trên thì bắt đầu người dùng đã chịu chi tiền cho smartphone nhiều hơn, số lượng người dùng chịu bỏ tiền ra mua ứng dụng để sử dụng trên smartphone cũng tăng lên theo từng năm. Do đó kéo theo nhu cầu về công việc liên quan đến mảng smartphone nói chung và lập trình mobile nói riêng tăng lên chóng mặt. Đưa ra những tiềm năng và thử thách dành cho những ai muốn phát triển theo hướng lập trình ứng dụng di động. Nhu cầu dạy và học lập trình ứng dụng di động đang dần trở thành xu hướng của xã hội hiện nay.

Giới thiệu với các bạn, gương mặt đứ con của chúng ta hôm nay đây rồi React Native. React Native là một framework do công ty công nghệ nổi tiếng Facebook phát triển nhằm mục đích giải quyết bài toán hiệu năng của Hybrid và bài toán chi phí khi mà phải viết nhiều loại ngôn ngữ native cho từng nền tảng di động. Chúng ta sẽ build được ứng dụng Native, và chúng ta cũng có thể build ứng dụng đó một cách đa nền tảng (multi-platform) chứ không phải là một “mobile web app”, không phải là “HTML5 app”, và cũng không phải là một “hybrid app” hay cũng không chỉ build trên iOS hay Android mà chúng ta build và chạy được cả hai hệ sinh thái luôn, sợ chưa!!! Một điểm hay hơn nữa mà mình có đề cập là giảm chi phí recompile của Native bằng cách sử dụng Hot-Loading tức là bạn không cần phải build lại ứng dụng từ đầu nên việc chỉnh sửa diễn ra rất nhanh chóng. Giúp cho lập trình viên có thể thấy được những chỉnh sửa của họ một cách nhanh chóng trực quan, không còn phải bỏ quá nhiều thời gian trong việc build và run ứng dụng nữa.

Và điểm lợi hại kế tiếp của React Native đó chính là chúng ta chỉ cần sử dụng JS để phát triển được một ứng dụng di động hoàn chỉnh, đồng thời giải quyết được các vấn đề mà Native App gặp phải mà mình đã nêu ở trên. Và rồi còn cả kết hợp với code native như Swift, Java, v.v...

2. Kiến thức cơ bản về ES6 trong React native

Dưới đây là các plugin ES6/7 mà React Native đang sử dụng và bài viết này mình tập trung vào những syntax mà mình thấy quan trọng nhất trong React Native.

```
'transform-es2015-arrow-functions',
'transform-es2015-block-scoping',
'transform-es2015-classes',
'transform-es2015-computed-properties',
'check-es2015-constants',
'transform-es2015-destructuring',
['transform-es2015-modules-commonjs', { strict: false, allowTopLevelThis: true }],
'transform-es2015-parameters',
'transform-es2015-shorthand-properties',
'transform-es2015-spread',
'transform-es2015-template-literals',
'transform-es2015-literals',
'transform-flow-strip-types',
'transform-object-assign',
'transform-object-rest-spread',
'transform-react-display-name',
'transform-react-jsx',
'transform-regenerator',
['transform-es2015-for-of', { loose: true }],
require('../transforms/transform-symbol-member')
```

Khai báo biến với **var**, **let**, và **const**

Khi khai báo biến với Const, biến đó sẽ là immutable variable, nghĩa là sẽ không thay đổi được giá trị của biến.

Với var và let, chúng ta đều có thể khai báo được 1 biến bất kỳ, biến này có thể thay đổi được giá trị. Điểm khác biệt giữa var và let đó là:

```
// var cho phép chúng ta khai báo lại 1 biến cũ, nhưng let thì không
var n = 1;
var n = 2; // no syntax error

let m = 1;
let m = 2; // syntax error

// var và let đều tác động vào function block như nhau, tuy nhiên ở trường hợp này let
// sẽ chỉ tác động vào block ngay sau nó:

function someFunc() {

    for( let i = 0; i <= 9; i++ ) {
        // Biến i lúc này chỉ tồn tại trong scope block của for
    }

    // Gọi biến i ngoài này sẽ bị lỗi
}

function someFunc() {

    for( var i = 0; i <= 9; i++ ) {
        // Biến i lúc này không chỉ tồn tại trong scope block của for
    }

    // mà còn tồn tại cả ở ngoài này nữa, lúc này biến i = 10
}
```


Arrow function cũng như function bình thường, chỉ khác về syntax và binding context:

```
//// File CurrencyConverter.js

const rate = 22650.0; // Private in file

// export for public use
export const bankName = 'ACB';
export const vnd2dollar = (vnd) => vnd / rate;

// export mặc định
export default dollar2vnd = (dolla) => dolla * rate;

//// File Other.js

import Convert from './CurrencyConverter';

//Convert chính là hàm dollar2vnd được export mặc định
alert(Convert(10)); // 226500

//// File Another.js

import Convert, { vnd2dollar, bankName } from './CurrencyConverter';

// Import thêm vnd2dollar và bankName trong file CurrencyConverter.js
alert(Convert(10)); // 226500
alert(vnd2dollar(226500)); // 10
```

```

function Pet() {
  this.age = 1;

  function showAgeFunc() {
    alert( 'Age in showAgeFunc: ' + this.age ); // Age in showAgeFunc: undefined
  }

  const showAgeArrowFunc = () => {
    alert( 'Age in showAgeArrowFunc: ' + this.age ); // Age in showAgeFunc: 1
  }

  setTimeout( showAgeFunc, 1000);
  setTimeout( showAgeArrowFunc, 1000);
}

new Pet();

```

Module import / export

Ứng dụng RN thường được phát triển trên nhiều file JS mà ta thường gọi là các module / component. Tất cả các biến và function trong 1 file JS sẽ chỉ được truy xuất trong file (hay còn gọi là file private). Để cho phép các thành phần có thể sử dụng từ các file khác, chúng ta cần dùng tới từ khoá export và import.

Function Parameter: default và rest

```
// url và method có giá trị mặc định nếu không được truyền vào khi gọi function.  
const requestURL = (url = '', method = 'GET') => {  
  
};  
  
requestURL(); // url = '', method = 'GET'  
requestURL('http://facebook.com/'); // url = 'http://facebook.com/', method = 'GET'  
  
const requestURLWithParams = (url = '', method = 'GET', ...params) => {  
    // params là array chứa giá trị các tham số thứ 3 trở đi  
}  
  
requestURLWithParams(); // params = []  
requestURLWithParams('someURL', 'GET', 'a', 1, {}); // params = ['a', 1, {}]
```

Object/Array Matching, Short Hand Notation

```
let arr = [1,2,3];  
let [n, , m] = arr; // n = 1, m = 3  
console.log(n, m); // 1, 3  
  
let person = { name : 'Viet Tran', age: 28, interestedIn: 'Coding' };  
let { name, age } = person; // name = 'Viet Tran', age = 28  
console.log(name, age); // 'Viet Tran', 28  
  
let { name , job = 'some job' } = person; // nếu person không có thuộc tính job thì job  
= 'some job' như một giá trị mặc định
```

Spread Operator

Đây là một trong những operator quan trọng chúng ta rất hay dùng trong RN. Trong clip hướng dẫn RN cơ bản mình cũng có demo quản lý style component con với Spread Operator.

```
let arr = [1,2,3];
let someArr = [...arr, 4]; // [1,2,3,4], ...arr gọi là spread operator
let thatArr = [4, ...arr]; // [4,1,2,3], tương tự nhưng chỉ khác vị trí các thành phần.

let p1 = { x: 0, y: 1 };
let p2 = { ...p1, z: 1 }; // { x: 0, y: 1, z: 1 }
let p3 = { ...p2, y: 2 }; // { x: 0, y: 2, z: 1 }, update y nếu y đã có.
let p4 = { y: 3, ...p3 }; // { y: 3, x: 0, z: 1 }, không update y vì nguyên tắc phía sau override phía trước
```

String interpolation

```
const number = 10;
const str = 'number = ' + 10; // đây là cách cũ thường được dùng trong ES5

const str = `number = ${ number }` // đây là cách mới trong ES6
const str = `number + 1 = ${ number + 1 }` // chúng ta có thể sử dụng expression trong '{}'

const str = `Some string`; // chuỗi bình thường cũng xài được
```

```

class SomeObject {}
let obj = new SomeObject(); // tạo biến obj là đối tượng của SomeObject

// Các biến và hàm trong class không cần khai báo với từ khoá var/let/constant/function

class Pet {

    // biến trong class
    food = `something eatable`;

    // Hàm (method) trong class
    eat() {
        console.log(`I can eat ${ this.food }`);
    }

    // Hàm khởi tạo (constructor) cho class Pet
    constructor(name, age) {
        this.name = name;
        this.age  = age;
    }
}

```

Classes

```

let myPet = new Pet('Beo', 2);
console.log(myPet); // object { food: 'something eatable', name: 'Beo', age: 1 }
myPet.eat(); // I can eat something eatable

// Khai báo class Cat kế thừa từ class Pet
class Cat extends Pet {

    static numberOfLegs = 4; // biến static trong class

    // Hàm static trong class
    static lazy() {
        console.log(`All cats are lazy`);
    }

    constructor(name, age) {
        super(name, age); // gọi lên hàm dựng của parent class: Pet
        this.food = `fishes`;
    }
}

let myCat = new Cat();
myCat.eat(); // I can eat fishes. Hàm eat được kế thừa từ class cha (Pet)
console.log(Cat.numberOfLegs); // 4
Cat.lazy(); // All cats are lazy

```

Promise và parallel promise

Việc sử dụng Promise là một giải pháp hiệu quả khi làm việc với các hàm callback, sourcecode chúng ta sẽ dễ đọc hơn.

```
const doSomething = (err) => {  
  return new Promise( (resolve, reject) => {  
    setTimeout( () => {  
      if (err) {  
        reject(err);  
      } else {  
        resolve(10);  
      }  
    }, 1000);  
  });  
}  
  
doSomething().then( result => {  
  console.log(result); // print `10` after 1s  
}).catch(err => {  
  console.log(err);  
});  
  
doSomething(`Something went wrong`).then( result => {  
  console.log(result);  
}).catch(err => {  
  console.log(err); // print `Something went wrong` after 1s  
});
```

Chúng ta có thể sử dụng hàm `then` như một middleware, ở mỗi bước `then` ta có thể return để làm tham số cho hàm `then` tiếp theo

```
doSomething().then( result => {
  console.log(result); // 10
  return result * 2;
}).then( result => {
  console.log(result); // 20
  return { n: result }
}).then( result => {
  console.log(result); // { n : 20 }
})
.catch(err => {
  console.log(err);
});
```

Chạy cùng lúc nhiều Promise với `Promise.all`. Việc này rất hiệu quả khi ta cần load 1 lúc nhiều APIs hoặc nhiều async tasks song song.

```
const doSomething = (result, timeout) => {
  console.log(result, timeout);
  return new Promise( (resolve, reject) => {
    setTimeout( () => {
      resolve(result);
    }, timeout);
  });
}

Promise.all([
  doSomething('OK', 2000),
  doSomething(100, 5000)
]).then( data => {
  console.log(data); // print ['OK', 100] after 5s

  let [ first, second] = data;
  console.log(first, second); // 'OK', 100
}, err => {
  console.log(err);
});
```

Async và Await

Thế giới của JS đầy rẫy những callback function và promise, thế nhưng lần lúc ta lại cần chúng chạy synchronize bình thường, hay nói đúng hơn ta sẵn sàng đợi cho chúng chạy xong. Source code sẽ chạy lần lượt từ trên xuống dưới.

```
async function doTask() { // có thể dùng `aync () => {}` để thay thế
  let result = await doSomething(100, 3000);
  console.log(result); // print 100 after 3s

  let nextResult = await doSomething(20, 2500);
  console.log(nextResult) // print 20 after 5.5s
}

doTask();
console.log(`Run here first`); // dòng này in trước khi doTask() chạy
```

lưu ý: Từ khoá wait chỉ chạy trong function được khai báo với từ khoá async. Vì function này async nên sẽ chạy bất đồng bộ (chạy ở 1 thread khác) nên ở trên ta thấy `Run here first` sẽ được in ra đầu tiên. Trong function doTask, từ khoá await sẽ khiến doSomething chạy như synchronize (block thread hiện tại để đợi kết quả).

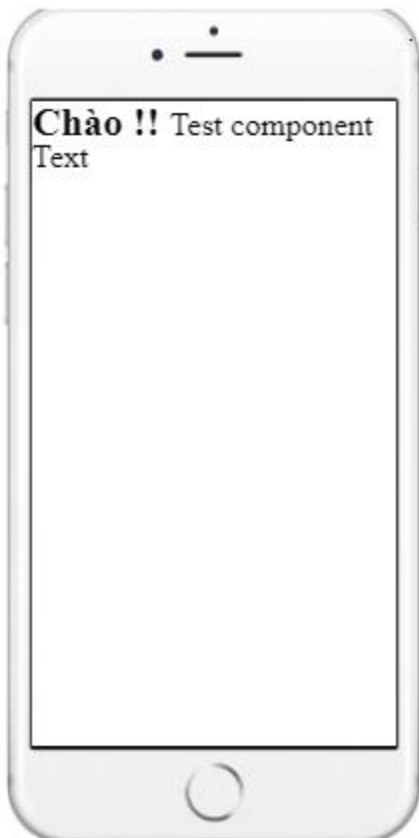
3. Component trong react native

A. The basic component

- View

```
class ViewColoredBoxesWithText extends Component {
  render() {
    return (
      <View
        style={{
          flexDirection: 'row',
          height: 100,
          padding: 20,
        }}>
        <View style={{backgroundColor: 'blue', flex: 0.3}} />
        <View style={{backgroundColor: 'red', flex: 0.5}} />
        <Text>Hello World!</Text>
      </View>
    );
  }
}
```

- Text : Giúp hiện thị văn bản lên màn hình



```
export default class TextInANest extends Component {
  constructor(props) {
    super(props);
    this.state = {
      titleText: "Chào !!",
      bodyText: 'Test component Text'
    };
  }

  render() {
    return (
      <Text style={styles.baseText}>
        <Text style={styles.titleText} onPress={this.onPressTitle}>
          {this.state.titleText}{'\n'}{'\n'}
        </Text>
        <Text numberOfLines={5}>
          {this.state.bodyText}
        </Text>
      </Text>
    );
  }
}
```

- Image: Đưa hình ảnh vào màn hình

```
import React, { Component } from 'react';
import { AppRegistry, View, Image } from 'react-native';

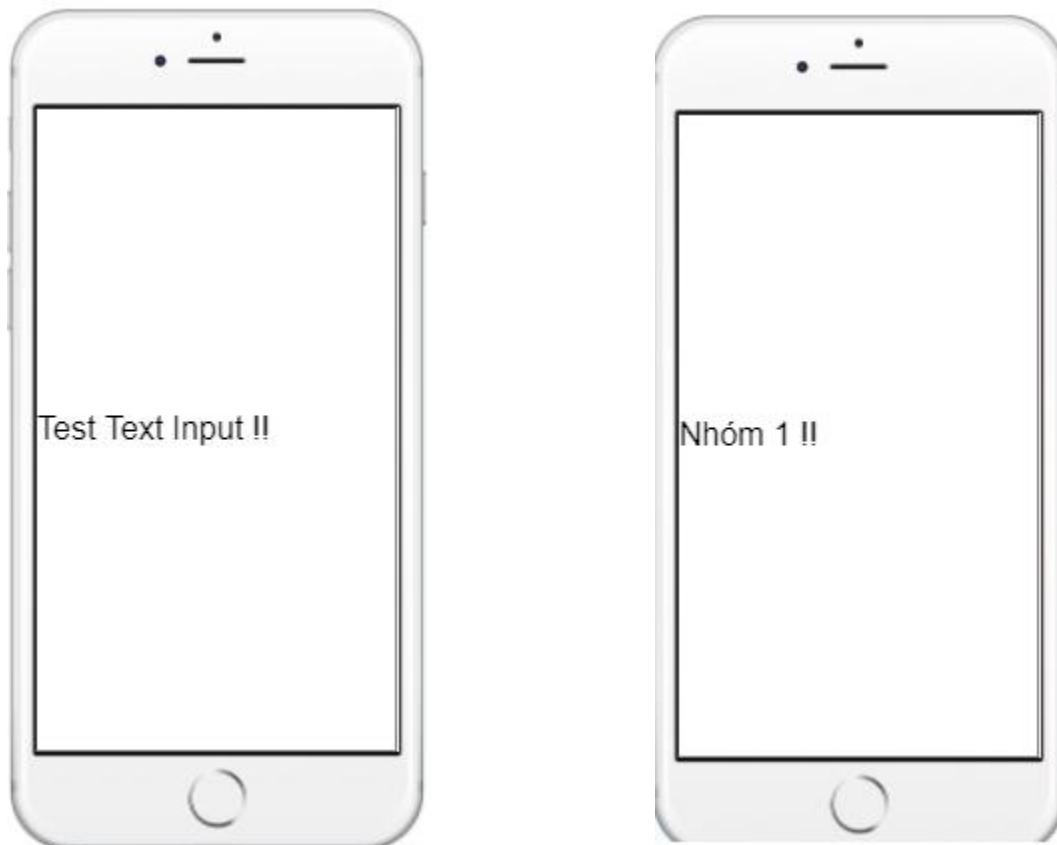
export default class DisplayAnImage extends Component {
  render() {
    return (
      <View>
        <Image
          source={require('/react-native/img/favicon.png')}
        />
        <Image
          style={{width: 50, height: 50}}
          source={{uri: 'https://upload.wikimedia.org/wikipedia/commons/thumb/2/21/Flag_of_Vietnam.svg/2000px-Flag_of_Vietnam.svg.png'}}
        />
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('DisplayAnImage', () => DisplayAnImage);
```

- Text Input : Nhập văn bản thông qua bàn phím

```
export default class UselessTextInput extends Component {
  constructor(props) {
    super(props);
    this.state = { text: 'Test Text Input !!' };
  }

  render() {
    return (
      <TextInput
        style={{height: 40, borderColor: 'gray', borderWidth: 1}}
        onChangeText={(text) => this.setState({text})}
        value={this.state.text}
      />
    );
  }
}
```

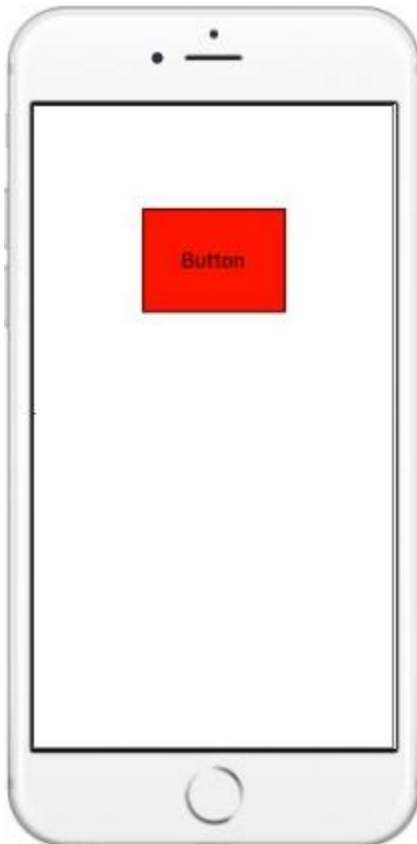


- StyleSheet : Thành phần dùng để viết CSS

```
const styles = StyleSheet.create({  
  container: {  
    borderRadius: 4,  
    borderWidth: 0.5,  
    borderColor: '#d6d7da',  
  },  
  title: {  
    fontSize: 19,  
    fontWeight: 'bold',  
  },  
  activeTitle: {  
    color: 'red',  
  },  
});
```

B. User interface

- Button



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

const App = () => {
  return (
    <View style = {styles.container}>
      <TouchableOpacity>
        <Text style = {styles.text}>
          Button
        </Text>
      </TouchableOpacity>
    </View>
  )
}
export default App

const styles = StyleSheet.create ({
  container: {
    alignItems: 'center',
    margin: 100
  },
  text: {
    borderWidth: 1,
    padding: 25,
    borderColor: 'black',
    backgroundColor: 'red'
  }
})
```

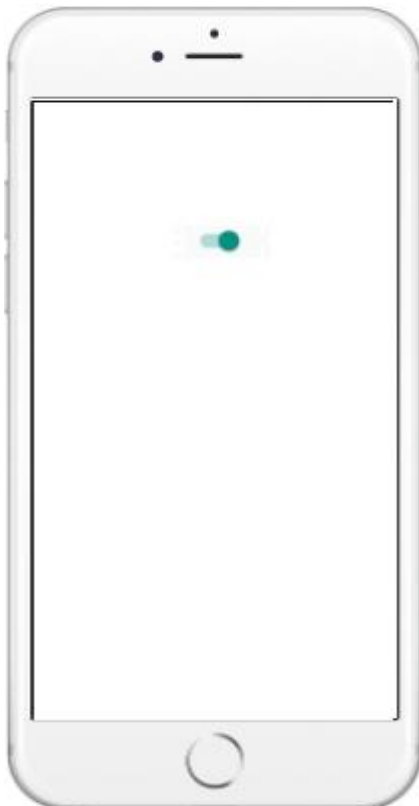
- Picker

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

class PickerExample extends Component {
  state = {user: ''}
  updateUser = (user) => {
    this.setState({ user: user })
  }
  render() {
    return (
      <View>
        <Picker selectedValue = {this.state.user} onValueChange = {this.updateUser}>
          <Picker.Item label = "Duy" value = "Duy" />
          <Picker.Item label = "Toàn" value = "Toàn" />
          <Picker.Item label = "Linh" value = "Linh" />
        </Picker>
        <Text style = {styles.text}>{this.state.user}</Text>
      </View>
    )
  }
}
export default PickerExample

const styles = StyleSheet.create({
  text: {
    fontSize: 30,
    alignSelf: 'center',
    color: 'red'
  }
})
})
```

- Switch



```

import React, { Component } from 'react'
import { View, Switch, StyleSheet }

from 'react-native'

export default SwitchExample = (props) => {
  return (
    <View style = {styles.container}>
      <Switch
        onChange = {props.toggleSwitch1}
        value = {props.switch1Value}/>
    </View>
  )
}

const styles = StyleSheet.create ({
  container: {
    flex: 1,
    alignItems: 'center',
    marginTop: 100
  }
})

```

4. API Trong React Native

React Native cung cấp module [Fetch API](#) để sử dụng cho việc kết nối network. Fetch sẽ rất thân thuộc nếu như bạn đã từng sử dụng XMLHttpRequest hoặc các networking APIs trước đây. Bạn có thể sẽ cần phải tham khảo hướng dẫn sử dụng Fetch [tại đây](#) để có được thêm các thông tin

Tạo một HTTP Request

Khi bạn muốn lấy nội dung bằng cách gọi đơn giản nhất từ một URL, rất đơn giản bạn chỉ cần đặt URL đó trong fetch:

```
fetch('https://mywebsite.com/mydata.json')
```

Fetch sẽ có một số tùy chọn tham số để bạn có thể tùy chỉnh HTTP request. Ví như bạn muốn thêm cụ thể một header nào đó và muốn gọi với phương thức POST. Ví dụ dưới đây sẽ cho các bạn thấy một cách đơn giản các tùy chọn:

```
fetch('https://mywebsite.com/endpoint/', {  
  method: 'POST',  
  headers: {  
    'Accept': 'application/json',  
    'Content-Type': 'application/json',  
  },  
  body: JSON.stringify({  
    firstParam: 'yourValue',  
    secondParam: 'yourOtherValue',  
  })  
})
```

Xử lý response

Networking bản chất là một hình thức bất đồng bộ (*Lan man một chút, vì sự bất đồng bộ này mà ở Android từ API 11 trở lên hệ điều hành đã ngăn cản việc chạy Network trên main thread để ngăn cản độ trễ của chương trình trong thời gian chờ dữ liệu mạng được trả về dưới client*). Phương thức Fetch sẽ trả về một [Promise](#), điều này sẽ dễ dàng để các bạn có thể viết các đoạn code xử lý cho các thao tác bất đồng bộ:

```
function getMoviesFromApiAsync() {
  return fetch('https://facebook.github.io/react-native/movies.json')
    .then((response) => response.json())
    .then((responseJson) => {
      return responseJson.movies;
    })
    .catch((error) => {
      console.error(error);
    });
}
```

Bạn cũng có thể sử dụng mẫu cấu trúc ES2017 về `async/await` trong ứng dụng React Native:

```
async function getMoviesFromApi() {
  try {
    let response = await fetch('https://facebook.github.io/react-native/movies.json');
    let responseJson = await response.json();
    return responseJson.movies;
  } catch(error) {
    console.error(error);
  }
}
```

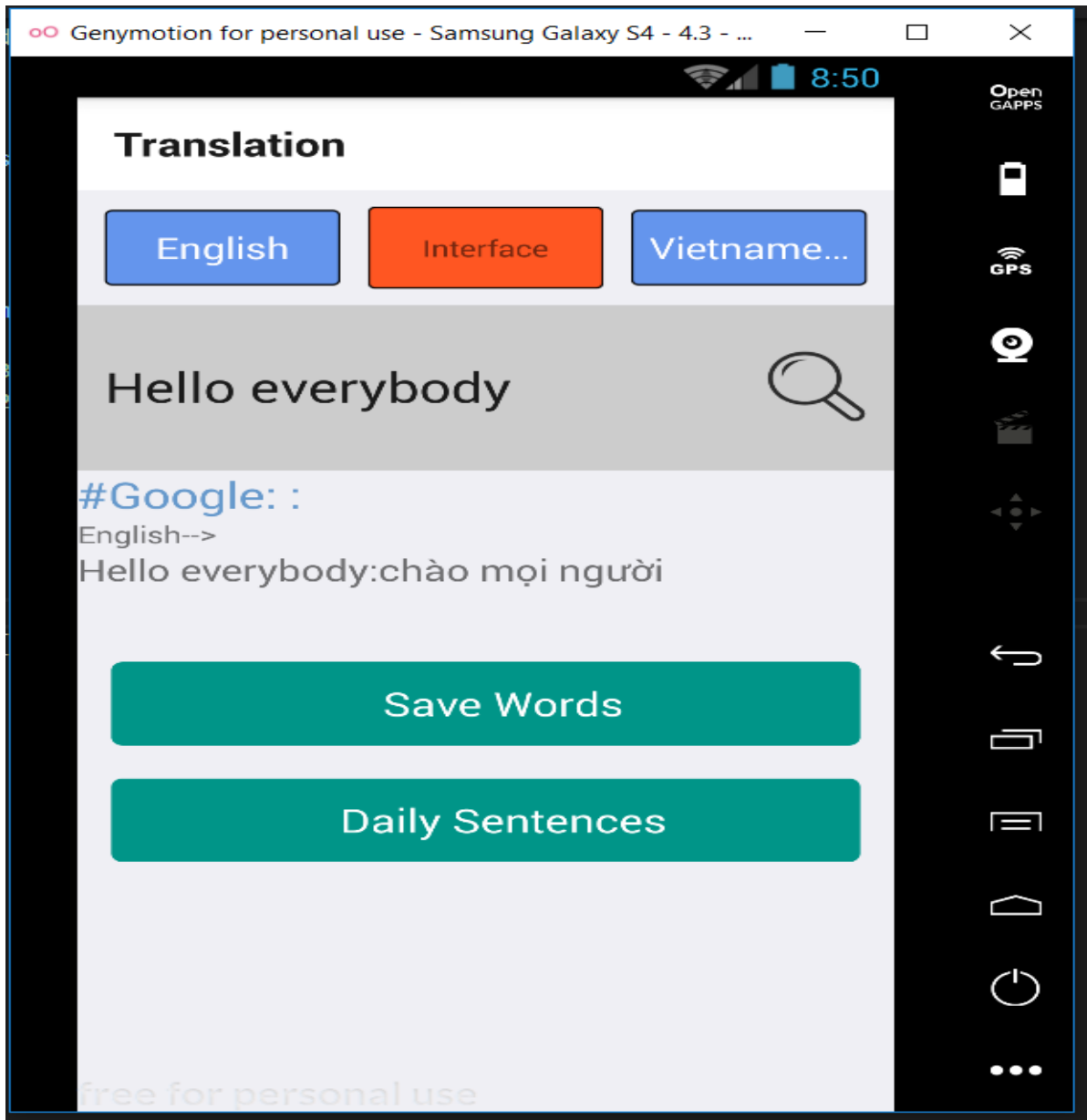
Đừng quên câu lệnh `catch` để bắt bất kỳ một lỗi nào xảy ra khi mà bạn thực hiện `fetch`. Bên cạnh đó chúng ta cũng không nên âm thầm bỏ qua các lỗi.

CHƯƠNG 2. PHÂN TÍCH THIẾT KẾ HỆ THỐNG

2.1 Phân tích hệ thống

2.1.1. Feature/Component #1: Dictionary screen

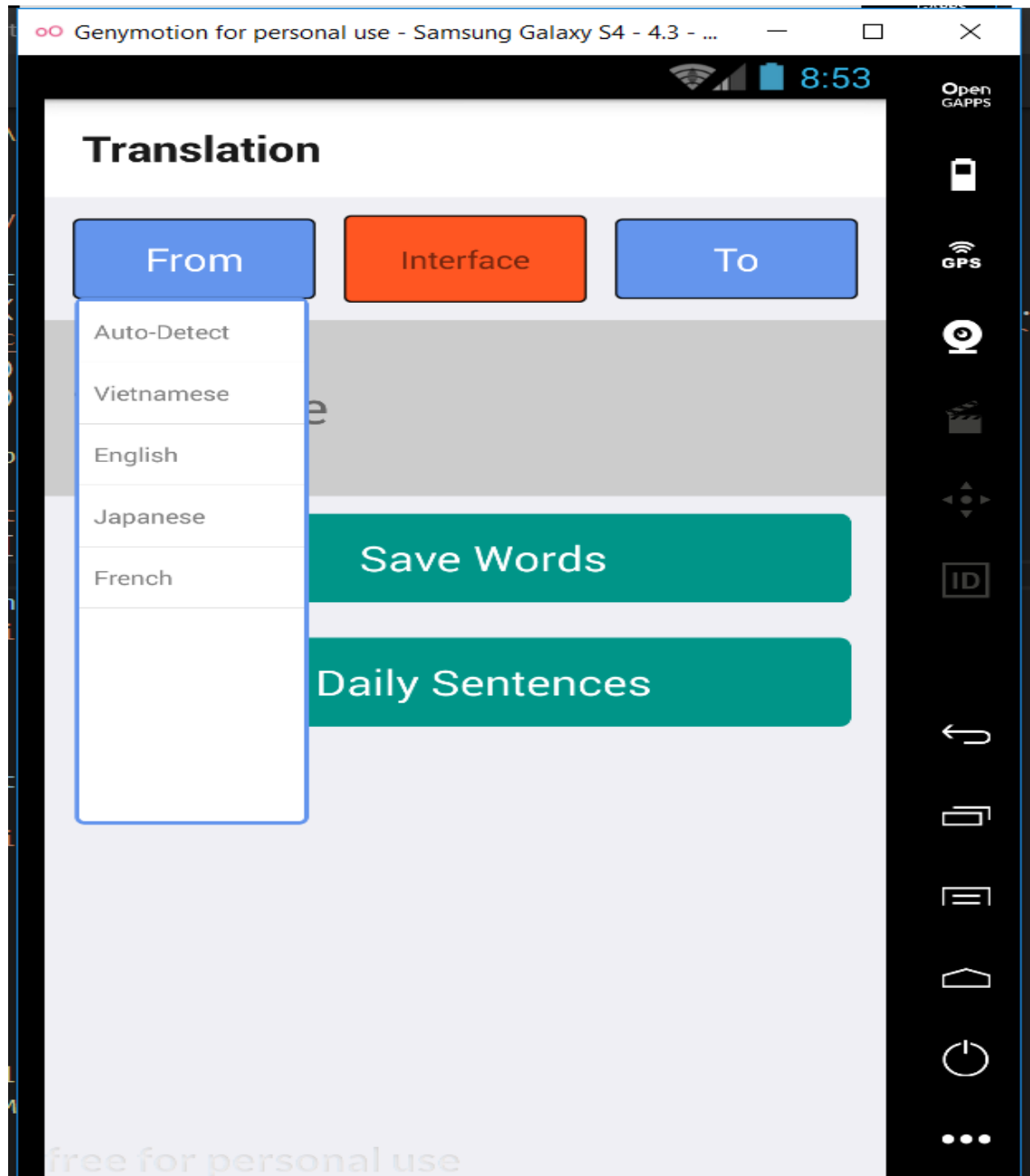
a. Chức năng dịch từ



Mô tả:

- Chức năng dịch cho phép người dùng nhập vào 1 từ hay 1 đoạn văn. Ứng dụng sẽ dịch sang nghĩa mà người dùng chọn ở chức năng chọn ngôn ngữ

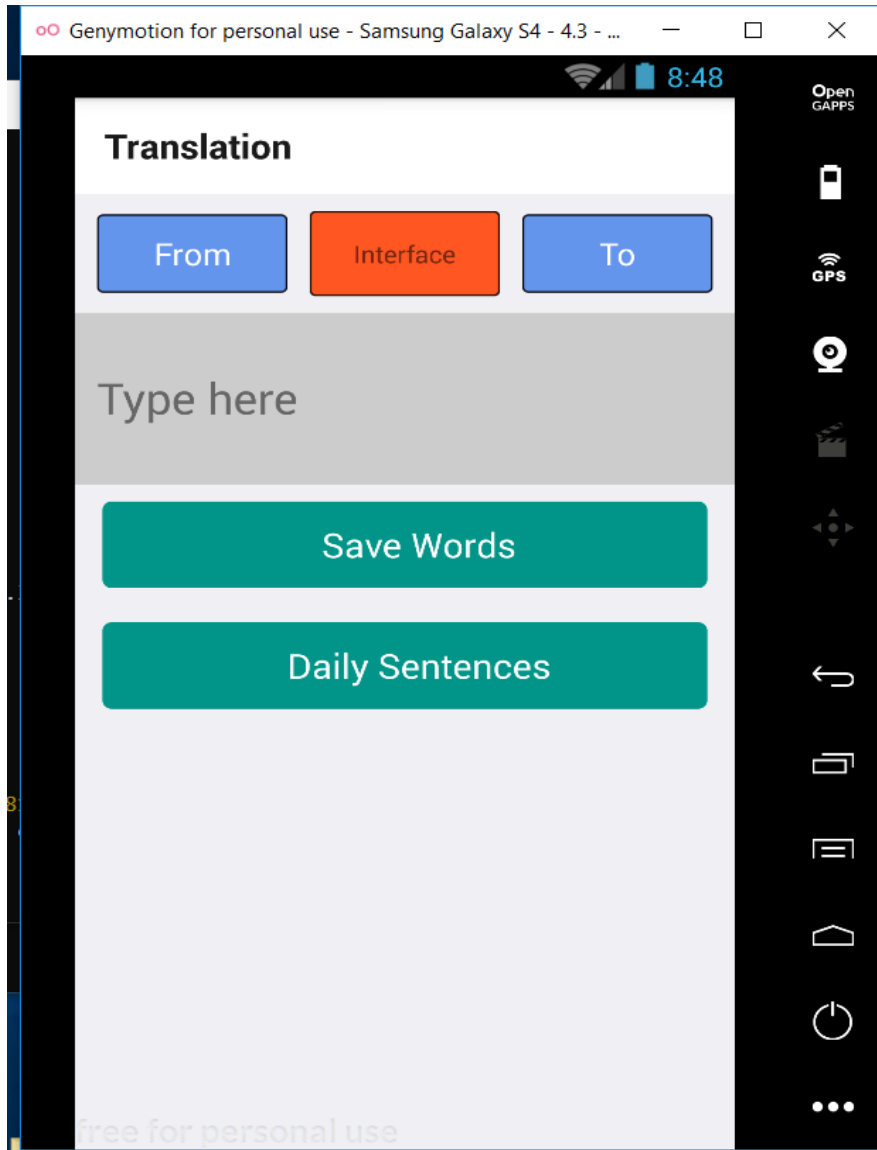
b. Chức năng chọn ngôn ngữ



Mô tả:

Người dùng có thể chọn ngôn ngữ mình nhập vào giúp cho ứng dụng trở nên linh hoạt hơn có thể dịch sang nhiều thứ tiếng khác nhau

2.1.1.1 User Interfaces



2.1.1.2. Functional Requirements

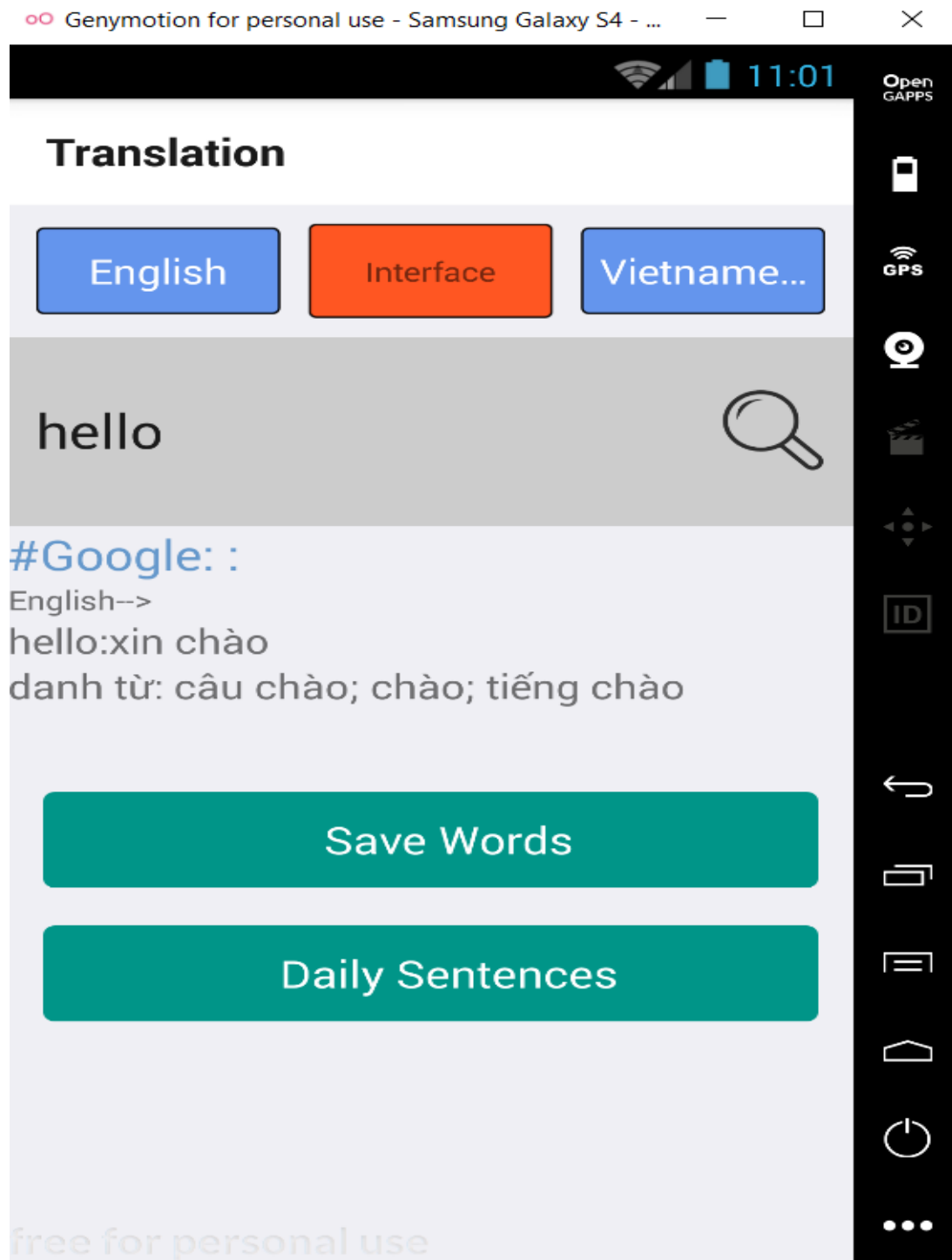
Item	Description	Action	Response
Chọn ngôn ngữ gốc	Bấm vào button chọn ngôn ngữ gốc, để muốn dịch sang ngôn ngữ đích. Có 4 loại ngôn ngữ: <ul style="list-style-type: none"> - Tiếng Nhật - Tiếng Anh - Tiếng Việt - Tiếng Pháp 	Bấm vào Button	Hiện thị ngôn ngữ được chọn
Chọn Interface (dùng nhiều loại API khác nhau: google, bing...)	Dùng nhiều loại API để dịch từ vựng khác nhau. Có thể sử dụng API của Google, Bing, Youdao, Baidu...	Bấm vào Button	Hiện thị các loại API muốn sử dụng
	Có thể chọn dịch một lúc nhiều API	Bấm chọn checkbox	Checkbox ticked
Chọn ngôn ngữ cần dịch sang	Bấm vào button chọn ngôn ngữ muốn dịch sang Có 4 loại ngôn ngữ: <ul style="list-style-type: none"> - Tiếng Nhật - Tiếng Anh - Tiếng Việt - Tiếng Pháp 	Bấm vào Button	Hiện thị ngôn ngữ được chọn
Điền từ cần dịch vào inputText	Điền từ cần dịch vào ô inputText	Bấm chọn vào inputText, điền word cần dịch vào	Hiện thị từ được nhập vào lên inputText cho người dùng nhìn thấy
Button tìm kiếm	Gọi hàm tìm kiếm và lấy từ cần dịch ở inputText	Bấm vào button kính lúp	Dịch từ vựng, và hiển tự kết quả vào listview

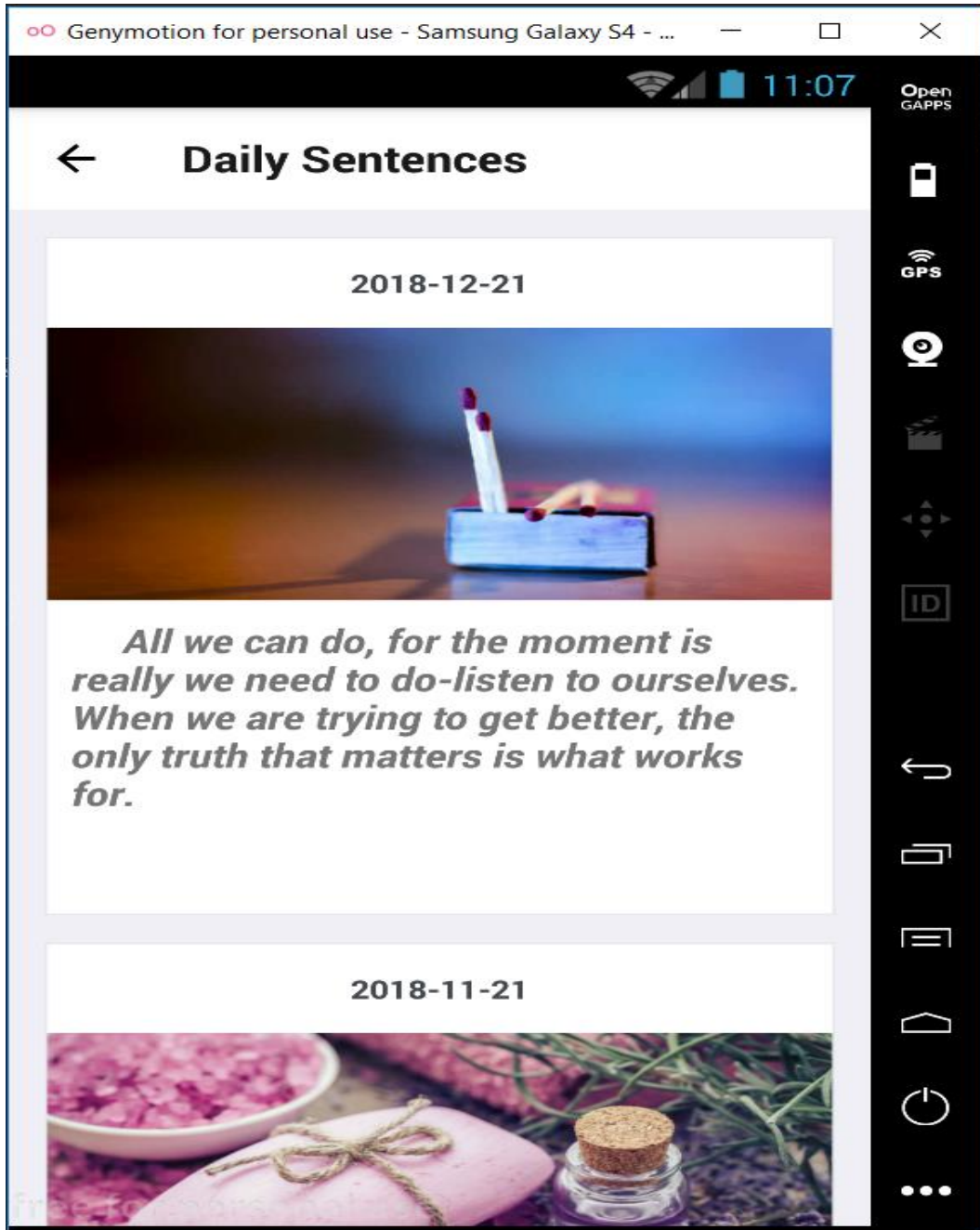
Button Save Words	Lưu các kết quả từ vựng đã tìm kiếm vào file json	Bấm vào button	Hiển thị popup đã lưu từ vựng
Exit	Closed application	Tap on Exit button	Application is closed
Button Daily Sentences	Hiển thị các nội dung, câu châm ngôn, hình ảnh theo ngày. (Được dùng API)	Bấm vào button	Navigation các được hiện ra, hiển thị các nội dung Sentences theo ngày

2.2 Thiết kế hệ thống

2.2.1. Dictionary Mobile Application Main Screen

2.2.1.1. Screen Shot for Dictionary Mobile Application





2.2.1.2. Objects and actions for Dictionary Mobile Application

Objects:

- Original Languages
- Languages translate
- Words translated
- Button change interface
- Save Word
- Daily Sentences

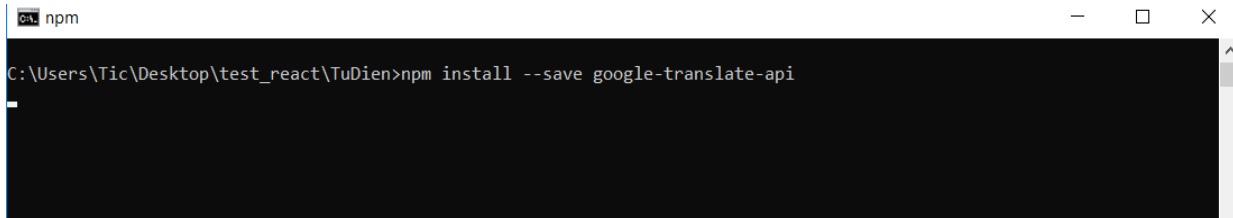
Actions:

- Translate words
- Save words
- Show words already
- Daily Sentences

CHƯƠNG 3. CÀI ĐẶT VÀ SỬ DỤNG GOOGLE TRANSLATE API

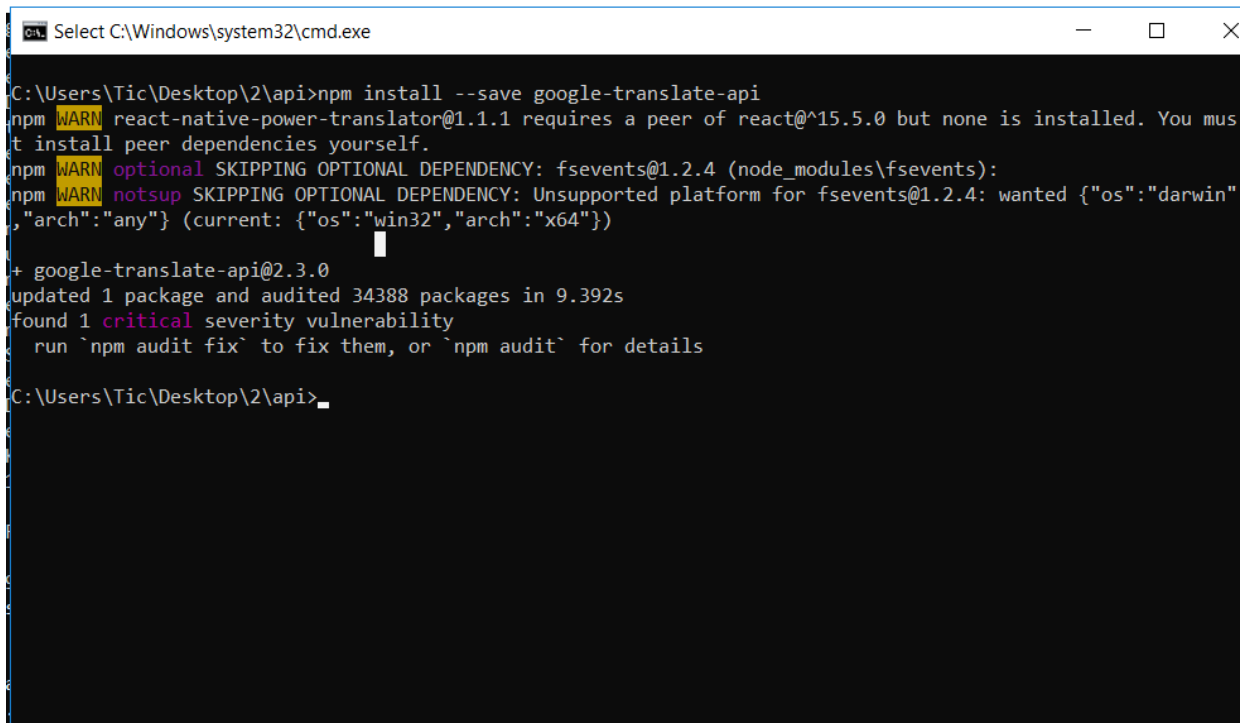
3.1. Cài đặt Google Translate Api:

Cần cài đặt packed vào thư mục react-native đã được cài đặt bằng lệnh:



```
npm
C:\Users\Tic\Desktop\test_react\TuDien>npm install --save google-translate-api
```

Hệ thống sẽ tiến hành cài đặt các gói mở rộng để có thể sử dụng google-translate-api.



```
Select C:\Windows\system32\cmd.exe
C:\Users\Tic\Desktop\2\api>npm install --save google-translate-api
npm WARN react-native-power-translator@1.1.1 requires a peer of react@^15.5.0 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ google-translate-api@2.3.0
updated 1 package and audited 34388 packages in 9.392s
found 1 critical severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
C:\Users\Tic\Desktop\2\api>
```

3.2. Khai báo và sử dụng hàm xử lý dịch

Sau khi cài đặt, chúng ta có thể sử dụng module của Google-translate api một cách dễ dàng.

Khai báo để sử dụng module, và chọn những loại ngôn ngữ muốn chuyển đổi, dịch:

```
1
2 const translate = require('./google-module/google-translate-api');
3
4 // map standard language tags into google language tags
5 const map = {
6   auto: 'auto',
7   zh: 'vi',
8   en: 'en',
9   ja: 'ja',
10  fr: 'fr',
11 };
12 // map google language tags into standard language tags
13 const map_inverse = {
14   auto: 'auto',
15   'zh-CN': 'vi',
16   en: 'en',
17   ja: 'ja',
18   fr: 'fr'
19 };
20
```

Có rất nhiều loại ngôn ngữ có thể dịch sang, API của Google đã hỗ trợ rất nhiều việc đó, chúng ta có thể thêm nhiều loại ngôn ngữ, và dịch sang ở trong module này.

Vì đây là một loại API có trả phí, khi sử dụng link get translate của google, cần phải lọc và replace những ký tự có trong chuỗi trả về (replace
, replace \r\n, ...)

Xây dựng hàm chuyển đổi giữa các ngôn ngữ, và xử lý các chuỗi json được trả về từ API

```
const google = (text, from, to) => {
  from = map[from];
  to = map[to];
  if (from === undefined || to === undefined || from === to)
    throw new Error(`google: unsupported source/destination: from ${from} to ${to}`);

  return translate(text, { from: from, to: to, raw: true })
    .then(res => {
      const json = JSON.parse(res.raw);

      let parts = [];
      try {
        parts = json[1].map((value, index) => `${value[0]}: ${value[1].join('; ')}`);
      } catch (e) {}
      // 'parts' of speech may not exist, such as sentences
      parts = [];

      let sentences = [];
      try {
        sentences = json[13][0].map(sentence => [
          sentence[0].replace(/<\/?b>/g, ''),
          ''
        ]);
      } catch (e) {
        // 'sentences' may not exist, such as when query is a sentence
        sentences = [];
      }

      let synonyms = new Set();
      try {
        json[1].forEach(part => {
          part[2].forEach(means => {
            means[1].forEach(mean => {
              synonyms.add(mean);
            });
          });
        });
        synonyms = [...synonyms];
      } catch (e) {
        synonyms = [];
      }

      let src_pron = '', dst_pron = '';
```

google-translate-api.js x JS google.js

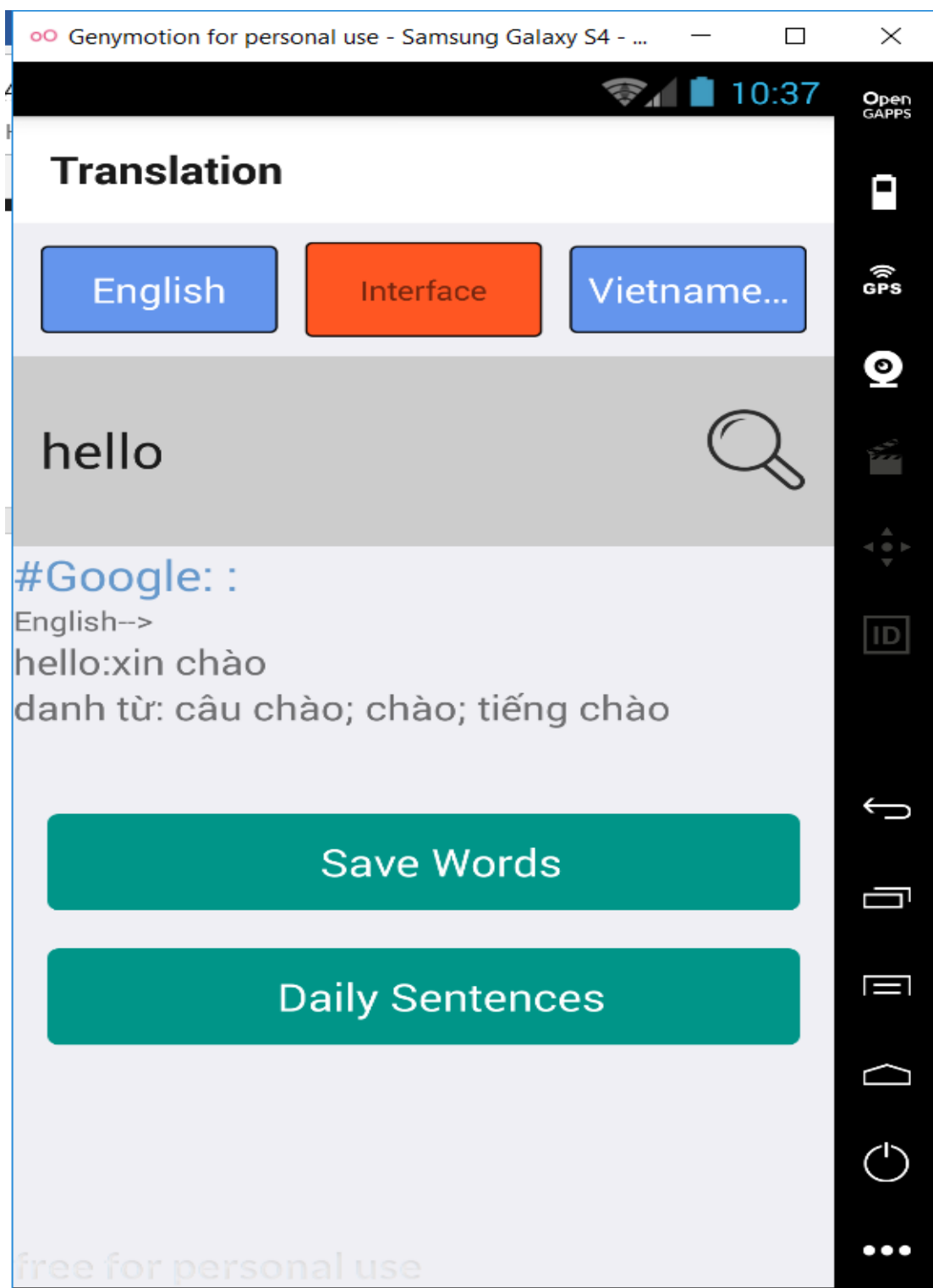
```
1 const { node_fetch } = require('../../cfg/develop_config');
2 if (node_fetch) {
3   eval('var fetch = require(\'node-fetch\')');
4 }
5
6 var token = require('./google-translate-token');
7 var languages = require('./google-translate-languages');
8
9 function translate(text, opts) {
10   opts = opts || {};
11
12   var e;
13   [opts.from, opts.to].forEach(function (lang) {
14     if (lang && !languages.isSupported(lang)) {
15       e = new Error();
16       e.code = 400;
17       e.message = 'The language \'' + lang + '\'' is not supported';
18     }
19   });
20   if (e) {
21     return new Promise(function (resolve, reject) {
22       reject(e);
23     });
24   }
25
26   opts.from = opts.from || 'auto';
27   opts.to = opts.to || 'en';
28
29   opts.from = languages.getCode(opts.from);
30   opts.to = languages.getCode(opts.to);
31
32   return token.get(text).then(function (token) {
33     var url = 'https://translate.googleapis.com/translate_a/single';
34     var data = {
35       client: 'gtx',
36       sl: opts.from,
37       tl: opts.to,
38       hl: opts.to,
39       dt: ['at', 'bd', 'ex', 'ld', 'md', 'qca', 'rw', 'rm', 'ss', 't'],
40       ie: 'UTF-8',
41       oe: 'UTF-8',
42       otf: 1,
43       ssel: 0,
44       tsel: 0,
45       kc: 7,
46       q: text
47     };
48     data[token.name] = token.value;
```

3.3. Test chương trình khi dùng Google-translate-API:

Tiến hành add các hàm trong API vào các button:

```
render() {  
  return (  
    <View>  
      <ScrollView>  
        <View style={{ flexDirection: 'row' }}>  
          <ModalDropdown style={styles.dropdown}>  
            textStyle={styles.dropdown_text}>  
            dropdownStyle={styles.dropdown_dropdown}>  
            options={language.from}>  
            defaultValue={ 'From' }>  
            defaultIndex={1}>  
            onSelect={(idx, value) => this.setState({ from: idx })}>  
          />  
          <TouchableHighlight style={styles.btn} onPress={() => this.setModalVisible(true)}>  
            <Text>  
              Interface  
            </Text>  
          </TouchableHighlight>  
          <ModalDropdown style={styles.dropdown}>  
            textStyle={styles.dropdown_text}>  
            dropdownStyle={styles.dropdown_dropdown}>  
            options={language.to}>  
            defaultValue={ 'To' }>  
            defaultIndex={1}>  
            onSelect={(idx, value) => this.setState({ to: idx })}>  
          />  
        </View>  
        <View style={styles.container}>  
          <AutoExpandingInput onChangeText={this._onChangeText}>  
            queryValue={this.state.querytext}>  
            style={{ flex: 1 }}>  
          />  
          {this.state.querytext == "" ? <Text /> :  
            <TouchableHighlight underlayColor='white' onPress={this._findText}>  
              <Image>  
                source={require('./app/res/search.png')}>  
                style={{ width: 50, height: 50, marginRight: 10 }}>  
              />  
            </TouchableHighlight>  
          }  
        </View>  
      </ScrollView>  
    </View>  
  )  
}
```

Chạy thử chương trình:



CHƯƠNG 4. CÀI ĐẶT VÀ KIỂM THỬ

4.1 Cài đặt

- Sử dụng Genymotion để cài đặt và phát triển
- Cài đặt react-native
- Thiết kế hệ thống, giao diện
- Cài đặt modules google-api-translate
- Triển khai code phần mềm
- Sửa lỗi, debug
- Tìm hiểu các components
- Áp dụng, sử dụng các components vào project
- Chỉnh sửa giao diện
- Kiểm thử

4.2 Kiểm thử

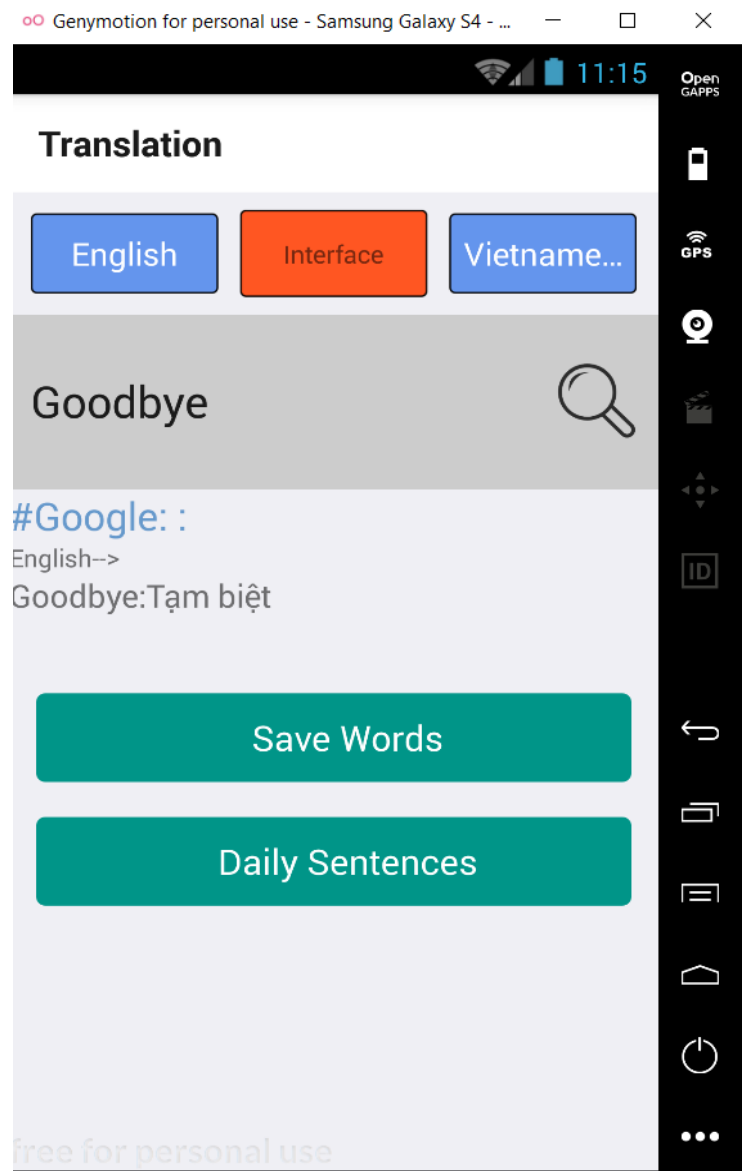
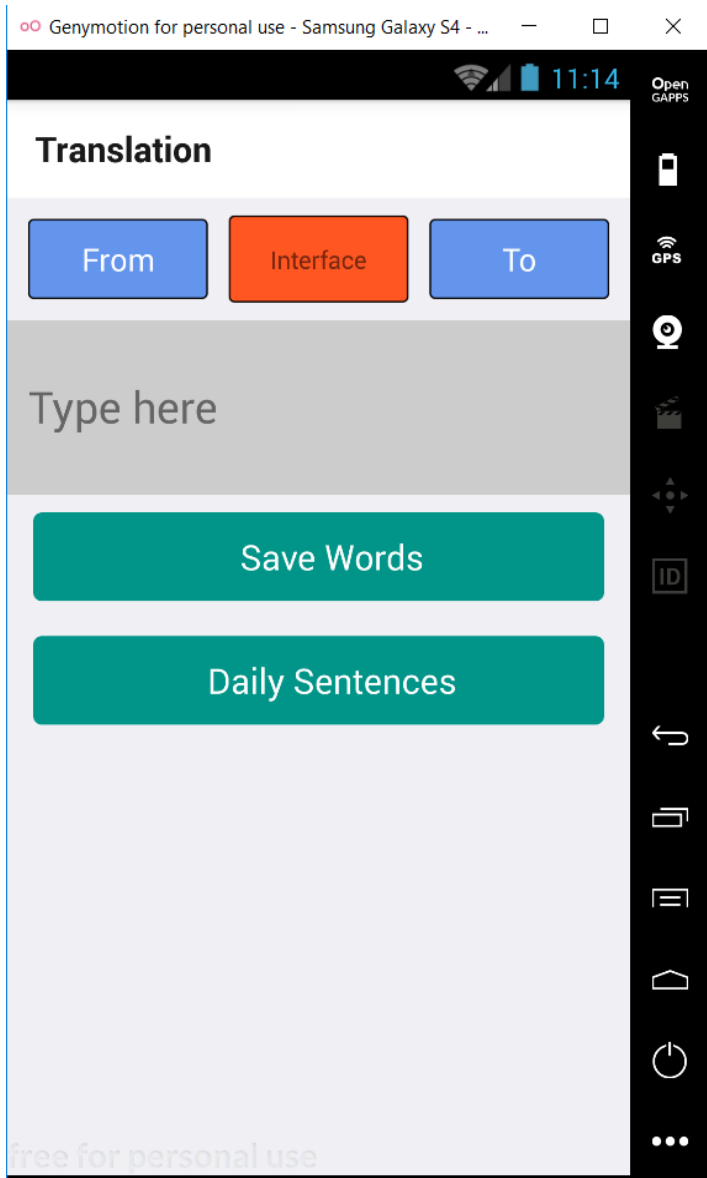
Step	Test Steps	Actual Result	Status (Pass/Fail)	Note
1	Test Interface action	1.Nhấp vào interface 2.Chọn google 3.Done	Pass	
2	Test From Action	1.Nhấp From 2.Chọn 1 ngôn ngữ trong 4 ngôn ngữ 3.Button From chuyển từ From sang ngôn ngữ ấy	Pass	
3	Test To Action	1.Nhấp To 2.Chọn 1 ngôn ngữ trong 4 ngôn ngữ 3.Button To chuyển từ To sang ngôn ngữ ấy	Pass	
4	Test Text Input	1.Click vào Text Input 2.Nhập từ cần dịch vào text input	Pass	
5	Test button Save Words	Tap on button	Pass	

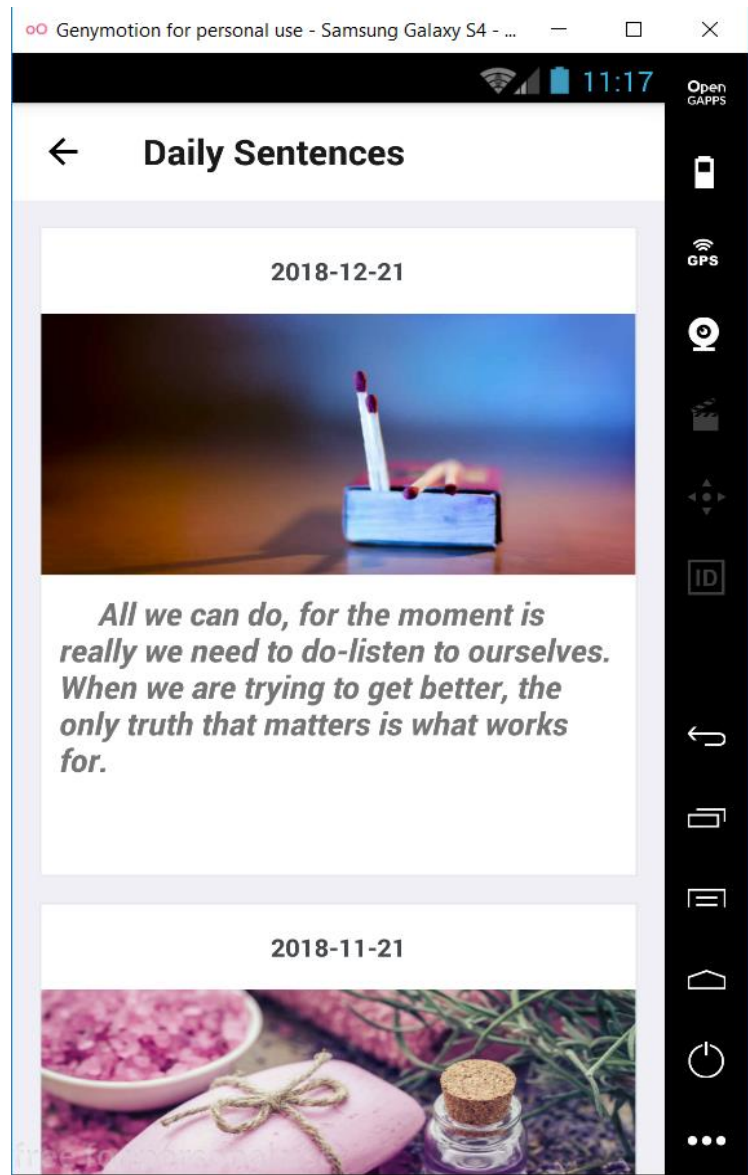
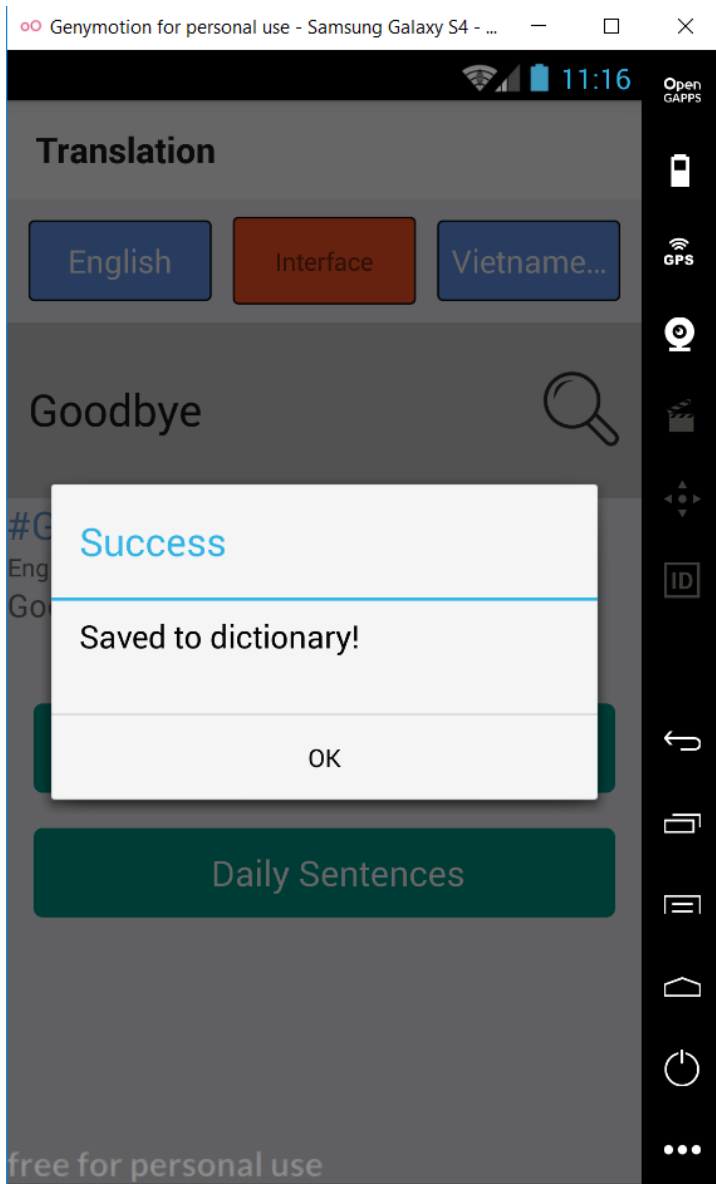
6	Test button interface	Tap on button	Pass	
7	Test button Daily Sentences	Tap on button	Pass	
8	Test From	1.Nhấp From 2. Hiện thị những ngôn ngữ	Pass	
9	Test To	1.Nhấp To 2. Hiện thị những ngôn ngữ	Pass	
10	Test find action	1.Chọn ngôn ngôn ngữ cần dịch và ngôn ngữ dịch 2.Nhập từ vào Text Input 3.Xuất hiện từ đã được dịch	Pass	
11	Test Save Words action	1.Dịch thành công từ cần dịch 2.Nhấp save words 3.Save data successfully	Pass	
12	Test Daily Sentences	1.Nhấp daily sentences 2.Xuất hiện Daily sentences layout	Pass	
13	Test the same language	1.Ví dụ chọn 2 ngôn ngữ là english và english 2. Xuất hiện thông báo the two languages are the same	Pass	
14	Test different languages	1.Ví dụ chọn 2 ngôn ngữ là english và french 2. Từ nhập "hello" 3.Hiện ra là "Boujour"	Pass	

CHƯƠNG 5. KẾT QUẢ ĐẠT ĐƯỢC

5.1 Kết quả đạt được

Các kết quả đạt được khi xây dựng ứng dụng:





5.2 Các kết luận và kiến nghị

- Dịch được và chuyển đổi được từ vựng sang ngôn ngữ khác
- Sử dụng và chuyển đổi được 4 loại ngôn ngữ
- Dịch ngược các loại văn bản
- Các chức năng bổ sung nếu có thêm thời gian:
 - + Sử dụng thêm nhiều loại API hơn
 - + Lưu những từ vựng được dịch vào database

TÀI LIỆU THAM KHẢO

1. Github – website & source code react native: <https://github.com/facebook/react-native>
2. Google – website: <https://google.com.vn/>
3. Wikipedia – website: [https://en.wikipedia.org/wiki/React_\(JavaScript_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library))
4. React-native from facebook – website: <https://facebook.github.io/react-native/>