fTRƯỜNG CAO ĐẲNG CÔNG NGHỆ THỦ ĐỨC

KHOA CÔNG NGHỆ THÔNG TIN

--------------------------------

BÁO CÁO KẾT THÚC MÔN HỌC

Lập trình di động 3

**TÊN ĐỀ TÀI**

Giảng viên hướng dẫn: Trương Bá Thái

Sinh viên thực hiện:

1. Lê Nguyễn Việt Thanh
2. Phạm Minh Nhàn
3. Ngô Hòa

Ngành: Công nghệ thông tin Lớp:LTDD3 Khoá: 15

*Tp. Hồ Chí Minh*, ngày tháng năm

**NHẬT KÝ HOẠT ĐỘNG NHÓM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Stt** | **Họ và tên** | **Công việc đã thực hiện** | **Tự đánh giá** | **Nhóm đánh giá** | **Chữ ký** |
| 1 | Lê Nguyễn Việt Thanh | 1. Cài đặt react native 2. So sánh let var arrow function 3. Widht, hight, flex trong react native 4. Code giao diện và chức năng trang quản lý lớp, quản lý sinh viên 5. Tìm hiểu listview 6. Tìm hiểu navigator 7. Tìm hiểu nội dụng để viết báo cáo |  |  |  |
| 2 | Phạm Minh Nhàn | 1. Cài đặt react native 2. Thiết kế màn hình Login, home 3. Tìm hiểu mảng trong ES6 4. Tìm hiểu style trong react native 5. Tìm hiểu webservice 6. Tìm hiểu navigator 7. Code giao diện và chức năng trang Home, Login |  |  |  |
| 3 | Ngô Hòa | 1. Tìm hiểu set trong ES6 2. Tìm hiểu flexbox trong react native 3. Tìm hiểu lisview 4. Tìm hiểu navigator |  |  |  |

MỤC LỤC

DANH MỤC BẢNG BIỂU, HÌNH VẼ, SƠ ĐỒ 5

CHƯƠNG 1. TỔNG QUAN VỀ REACT NATIVE 6

1. Giới thiệu tổng quan về React native 6

2. Kiến thức cơ bản về ES6 trong React native 6

Các phương pháp mảng 10

dùng để kết nối 2 hay nhiều mảng với nhau. Phương thức này không làm thay đổi các mảng đã có mà thay vào đó sẽ trả về 1 mảng mới. 10

every() 10

trả về giá trị nếu đùng điều kiện. 10

filter() 10

dùng để tạo một mảng mới với tất cả các phần tử thỏa điều kiện của một hàm test. 10

forEach() 10

duyệt qua từng phần tử của mảng. 10

4. Phương pháp mảng 10

Array.prototype.find 10

cho phép bạn lặp qua một mảng và lấy lại phần tử đầu tiên làm cho hàm gọi lại đã cho trả về true. Khi một phần tử đã được tìm thấy, hàm này sẽ trả về ngay lập tức. 10

Array.prototype.findIndex 11

hoạt động tương tự như tìm, nhưng thay vì trả về phần tử đã khớp, nó trả về chỉ mục của phần tử đó. 11

Array.prototype.entries 11

là một hàm trả về một Iterator mảng có thể được sử dụng để lặp qua các khóa và giá trị của mảng. Các mục sẽ trả về một mảng các mảng, trong đó mỗi mảng con là một mảng của [chỉ mục, giá trị]. 11

Array.from 11

Array.prototype.keys () 12

3. Component trong react native 16

4. API Trong React Native 26

CHƯƠNG 2. PHÂN TÍCH THIẾT KẾ HỆ THỐNG 28

2.1 Phân tích hệ thống 28

1. Introduction 30

1.1 Overview 30

1.2 Scope 30

2. Functional Requirements 31

Feature/Component #1: login page 31

User Interfaces 31

Functional Requirements 32

User Interfaces 33

Functional Requirements 33

Feature/Component #2: Quản lý screens 35

User Interfaces 35

Functional Requirements 35

Feature/Component #3: Quan Ly Sinh Vien Page 36

Quan Ly Sinh Vien Screens 37

Functional Requirements 37

Feature/Component #4: Quan Ly Lop page 39

User Interfaces 39

Functional Requirements 39

2.2 Thiết kế hệ thống 40

\*A - Added M - Modified D – Deleted 2

<Position>member 3

<Position> 3

<Position 3

1 INTRODUCTION 5

1.1 Purpose 5

1.2 Overview 5

1 SCREEN FLOW 6

1.1 Screen Login 6

1.2 Screen Quản lý 9

1.3 Quản lý sinh viên screen 11

1.4 Quản Lý lớp screen 15

CHƯƠNG 3. CÀI ĐẶT VÀ KIỂM THỬ 16

3.1 Cài đặt 16

3.2 Kiểm thử 16

CHƯƠNG 4. KẾT QUẢ ĐẠT ĐƯỢC 18

4.1 Kết quả đạt được 18

4.2 Các kết luận và kiến nghị 18

PHỤ LỤC 19

TÀI LIỆU THAM KHẢO 20

# DANH MỤC BẢNG BIỂU, HÌNH VẼ, SƠ ĐỒ

# CHƯƠNG 1. TỔNG QUAN VỀ REACT NATIVE

## Giới thiệu tổng quan về React native

- React Native là công nghệ được tạo bởi Facebook cho phép các dev sử dụng JavaScript để làm mobile apps trên cả Android và iOS với cảm nhận và giao diện native.

Ưu điểm:

* Code một lần, chạy nhiều chỗ
* Hoạt động đa hệ điều hành rất tốt
* Tốc độ xử lý cao
* Có nhiều tính năng mới

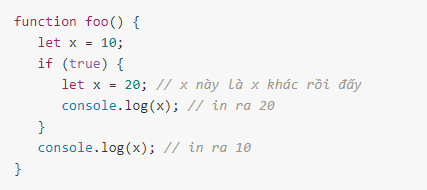
nhược điểm:

* Còn thiếu nhiều component view
* Chỉ hỗ trợ cho 2 nền tảng android và IOS
* Windows và linux chưa có official support từ React Native
* Không thể build được ứng dụng quá phức tạp như native
* Không nên dùng để viết game có tính đồ họa và cách chơi phức tạp
* Khó tiếp cận
* Chưa có IDE hỗ trợ

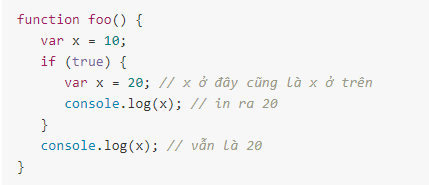
## Kiến thức cơ bản về ES6 trong React native

1. **Sự khác nhau Let và Var trong ES6(Nhàn)**

Let: tạo ra một biến chỉ có thể truy cập được trong block bao quanh nó và sẽ tự hủy sau khi kết thúc {}

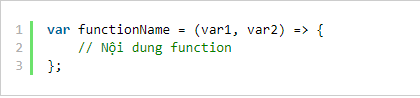


Var: tạo ra một biến có phạm vi truy cập trong function của nó

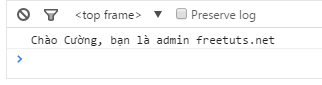
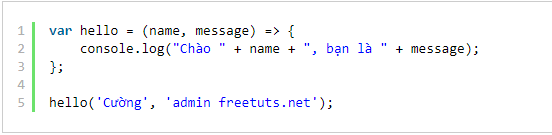


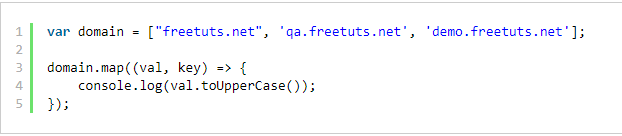
1. **Arrow Function trong ES6**

+ Cú pháp căn bản:



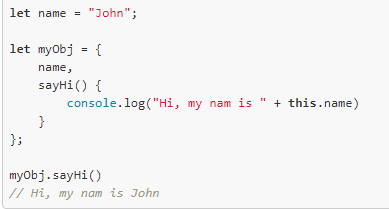
+ Ví dụ:





1. **Object trong ES6**

+ Object trong JavaScript có thể được tạo theo một số cách khác nhau. Object mới trong ES6 tạo object sử dụng cặp dấu ngoặc {} và bên trong là danh sách các property ( thuộc tính ) của object



1. **Class trong ES6**

ES6 hiện nay đã hỗ trợ khai báo Class thay vì phiên bản cũ là ES5 lại không hỗ trợ. Giúp cho những người yêu thích lập trình hướng đối tượng đã có thể khai báo Class trong phiên bản ES6 hiện tại với những cú pháp đơn giản sau:



Nhóm 4.

Cách xử lý mảng trong ES6

1. Khai báo và khởi tạo mảng.

Để khai báo mảng sử dụng câu lệnh sau :

var array\_name; //declaration

array\_name = [val1,val2,valn..] //initialization

OR

var array\_name = [val1,val2…valn]

Cặp [] được gọi là thứ nguyên của mảng.

1. Truy cập các phần tử mảng.

Truy cập bằng chỉ mục của mảng

Cú pháp

array\_name[subscript]

ví dụ:

mảng 1 chiều:

var alphas;

alphas = ["1","2","3","4"]

console.log(alphas[0]);

console.log(alphas[1]);

kết quả :

1

2

var nums = [1,2,3,3]

console.log(nums[0]);

console.log(nums[1]);

console.log(nums[2]);

console.log(nums[3]);

kết quả :

1

2

3

3

1. Đối tượng mảng

Một mảng cũng có thể được tạo ra bằng cách sử dụng đối tượng mảng. Các nhà xây dựng mảng có thể được thông qua như sau :

* Một giá trị số đại diện cho kích thước của mảng hoặc.
* Danh sách các giá trị được phân cách bằng dấu phẩy.
* var arr\_names = new Array(4)
* for(var i = 0;i<arr\_names.length;i++) {
* arr\_names[i] = i \* 2
* console.log(arr\_names[i])
* }

var names = new Array("Mary","Tom","Jack","Jill")

for(var i = 0;i<names.length;i++) {

console.log(names[i])

}

## Các phương pháp mảng

[**concat ()**](https://www.tutorialspoint.com/es6/es6_array_method_concat.htm)

## dùng để kết nối 2 hay nhiều mảng với nhau. Phương thức này không làm thay đổi các mảng đã có mà thay vào đó sẽ trả về 1 mảng mới.

## [every()](https://www.tutorialspoint.com/es6/es6_array_method_every.htm)

## trả về giá trị nếu đùng điều kiện.

## [filter()](https://www.tutorialspoint.com/es6/es6_array_method_filter.htm)

## dùng để tạo một mảng mới với tất cả các phần tử thỏa điều kiện của một hàm test.

## [forEach()](https://www.tutorialspoint.com/es6/es6_array_method_foreach.htm)

## duyệt qua từng phần tử của mảng.

## Phương pháp mảng

### Array.prototype.find

## cho phép bạn lặp qua một mảng và lấy lại phần tử đầu tiên làm cho hàm gọi lại đã cho trả về true. Khi một phần tử đã được tìm thấy, hàm này sẽ trả về ngay lập tức.

var numbers = [1, 2, 3];

var oddNumber = numbers.find((x) => x % 2 == 1);

console.log(oddNumber); // 1

### Array.prototype.findIndex

## hoạt động tương tự như tìm, nhưng thay vì trả về phần tử đã khớp, nó trả về chỉ mục của phần tử đó.

var numbers = [1, 2, 3];

var oddNumber = numbers.findIndex((x) => x % 2 == 1);

console.log(oddNumber); // 0

### Array.prototype.entries

## là một hàm trả về một Iterator mảng có thể được sử dụng để lặp qua các khóa và giá trị của mảng. Các mục sẽ trả về một mảng các mảng, trong đó mỗi mảng con là một mảng của [chỉ mục, giá trị].

var numbers = [1, 2, 3];

var val = numbers.entries();

console.log(val.next().value);

console.log(val.next().value);

console.log(val.next().value);

### Array.from

cho phép tạo một mảng mới từ một mảng như đối tượng. Chức năng cơ bản của Array.from () là chuyển đổi hai loại giá trị thành mảng -

* Giá trị giống như mảng.
* Các giá trị lặp lại như Set và Map.
* "use strict"
* for (let i of Array.from('hello')) {
* console.log(i)
* }
* Đầu ra sau được hiển thị khi thực hiện thành công mã trên.
* h
* e
* l
* l
* o

### Array.prototype.keys ()

Hàm này trả về các chỉ mục mảng.

Thí dụ

console.log(Array.from(['a', 'b'].keys()))

Đầu ra sau được hiển thị khi thực hiện thành công mã trên.

[ 0, 1 ]

Người ta có thể sử dụng cho ... trong vòng lặp để đi qua một mảng.

"use strict"

var nums = [1001,1002,1003,1004]

for(let j in nums) {

console.log(nums[j])

}

Vòng lặp thực hiện truyền tải mảng dựa trên chỉ mục. Đầu ra sau được hiển thị khi thực hiện thành công mã trên.

1001

1002

1003

1004

Cách sử dụng Map trong ES6(Thanh)

-Map là một kiểu dữ liệu tương tự như Set, tuy nhiên Map có dạng cấu trúc dạng key => value còn với Set thì chỉ có value.

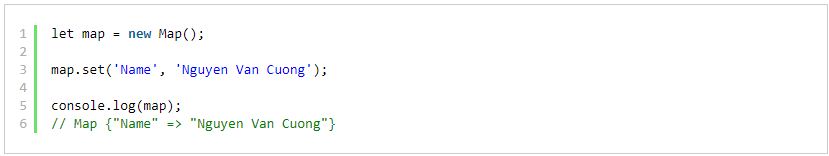
\*Các thao tác chính với map:  
Khởi tạo: let map = new Map()  
Thêm phần tử: map.set('Name', 'Nguyen Van Cuong');  
Xóa phần tử: map.delete("Name");  
Kiểm tra phần tử tồn tại: map.has('Name')  
Đếm tổng số phần tử: map.size  
Xóa toàn bộ phần tử: map.clear();

-Đối với Map các key không được trùng, vì vậy nếu truyền vào hai key giống nhau thì nó chỉ lưu đè vào một key duy nhất.

1. Khởi tạo: Lúc khởi tạo sẽ có một tham số truyền vào và giá trị của tham số này là một mảng chứa nhiều mảng con, mỗi mảng con sẽ có hai pần tử đại diện cho key và value.



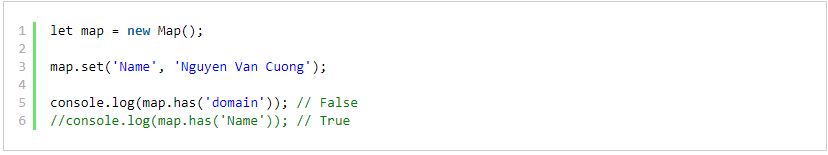
1. Thêm phần tử



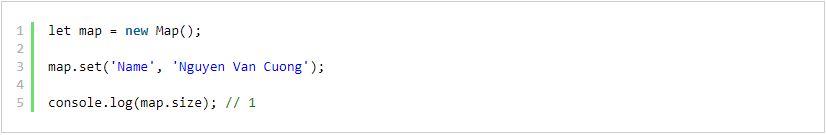
1. Xóa phần tử



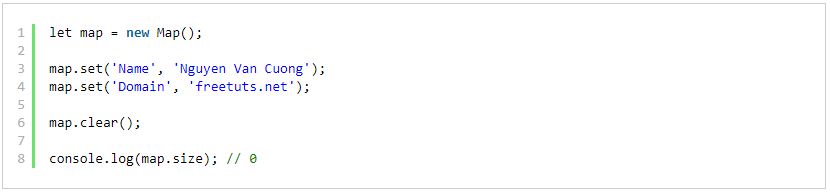
1. Kiểm tra phần tử tồn tại



1. Đếm tổng số phần tử



1. Xóa toàn bô phần tử



**Set trong ES6**

có bốn thao tác chính

* **Khởi tạo**: let set = new Set(); ->Lúc khởi tạo sẽ có một tham số truyền vào, tham số này bắt buộc phải là một mảng.

var numbers = new Set([1, 2, 3, 4]);

* **Thêm phần tử**: set.add(value);

var numbers = new Set([1, 2, 3, 4]);

numbers.add(5); // numbers = Set {1, 2, 3, 4 ,5}

* **Xóa phần tử**: set.delete(value);

var numbers = new Set([1, 2, 3, 4]);

numbers.delete(2); // numbers = Set {1, 3, 4}

* **Kiểm tra tồn tại giá trị**: set.has(value);

var numbers = new Set([1, 2, 3, 4]);

console.log(numbers.has(1)); // True

console.log(numbers.has(5)); // False

* **Đếm tổng số phần tử**: set.size;

var numbers = new Set([1, 2, 3, 4]);

console.log(numbers.size); // 4

* **Xóa toàn bộ phần tử**: set.clear();

var numbers = new Set([1, 2, 3, 4]);

numbers.clear(); // numbers = Set()

Set thì các giá trị không được trùng

## Component trong react native

**Props**

props là viết tắt của Properties. Một điều mà bạn cần phải nhớ khi sử dụng props đó là không bao giờ nên thay đổi giá trị của nó, hay nói cách khác, đây là một dữ liệu immutable.

Các component nhận props từ component cha. Bạn không được thay đổi giá trị của props trong các component này mà chỉ được phép đọc giá trị ra thôi. Trong React thì dữ liệu sẽ đi theo một chiều, có nghĩa là từ component cha => các component con.

Bạn có thể tạo ra component sử dụng props. Ý tưởng của props đó là việc trừu tượng hoá các component để có thể sử dụng được ở nhiều chỗ khác nhau trong app. Hãy xem một ví dụ đơn giản sau:

// Parentexport default class ScreenOne extends React.Component {

render () {

return (

<View>

<Heading message={'Custom Heading for Screen One'}/>

</View>

)

}}

// Child componentexport default class Heading extends React.Component {

render () {

return (

<View>

<Text>{this.props.message}</Text>

</View>

)

}}

Heading.propTypes = {

message: PropTypes.string}

Heading.defaultProps = {

message: 'Heading One'}

**State**

state thì hoạt động khác với props. state là dữ liệu nội bộ của một Component, trong khi props là dữ liệu được truyền cho Component. Chính vì vậy chúng ta hoàn toàn có thể thay đổi state, và coi nó là một kiểu dữ liệu mutable.

Tuy vậy, hãy nhớ rằng đừng bao giờ thay đổi trực tiếp biến this.state. Thay vào đó hãy dùng hàm setState để cập nhật giá trị. Sở dĩ chúng ta cần dùng hàm này là do nó sẽ kích hoạt việc render lại component và tất cả component con nằm trong nó, còn thay đổi this.state thì không. Còn một vấn đề nữa, đó là setState chạy bất đồng bộ, vậy nên nếu bạn tiến hành đọc ra giá trị state ngay sau khi setState thì chưa chắc giá trị sẽ được update lên mới nhất đâu.

Ví dụ nếu bạn cần phải điền một cái form với nhiều text input, thì mỗi text input sẽ giữ đoạn text mà bạn gõ vào trong state của nó. Nếu bạn gõ một đoạn text khác vào, state của text input đó sẽ thay đổi và kích hoạt việc render lại component, và text mới của bạn sẽ được vẽ lên.

class Form extends React.Component {

constructor (props) {

super(props)

this.state = {

input: ''

}

}

handleChangeInput = (text) => {

this.setState({ input: text })

}

render () {

const { input } = this.state

return (

<View>

<TextInput style={{height: 40, borderColor: 'gray', borderWidth: 1}}

onChangeText={this.handleChangeInput}

value={input}

/>

</View>

)

}

}

Style Trong React Native

Với React-Native hỗ trợ style cho tất cả các components . Nó đơn giản như bạn viết CSS cho Web. Tuy nhiên có một số thay đổi về quy định cách thức viết các selector, hay cách gọi css…các bạn cần nắm rõ các quy định đó. Ví dụ trong CSS sử dụng cho HTML chúng ta viết background-color thì sang React-Native chúng ta sử dụng backgroundColor Thao tác với css cho đối tượng, tức là bạn sẽ thao tác trong class hiển thị ra phần view. Có 3 kiểu khai báo styles:

1. Khai báo trực tiếp trong Component

import React, { Component } from 'react';

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

class StylesInComponent extends Component {

render() {

return (

<View style={{

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: 'red',

}}>

<Text style={{

fontSize: 20,

textAlign: 'center',

margin: 10,

color: 'white',

}}>

Welcome to React Native!

</Text>

</View>

);

}

}

Mỗi component View, Text... sẽ có thuộc tính style Lưu ý cách viết : style ={{backgroundColor : ‘red’}} nhé Trong 2 dấu {{}} thì 1 dấu là viết cho JSX , 1 dấu là kiểu đối tượng.

1. Khai báo ngoài Component nhưng vẫn nằm trong một file Đầu tiên bạn cần import thư viện StyleSheet Tiếp theo bên ngoài Component bạn khai báo biến styles kiểu StyleSheet. Bạn sẽ tạo các selector CSS :

* container -> View
* welcome -> Text

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#F5FCFF',

},

welcome: {

fontSize: 20,

textAlign: 'center',

margin: 10,

},

});

Quay vào trong class cha chứa các component, bạn đơn giản chỉ cần gọi biến styles để lựa chọn các selector tương ứng cho component

class StylesOutComponent extends Component {

render() {

return (

<View style={styles.container}>

<Text style={styles.welcome}>

Welcome to React Native!

</Text>

</View>

);

}

}

});

1. Khai báo vào file riêng

Tạo file có tên Styles.js với nội dung như sau :

import { StyleSheet,} from 'react-native';

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#F5FCFF',

},

welcome: {

fontSize: 20,

textAlign: 'center',

margin: 10,

}

});

export default styles;

Đầu tiên bạn cần import thư viện StyleSheet Tương tự bạn tạo cái selector CSS : container, welcome cho biến styles Quan trọng nhất là bạn phải thêm dòng export default styles; vào cuối file. Khi bạn đã export thì có nghĩa là bạn có thể import styles ở bất kì đâu để sử dụng. Sử dụng câu lệnh import styles from './Styles' . Trong đó './Styles' là đường dẫn của tới Styles.js

import styles from './Styles'

class StylesNewFile extends Component {

render() {

return (

<View style={styles.container}>

<Text style={styles.welcome}>

Welcome to React Native!

</Text>

</View>

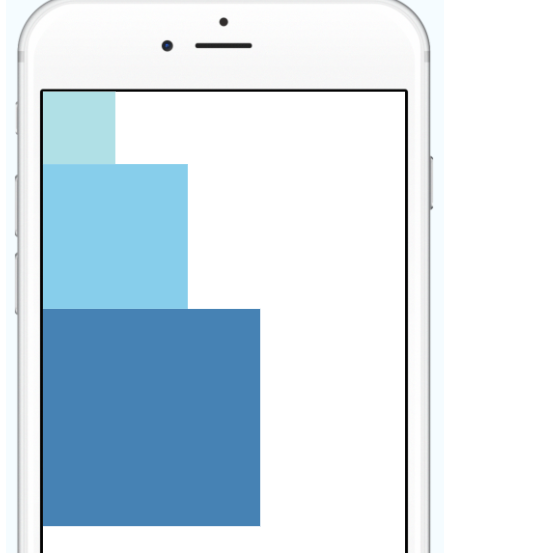
);

}

}

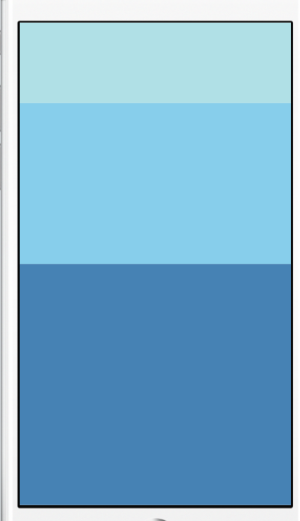
**Width, Height và Flex(Hòa)**

Width và Height: là chiều dài và chiều rộng của components sẽ xác định kích thước trong màn hình ( kích thước cố định ) và đều tuân theo đơn vị pixel



Flex: sử dụng Flex ( kích thước linh động ) trong components để thay đổi vùng hiển thị một cách linh động

Ta sẽ sử dụng code:  **flex: 1**



Flexbox

* **Khái niệm.**
* Là một kiểu dàn trang (layout mode) mà nó sẽ tự cân đối kích thước của các phần tử bên trong để hiển thị trên mọi thiết bị.
* Bạn không cần thiết lập kích thước của phần tử, không cần cho nó float, chỉ cần thiết lập nó hiển thị chiều ngang hay chiều dọc, lúc đó các phần tử bên trong có thể hiển thị theo ý muốn.
* Bố cục Flex được thiết lập từ một khung lớn (parent container) đóng vai trò là khung linh hoạt (flex containter) và các thẻ con ngay trong nó (immediate children) đóng vai trò các mục nhỏ linh hoạt (flex items).
* **Các công cụ hỗ trợ**
* Chrome 29+ Firefox 28+ Internet Explorer 11+ Opera 17+ Safari 6.1+ (sử dụng tiền tố -webkit-) Android 4.4+ iOS 7.1+ (sử dụng tiền tố -webkit-)
* **Cách sử dụng**
* Để sử dụng Bố cục Flexbox bạn chỉ cần đặt giá trị cho thuộc tính display trên khung lớn (parent container). Hoặc bạn muốn nó hiển thị như một phần tử inline….
* Chú ý: Bạn chỉ cần đặt thuộc tính trên vào khung lớn là các thẻ con sẽ lập tức trở thành các mục linh hoạt.
* **Các thuộc tính**

Chúng có thể được trình bày theo hai hướng, giống như hàng ngang hay hàng dọc.

* Theo hàng ngang (row), các mục linh hoạt sẽ được sắp xếp theo một hàng từ trái qua phải.
* Theo hàng đảo ngược (row-reverse), các mục linh hoạt sẽ được sắp xếp theo một hàng ngược lại.
* Tương tự với hàng dọc (column), các mục linh hoạt sẽ được sắp xếp theo một cột từ trên xuống dưới. Và ngược lại…
* **flex-wrap**
* đặt các mục linh hoạt theo một hàng duy nhất. Nhưng sẽ thế nào nếu chúng ta muốn có một bố cục với các mục xếp thành nhiều hàng? Thuộc tính flex-wrap được tạo ra để giúp chúng ta giải quyết điều này.
* Các mục linh hoạt được hiển thị trên cùng một hàng, mặc định chúng sẽ tự động dãn hoặc thu hẹp để vừa với chiều rộng của khung lớn.
* Các mục linh hoạt có thể hiển thị nhiều hàng từ trái qua phải và từ trên xuống dưới nếu cần
* Thậm chỉ là ngược lại nếu bạn muốn
* Giá trị mặc định: nowrap Chú ý: Thuộc tính này phụ thuộc vào chế độ viết cho nên ở chế độ viết từ phải qua trái (rtl) thì chúng sẽ tự động được đảo ngược.
* **flex-flow**
* Thuộc tính này là một dạng viết tắt (shorthand) cho hai thuộc tính flex-direction và flex-wrap.
* Giá trị mặc định: row nowrap
* **justify-content**
* sắp xếp các mục linh hoạt theo trục chính của dòng hiện tại trong khung linh hoạt. Nó giúp bổ sung không gian còn thừa ngay cả khi các mục linh hoạt trên một dòng không thể co giãn hoặc đã đạt đến kích thước tối đa.
* Các mục linh hoạt được sắp xếp ở bên trái của khung linh hoạt theo dạng ltr
* Các mục linh hoạt được sắp xếp ở bên phải của khung linh hoạt theo dạng ltr
* Các mục linh hoạt được sắp xếp ở giữa khung linh hoạt
* Các mục linh hoạt sẽ được hiển thị với khoảng cách tương đương nhau ở giữa chúng, mục linh hoạt đầu tiên và cuối cùng được xếp vào 2 bên của khung linh hoạt
* Các mục linh hoạt được hiển thị với khoảng cách bằng nhau xung quanh mọi mục linh hoạt, kể cả mục linh hoạt đầu tiên và cuối cùng
* Giá trị mặc định: flex-start

## 4. API Trong React Native

# CHƯƠNG 2. PHÂN TÍCH THIẾT KẾ HỆ THỐNG

## 2.1 Phân tích hệ thống

Software Requirement Specification

NHOM8-SRS-001

Version 1.0

**Revision History**

| Date | Version | Description | Author | Revised by |
| --- | --- | --- | --- | --- |
| 3/12/2018 | 1.0 | First version | Lê Nguyễn Việt Thanh | Phạm Minh Nhàn |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Introduction

## Overview

This software supports users to view the phone on the online and can order the product.

## Scope

Demo version

Demo version is for display purpose only, not link to Database, not link to affiliated sites

Android OS: Android: OS 4.0.X to 6.0.X

Android screen resolutions: 480x800, 720x1280

Support screen orientation in landscape mode only

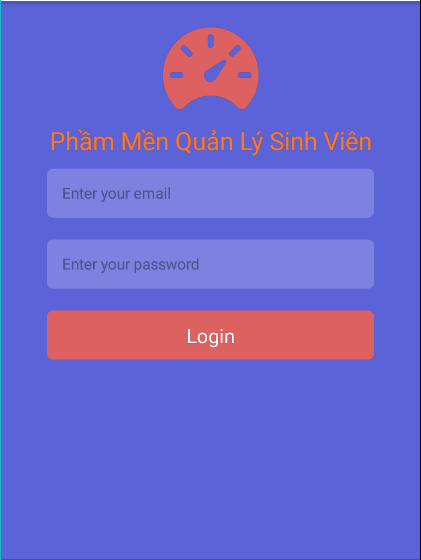
# Functional Requirements

## Feature/Component #1: login page

### User Interfaces

**Feature/Component #2:** **Registration page**

### User Interfaces



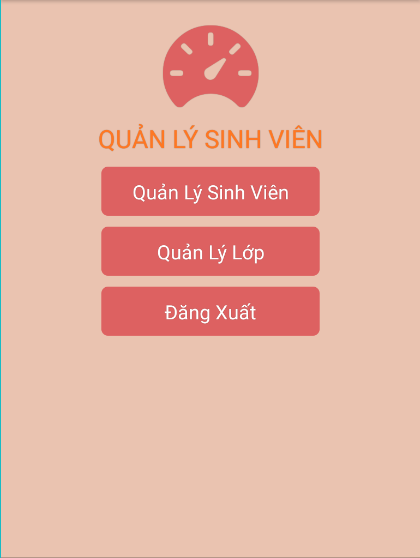
*Image1:* login page

### Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Description** | **Action** | **Response** |
| login page | Include:   * 1 Button * 2 Text Input * Text * 1 Image | N/A | N/A |
| Button “Login” | Log in successfully if correct account information | Click | Logged in successfully,  Switch to home page |
| Image | Contains the image of the application | N/A | N/A |
| Text Input  “Enter your name ” | Allows the user to enter a login name | N/A | N/A |
| Text Input “Enter your password” | enter password | N/A | N/A |
| Button “Sign up” | For customers who register without an account | Click | Go to the Quản Lý Sinh Viên page, switch to Subscriptions  Page |

## Feature/Component #2: Quản lý screens

### User Interfaces



*Image2: Quản lý screens*

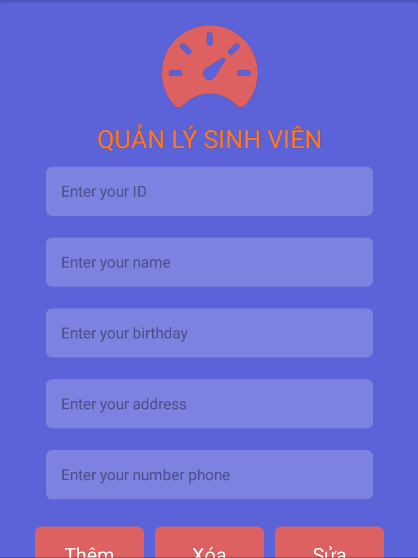
### Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Action** | **Response** |
| Registration page | Include:   * 1 Image * 3 Button * Text | N/A | N/A |
| Button “Quản lý sinh viên” | Registration is successful if the correct information is registered | Click | When successful registration will be announced "Successful registration" |
| Button “Quản lý lớp” | If you do not want to continue signing up for an account | Click | Go to the quản lý lớp page |
| Button “Đăng xuất” | If you do not want to continue signing up for an account | Click | Go back to the home page |
| Text “Quản Lý Sinh Viên“ | Name Sign up | N/A | N/A |
| Image | Contains the image of the application | N/A | N/A |

## 

## Feature/Component #3: Quan Ly Sinh Vien Page

### Quan Ly Sinh Vien Screens



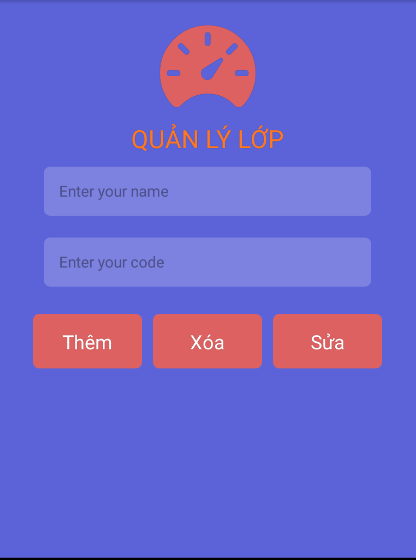
*Image3 : Quản lý sinh viên screens*

### Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Action** | **Response** |
| Home page | Include:   * 1 Image * List view * 5 Text Input * 3 Button | N/A | N/A |
| Image | Contains the image of the application | N/A | N/A |
| Text Input  “Enter your ID ” | Allows the user to enter a ID | N/A | N/A |
| Text Input “Enter your name” | Allows the user to enter a name | N/A | N/A |
| Text Input  “Enter your birthday ” | Allows the user to enter a birthday | N/A | N/A |
| Text Input “Enter your Address” | Allows the user to enter a address | N/A | N/A |
| Text Input  “Enter your number phone ” | Allows the user to enter a number phone | N/A | N/A |

## Feature/Component #4: Quan Ly Lop page

### User Interfaces



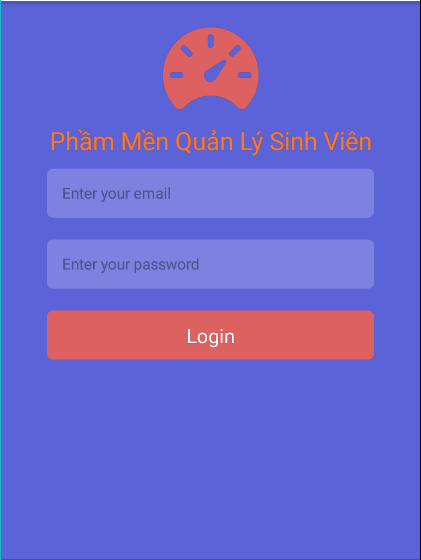
*Image4: Quan ly lop page*

### Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Description** | **Action** | **Response** |
| login page | Include:   * 3 Button * 2 Text Input * Text * 1 Image | N/A | N/A |
| Button “Thêm” | Thêm in successfully if correct account information | Click | Thêm in successfully |
| Image | Contains the image of the application | N/A | N/A |
| Text Input  “Enter your name ” | Allows the user to enter a login name | N/A | N/A |
| Text Input “Enter your code” | enter code | N/A | N/A |
| Button “Sửa” | For customers who register edit sinhvien | Click | Click |
| Button “Xóa” | For customers who register delete sinhvien | Click | Click |

## 2.2 Thiết kế hệ thống

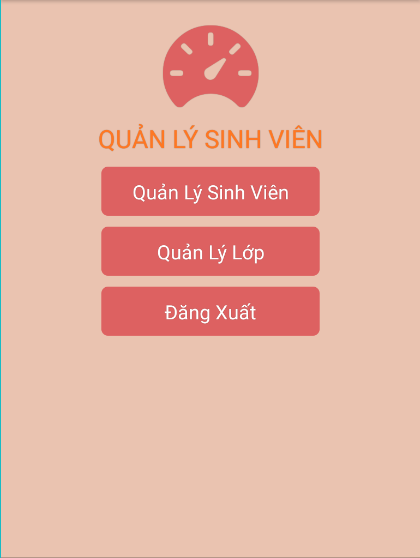
## Screen Login



**Image 1: screen Login**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Component Name** | **Required** | **Format** | **Reference** | **Note** |
|  | 1 field for input email/password |  |  | * Use Text : * Text : Email/ Username * Use TextInput : * fontSize: 20 * Color :#FFFFFF * Backgroudcolor:# 1c313a |  |
|  | 1 field for input password |  |  | * Use Text : * Text : Password * Use TextInput : * fontSize: 20   Color :#FFFFFF |  |
|  | 1 button for login |  |  | * Use Buttom : * Text : Login * fontSize :15 * Color :#FFFFFF * Background :#FF0000   textAlign="center". |  |

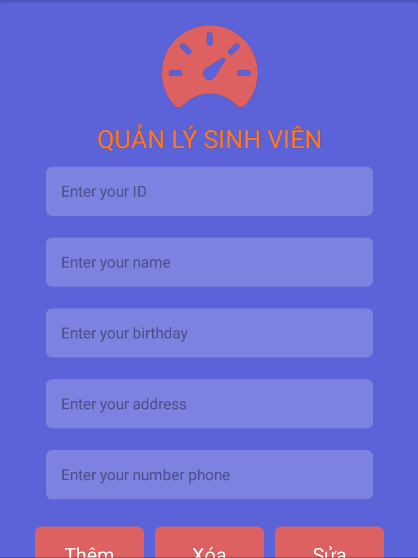
## Screen Quản lý



**Image 2: Screen quản lý**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Component Name** | **Required** | **Format** | **Reference** | **Note** |
|  | 1 button |  |  | * Use Buttom : * Text : Xóa * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |
|  | 1 button |  |  | * Use Buttom : * Text : Sửa * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |
|  | 1 button |  |  | * Use Buttom : * Text : Thêm * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |

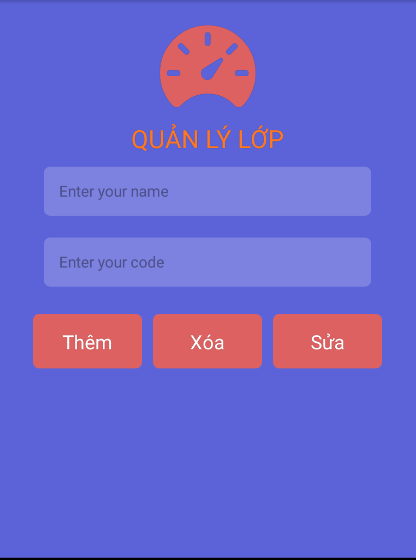
## Quản lý sinh viên screen



**Image 3: Quản lý sinh viên screen**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Component Name** | **Required** | **Format** | **Reference** | **Note** |
|  | 1 button |  |  | * Use Buttom : * Text : Xóa * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |
|  | 1 button |  |  | * Use Buttom : * Text : Thêm * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |
|  | 1 field for input ID |  |  | * Use Text : * Text : ID * Use TextInput : * fontSize: 20 * Color :#FFFFFF * Backgroudcolor:# 1c313a |  |
|  | 1 field for input Name |  |  | * Use Text : * Text : Name1 * Use TextInput : * fontSize: 20   Color :#FFFFFF |  |
|  | 1 field for input Birthday |  |  | * Use Text : * Text : Birthday * Use TextInput : * fontSize: 20 * Color :#FFFFFF * Backgroudcolor:# 1c313a |  |
|  | 1 field for input address |  |  | * Use Text : * Text : address * Use TextInput : * fontSize: 20   Color :#FFFFFF |  |
|  | 1 field for input number phone |  |  | * Use Text : * Text : number phone * Use TextInput : * fontSize: 20   Color :#FFFFFF |  |
|  | 1 button |  |  | * Use Buttom : * Text : Sửa * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |

## Quản Lý lớp screen

 **Image 4: Quản lý lớp page**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Component Name** | **Required** | **Format** | **Reference** | **Note** |
|  | 1 field for input name |  |  | * Use Text : * Text : name * Use TextInput : * fontSize: 20   Color :#FFFFFF |  |
|  | 1 field for input code |  |  | * Use Text : * Text : code phone * Use TextInput : * fontSize: 20   Color :#FFFFFF |  |
|  | 1 button |  |  | * Use Buttom : * Text : Thêm * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |
|  | 1 button |  |  | * Use Buttom : * Text : Xóa * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |
|  | 1 button |  |  | * Use Buttom : * Text : Sửa * fontSize :15 * Color :#FFFFFF * Background :#FF0000 * textAlign="center". |  |

# CHƯƠNG 3. CÀI ĐẶT VÀ KIỂM THỬ

## 3.1 Cài đặt

Mô tả cách thức hiện thực hoá cho từng Activity, Service… của hệ thống

## 3.2 Kiểm thử

Các Test case của hệ thống là gì?

Kết quả kiểm thử ra sao?

# CHƯƠNG 4. KẾT QUẢ ĐẠT ĐƯỢC

## 4.1 Kết quả đạt được

Mô tả các kết quả khi chạy ứng dụng (chụp các màn hình khi chương trình chạy, mô tả cách chạy, kết quả ra sao…

## 4.2 Các kết luận và kiến nghị

- Những điểm đã làm được

* Biết tạo 1 app bán hàng đơn giản bằng react native
* Biết cách sử dụng webservice

- Những điểm chưa làm được

* Ở trang Home, tụi em chưa làm được chức năng lọc sản phẩm và nút tìm kiếm
* Xử lý nút “Add to cart”, vì khi có giỏ hàng thì app sẽ hoàn chỉnh hơn.
* Xử lý chức năng thêm xóa sản phẩm đã chọn của khách hàng trong trang user.

- Các chức năng bổ sung nếu có thêm thời gian…

Nếu có thời gian, tụi em sẽ xử lý 3 việc như sau để app có thể hoàn chỉnh hơn:

* Xử lý chức năng lọc sản phẩm và làm thêm nút tìm kiếm ở trang Home
* Ở Trang Detail chúng em sẽ xử lý nút “Add to cart” để thực hiện chức năng mua sản phẩm
* Ở trang user, tui em sẽ thêm chức năng thêm và xóa các sản phẩm khách hàng đã chọn.

# PHỤ LỤC

# TÀI LIỆU THAM KHẢO