**VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY**  
**INTERNATIONAL UNIVERSITY**  
**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**



THESIS REPORT

**Nguyễn Thành Thiện**

**ITITIU14089**

**SPORT EVENTS MANAGEMENT MOBILE APP**

by

**NGUYEN THANH THIEN**

***Submitted to:*** School of Computer Science and Engineering - International University, VNU-HCM

APPROVED BY SUPERVISOR APPROVED BY: COMMITTEE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tran Thanh Tung, Ph.D. Assoc. Prof. Chair Tran Manh Ha,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Thesis committee**

**TABLE OF CONTENTS**

**LIST OF FIGURES**

**I. INTRODUCTION**

**1. Objective**

With the development of society, the application of modern technology is more and more popular in our world, especially of using phone in daily life. Today's phones can meet most user requirements, from keeping contacts, taking photos/videos to serve entertainment purpose such as music, games, etc.

One of the benefits of a modern phone (smartphone) is that users can select their favorite applications and use them independently as regular web applications.

Applications that are the most popular have basic requirements such as tracking the status of an in-app object, finding an object, controlling personal information from the application and the interaction between users and apps, etc.

With my product (called "Sport News"), you can use the functions needed for specific sport community events in your area.

**2. Problems and Solutions:**

*\* Problems:*

- With customer:

+ There are too less applications used to track community sports events in smartphone.

+ It is difficult to keep tracking of favorite kind of sports.

+ The event positioning system has not been applied.

+ Review tournaments with pictures is hard.

- With owner:

+ Upload event into phone application is limited.

+ Hard to manage the own events in smartphone.

*\* Solution:*

- Realtime Push Notifications about the event into customer’s phone.

- Capture their moment using to review the event.

- Tracking events by real time map and GPS.

- A web application for event’s hosts.

**3. Technical**

***3.1/ React (React Native and ReactJs)***

React was introduced and developed by Facebook, the biggest community network in the world. React was created to optimize the DOM handling (by saving the state of an application and just change the DOM when state changed). By this way, it is really helpful with constant data.

Beside that, React Native is a framework that lets you build native iOS and Android apps with one unique language of Javascript. React Native provides a virtual DOM (Document Object Model) which is a generic representation of what your UI will look like independent of the two platforms. React Native then renders the native control based on your UI markup representation which then renders native controls. It is able to take native platform components (sliders, switches, labels, tab bars) and wrap them in React component counterparts.

***Why React Native***

**1. Saving time and reducing development costs.**

*\* Easy to learn*

- Getting started with React Native is easy - especially for JavaScript professionals. Just download the open-source code from Github and make sure you’re familiar with a few tools and constructs in the React Native library including NodeJS, the CSS Flexbox system, ECMAScript 6, and JSX.

### *\* Cross-platform*

- React Native was originally only developed for iOS support, but due to its success and popularity, Facebook decided to develop support for Android as well, so which translates to lower development costs.

*\* The ability to combine good features and build apps*

- Whereas native app development is usually associated with inefficiency, less developer productivity, and slower time to deployment, React Native brings the speed and agility of web app development to the hybrid space with native result (the combination between pros of Hybrid and Native apps).

- React Native provided "Hot Reloading" (the previous version is “Live Reloading”) that keeps the app running and to inject new versions of the files that you edited at runtime. By this, you don't lose any of your state which is especially useful if you are tweaking the UI. The result is times between you save a file and be able to get this feedback loop can be under 1 second, even as your app grows.

**2. Performance**

- Traditionally, “Hybrid” or “Mobile web” apps that work on both iOS and Android tends to go down for more complicated apps. Apps built through React Native are compiled into natively written code, so they not only work on both operating systems, but also function the same as a natively written app (**“learn once, write anywhere”**).

- React Native is focused solely on building a mobile UI, making it more like a JavaScript library than a framework. It is developed based on Facebook’s popular ReactJS UI library for web applications, so it brings all of ReactJS’s better app performance: DOM abstraction, and simplified programming methods to hybrid mobile development. The resulting UI is highly responsive and feels fluid (the app will have quicker load times than a typical hybrid app, and a smoother feel) thanks to asynchronous between Javascript and Native.

**3. Reusability**

- Reusability is key in React Native. That mean the building blocks are treated as native components that can be compiled directly into the native languages (Objective-C for iOS and Java for Android). This is a huge bonus for businesses that want to augment an existing app but don’t want to overhaul it by Incorporate React Native components into the app’s code.

- This is possible to exclude the WebView components of other Hybrid mobile apps because React Native’s building blocks are reusable “native components” that compile directly to native. For example, if you’re adding Google Maps functionality to your app, React Native lets you link the plugin with a native module without have to rely on a WebView for certain functions. Components you would use in platforms have counterparts right in React, so you’ll get a consistent look and feel.

**4. Maintainablity**

- React Native uses Javascript and this, coupled with the intuitive architecture of the framework itself, allows engineers to jump to and from each other’s projects on Native apps with relative ease.

- The typical workflow would consist of writing most of the code on React Native and then using the native languages to optimize certain elements of the app. Parts of a specific app written in React Native will have no trouble combining with parts written in the native languages for iOS or Android. It is completely up to the developer to build the app, it not only can the native languages of an OS be used to optimize an app, they can also be used to write parts of the app.

***3.2/ Firebase***

**4. Goal**

After developed the project, I want to gain the needs that:

- Having an application for user to track sports community events (in both client-server sides).

- Using React to build application (on both smartphone and web environment) with smoothly.

- Applying Firebase into future projects as a steady server to store and analyze data.

**II. SYSTEM ARCHITECTURE**

**1. Background**

***1.1/ Flux and Redux***

**FLUX:**

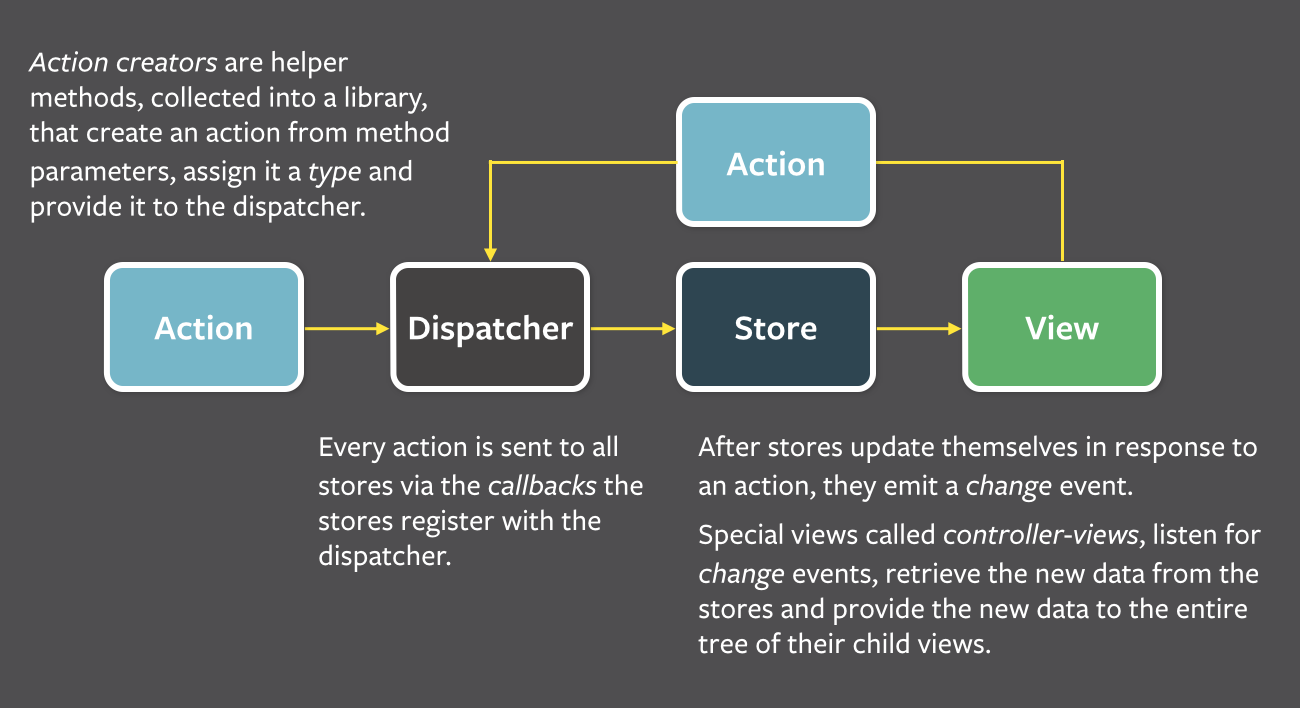
- Flux is an architecture for creating data layers in JavaScript applications. It was designed along with the React view library. It places a focus on creating explicit and understandable update paths for your application's data, which makes tracing changes during development simpler and makes bugs easier to track down and fix.

- Flux consists of the following basic components:

**+ Action:** This is the place to register the functions that will be called when View needed.

**+ Dispatcher:** Has the role of transmitting calls from Action to the Store. When an Action is called, the Dispatcher will broadcast an event to all Stores, with Action Type, or any other required data.

**+ Store:** The place to store data, is also the only place where you can add, edit or delete data. The store will listen to the events that come from the Action through the Dispatcher, check that the event is under its processing, and make the necessary data changes, corresponding to each event. After changing the data, Store will give another event to report its change.

**+ View:** Retrieves data from functions.

- React using **one-way data bindings**, because they think that two-way data bindings will become "When one object update, it many other objects will update, and then makes more updates". With one-way, when all the changes go to the Dispatcher, they can easily find where the change comes from, so the system will become "predictable".

- Flux follows the concept of **Unidirectional Data Flow** (UDF is to keep the data flow in the application moving in a single direction. When the data changes, this stream reboot from scratch) making it much easier to zero in of where the error lies. The data goes through a strict pipeline through your application:

+ Any changes, or Actions, **must go** through the Dispatcher!

+ Store is just only public getter, not public setter. Data change handlers can only be called inside the Store itself. This means:

+ You cannot change the data in the Store directly from the View.

+ You also can not change the data in one Store from another. As mentioned above, Store is not public setter. To do that, you must go through Action and Dispatcher. This makes the data inside your Store more manageable, you can easily debug where the change comes from when data changes.

**DIFFERENT OF FLUX WITH MVC**

**1. Easier to understand the data flow.**

- In the bidirectional data flow, you have the typical data flow   MVC. But when applications became more complex, the Controller takes the huge responsibility of maintaining both the application state and the data. Also, the cascading updates makes the app really difficult to understand and debug.

- With UDF, changes in the application view layer will trigger an action in the data layer. These changes will then be reflected in the View. The View does not directly affect application data.

**2. Reducing the role of Controller.**

- Dispatcher is not a controller, it does not contain business logic. It simply is a "coordinating center", which send an Action to every Store. In the MVC model, you can design any Controller as you can. But in Flux, there is only one Dispatcher and all Actions must go through it. The appearance of the Dispatcher is important because it ensures the design of UDF.

**3. Store can handle many objects.**

- In the MVC model, the Model usually manage a particular object. In opposition to the Model, the Store may not handle any data or handle multiple states of the application, or manage multiple objects at once.

**REDUX:**

- Redux is an implementation library of Flux with minimal API but completely predictable behavior, so it is possible to implement logging, hot reloading, time travel, universal apps, record and replay, without any buy-in from the developer.

- Redux can be used together with React, or with any other view library, and following 3 principles:

+ All application state is contained within a ***single store***, which is most often a JavaScript object.

+ The application's state is ***immutable***. This means that at no point should the object representing the state be modified in any way by any component.

+ All functions that compute the new state (called reducer functions) must be **pure functions**. Pure functions are functions that produce no side-effects and are deterministic - for a given set of inputs, the output will always be the same.

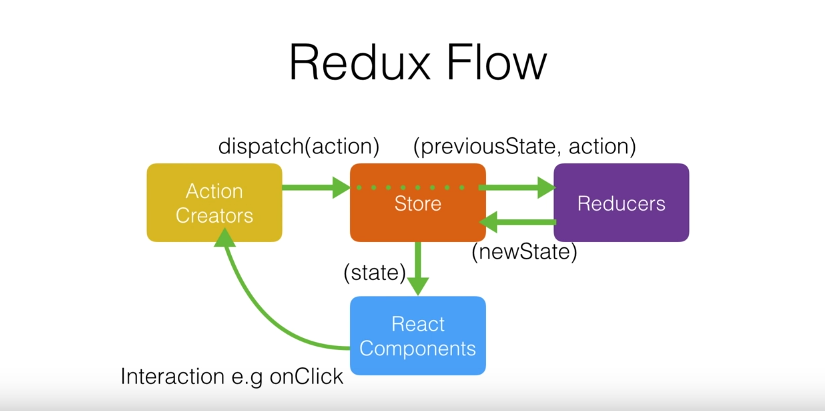
- General structure of Redux design pattern:

+ ***actions*:** Are payloads of information that send data from your application to your store. They are the only source of information for the store

+ ***reducers*:** Reducers specify how the application's state changes in response.

+ ***stores*:** The actions that represent the facts about “what happened” and the reducers that update the state according to those actions. The Store is the object that brings them together.

+ ***components:*** Folder contains the components that is only responsible for view and receive user interaction (How things look: markup, styles). These files read and change data from *props* => Dump components.

+ ***container:*** The components that are aware of Redux, Router, etc. They are more coupled to the app. They transmit the data to dump components with *props* (How things work: data fetching, state update) => Smart components.

***1.2/ Firebase - Cloud Function***

***1.3/ Firebase – Real time database***

***1.4/ Firebase – Authentication***

***1.5/ Expo***

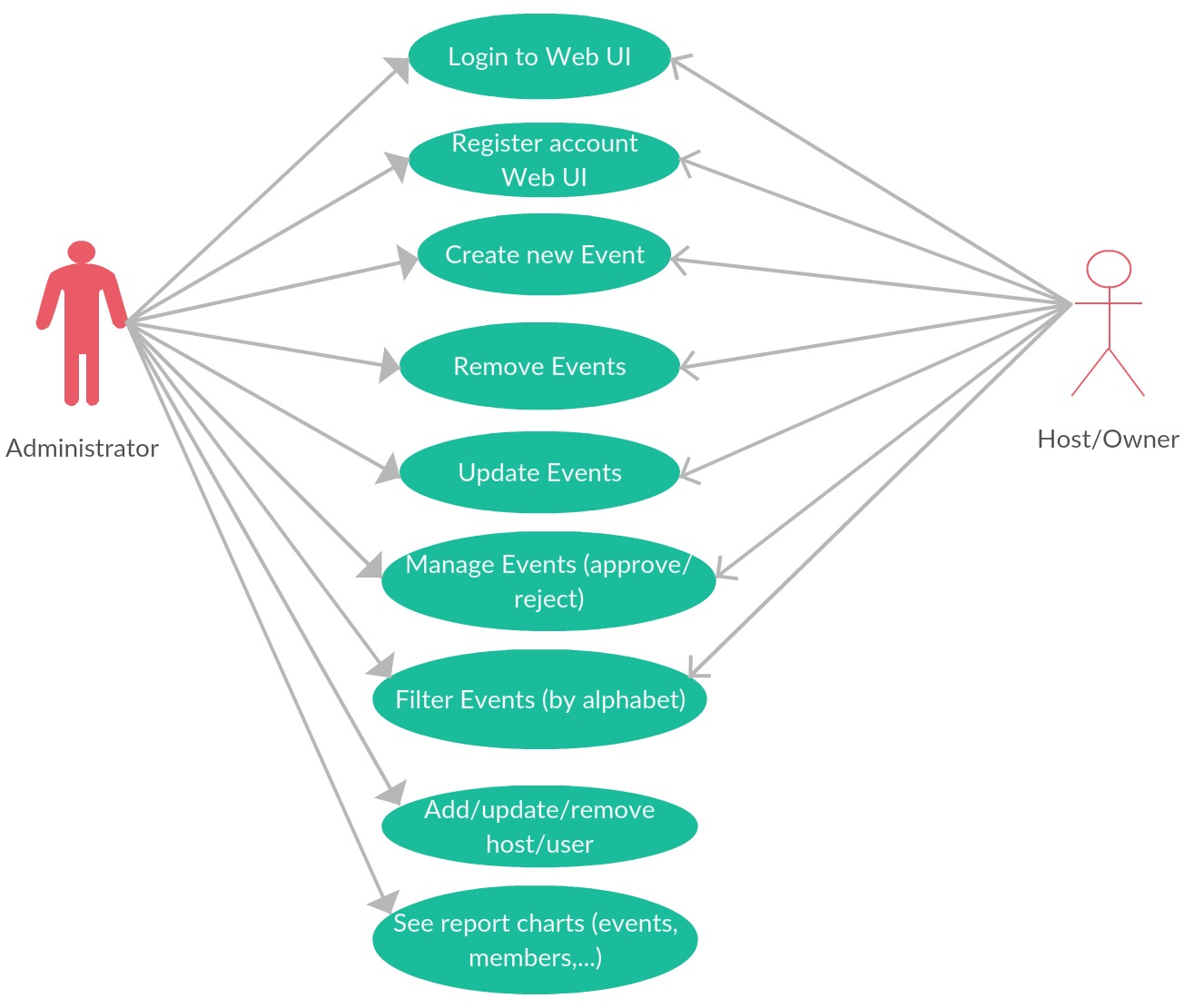
***1.6/ Open source***

**2. System requirements**

\* I will divide my group users into 2 groups:

**- User (visitor), member:** Customers using my app on smartphone.

**- Administrator, owner:** The event’s hosts using web app to manage their events.

***2.1/ Diagrams***

****

**III. IMPLEMENTATION**

**IV. CONCLUSION**

**REFERENCES**