Sean Grace Puy

react native

Informe creación AppGaztaroa

Tabla de contenido

[PRIMEROS PASOS EN REACT NATIVE 2](#_Toc101722066)

[COMPONENTES REACT NATIVE 2](#_Toc101722067)

[COMPONENTES FUNCIONALES EN REACT NATIVE 2](#_Toc101722068)

[STACK NAVIGATION 2](#_Toc101722069)

[DRAWER NAVIGATION 2](#_Toc101722070)

[COMPONENTES Y NAVEGACIÓN 2](#_Toc101722071)

[BOTONES E ICONOS 3](#_Toc101722072)

[SERVIDOR JSON 3](#_Toc101722073)

[REDUX Y THUNK 3](#_Toc101722074)

# PRIMEROS PASOS EN REACT NATIVE

En esta primera parte, se configura todo el entorno de trabajo. Algunas de las herramientas también han sido utilizadas en la primera parte de la asignatura (Github, Trello, NodeJS, NPM). A parte de estas, se van a emplear otras distintas como son Yarn que es otro gestor de paquetes tipo npm y Expo que es una herramienta que nos permite construir la aplicación y enviarla para probarla directamente en nuestro teléfono móvil.

# COMPONENTES REACT NATIVE

En esta parte del desarrollo, se comienzan a añadir componentes a nuestra aplicación. Antes de eso, se generan algunas carpetas para tenerla organizad. Se crea una carpeta componentes en la que se meterán los componentes principales de la aplicación y dos carpetas más. Una carpeta llamada imágenes en la que se guardaran las imágenes que utilizaran los distintos componentes y otra carpeta común en la que se colocaran los elementos comunes a los distintos componentes principales.

# COMPONENTES FUNCIONALES EN REACT NATIVE

En esta parte creamos un nuevo componente funcional y vemos como trabaja exactamente igual que en react como se vio en la primera parte de la asignatura.

# STACK NAVIGATION

En esta parte se añade la posibilidad de navegar en la aplicación haciendo uso de una pila de vistas. Para facilitar esto y que cada componente mantenga su estado vamos a convertir los componentes funcionales en componentes de clase. Una vez hecho esto, se implementa el stack navigation en campo base. Ahora, al realizar el navigate para acceder a otra ventana en la pila se le pueden pasar parámetros como variables y se pueden acceder a ellas desde route.params

# DRAWER NAVIGATION

Se incluye un navegador lateral a la aplicación. Tener en cuenta que para que funcione correctamente con el otro navegador se introduce uno dentro del otro de forma enlazada. Tambien tener en cuenta que la aplicación da unos warnings cuando existen distintas Screens con el mismo nombre y podria causar problemas.

# COMPONENTES Y NAVEGACIÓN

Para poder tener scroll en la pantalla de ‘Quienes Somos’ si utilizamos Scroll View nos da un warning de que el listview podría no funcionar adecuadamente. Para evitar esto, una opción es eliminar el scrollview y añadir en el listview el elemento ListHeaderComponent que permite añadir lo que quieras antes de la lista. El problema que he encontrado de utilizar esto, es que no he sabido como hacer que se rendericen los elementos dentro del card de actividades y recursos, mantener el card de historia y que se pueda hacer scroll.

Pese a tener el warning, parece que funcionaba bien por lo que lo he dejado con el scrollview.

# BOTONES E ICONOS

En esta parte se incluyen botones sin emplear el componente Button y además se incluyen iconos para los elementos del menú. Para que se comporte como botón un elemento Icon, basta con añadir la propiedad onclick y enlazarla con una función.

En esta sección también se muestra como crear el botón típico de menú que se muestra en la parte superior izquierda de la aplicación. Para ello, hay que crear un icono en el stack navigator y utilizar el toggle drawer para que despliegue el menú.

# SERVIDOR JSON

En este apartado se modifica la localización de los ficheros comunes (json e imágenes) que emplea la aplicación. En lugar de obtenerlos del almacenamiento local de la aplicación, se utilizar un servidor REST para servir los ficheros simulando un uso real. De esta manera, modificando estos ficheros que se encuentran en el servidor se puede cambiar el contenido que se visualiza en las aplicaciones sin tener que realizar modificaciones en ellas.

# REDUX Y THUNK

En este apartado se implementa redux para tener una implementación ordenada de la aplicación y sus estados. A continuación, se muestra un esquema de la estructura de este sistema. En este esquema faltaría el provider que engloba al store y hace de unión con los componentes y el middleware que estaría de intermediario entre el dispatcher y el store para poder funcionar de manera asíncrona. En la Figura 2, se muestra un esquema de como se ha quedado la implementación Redux en nuestra aplicación.

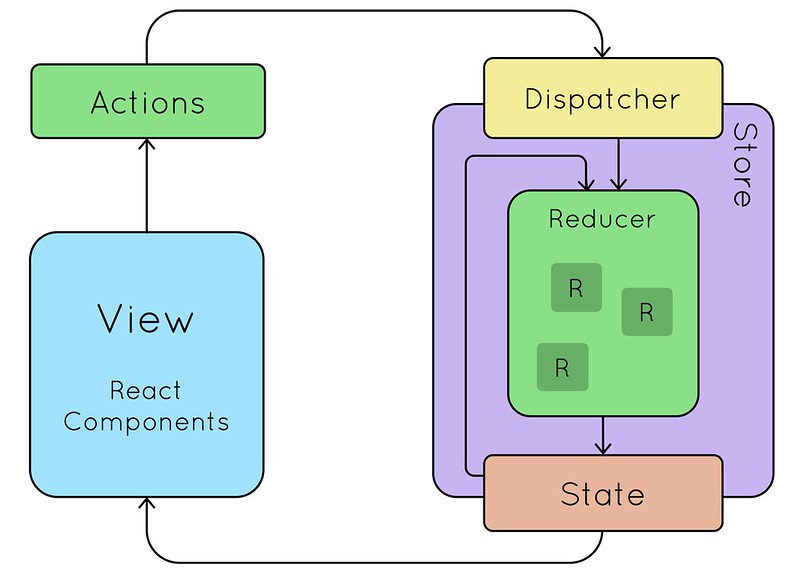


Figura . Estructura applicación redux



Figura . Estructura appGaztaroa

# ACTIVITY INDICATOR Y ADDFAVORITOS

En este ejercicio, se añade un activity indicator para que muestre el circulito típico de carga mientras el programa está realizando la carga de la información de la base de datos. Se implementa en varias zonas del programa. Para comprobar que esta funcionando correctamente, se puede cambiar la ip del servidor json que podemos encontrar en el archivo común.js. Si hacemos esto, veremos como nos aparece la pantalla de carga en las secciones en las que lo tengamos implementado.

Otra de los elementos que se implementa en esta parte es el botón de addfavoritos. Este botón ya estaba preparado pero no era funcional en el sentido de que al salir de la ventana se reseteaba la información. En esta parte se implementa en nuestra infraestructura redux y se deja listo para poder implementarlo con la base de datos

# EJERCICIO MODALS Y FORMULARIOS CON REDUX

En este ejercicio se implementa un modal con un formulario para poder añadir comentarios. Se divide en dos partes. Por un lado esta la parte visual y otra la parte de lo que ocurre por detrás con la implementación en la arquitectura redux y envio al servidor.

Como observación, al implementar el modal he tenido problemas por haber seleccionado la librería de manera automática con Visual Studio Code. En lugar de importarme la librería desde ‘react-native’ lo hacia desde ‘react-native-web’ que no daba problemas de compilación, pero si a nivel de funcionalidades ya que ni siquiera se mostraba.

Otro detalle que afecta a lo visual es que, si queremos modificar los estilos de los componentes ‘Button’, he necesitado implementarlo dentro de un View y aplicarle las propiedades de estilo desde ahí. Esto es necesario ya que el componente no tiene implementado la prop de style. Esto se puede ver en la Figura 3.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . Configurar estilos de un componente tipo Button

# FUNCIONALIDADES EXTRA

* Configurar backend firebase
  + API Rest
  + Imágenes
  + Upload: comentarios
  + Storage: Upload archivos GPS.
* UI y Capacidades nativas
  + Subir track GPS (admin)
  + Modificar/Añadir Imagen (admin)
* Autentificacion
  + Login para enviar comentarios
  + Si es usuario tipo admin funcionalidades extra(añadir track, modificar imagen)

/\*fetch(baseUrl + 'comentarios')

.then(response => response.json())

.then(function(data) {

for (let index = 0; index < data.length; index++) {

console.log(data[index]);

console.log(index);

setDoc(doc(db, "comentarios", data[index].id+'\_'+data[index].autor), {

autor: data[index].autor,

comentario: data[index].comentario,

dia: data[index].dia,

excursionId: data[index].excursionId,

id: data[index].id,

valoracion: data[index].valoracion

});

}

})\*/

fetch(baseUrl + 'actividades')

.then(response => response.json())

.then(function(data) {

for (let index = 0; index < data.length; index++) {

console.log(data[index]);

console.log(index);

setDoc(doc(db, "actividades", data[index].id+'\_'+data[index].nombre), {

id: data[index].id,

nombre: data[index].nombre,

imagen: data[index].imagen,

destacado: data[index].destacado,

descripcion: data[index].descripcion

});

}

})

fetch(baseUrl + 'excursiones')

.then(response => response.json())

.then(function(data) {

for (let index = 0; index < data.length; index++) {

console.log(data[index]);

console.log(index);

setDoc(doc(db, "excursiones", data[index].id+'\_'+data[index].nombre), {

id: data[index].id,

nombre: data[index].nombre,

imagen: data[index].imagen,

destacado: data[index].destacado,

descripcion: data[index].descripcion

});

}

})

fetch(baseUrl + 'cabeceras')

.then(response => response.json())

.then(function(data) {

for (let index = 0; index < data.length; index++) {

console.log(data[index]);

console.log(index);

setDoc(doc(db, "cabeceras", data[index].id+'\_'+data[index].nombre), {

id: data[index].id,

nombre: data[index].nombre,

imagen: data[index].imagen,

destacado: data[index].destacado,

descripcion: data[index].descripcion

});

}

})