React Native Con TS

Reforzamiento React

Type Script:

TS es un lenguaje de tipado extricto basado en javascript, para reac se puede usar JS o TS, sin embargo por calidad de código es preferible usar TS

Iniciado en react con TS

para iniciar un proyecto en react se puede ir a la siguiente seccion de la documentacion oficial de react https://create-react-app.dev/docs/getting-started/ y elegir el comando que mas se ajuste bien sea con JS o con TS.

para este caso el comando con TS es: `npx create-react-app “project-name --template typescript`

recordando que si se desean dejar espacios en el nombre del proyecto se debe usar - en vez del espacio normal

una vez inicializado el proyecto desde la terminal lo podemos lanzar con el `npm start`

Tipos de datos:

let → define una variable

const → Define una constante

TS puede inferir el tipo de dato pero algo novedoso es que puede ser de varios tipos ejemplo:

let nombre: string | number = ‘nombre tal’

nombre = 1234 y esto no generaria error

JSON.stringify(object) → Transforma un objeto a su interpretación JSON

la estructura de dato sea class o type si implementa una interface debe contener todo tal cual esta la interfaz, y se implementa como si fuera un protocolo en swift

Los nombres de interface se deben realizar en mayuscula la primera

Archivo de parctica: https://github.com/IvanCardona01/react-practice

React Native CLI

Documentacion como empezar con react native:
https://reactnative.dev/docs/environment-setup

Iniciar proyecto react native con TS

```
npx react-native init AwesomeTSProject --template react-native-template-typescript
```

Correr proyecto de react native en emulador

```
npx react-native start
```

```
npx react-native run-android
```

Hooks:

useState ⇒

Sirve para guardar variables que necesitemos modificar o representar estados con ella:

Sintaxis para Ts ⇒ const [stateName, setStateName] = useState(initValue);

stateName es la variable, se modifica con su funcion setStateName y se inicializa el useState con un valor inicial

useEffect ⇒

Sirve para realizar una o varias acciones cuando ocurra un cambio en el sistema y a la cual se le puede especificar cuando debe efectuar la accion:

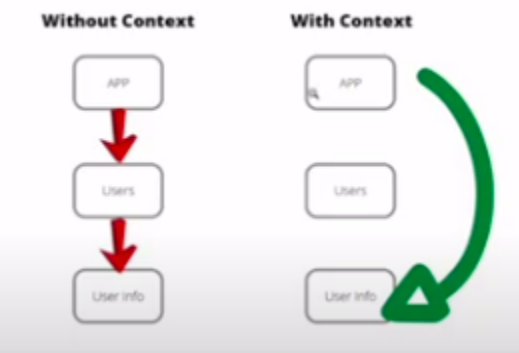
Sintaxis para Ts:

useEffect(( ) ⇒ {

//Action ⇒ Aqui se especifica la accion

}, [variable]) //Si se especifica una variable o componente aqui esta accion se ejecutara siempre que esta variable tenga algun cambio, si no se especifica alguna variable esta accion solo se realizara la primera vez que inicie el sistema

useContext ⇒



El useContext nos permite darle ciertas propiedades o datos a un componente padre donde todos sus hijos podran acceder a ella sin tener que pasar componente a componente un dato, estas propiedades a la que los componentes hijos puede acceder se definen en las props del componente padre, y puede ser tambien funciones que desde cualquier hijo pueda modificar algun estado del sistema siendo persistentes los datos

sintaxis ejemplo:

```
import React from "react";

//Definir como luce la informacion que tendra en context
import { createContext } from "react";

export interface AuthState {
    isLoggedIn: boolean;
   name?: string;
   favoriteIcon?: string;
}

//Estado inicial
export const authInitialState: AuthState = {
    isLoggedIn: false,
```

```
}

//Decir a React como luce y que expone el context
export interface AuthContextProps {
    authState: AuthState;
    singIn: () => void
}

export const AuthContext = createContext( {} as AuthContextProps);

//Componente provedor de estado
export const AuthProvider = ({ children }: any) => {
    return(
    <AuthContext.Provider value={{ authState: authInitialState, singIn: () => {} }}>
        { children }
    </AuthContext.Provider>
    )
}

En la clase hijo que va a utilizar los datos del useContext
const { authState } = useContext(AuthContext)
```

useRef ⇒

Hace una referencia a un componente del sistema, por ejemplo sirve para editar componentes sea en estado o diseño, ya por este podemos tomar sus propiedades, sea un campo te texto tomar su valor, o un view para poder editar en alguna funcion que llama algun boton su diseño

useReducer ⇒

Funciona de una forma muy similar al useState pero este hook se implementa cuando se usa una lógica mas compleja ya que permite no solo cambiar el estado del store que guarda si no que puede implementar un proceso de validacion complejo para saber que el directamente lo que va a guardar ejemplo:

sintaxis Ts:

cosnt [state, dispatch] = useReducer(reduce, initialState);

state ⇒ Es la variable o esta que manejara el useReducer

dispactch ⇒ Es la funcion que dispara el reduce para cambiar el estado

reduce ⇒ Es la funcion que cambia el estado y donde se concentra mayor parte de la logica

initialState ⇒ Estado inicial

ejemplo practico:

```
const initialState = "undefine"
const reducer = (state: string, age: number): string => {
    if (age >= 18){
        return 'is adult'
    }else {
        return 'is not adult'
    }
}

const [state, dispatch] = useReducer(reducer, initialState);

<Button title='Reducer' onPress={() => dispatch(12)} />
<Text style={ {
       marginVertical: 20,
       fontSize: 15,
       color: 'black',}}>Estado del reducer: {state} </Text>
```

useMemo ⇒

Se utiliza para guardar resultados cuya operacion consume muchos recursos del sistema o simplemente no tiene una gran variacion, este hook guarda el resultado de dicha funcion y no la vuelve a ejecutar hasta que algun estado cambie.

Ejemplo:

```
const calculateMultiply = (number1: number , number2: number ) ⇒ {
    return number1 * number2;
}
cosnt change: boolean = false
const memorizedState = useMemo(() ⇒ calculateMultiply(number1, number2),[ change ])
```

lo que hara en este caso el useMemo es memorizar el resultado y no volver a realizar esta funcion a menos que la variable change cambie de estado. sirve mucho para la optimización de recursos

useCallback ⇒

Este hook es muy similar al useMemo solo que useMemo solo memoriza un estado o valor, en cambio el useCallback puede memorizar una funcion con el obetivo de hacer la aplicación mucho mas optima consumiendo menos recursos.

Ejemplo

```
const [count, setCount] = useState( 0 );
```

```
const printCount ⇒ ( ( ) ⇒ {
    alert(count)
```

`}, [  ] )` //Tambien se puede definir en este campo que se repida cuando alla un cambio

useLayaoutEffect ⇒

Funciona de una forma muy similiar al userEffect solo que este es avecese mas utilizado para consultas asincronas debido a que este hook no permite que la app continue si no ha terminao de hacer la consulta o lo que sea que tena y tambien se puede indicar cuando volver a repetir

Bloques de vista principales en react native

Box Object Model

Alto, Ancho, Margen, padding, borde

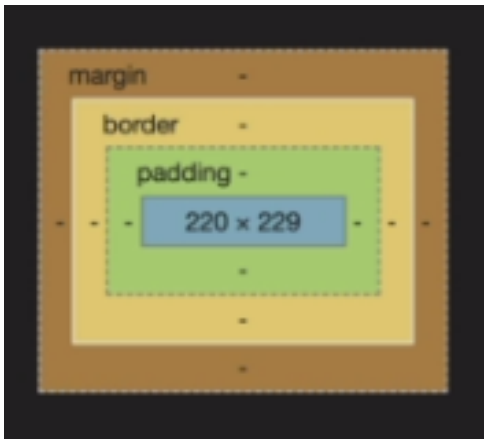
Position

absoluta, relativa, top, left, right, bottom

Flex box

Direccion, ubicacion, alineamiento, estirar, encoger, proporcionales

Box Object Model



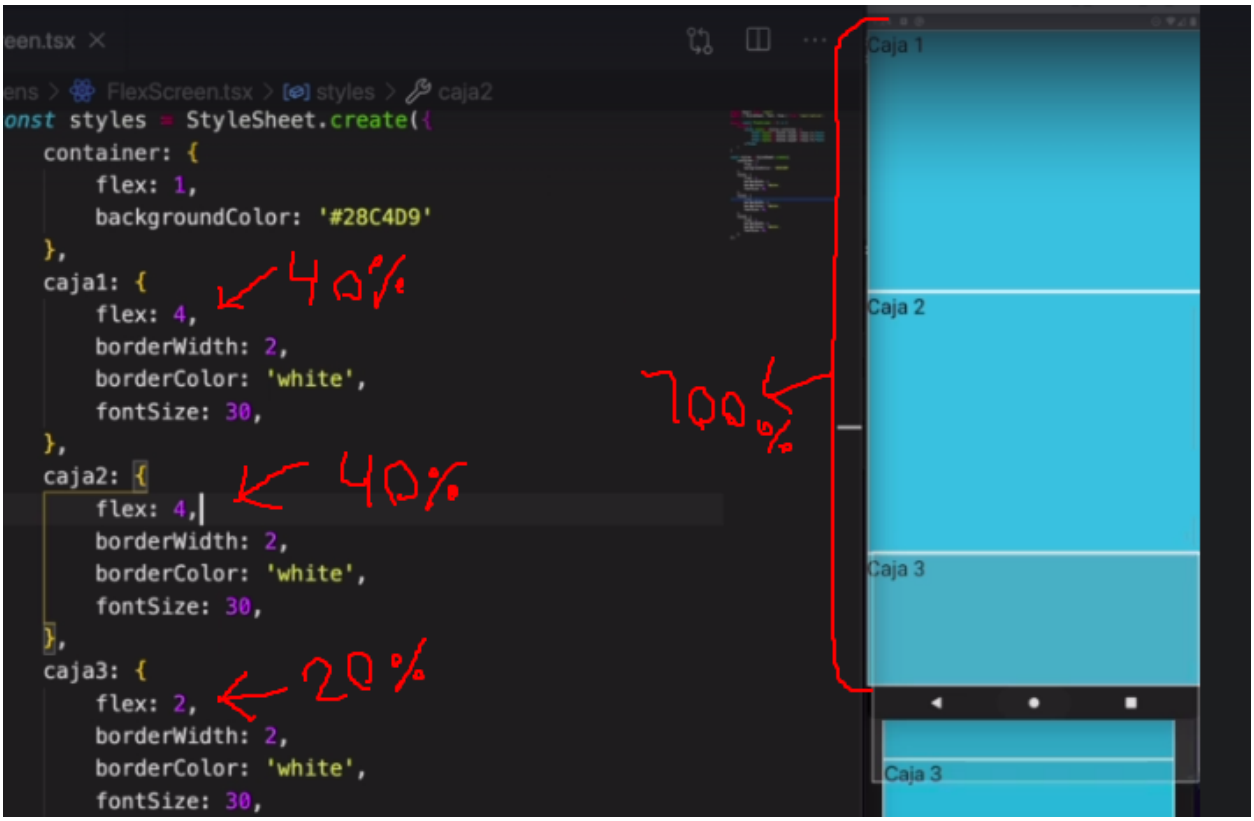
Como obtener la dimension de la pantalla?

```
const { width, height } = Dimensions.get('window');
```

Flex Box

Nos ayuda a mantener una estructura consistente entre diferentes pantallas

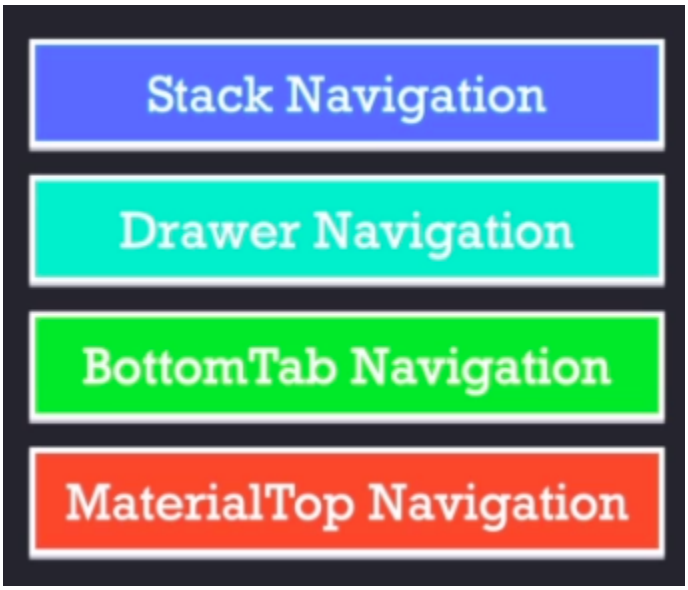
La propiedad flex se puede interpretar de la siguiente manera



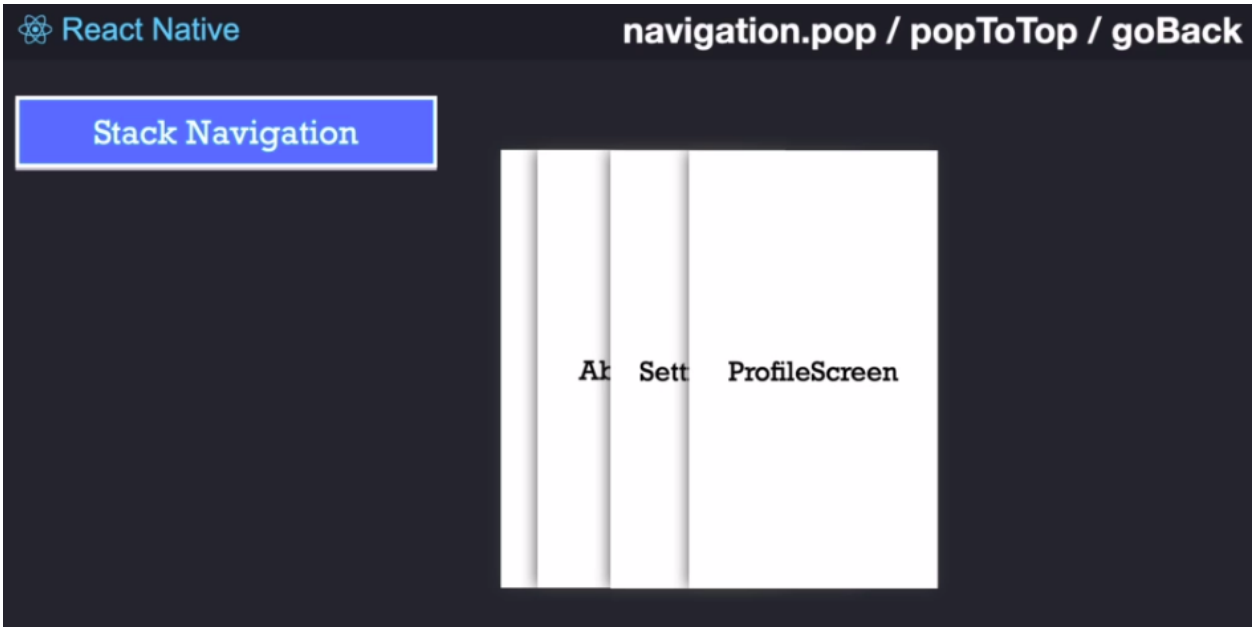
Flex Wrap

Nos permite darle tratamiento a los componentes hijos que superan el tamaño de su padre

Navegacion en React Native



Stack Navigation



Es sobre poner pantallas mientras las de atras se siguen ejecutando, la primera base es el Top de screen.

navigation.pop ⇒ Destruye la ultima pantalla lanzada (la mas sercana al usuario)

goBack ⇒ Vuelve una pantalla atras

popToTop ⇒ Vuelve a la pantalla principal

Para usar el Stack navigation es necesario intalar unas dependencias:

para hacerlo solo debemos seguir los pasos de la documentación oficial

https://reactnavigation.org/docs/getting-started

En esta documentación basicamente lo que nos dice es que debemos instalar lo siguiente

```
npm install @react-navigation/stack
```

Y en todo proyecto con navegacion es indispensable colocar los siguientes datos en el MainActivity.java

```
import 'react-native-gesture-handler';
```

Forma correcta para pasar datos entre ventanas:

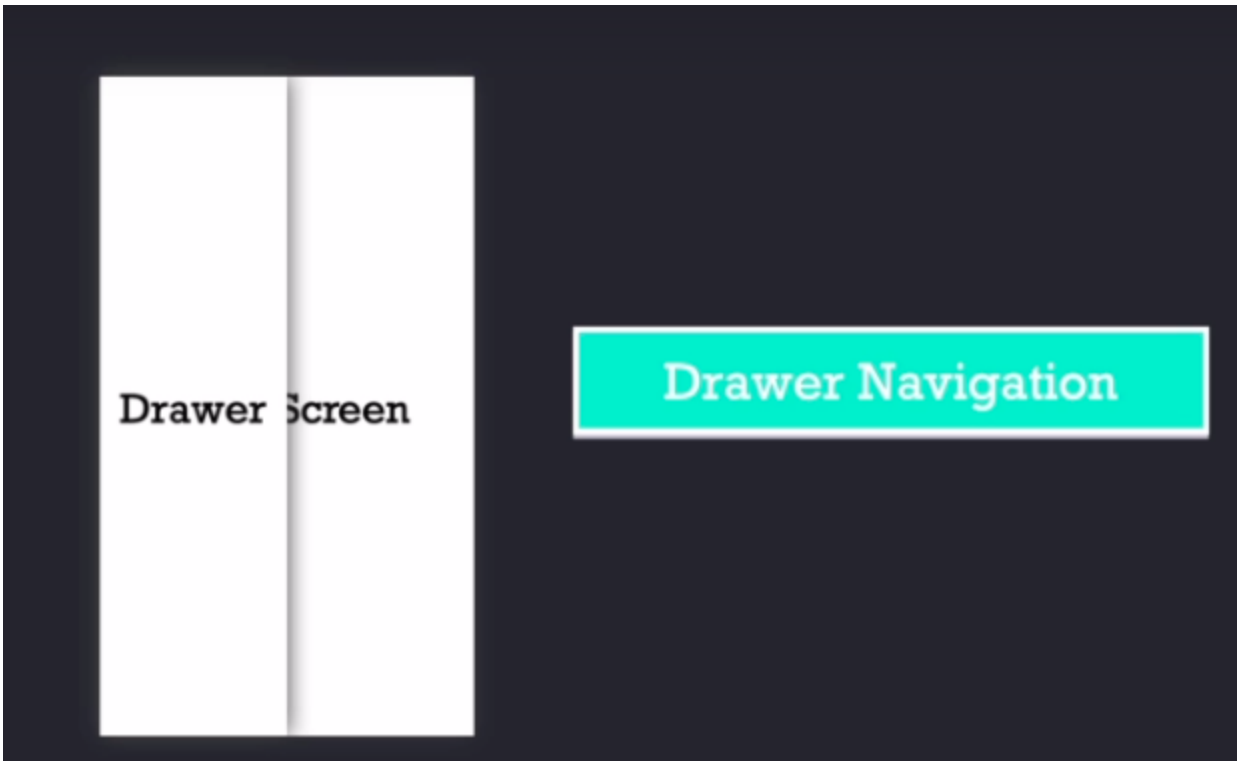
StackNavigator ⇒

```tsx
import { createStackNavigator } from '@react-navigation/stack';
import { Pagina1Screen } from '../screens/Pagina1Screen';
import { Pagina2Screen } from '../screens/Pagina2Screen';
import { Pagina3Screen } from '../screens/Pagina3Screen';
import { PersonaScreen } from '../screens/PersonaScreen';

export type RootStackParams = {
  Pagina1Screen: undefined,
  Pagina2Screen: undefined,
  Pagina3Screen: undefined,
  PersonaScreen: { id: number, nombre: string },
}


const Stack = createStackNavigator<RootStackParams>();
```

PersonaScreen ⇒

```tsx
import { RootStackParams } from '../navigator/StackNavigator';
import { styles } from '../theme/appTheme';

// interface RouterParams {
//     id: number;
//     nombre: string
// }

interface Props extends StackScreenProps<RootStackParams, 'PersonaScreen'>{};

export const PersonaScreen = ( { route, navigation }: Props ) => {

    // const params = route.params as RouterParams;
    const params = route.params;

    useEffect( () => {

        navigation.setOptions({
            title: params!.nombre
        })

    },[])
```

Drawer Navigation



Instalamos los ficheros restantes para el Drawer Navigation

```
npm install @react-navigation/drawer
```

Y para poder empezar tambien debemos intalar la siguiente dependencia:

npm install

Sintaxis del navigator

```
import { createDrawerNavigator } from '@react-navigation/drawer';

const Drawer = createDrawerNavigator();

function MyDrawer() {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Feed" component={Feed} />
      <Drawer.Screen name="Article" component={Article} />
    </Drawer.Navigator>
  );
}
```

Recomendacion al crear un proyecto con el drawer navigation u otras animaciones:

Pues he tenido que crear otro proyecto desde cero pero con los paquetes de las sección 7 actualizados hasta la fecha (Por el momento voy en el video 109):

```
npx react-native init NavegacionAppEnero2022 --template react-native-template-typescript
npm install @react-navigation/native
npm install react-native-screens react-native-safe-area-context
npm install @react-navigation/stack
npm install react-native-gesture-handler react-native-reanimated
npm install @react-native-masked-view/masked-view
npm install @react-navigation/drawer
```

Configurar reanimated y otras cosillas:

Para reanimated:

Add Reanimated's babel plugin to your `babel.config.js`:

```
module.exports = {
   presets: ['module:metro-react-native-babel-preset'],
   plugins: ['react-native-reanimated/plugin',],
};
```

Turn on Hermes engine by editing `android/app/build.gradle:`

```
project.ext.react = [
   enableHermes: true,
]
```

Edit `MainApplication.java` file which is located in `android/app/src/main/java/<your package name>/ MainApplication.java`.

Plug Reanimated in `MainApplication.java:`

```
import com.facebook.react.bridge.JSIModulePackage; // <- add
import com.swmansion.reanimated.ReanimatedJSIModulePackage; // <- add
```

pegar debajo del

```
@Override
protected String getJSMainModuleName() {
   return "index";
}
```

Lo siguiente:

```
@Override
protected JSIModulePackage getJSIModulePackage() {
   return new ReanimatedJSIModulePackage(); // <- add
}
```

Correr esto para limpiar cache:

```
npx react-native start --reset-cache
```

luego control c y arrancar de nuevo con:

```
npx react-native run-android
```

Más info: https://docs.swmansion.com/react-native-reanimated/docs/fundamentals/installation

Para native-screens (Un plus por si acaso que vi en la documentación oficial):

`react-native-screens` package requires one additional configuration step to properly work on Android devices. Edit `MainActivity.java` file which is located in `android/app/src/main/java/<your package name>/MainActivity.java`.

Add the following code to the body of `MainActivity` class:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(null);
}
```

and make sure to add an import statement at the top of this file:

```
import android.os.Bundle;
```

Si reanimated sigue sin funcionar entonces:

funciona pero con warnings:

```
npm remove react-native-reanimated
npm install react-native-reanimated@2.2.4
```

Como generar build:

https://www.skyzer.com.co/como-puedo-generar-un-apk-en-mi-proyecto-de-react-native/