

Ejercicio 1: Primeros pasos en React Native

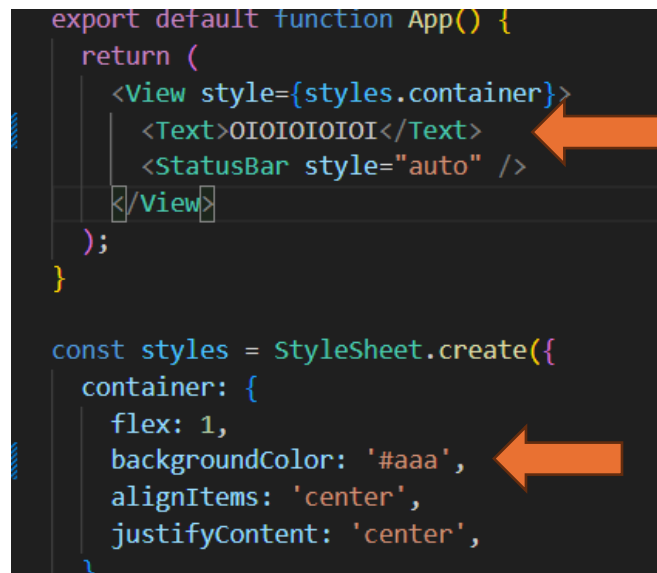
Tiempo invertido en el ejercicio: 1 h, 15 min

Pasos realizados:

- Se han actualizado los programas de Git, NodeJS, NPM y se ha instalado Expo
- Se ha creado el proyecto de Gaztaroa con el comando `npx create-expo-app`
- Se ha lanzado el proyecto de Gaztaroa con el comando `npx expo start`
- Se ha accedido desde el teléfono móvil a la aplicación desplegada, a través de la aplicación Expo Go
- Se han creado los repositorios local y en GitHub, y se han sincronizado
- Se ha hecho un primer commit con el proyecto por defecto
- Se han cambiado los valores del color de fondo y el texto desplegado en *App.js*
- Se ha hecho un segundo commit "*Primeros pasos en React Native*" con los cambios y con esta explicación

Notas importantes:

- Con Expo, se pueden realizar cambios en el código de la aplicación y observarlos *en vivo* en el teléfono
- Se puede cambiar el texto mostrado y el color de fondo a través del componente *Text* y el estilo *container*, como se muestra en la imagen



```
export default function App() {
  return (
    <View style={styles.container}>
      <Text>OI OI OI OI OI</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#aaa',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Ejercicio 2: Componentes React Native

Tiempo invertido en el ejercicio: 30 min

Pasos realizados:

- Se han incluido las carpetas de imágenes y el archivo *excursiones.js* al proyecto tal y como se indica en el guión
- Se han creado los componentes *Campobase* y *Calendario*, definidos en los archivos *CampobaseComponent.js* y *CalendarioComponent.js*

Notas importantes:

- Aquí se ha visto cómo se crean listas de elementos, a través del elemento *FlatList*
- El *Safe Area Context* tiene una función equivalente a los contextos en React normal

Ejercicio 3: Componentes funcionales React Native

Tiempo invertido en el ejercicio: 30 min

Pasos realizados:

- Se ha creado el componente *DetalleExcursion*, definido en el archivo *DetalleExcursionComponent.js*

Notas importantes:

- Se han introducido funcionalidades de botones a través de la función *onPress*
- Se ha utilizado el componente *Card* para mostrar información de una excursión en concreto

Ejercicio 4: Stack Navigation

Tiempo invertido en el ejercicio: 10 min

Pasos realizados:

- Se han instalado las librerías necesarias para hacer la navegación a través de las diferentes vistas
- Se han descargados los archivos modificados y se han implementado en el código, actualizando el programa con las modificaciones sugeridas por el guión

Respuesta a las preguntas:

- El *StackNavigator* es según la documentación, un componente que gestiona el árbol de navegación y guarda el estado de navegación. En su interior, podrá haber un componente creado con la función *createNativeStackNavigator()*
- El componente creado con la función *createNativeStackNavigator()* tiene dos propiedades: el *Navigator* y las *Screens*. El *Navigator* contiene *Screens* como componentes hijas para poder navegar entre ellas
- Las *Screens* son las diferentes pantallas que se pueden acceder dentro de la aplicación, a las que se les puede pasar el componente que renderiza el contenido correspondiente
- Para pasar el identificador de la excursión al componente *DetalleExcursion*, el *CalendarioComponent* utiliza la línea:
`onPress={() => navigate('DetalleExcursion', { excursionId: item.id })}`
Donde *excursionId* se recibe en *DetalleExcursion* a través de la variable *this.props.route.params*, donde se recibe el JSON argumento de la función *navigate*
- La función *navigate* se recibe en el calendario como *this.props.navigation*, mientras que los parámetros de ruta se reciben en el detalle como *this.props.route.params*

Notas importantes:

- Se ha visto cómo hacer navegación entre vistas con el componente *StackNavigator*