

Ejercicio 1: Primeros pasos en React Native

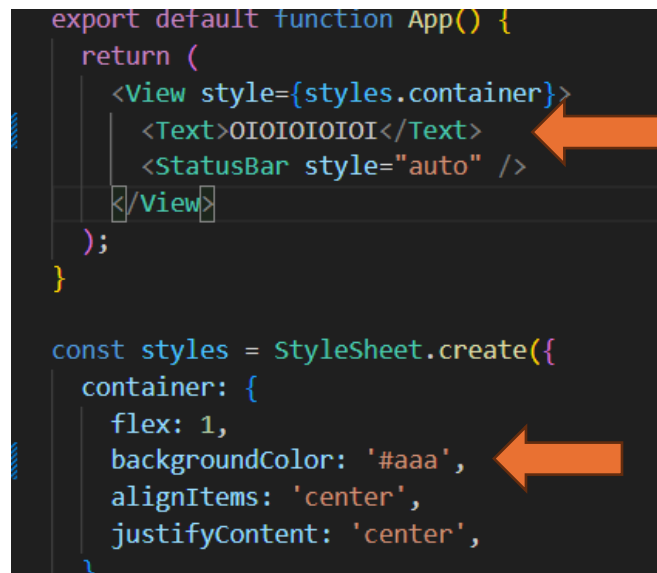
Tiempo invertido en el ejercicio: 1 h, 15 min

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*

Ejercicio 5: Drawer Navigation

Tiempo invertido en el ejercicio: 30 min

Pasos realizados:

- Se han instalado las librerías necesarias para hacer la navegación con *drawers*
- Se han descargados los archivos de *HomeComponent* y las listas de actividades y cabeceras
- Se ha modificado el componente *CampobaseComponent* para permitir la navegación con *drawers*:
 - o Se ha añadido otro navegador de *Stacks*, *HomeNavigator*
 - o Se ha creado un nuevo navegador, *DrawerNavigator*

Notas importantes:

- Se ha visto cómo hacer navegación entre vistas con el componente *DrawerNavigator*, de una manera extremadamente similar a *StackNavigator*
- Para forzar actualizaciones de la aplicación en el móvil, usar la opción de *reload app* de la consola de Expo

Ejercicio 6: Ejercicio Componentes y Navegación

Tiempo invertido en el ejercicio: 3 h, 30 min

Pasos realizados:

- Se han creado los componentes de clase *ContactoComponent* y *QuienesSomosComponent*
- Se han creado los componentes funcionales *RenderItem* e *Historia*
- Se ha actualizado el navegador *DrawerNavegador*, incorporando las opciones de las páginas de “Contacto” y “Quiénes somos”
- Se han creado los navegadores *ContactoNavegador* y *QuienesSomosNavegador*, correspondientes a las páginas de “Contacto” y “Quiénes somos”
- Se han creado y aplicado hojas de estilos para mostrar el título de las *Cards* en *HomeComponent* y *DetalleExcursionComponent* de color chocolate y sobre las imágenes

Notas importantes:

- Se han reforzado conceptos de ejercicios anteriores
- Para anidar elementos que incorporan un *scroll*, como un *FlatList* en un *ScrollView*, poner a *False* algún campo *scrollEnabled*

Ejercicio 7: Botones o Iconos

Tiempo invertido en el ejercicio: 3 h, 30 min

Pasos realizados:

- Se ha añadido el archivo JSON *comentarios.js* al directorio *comun*
- Se ha añadido, en *DetalleExcursion.js*, una nueva *Card* con los comentarios
- Se ha añadido un icono para poder marcar una excursión como favorita
- Se ha estilizado el menú lateral incluyendo imágenes para las páginas
- Se ha añadido un botón para abrir el menú lateral sin necesitar deslizar

Notas importantes:

- La adición de excursiones a favoritos no está implementada a nivel de aplicación, sólo se entra la página en concreto
- Hay que tener mucho cuidado con qué elementos son capaces de acceder a la variable *navigation* para poder hacer los despliegues del menú

Ejercicio 8: Servidor JSON

Tiempo invertido en el ejercicio: 1 h, 30 min

Pasos realizados:

- Se ha copiado la estructura de ficheros del directorio *comun*, tal y como se proporciona desde MiAulario
- Se ha instalado la librería *json-server*
- Se ha creado *comun.js*, con la información necesaria para acceder a los datos del servidor JSON y con los colores correspondientes al tema
- Se han actualizado todos los ficheros para incorporar los colores del tema desde *comun.js*
- Se han actualizado todos los ficheros para tomar las imágenes desde el servidor JSON, particularizando la petición para conseguir la imagen correspondiente al elemento en concreto
- En *DetalleExcursionComponent.js*, se han modificado los títulos, de forma que se muestren de color blanco

Notas importantes:

- Se ha visto cómo crear un servidor JSON y cómo acceder a él para obtener objetos, como imágenes o listas
- Se ha centralizado la gestión de los colores del tema de la aplicación

Ejercicio 9: Redux y Thunk

Tiempo invertido en el ejercicio: 1 h, 15 min

Pasos realizados:

- Se han instalado las librerías necesarias para las extensiones de Redux y Thunk
- Se ha configurado el *store* y se han incluido los archivos correspondientes a los *reducers* y las acciones
- Se ha declarado el *store* en un *Provider* en *App.js* y se han conectado los componentes necesarios
- Se han preparado las acciones *fetch* al cargarse el componente *CampobaseComponent.js*
- Se ha refactorizado la aplicación, para tomar los datos necesarios del *store* y no de los archivos JSON

Notas importantes:

- Se han implementado Redux y Thunk en la aplicación

Diagrama de flujo de la aplicación:

El diagrama de flujo se encuentra adjunto en el archivo PDF correspondiente, con detalles indicando los tramos de código relativos a cada bloque del diagrama.

Ejercicio 10: Activity Indicator y addFavoritos

Tiempo invertido en el ejercicio: 1 h

Pasos realizados:

- Se ha creado un nuevo componente de tipo *ActivityIndicator*
- Se ha modificado el renderizado de los componentes *CalendarioComponent*, *HomeComponent* y *QuienesSomosComponent*, mostrando el componente *ActivityIndicatorComponent* cuando los datos se estén cargando, mostrando el mensaje de error si se produjera un error y mostrando la información correspondiente en caso de haberse cargado todo bien
- Se ha creado un nuevo par de acciones, *ADD_FAVORITO* y *POST_FAVORITO*
- Se ha creado un nuevo *reducer*, *favoritos*
- Se han creado los *ActionCreators* *addFavorito* y *postFavorito*
- Se ha actualizado *DetalleExcursionComponent* para operar con la variable *favoritos* del estado centralizado, no el propio del componente
- Se ha actualizado *configureStore* para incluir el nuevo *reducer*

Notas importantes:

- Se ha creado un nuevo *reducer* para tratar la información de las excursiones favoritas
- Se pueden utilizar *ActivityIndicators* para mostrar elementos de carga de la aplicación

Ejercicio 11: Ejercicio Formularios y Modals con Redux

Tiempo invertido en el ejercicio: 4 h, 30 min

Pasos realizados:

- Se ha instalado la librería *react-native-ratings*
- Se ha creado un nuevo icono en el componente *DetalleExcursionComponent*, consistente en un lápiz azul
- Se han centrado los dos iconos en *DetalleExcursionComponent*
- Se ha creado un modal, que se puede lanzar desde el icono del lápiz
- Se han introducido los campos necesarios en el modal
- Se ha implementado la función necesaria para limpiar los campos del formulario
- Se han creado los campos del estado local de *DetalleExcursionComponent* necesarios para guardar los valores de los campos del formulario
- Se han creado los tipos de acciones correspondientes a *POST_COMENTARIO* y *ADD_COMENTARIO*
- Se han creado las acciones *postComentario* y *addComentario*. La primera es una función que genera una cadena de texto con la fecha actual y pasa los argumentos a la función *addComentario* tras un retraso de 2000 ms. La segunda es una función que provoca la acción del reducer *comentarios*
- Se ha creado, en el reducer *comentarios*, la acción correspondiente a la adición de un nuevo comentario al estado de Redux. La función genera un nuevo *id* de comentario correspondiente a la longitud del *array* de comentarios previo, y concatena el comentario pasado como argumento (además del *id* generado) a dicho *array*

Notas importantes:

- Se ha practicado a crear nuevas acciones en los *reducers* existentes
- Se ha utilizado el campo *ratings* para poder añadir valoraciones
- Se ha creado un modal para poder mostrar formularios sobre la pantalla previa