**Ejercicio 1** (30 min aprox)

Brevemente comento que es lo que he asimilado con este ejercicio. En primer lugar, he actualizado todas las tecnologías a emplear para este proyecto a su última versión: nodejs, npm…

Seguidamente, he abierto en Visual Studio Code la ruta de la carpeta con los contenidos, me he creado un repo GitHub y he sincronizado el entorno de Visual Studio con el repositorio. Me he parado a analizar cómo está compuesta la aplicación y su estructura de carpetas.

Por último, he lanzado exitosamente la app en mi móvil. Además, me he creado una cuenta personal de trello.

**Ejercicio 2** (30min aprox)

En este ejercicio he asimilado la estructura y funcionamiento de un componente. Además, he hecho uso de librerías externas, instalándolas e importándolas para su uso.

En este ejercicio se ha agregado a la página principal un componente llamado ‘Campobase’. Este componente está definido en ‘CampobaseComponent.js’ y exportado como ‘Campobase’ para su uso. Lo que hace este componente es gestionar las excursiones que obtiene de la carpeta comun y agregarlas al estado de la aplicación. Estas excursiones se pasan al componente ‘Calendario’.

‘Calendario’ es un componente funcional que recibe unos parámetros y que al igual que el anterior, es exportado para su uso (esto hay que hacerlo siempre si se quiere utilizar el componente en el resto de la aplicación). Este componente recorre la lista de excursiones mediante FlatList y para cada excursión genera un ListItem, que tendrá un título y una descripción.

FlatList y ListItem funcionan con una lista de elementos y una key para cada elemento, para identificarlos y poder trabajar con ellos.

Al final de este ejercicio obtengo en el móvil el listado de las excursiones, con su nombre y su descripción.

**Ejercicio 3** (40 min aprox)

Antes de comenzar a programar en este ejercicio, he comprendido la diferencia entre un componente funcional y uno de clase. Un componente de clase es aquel que se define como una clase y hace uso de React.Component. Un componente de clase puede utilizar el estado y el ciclo de vida directamente. Un componente funcional es más sencillo de programar e interpretar, pero no tiene acceso directo al estado ni al ciclo de vida. Sin embargo, puede usar useState como estado y UseEffect como ciclo de vida accediendo a ellos mediante Hooks. El componente clásico de clase no puede usar Hooks.

Hasta ahora, siempre había utilizado componentes funcionales, hasta el ejercicio 2 en el que hice uso del componente de clase llamado ‘Campobase’.

Lo primero que se hace en este ejercicio es crear un componente llamado DetalleExcursion, que se trata de un componente funcional. Si se analiza su código, se ve que su objetivo es recibir una excursión mediante props y desglosarla en un elemento Card dando información de su nombre, una imagen y su descripción. Además, antes de esto comprueba que la excursión recibida no esté vacía, y, de lo contrario, devuelve un elemento View vacío.

El siguiente paso en el ejercicio es editar el componente ‘Campobase’ para que este renderice el detalle de una excursión. Si se analiza el nuevo código, se observa que el componente ‘Calendario’ empleado por ‘Campobase’ tiene ahora una nueva propiedad onPress que se utilizará para establecer el estado del nuevo parámetro seleccionExcursion (para determinar la excursión seleccionada) al id de la excursión presionada por el usuario.

A su vez, se añade el nuevo componente ‘DetalleExcursion’ en el renderizado de ‘Campobase’ para que reciba como propiedad la excursión seleccionada por el usuario. Para ello, se filtra todo el listado de excursiones con el id almacenado en el parámetro de estado seleccionExcursion.

Por último, se edita también ‘Calendario’ para que pase el id de la excursión a ‘Campobase’.

En este punto, al clicar en una de las excursiones en el teléfono móvil se obtiene el detalle de la misma.

**Ejercicio 4** (40min aprox)

El primer paso en este ejercicio es la instalación de react-navigation para poder realizar un menú con el que navegar entre los diferentes componentes de la aplicación.

Para crear este menú y hacer posible la navegación se han realizado varios cambios en los componentes de la aplicación y en el fichero ‘App.js’.

La navegación se hace posible gracias al componente createStackNavigator, que es una función que crea un menú con una pila de navegación. El NavigationContainer envuelve toda la navegación de la aplicación y gestiona la navegación entre pantallas. Las Screens son las pantallas definidas en el menú y cada una tienen un nombre y un componente que se renderiza.

En el componente ‘Campobase’ es donde se define el menú de navegación en esta aplicación. Se crea un menú con dos screens. Una screen se llama ‘Calendar’ y renderiza el componente ‘Calendario’ y la otra screen tiene el nombre ‘DetalleExcursión’ y renderiza el componente ‘DetalleExcursion’. Ambas screens están incluidas en el StackNavigator. El componente ‘Campobase’ ahora devuelve un NavigationContainer que engloba el StackNavigator devuelto por la función ‘CalendarioNavegador’.

El componente ‘Calendario’ ha sido editado para que al clicar en cada elemento se navegue a la screen de nombre ‘DetalleExcursion’, es decir, al componente ‘DetalleExcursion’. Además, al realizar este paso se envía el id de la excursión mediante la navegación para que luego el componente ‘DetalleExcursion’ obtenga este id de los parámetros de navegación y renderice el componente para mostrar los detalles de esa excursión en el Card.

**Ejercicio 5** (40min aprox)

En este ejercicio, he implementado una barra lateral en la aplicación que permite navegar entre los componentes. En esta barra lateral se encuentran dos opciones de navegación. Una de ellas es la opción de navegar al calendario de excursiones y la otra es la opción de navegar al campo base con información de la aplicación (navegar al nuevo componente Home).

Para hacer esto posible, ha sido necesario realizar varios cambios en los componentes e integrar Drawer Navigation en la aplicación.

En primer lugar, he generado el componente ‘Home’ que contiene tres Cards con información proporcionada por ‘excursiones.js’ y por los ficheros añadidos ‘actividades.js’ y ‘cabeceras.js’.

El cambio más importante en este ejercicio radica en el componente ‘Campobase’.

En este ejercicio en lugar de devolver ‘CalendarioNavegador’, el componente devuelve ‘DrawerNavegador’, que es una función que gestiona la navegación entre las dos screens de drawer disponibles en la aplicación. Estas dos screens de drawer son el StackNavigator ‘CalendarioNavegador’ creado en el ejercicio anterior y el nuevo StackNavigator ‘HomeNavegador’.

A su vez, estas screens de drawer, que son elementos StackNavigator, contienen sus propias Stack.Screen, como en el ejercicio anterior. De esta manera, se logra navegar entre dos screens principales de la barra lateral de la aplicación (drawers) y dentro de estas screens también se puede navegar como se hacía en el ejercicio anterior.

**Ejercicio 6** (2-3h)

Este ejercicio ha el más laborioso. El objetivo del ejercicio era implementar dos nuevos componentes y actualizar el Drawer Navigation para incluir estos nuevos componentes. He creado los componentes y he actualizado la navegación sin problema. El problema ha venido al integrar FlatList con ScrollView sin que diera el siguiente error:

|  |
| --- |
| VirtualizedList should never be nested inside plain ScrollViews with the same orientation because it can break windowing and other functionality |

Al final lo he solucionado agregando en FlatList la propiedad: scrollEnabled={false}.

El tema de los estilos también ha sido un poquito más complicado que la generación de los componentes. He creado un archivo llamado ‘EstilosComponentes.js’ para exportar los estilos y luego usarlos en ‘HomeComponent.js’ y en ‘DetalleExcursion.js’. El problema ha venido en el uso de stylesheet en los diferentes elementos, pero lo he solucionado rápido.

En general, con este ejercicio he aprendido a utilizar los conocimientos adquiridos en los ejercicios anteriores.

**Ejercicio 7** (2h aprox)

En la primera parte de este ejercicio he incorporado comentarios al detalle de la excursión, así como un botón para marcar la excursión como favorita (a pesar de ser un estado que no se guarda, por el momento).

Para mostrar los comentarios he utilizado el archivo ‘comentarios.js’ ubicado en *commun*, y he renderizado el contenido del archivo para mostrar en el detalle de cada excursión sus comentarios correspondientes. Para hacer el filtrado de comentarios por excursión, utilizo la propiedad excursionId de cada comentario que se compara con el id de la excursión cuyo detalle se está mostrando. Una vez obtenidos los comentarios que pertenecen a esa excursión mapeo el array de comentarios para mostrar su información.

En adición, he implementado una función para formatear las fechas y que se vean de una manera legible para el usuario. Para hacer esto posible, he hecho uso de Date() y un regex para remplazar los caracteres que imposibilitan la lectura del *string* para Date(). Por último, he utilizado toLocaleString() para personalizar la fecha.

En cuanto al botón utilizado para marcar la excursión como favorita, utilizo la libraría *Icon*. Para que el icono funcione como un botón, lo que hago es incluir el icono dentro del renderizado de la excursión. A esa función de renderizado de la excursión se le pasa como propiedad la función que agrega el id de la excursión al array de favoritos del estado. El icono recibe esa función y cada vez que se clica en el icono, si ya se había clicado antes muestra una alerta indicándolo, y, si no, llama a la función que ha recibido desde arriba.

En la segunda parte de este ejercicio, he editado la barra lateral empleando drawerIcon y Font-awesome. He añadido iconos a cada elemento de navegación de la barra lateral, además de editar el título de la barra lateral incluyendo el nombre de la aplicación. Para esto último, he creado una función CustomDrawerContent que hace uso de DrawerContentScrollView y SafeAreaView. Esta función se utiliza en el propio Drawer.Navigator.

En la tercera y última parte de este ejercicio he incluido en cada navegador del Drawer (Stack.Navigator) una propiedad llamada headerLeft a la que se le pasa un icono cuya funcionalidad al clicar es abrir la barra lateral.

**Ejercicio 8** (1h aprox)

En primer lugar, en este ejercicio corrijo la navegación del ejercicio anterior ya que había incluido el headerleft en el navegador del calendario que engloba el calendario y el detalle de la excursión en lugar de sólo en el calendario. También he editado los estilos de los títulos que aparecen en las fotos para que se adecuen a lo que se ve en el vídeo.

En cuanto al trabajo acometido relativo al servidor json, lo primero que he hecho ha sido incluir la librería y ajustar la estructura de carpetas tal y como indica el ejercicio. He buscado la ip a la que están conectados mi ordenador y mi móvil y en otra terminal he lanzado el comando indicado por el guion.

Tras haber editado todas las imágenes de la aplicación para que consulten la imagen mediante ‘baseUrl’ obtengo el resultado esperado en el vídeo para las imágenes.

También acometo el cambio indicado respecto a los estilos para que los colores se obtengan de ese fichero compartido ‘comun.js’.

Por último, he de comentar que no he eliminado ni los ficheros en comun ni las imágenes en componentes.

**Ejercicio 9** (3h: 1h ejercicio + 2h asimilar redux y diagrama)

En el primer apartado de este ejercicio he podido entender de una manera más detallada cómo es la estructura de redux y thunk. Si bien el diagrama es muy didáctico, ver los archivos me ha ayudado a comprender los conceptos en mayor profundidad.

El segundo apartado de este ejercicio me ha ayudado a comprender de manera práctica los conceptos interiorizados en el apartado anterior. He comprendido como interactúan los diferentes componentes de redux en una aplicación real, el concepto de los estados y cómo se aplican y la manera en la que los reducers funcionan. Además, me parece didáctico y útil separar los reducers para estructurar la gestión de cada uno de los tipos de datos de manera más organizada.

He reestructurado la app para que los componentes consuman los datos con el nuevo repositorio redux.

Diagrama

El contenido generado por IA puede ser incorrecto.Adjunto el diagrama de comprensión de conceptos redux-thunk:

Diagrama

El contenido generado por IA puede ser incorrecto.Versión 2 (después de la presentación):

**Ejercicio 10** (2h)

El primer apartado del ejercicio ha consistido en incluir en la aplicación un elemento que indica al usuario si la aplicación está llevando a cabo una actividad en concreto. He creado el componente ‘IndicadorActividad’ empleando *ActivityIndicator* de react, que se encarga de proporcionar la funcionalidad deseada. Así que, lo que he hecho ha sido añadir el nuevo componente en el resto de los componentes que solicitan información al estado. Estos componentes son: ‘Calendario’, ‘DetalleExcursion’, ‘Home’ y ‘QuienesSomos’. La manera de incluir el indicador de actividad en estos varía ya que no todos los componentes se comportan por igual, pero en todos se ha incluido con un código que favorece la optimización.

En la segunda parte del ejercicio, he cambiado la funcionalidad de agregar excursiones como favoritas para que deje de guardarse el array de favoritos en una variable local y pase a guardarse en el store. He incluido un nuevo reducer llamado ‘favoritos’, que se encarga de añadir al array de favoritos del store el idExcursion que le llega en el payload. Para utilizar este reducer, he creado una función thunk (postFavorito) que se llama desde el componente ‘DetalleExcursion’ mediante dispatch. A su vez, esta función (postFavorito) llama mediante dispatch a otra función (addFavorito) que es la que envía al reducer el estado ADD\_FAVORITO y el id de la excursión a marcar en favoritos en el payload.

Gracias a esta segunda parte del ejercicio, he podido interiorizar un poco más el concepto del middleware (thunk).