FrontendMayo25-Orders.pdf

pamaro

Introducción a la Ingeniería del Software y los Sistemas de Información II

2º Grado en Ingeniería Informática - Ingeniería del Software

Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

Examen Mayo 25

Examen DeliverUS - Modelo A - Pedidos (Orders)

Recuerde que DeliverUS está descrito en: https://github.com/IISSI2-IS-2025

Enunciado del examen

Se ha de implementar la interfaz gráfica de algunos requisitos funcionales de los propietarios, en concreto:

RF.01 – Visualización de pedidos por restaurante

Como

propietario,

Quiero

ver un listado con todos los pedidos realizados a uno de mis restaurantes. Para cada pedido, necesito conocer su fecha de creación, el estado, la dirección de entrega y el precio,

Para

poder hacer un seguimiento y gestión de los mismos.

RF.02 – Edición de algunas propiedades de pedidos

Como

propietario,

Quiero

poder editar la dirección de entrega y el precio de los pedidos realizados a mis restaurantes,

Para

corregir errores o actualizar información relevante.

Pruebas de aceptación

- El propietario puede acceder a la edición de un pedido desde el listado de pedidos de su restaurante.
- Solo se muestran los campos dirección de entrega y precio total y han de presentarse con los valores actuales del pedido que se desea editar.
- Ambos campos (dirección de entrega y precio total) son obligatorios.
- Si se intenta guardar un pedido sin completar la dirección de entrega, se muestra un mensaje de error indicando que el campo es obligatorio.
- Si se intenta guardar un pedido sin establecer el precio total, se muestra un mensaje de error indicando que el campo es obligatorio.
- Si se introduce un precio igual o inferior a 0, se muestra un mensaje de error indicando que el valor debe ser mayor que 0.
- Al guardar correctamente los cambios, se actualiza la información del pedido y se muestra un mensaje de confirmación.
- Tras la correcta actualización de datos, se ha de navegar automáticamente hacia el listado de pedidos que debe presentar la información actualizada.

RF.03 – Visualización de analíticas de pedidos de restaurante

Como

propietario,

Quiero

visualizar analíticas sobre los pedidos de cada uno de mis restaurantes. Para cada restaurante quiero conocer: el importe facturado hoy, el número de pedidos en estado `pending`, el número de pedidos de hoy en estado `delivered` y el número de pedidos realizados ayer.

Para

tomar decisiones informadas sobre la gestión de mi restaurante.

RF.04 – Cambio de estado de un pedido

Como

propietario,

Quiero

poder cambiar el estado actual de los pedidos realizados a mis restaurantes. Desde el estado `pending`, puedo pasarlo a `in process`; de `in process` a `sent`; y de `sent` a `delivered`,

Para

mantener actualizado el estado de mis pedidos y que el cliente esté informado de ello.

Pruebas de aceptación

- Al cambiar correctamente el estado de un pedido, se ha de reflejar en el listado y se ha de actualizar la sección de analíticas. Además se ha de mostrar un mensaje de éxito.
- Si no se ha podido cambiar el estado se ha de mostrar un mensaje de error.
- Los pedidos en estado ´delivered´ no pueden cambiar de estado ni se debe presentar la opción de cambiarlo.

Ejercicios

1. Listado de pedidos. 3 puntos

Trabaje el RF.01 en el fichero `./DeliverUS-Frontend-Owner/src/screens/orders/OrdersScreen.js` realizando todos los cambios necesarios para mostrar una interfaz como se muestra en la Figura 1.

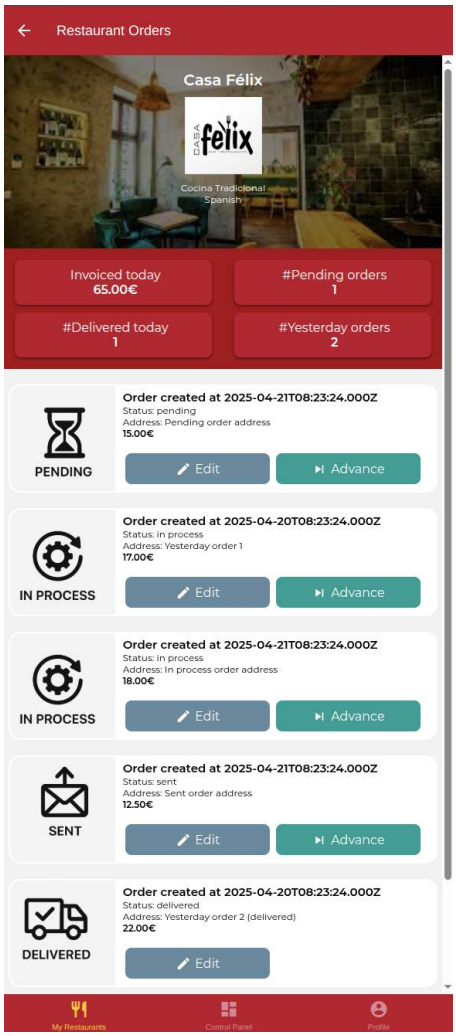


Figura 1: listado de pedidos y analíticas

Aspectos a tener en cuenta:

- La pantalla de listado de pedidos es accesible desde el botón de `Check orders` que se ha incorporado en la pantalla de detalles de restaurante `RestaurantDetailScreen`
- La imagen que representa el estado del pedido puede obtenerse a través de la función `getOrderImage(status)` que ya se le proporciona en el mismo fichero.
- El listado de pedidos puede obtenerse a través de la función `getRestaurantOrders (restaurantId)` que ya se le proporciona en `./DeliverUS-Frontend-Owner/src/api/RestaurantEndpoints.js` y que realiza la petición necesaria a backend.
- El botón de edición de pedido se realizará en el ejercicio 2.
- La sección de analíticas se realizará en el ejercicio 3.
- El botón de cambio de estado se realizará en el ejercicio 4.

2. Edición de pedido. 3 puntos

Trabaje el RF.02 en el fichero `./DeliverUS-Frontend-Owner/src/screens/orders/EditOrderScreen.js` tal y como se muestra en la Figura 2.

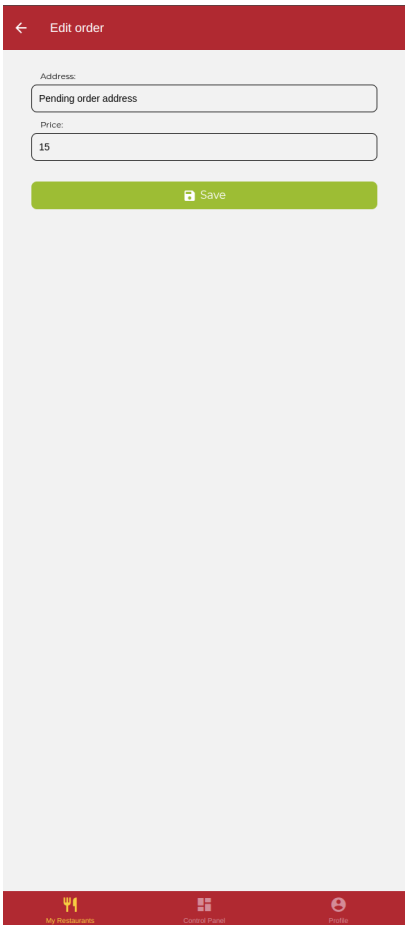


Figura 2: Edición de pedido

Aspectos a tener en cuenta:

- Debe incorporar un botón para navegar a la edición de un pedido desde el listado de pedidos en el fichero `./DeliverUS-Frontend-Owner/src/screens/orders/OrdersScreen.js`.
- Debe incorporar lo necesario para habilitar la navegación hacia la pantalla de edición de pedido.

3. Analíticas de restaurante. 2 puntos

Trabaje el RF.03 en el fichero `./DeliverUS-Frontend-Owner/src/screens/orders/OrdersScreen.js` realizando todos los cambios necesarios para mostrar la sección de analíticas tal y como se presenta en la Figura 1.

Aspectos a tener en cuenta:

- Se le proporcionan los estilos `analyticsContainer`, `analyticsRow` y `analyticsCell`, para mostrar la sección de analíticas y ha de completar las propiedades relacionadas con Flex para ajustarlo fielmente al estilo que se le presenta e la Figura 1.
- El objeto de analíticas que devuelve el backend tiene la siguiente estructura (valores de ejemplo):

```
{
   restaurantId: 1,
   numYesterdayOrders: 2,
   numPendingOrders: 1,
   numDeliveredTodayOrders: 1,
   invoicedToday: 65.0
}
```

4. Cambio de estado. 2 puntos

Trabaje el RF.04 en el fichero `./DeliverUS-Frontend-Owner/src/screens/orders/OrdersScreen.js` realizando todos los cambios necesarios para permitir avanzar el estado de cada pedido tal y como se presenta en la Figura 1.

Aspectos a tener en cuenta:

- Se le proporciona la función `nextStatus(order)` en el fichero `./DeliverUS-Frontend-Owner/src/api/OrderEndpoints.js` que realiza la petición a backend para avanzar en el estado.
- El nombre del `MaterialCommunityIcon` del botón `Advance` es `skip-next`

Procedimiento de entrega

1. Borrar las carpetas **DeliverUS-Backend/node_modules**, **DeliverUS-Frontend-Owner/node_modules** y **DeliverUS-Frontend-Owner/.expo**.
2. Crear un ZIP que incluya todo el proyecto. **Importante: Comprueba que el ZIP no es el mismo que te has descargado e incluye tu solución**
3. Avisa al profesor antes de entregar.
4. Cuando el profesor te dé el visto bueno, puedes subir el ZIP a la plataforma de Enseñanza Virtual. **Es muy importante esperar a que la plataforma te muestre un enlace al ZIP antes de pulsar el botón de enviar**. Se recomienda descargar ese ZIP para comprobar lo que se ha subido. Un vez realizada la comprobación, puedes enviar el examen.

Preparación del Entorno

a) Windows

- Abre una terminal y ejecuta el siguiente comando:

```
npm run install:all:win
```

b) Linux/MacOS

- Abre una terminal y ejecuta el siguiente comando:

```
npm run install:all:bash
```

Ejecución

Backend

- Para **recrear las migraciones y seeders**, abre una terminal y ejecuta el siguiente comando:

```
npm run migrate:backend
```

- Para **iniciar el backend**, abre una terminal y ejecuta el siguiente comando:

```
npm run start:backend
```

Frontend

- Para **ejecutar la aplicación frontend del** `owner`, abre una nueva terminal y ejecuta el siguiente comando:

```
npm run start:frontend
```

Depuración

- Para **depurar el frontend**, asegúrate de que **SÍ** haya una instancia en ejecución del frontend que deseas depurar, y usa las herramientas de depuración del navegador.

Cacaolat

Fundada 1933

TU CUERPO NO VIVE SOLO DE APUNTES Y CAFÉS. DALE PROTEÍNA. DALE CACAOLAT PRO.

25g de proteína y muuuucho sabor.

DESCUBRE MÁS AQUÍ

SOLUCIONES:

Soluciones paso a paso y al final los archivos completos

Ejercicio 1: Listado de pedidos

Nos vamos al fichero `./DeliverUS-Frontend-Owner/src/screens/orders/OrdersScreen.js` para hacer lo cambios para la Figura 1.

Añadimos la constante

```
const [orders, setOrders] = useState({})
```

y añadimos la función `fetchRestaurantOrders()` al useEffect()

```
useEffect(() => {
  fetchRestaurantDetail()
  fetchRestaurantOrders()
}, [route])
```

añadimos a dicha función, la función de `getRestaurantOrders` que nos viene dada, con los parámetros id y que haga setOrders con dichos pedidos.

```
const fetchRestaurantORders = async => {
try{
  const fetchedOrders = await getRestaurantOrders(route.params.id)
  setOrders(fetchedOrders)
} catch (error) {
  showMessage({
    message: `There was an error while retrieving restaurant orders (id ${route.params.id}). ${error}`,
    type: 'error',
    style: GlobalStyles.flashStyle,
    titleStyle: GlobalStyles.flashTextStyle
    })
  }
}
```

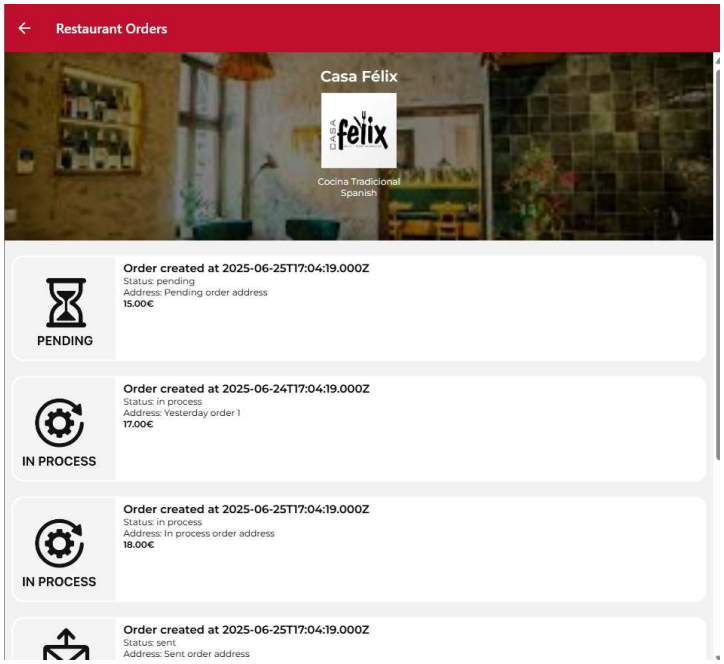
Por último tenemos que añadir en `renderOrder` la lista de pedidos:

```
const renderOrder = ({ item }) => {
  return (
    <ImageCard
      imageUri={getOrderImage(item.status)}
      title={`Order created at ${item.createdAt}`}
      >
        <TextRegular numberOfLines={2}>Status: {item.status}</TextRegular>
        <TextRegular numberOfLines={2}>Address: {item.address}</TextRegular>
        <TextSemiBold>{item.price.toFixed(2)}€</TextSemiBold>
      </ImageCard>
  )
}
```

y añadimos en el return

```
return (
  <FlatList
  ListHeaderComponent={renderHeader}
  ListEmptyComponent={renderEmptyOrdersList}
  style={styles.container}
  data={orders}
  renderItem={renderOrder}
  keyExtractor={item => item.id.toString}
  />
 )
}
```

debería de quedar así, para comprobarlo vamos a la pantalla myRestaurants y clickamos en uno de ellos, y luego en el botón “Check orders” :



Ejercicio 2: Edición de pedido

Para este ejercicio trabajamos en `/DeliverUS-Frontend-Owner/src/screens/orders/EditOrderScreen.js` aunque para navegar a esta pantalla primero tenemos que añadir un botón desde `OrdersScreen` en la función de `renderOrder` para que así nos aparezca dentro de cada pedido. Podemos copiarlo del botón de Edit de `RestaurantScreens` y cambiar los datos para que reciba orders y para que vaya a la pantalla correcta:

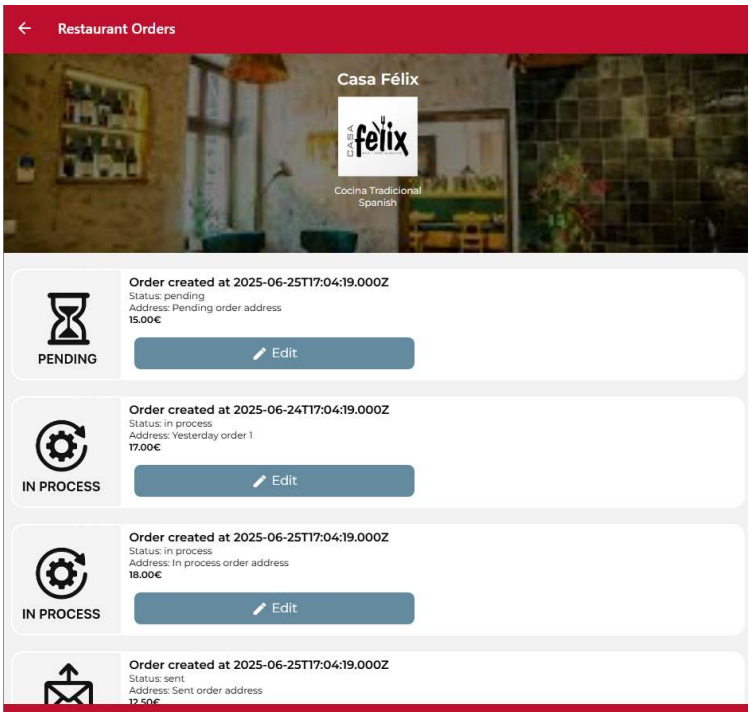
```
<View style={styles.actionButtonsContainer}>
 <Pressable
  onPress={() => navigation.navigate('EditOrdersScreen', { orderId: item.id })
  }
  style={({ pressed }) => [
   {
```

```
          backgroundColor: pressed
            ? GlobalStyles.brandBlueTap
            : GlobalStyles.brandBlue
        },
        styles.actionButton
      ]}>
      <View style={[{ flex: 1, flexDirection: 'row', justifyContent: 'center' }]}>
        <MaterialCommunityIcons name='pencil' color={'white'} size={20}/>
        <TextRegular textStyle={styles.text}>
          Edit
        </TextRegular>
      </View>
    </Pressable>
      </View>
```

Para que esta pantalla funcione tenemos que añadirla al stack, es decir a `RestaurantStack.js` , copiamos el formato de las demás pantallas e introducimos los datos acordes a la pantalla para editar el pedido:

```
<Stack.Screen
name='EditOrdersScreen'
component={EditOrderScreen}
options={{
  title: 'Edit order'
}} />
```

Ahora debe aparecer nuestro botón:



Aunque al pulsarlo, la pantalla está todavía en blanco. Para ello debemos introducir el siguiente contenido en `EditOrderScreen.js` , para ello podemos tomar el formato de `EditRestaurantScreen.js` :

Añadimos los valores iniciales:

```
const [backendErrors, setBackendErrors] = useState([])
const [order, setOrder] = useState({})

const [initialValues, setInitialOrderValues] = useState({ address: '', price: '' })
const validationSchema = yup.object().shape({
 address: yup
  .string()
  .max(255, 'Address too long')
  .required('Address is required'),
 price: yup
  .number()
  .required('Price is required').moreThan(0)
})
```

el useEffect para la función de buscar pedidos:

```
useEffect(() => {
 async function fetchOrder () {
  try {
   const fetchedOrders = await getById(route.params.orderId)
   setOrder(fetchedOrders)
   setInitialOrderValues({
    address: fetchedOrders.address,
    price: fetchedOrders.price.toString()
   })
  } catch (error) {
   showMessage({
    message: `There was an error while retrieving order details (id ${route.params.orderId}). ${error}`,
    type: 'danger',
    style: GlobalStyles.flashStyle,
    titleStyle: GlobalStyles.flashTextStyle
   })
  }
 }
 fetchOrder()
}, [route.params.orderId])
```

y la función `updateOrder` que tenemos que implementar:

```
const updateOrder = async (values) => {
 setBackendErrors([])
 try {
  await update(order.id, values)
  showMessage({
   message: `Order ${order.id} updated successfully`,
   type: 'success',
   style: GlobalStyles.flashStyle,
   titleStyle: GlobalStyles.flashTextStyle
  })
  navigation.navigate('OrdersScreen', {
   id:
   order.restaurantId,
   dirty: true
  })
```

```
  } catch (error) {
    setBackendErrors(error.errors || [])
  }
}
```

Por último tenemos que retornar todos los botones que nos ayudaran a editar el pedido:

```
return (
  <Formik
    enableReinitialize
    validationSchema={validationSchema}
    initialValues={initialValues}
    onSubmit={updateOrder}
  >
    {({ handleSubmit }) => (
      <ScrollView>
        <View style={{ alignItems: 'center' }}>
          <View style={{ width: '60%' }}>
            <InputItem name = "address" label = "Address:"></InputItem>
            <InputItem name = "price" label = "Price:" keyboardType="numeric"></InputItem>

            {backendErrors.length > 0 && backendErrors.map((error, index) => (
              <TextError key={index}>{error.param} - {error.msg}</TextError>
            ))}

            <Pressable
              onPress={handleSubmit}
              style={({ pressed }) => [
                {
                  backgroundColor: pressed
                    ? GlobalStyles.brandSuccessTap
                    : GlobalStyles.brandSuccess
                },
                styles.button
              ]}
              >
              <View style={[{ flex: 1, flexDirection: 'row', justifyContent: 'center' }]}>
                <MaterialCommunityIcons name='content-save' color={'white'} size={20}/>
                <TextRegular textStyle={styles.text}>
                  Save
                </TextRegular>
              </View>
              </Pressable>
            </View>
          </View>
        </ScrollView>
      )}
    </Formik>
  )
}
```

Y ya obtendríamos la pantalla como en la Figura 2

Ejercicio 3: Analíticas de restaurante

Vamos al fichero `./DeliverUS-Frontend-Owner/src/screens/orders/OrdersScreen.js`

Primero añadimos la constante y la función en el useEffect

```
// Analytics
const [analytics, setAnalytics] = useState(null)

useEffect(() => {
 fetchRestaurantDetail()
 fetchRestaurantOrders()
 fetchRestaurantAnalytics()
}, [route])
```

luego la desarrollamos dicha función:

```
const fetchRestaurantAnalytics = async () => {
 try {
  const fetchedAnalytics = await getRestaurantAnalytics(route.params.id)
  setAnalytics(fetchedAnalytics)
 } catch (error) {
  showMessage({
   message: `There was an error while retrieving restaurant analytics (id ${route.params.id}). ${error}`,
   type: 'error',
   style: GlobalStyles.flashStyle,
   titleStyle: GlobalStyles.flashTextStyle
  })
 }
}
```

Estas analiticas tienen que ser mostradas, por lo que cambiamos la función `renderAnalytics` para que muestre los valores adecuados:

```
const renderAnalytics = () => {
 return (
  <View style={styles.analyticsContainer}>
   <View style={styles.analyticsRow}>
    <View style={styles.analyticsCell}>
     <TextRegular textStyle={styles.text}>
      Invoiced today
     </TextRegular>
     <TextSemiBold textStyle={styles.text}>
     {analytics.invoicedToday.toFixed(2)}€
     </TextSemiBold>
    </View>
    <View style={styles.analyticsCell}>
     <TextRegular textStyle={styles.text}>
      #Pending orders
     </TextRegular>
     <TextSemiBold textStyle={styles.text}>
     {analytics.numPendingOrders}
     </TextSemiBold>
    </View>
   </View>
```

```
        <View style={styles.analyticsRow}>
          <View style={styles.analyticsCell}>
            <TextRegular textStyle={styles.text}>
              #Delivered today
            </TextRegular>
            <TextSemiBold textStyle={styles.text}>
            {analytics.numDeliveredTodayOrders}
            </TextSemiBold>
          </View>
          <View style={styles.analyticsCell}>
            <TextRegular textStyle={styles.text}>
              #Yesterday orders
            </TextRegular>
            <TextSemiBold textStyle={styles.text}>
            {analytics.numYesterdayOrders}
            </TextSemiBold>
          </View>
        </View>
      </View>
    )
  }
```

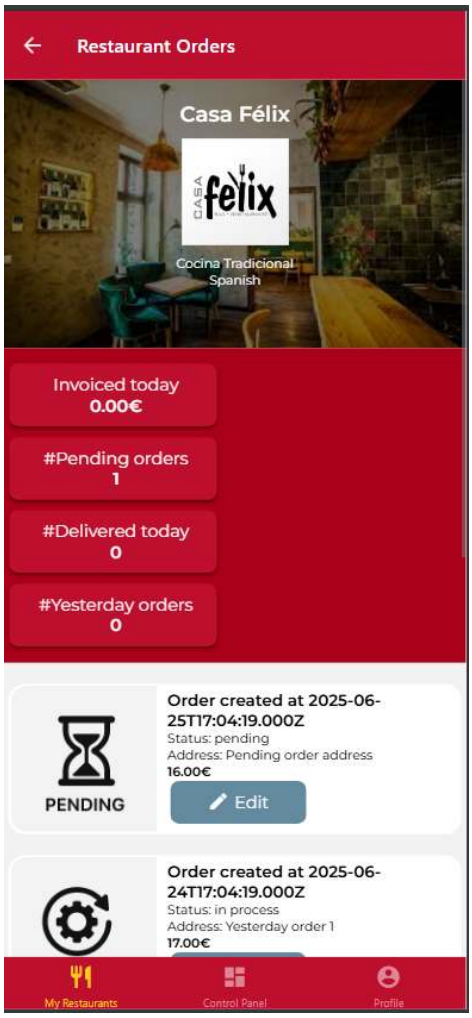
y añadimos dicha función a `renderHeader`

```
  const renderHeader = () => {
    return (
      <View>
        <ImageBackground source={(restaurant?.heroImage) ? { uri: API_BASE_URL + '/' + restaurant.heroImage, cache:
'force-cache' } : undefined} style={styles.imageBackground}>
          <View style={styles.restaurantHeaderContainer}>
            <TextSemiBold textStyle={styles.textTitle}>{restaurant.name}</TextSemiBold>
            <Image style={styles.image} source={restaurant.logo ? { uri: API_BASE_URL + '/' + restaurant.logo, cache: 'forc
e-cache' } : undefined} />
            <TextRegular textStyle={styles.description}>{restaurant.description}</TextRegular>
            <TextRegular textStyle={styles.description}>{restaurant.restaurantCategory ? restaurant.restaurantCategory.na
me : ''}</TextRegular>
          </View>
        </ImageBackground>
        {analytics !== null && renderAnalytics()}
      </View>
    )
  }
```

Sin embargo, con estos cambios no es suficiente y debemos añadir contendio a la función `getRestaurantAnalytics` de `RestaurantEndpoints.js`

```
function getRestaurantAnalytics (restaurantId) {
  return get(`/restaurants/${restaurantId}/analytics`)
}
```

Con estos cambios nos daría este resultado:



Este no es igual al de la Figura 1 del enunciado, por lo que tenemos que hacer un cambio en los estilos que nos dan, vemos que el estilo `analyticsRow` está vacío y tenemos que cambiarlo:

```
analyticsRow: {
  flexDirection: 'row',
  justifyContent: 'space-around'
},
```

Y ahora si que nos queda como en la Figura 1.

Ejercicio 4: Cambio de estado

Tambien en el archivo `OrderScreen.js` , completamos la función `handleNextStatus` utilizando la función `nextStatus(order)` que nos dan

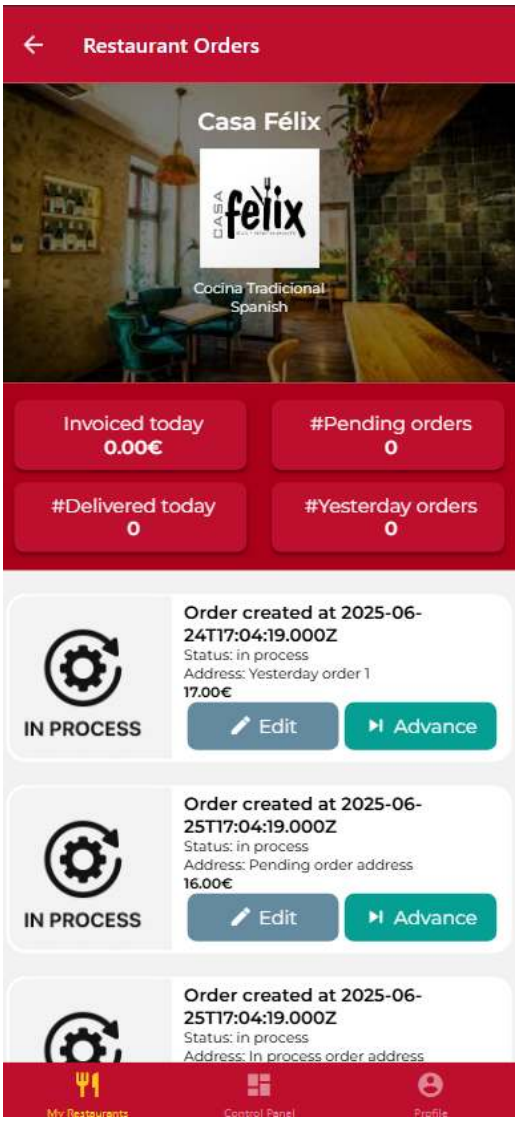
```
const handleNextStatus = async (order) => {
  try {
    await nextStatus(order)
    showMessage({
      message: `Order ${order.id} status updated`,
```

```
        type: 'success',
        style: GlobalStyles.flashStyle,
        titleStyle: GlobalStyles.flashTextStyle
      })
      fetchRestaurantOrders()
      fetchRestaurantAnalytics()
    } catch (error) {
      showMessage({
        message: `There was an error while advancing order status ${error}`,
        type: 'danger',
        style: GlobalStyles.flashStyle,
        titleStyle: GlobalStyles.flashTextStyle
      })
    }
  }
```

Luego añadimos el botón para cada pedido en `renderOrder` , lo ponemos después del botón de edit de esta manera:

```
{item.status !== 'delivered' &&
<Pressable
  onPress={() => handleNextStatus(item)}
  style={({ pressed }) => [
    {
      backgroundColor: pressed
        ? GlobalStyles.brandGreenTap
        : GlobalStyles.brandGreen
    },
    styles.actionButton
  ]}>
  <View style={[{ flex: 1, flexDirection: 'row', justifyContent: 'center' }]}>
  <MaterialCommunityIcons name='skip-next' color={'white'} size={20}/>
  <TextRegular textStyle={styles.text}>
    Advance
  </TextRegular>
</View>
</Pressable>
}
```

ahora al presionar el botón avanzará de estado y queda de la siguiente manera:



Este es el código completo por archivo del siguiente proyecto base:

https://github.com/IISSI2-IS-2025/ExLab-Frontend-Curso-Orders

OrderScreen.js:

```
/* eslint-disable react/prop-types */
import React, { useEffect, useState } from 'react'
import { StyleSheet, View, FlatList, ImageBackground, Image, Pressable } from 'react-native'
import { showMessage } from 'react-native-flash-message'
import { MaterialCommunityIcons } from '@expo/vector-icons'
import { getDetail, getRestaurantAnalytics, getRestaurantOrders } from '../../api/RestaurantEndpoints'
import { nextStatus } from '../../api/OrderEndpoints'

import TextRegular from '../../components/TextRegular'
import TextSemiBold from '../../components/TextSemibold'
import * as GlobalStyles from '../../styles/GlobalStyles'
import { API_BASE_URL } from '@env'
import pendingOrderImage from '../../../assets/order_status_pending.png'
```

```
import inProcessOrderImage from '../../../assets/order_status_in_process.png'
import sentOrderImage from '../../../assets/order_status_sent.png'
import deliveredOrderImage from '../../../assets/order_status_delivered.png'
import ImageCard from '../../components/ImageCard'

export default function OrdersScreen ({ navigation, route }) {
 const [restaurant, setRestaurant] = useState({})
 // Orders listing
 const [orders, setOrders] = useState({})
 // Analytics
 const [analytics, setAnalytics] = useState(null)

 useEffect(() => {
  fetchRestaurantDetail()
  fetchRestaurantOrders()
  fetchRestaurantAnalytics()
 }, [route])

 const fetchRestaurantAnalytics = async () => {
  try {
   const fetchedAnalytics = await getRestaurantAnalytics(route.params.id)
   setAnalytics(fetchedAnalytics)
  } catch (error) {
   showMessage({
    message: `There was an error while retrieving restaurant analytics (id ${route.params.id}). ${error}`,
    type: 'error',
    style: GlobalStyles.flashStyle,
    titleStyle: GlobalStyles.flashTextStyle
   })
  }
 }

 const fetchRestaurantOrders = async () => {
  try {
   const fetchedOrders = await getRestaurantOrders(route.params.id)
   setOrders(fetchedOrders)
  } catch (error) {
   showMessage({
    message: `There was an error while retrieving restaurant orders (id ${route.params.id}). ${error}`,
    type: 'error',
    style: GlobalStyles.flashStyle,
    titleStyle: GlobalStyles.flashTextStyle
   })
  }
 }

 const handleNextStatus = async (order) => {
  try {
   await nextStatus(order)
   showMessage({
    message: `Order ${order.id} status updated`,
    type: 'success',
    style: GlobalStyles.flashStyle,
    titleStyle: GlobalStyles.flashTextStyle
   })
   fetchRestaurantOrders()
   fetchRestaurantAnalytics()
  } catch (error) {
```

```
      showMessage({
        message: `There was an error while advancing order status ${error}`,
        type: 'danger',
        style: GlobalStyles.flashStyle,
        titleStyle: GlobalStyles.flashTextStyle
      })
    }
  }

  const renderAnalytics = () => {
    return (
      <View style={styles.analyticsContainer}>
        <View style={styles.analyticsRow}>
          <View style={styles.analyticsCell}>
            <TextRegular textStyle={styles.text}>
              Invoiced today
            </TextRegular>
            <TextSemiBold textStyle={styles.text}>
            {analytics.invoicedToday.toFixed(2)}€
            </TextSemiBold>
          </View>
          <View style={styles.analyticsCell}>
            <TextRegular textStyle={styles.text}>
              #Pending orders
            </TextRegular>
            <TextSemiBold textStyle={styles.text}>
            {analytics.numPendingOrders}
            </TextSemiBold>
          </View>
        </View>

        <View style={styles.analyticsRow}>
          <View style={styles.analyticsCell}>
            <TextRegular textStyle={styles.text}>
              #Delivered today
            </TextRegular>
            <TextSemiBold textStyle={styles.text}>
            {analytics.numDeliveredTodayOrders}
            </TextSemiBold>
          </View>
          <View style={styles.analyticsCell}>
            <TextRegular textStyle={styles.text}>
              #Yesterday orders
            </TextRegular>
            <TextSemiBold textStyle={styles.text}>
            {analytics.numYesterdayOrders}
            </TextSemiBold>
          </View>
        </View>
      </View>
    )
  }
  const renderHeader = () => {
    return (
      <View>
        <ImageBackground source={(restaurant?.heroImage) ? { uri: API_BASE_URL + '/' + restaurant.heroImage, cache:
'force-cache' } : undefined} style={styles.imageBackground}>
          <View style={styles.restaurantHeaderContainer}>
```

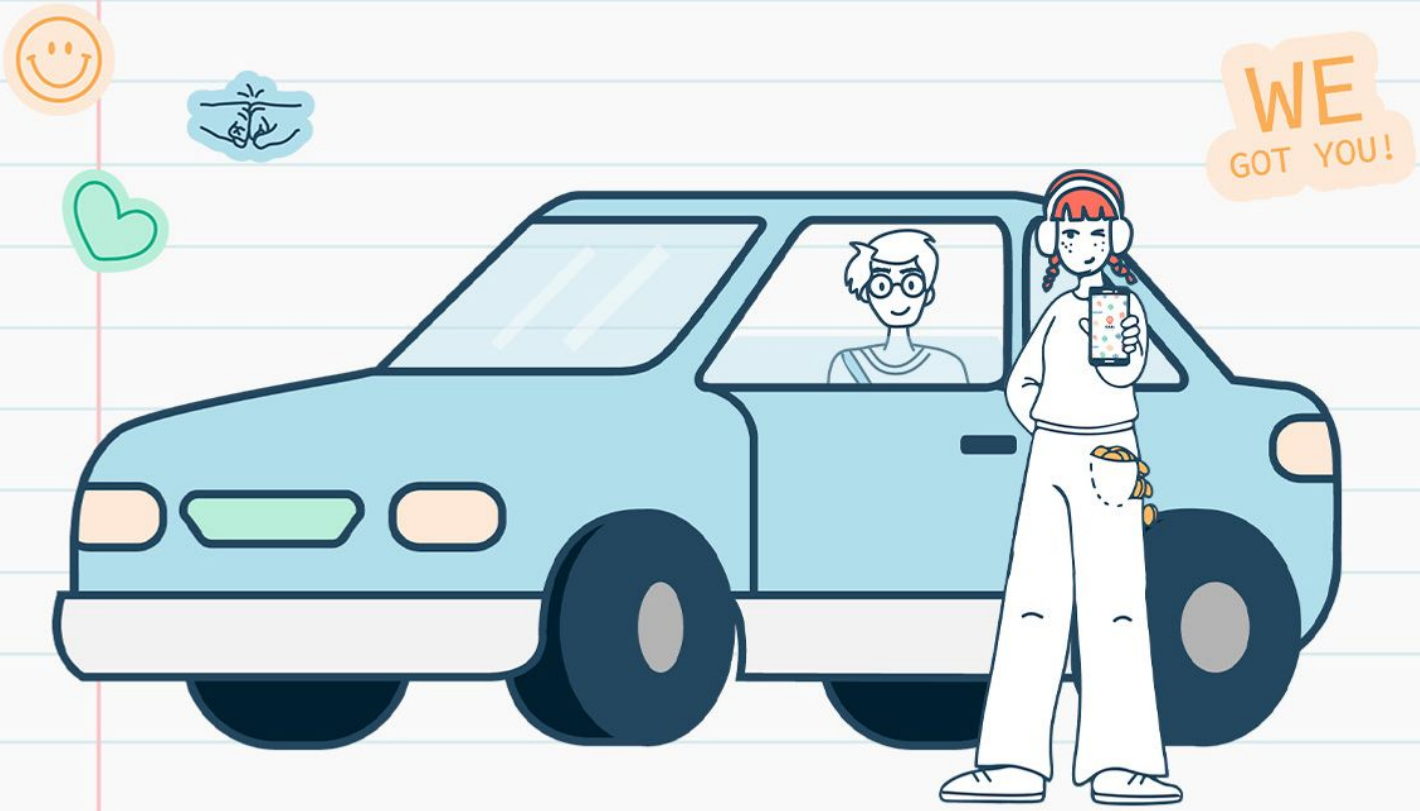
```
          <TextSemiBold textStyle={styles.textTitle}>{restaurant.name}</TextSemiBold>
          <Image style={styles.image} source={restaurant.logo ? { uri: API_BASE_URL + '/' + restaurant.logo, cache: 'forc
e-cache' } : undefined} />
          <TextRegular textStyle={styles.description}>{restaurant.description}</TextRegular>
          <TextRegular textStyle={styles.description}>{restaurant.restaurantCategory ? restaurant.restaurantCategory.na
me : ''}</TextRegular>
        </View>
      </ImageBackground>
      {analytics !== null && renderAnalytics()}
    </View>
  )
 }

 const getOrderImage = (status) => {
  switch (status) {
   case 'pending':
    return pendingOrderImage
   case 'in process':
    return inProcessOrderImage
   case 'sent':
    return sentOrderImage
   case 'delivered':
    return deliveredOrderImage
  }
 }

 const renderOrder = ({ item }) => {
  return (
   <ImageCard
    imageUri={getOrderImage(item.status)}
    title={`Order created at ${item.createdAt}`}
    >
     <TextRegular numberOfLines={2}>Status: {item.status}</TextRegular>
     <TextRegular numberOfLines={2}>Address: {item.address}</TextRegular>
     <TextSemiBold>{item.price.toFixed(2)}€</TextSemiBold>

    <View style={styles.actionButtonsContainer}>
     <Pressable
      onPress={() => navigation.navigate('EditOrdersScreen', { orderId: item.id })
      }
      style={({ pressed }) => [
       {
        backgroundColor: pressed
         ? GlobalStyles.brandBlueTap
         : GlobalStyles.brandBlue
       },
       styles.actionButton
      ]}>
      <View style={[{ flex: 1, flexDirection: 'row', justifyContent: 'center' }]}>
       <MaterialCommunityIcons name='pencil' color={'white'} size={20}/>
       <TextRegular textStyle={styles.text}>
        Edit
       </TextRegular>
      </View>
     </Pressable>

     {item.status !== 'delivered' &&
     <Pressable
```

```
        onPress={() => handleNextStatus(item)}
        style={({ pressed }) => [
         {
          backgroundColor: pressed
           ? GlobalStyles.brandGreenTap
           : GlobalStyles.brandGreen
         },
         styles.actionButton
        ]}>
        <View style={[{ flex: 1, flexDirection: 'row', justifyContent: 'center' }]}>
        <MaterialCommunityIcons name='skip-next' color={'white'} size={20}/>
        <TextRegular textStyle={styles.text}>
         Advance
        </TextRegular>
       </View>
      </Pressable>
      }
       </View>
      </ImageCard>
    )
  }

  const renderEmptyOrdersList = () => {
   return (
    <TextRegular textStyle={styles.emptyList}>
     This restaurant has no orders yet.
    </TextRegular>
   )
  }

  const fetchRestaurantDetail = async () => {
   try {
    const fetchedRestaurant = await getDetail(route.params.id)
    setRestaurant(fetchedRestaurant)
   } catch (error) {
    showMessage({
     message: `There was an error while retrieving restaurant details (id ${route.params.id}). ${error}`,
     type: 'error',
     style: GlobalStyles.flashStyle,
     titleStyle: GlobalStyles.flashTextStyle
    })
   }
  }

  return (
    <FlatList
    ListHeaderComponent={renderHeader}
    ListEmptyComponent={renderEmptyOrdersList}
    style={styles.container}
    data={orders}
    renderItem={renderOrder}
    keyExtractor={item => item.id.toString}
    />
   )
  }

  const styles = StyleSheet.create({
   container: {
```

CARi

Ahorra dinero compartiendo tu ruta

Comparte coche con tus compis de la uni
y ahorra en tus trayectos



Con CARi comparte tu coche con gente de tu uni y ahorra en cada viaje
sin complicaciones

¡Escanea!

```
  flex: 1
},
restaurantHeaderContainer: {
  height: 250,
  padding: 20,
  backgroundColor: 'rgba(0,0,0,0.5)',
  flexDirection: 'column',
  alignItems: 'center'
},
imageBackground: {
  flex: 1,
  resizeMode: 'cover',
  justifyContent: 'center'
},
image: {
  height: 100,
  width: 100,
  margin: 10
},
description: {
  color: 'white'
},
textTitle: {
  fontSize: 20,
  color: 'white'
},
emptyList: {
  textAlign: 'center',
  padding: 50
},
text: {
  fontSize: 16,
  color: 'white',
  alignSelf: 'center',
  marginLeft: 5
},
actionButton: {
  borderRadius: 8,
  height: 40,
  marginTop: 12,
  margin: '1%',
  padding: 10,
  alignSelf: 'center',
  flexDirection: 'column',
  width: '50%'
},
actionButtonsContainer: {
  flexDirection: 'row',
  bottom: 5,
  position: 'absolute',
  width: '90%'
},
analyticsContainer: {
  backgroundColor: GlobalStyles.brandPrimaryTap,
  paddingVertical: 10
},
analyticsRow: {
  flexDirection: 'row',
```

```
    justifyContent: 'space-around'
  },
  analyticsCell: {
    margin: 5,
    color: 'white',
    fontSize: 12,
    width: '45%',
    backgroundColor: GlobalStyles.brandPrimary,
    borderRadius: 8,
    paddingVertical: 10,
    shadowColor: '#000',
    shadowOffset: {
      width: 0,
      height: 2
    },
    shadowOpacity: 0.41,
    shadowRadius: 3.11,
    elevation: 2
  }
})
```

EditOrderScreen.js

```
import React, { useEffect, useState } from 'react'
import { ScrollView, StyleSheet, View, Pressable } from 'react-native'
import { MaterialCommunityIcons } from '@expo/vector-icons'
import * as yup from 'yup'
import { Formik } from 'formik'
import { getById, update } from '../../api/OrderEndpoints'
import InputItem from '../../components/InputItem'
import TextRegular from '../../components/TextRegular'
import TextError from '../../components/TextError'
import * as GlobalStyles from '../../styles/GlobalStyles'
import { showMessage } from 'react-native-flash-message'

export default function EditOrderScreen ({ navigation, route }) {
  const [backendErrors, setBackendErrors] = useState([])
  const [order, setOrder] = useState({})

  const [initialValues, setInitialOrderValues] = useState({ address: '', price: '' })
  const validationSchema = yup.object().shape({
    address: yup
      .string()
      .max(255, 'Address too long')
      .required('Address is required'),
    price: yup
      .number()
      .required('Price is required').moreThan(0)
  })

  useEffect(() => {
    async function fetchOrder () {
      try {
        const fetchedOrders = await getById(route.params.orderId)
        setOrder(fetchedOrders)
        setInitialOrderValues({
```

```
        address: fetchedOrders.address,
        price: fetchedOrders.price.toString()
      })
    } catch (error) {
      showMessage({
        message: `There was an error while retrieving order details (id ${route.params.orderId}). ${error}`,
        type: 'danger',
        style: GlobalStyles.flashStyle,
        titleStyle: GlobalStyles.flashTextStyle
      })
    }
  }
  fetchOrder()
}, [route.params.orderId])

const updateOrder = async (values) => {
  setBackendErrors([])
  try {
    await update(order.id, values)
    showMessage({
      message: `Order ${order.id} updated successfully`,
      type: 'success',
      style: GlobalStyles.flashStyle,
      titleStyle: GlobalStyles.flashTextStyle
    })
    navigation.navigate('OrdersScreen', {
      id:
      order.restaurantId,
      dirty: true
    })
  } catch (error) {
    setBackendErrors(error.errors || [])
  }
}

return (
  <Formik
    enableReinitialize
    validationSchema={validationSchema}
    initialValues={initialValues}
    onSubmit={updateOrder}
  >
    {({ handleSubmit }) => (
      <ScrollView>
        <View style={{ alignItems: 'center' }}>
          <View style={{ width: '60%' }}>
            <InputItem name = "address" label = "Address:"></InputItem>
            <InputItem name = "price" label = "Price:" keyboardType="numeric"></InputItem>

            {backendErrors.length > 0 && backendErrors.map((error, index) => (
              <TextError key={index}>{error.param} - {error.msg}</TextError>
            ))}

            <Pressable
              onPress={handleSubmit}
              style={({ pressed }) => [
                {
                  backgroundColor: pressed
```

```
                ? GlobalStyles.brandSuccessTap
                : GlobalStyles.brandSuccess
            },
            styles.button
          ]}
          >
          <View style={[{ flex: 1, flexDirection: 'row', justifyContent: 'center' }]}>
            <MaterialCommunityIcons name='content-save' color={'white'} size={20}/>
            <TextRegular textStyle={styles.text}>
              Save
            </TextRegular>
          </View>
            </Pressable>
        </View>
      </View>
    </ScrollView>
  )}
  </Formik>
 )
}

const styles = StyleSheet.create({
 button: {
  borderRadius: 8,
  height: 40,
  padding: 10,
  width: '100%',
  marginTop: 20,
  marginBottom: 20
 },
 text: {
  fontSize: 16,
  color: 'white',
  textAlign: 'center',
  marginLeft: 5
 }
})
```

RestaurantEndpoints.js

```
import { get, post, put, destroy } from './helpers/ApiRequestsHelper'
function getAll () {
 return get('users/myrestaurants')
}

function getDetail (id) {
 return get(`restaurants/${id}`)
}

function getRestaurantCategories () {
 return get('restaurantCategories')
}

function create (data) {
 return post('restaurants', data)
}
```

```
function update (id, data) {
  return put(`restaurants/${id}`, data)
}

function remove (id) {
  return destroy(`restaurants/${id}`)
}

function getRestaurantOrders (restaurantId) {
  return get(`/restaurants/${restaurantId}/orders`)
}

function getRestaurantAnalytics (restaurantId) {
  return get(`/restaurants/${restaurantId}/analytics`)
}

export { getRestaurantOrders, getRestaurantAnalytics, getAll, getDetail, getRestaurantCategories, create, update, remo
ve }
```

RestaurantStack.js

```
import { createNativeStackNavigator } from '@react-navigation/native-stack'
import React from 'react'
import CreateProductScreen from './CreateProductScreen'
import CreateRestaurantScreen from './CreateRestaurantScreen'
import EditProductScreen from './EditProductScreen'
import EditRestaurantScreen from './EditRestaurantScreen'
import RestaurantDetailScreen from './RestaurantDetailScreen'
import RestaurantsScreen from './RestaurantsScreen'
import OrdersScreen from '../orders/OrdersScreen'
import EditOrderScreen from '../orders/EditOrderScreen'

const Stack = createNativeStackNavigator()

export default function RestaurantsStack () {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name='RestaurantsScreen'
        component={RestaurantsScreen}
        options={{
          title: 'My Restaurants'
        }} />
      <Stack.Screen
        name='RestaurantDetailScreen'
        component={RestaurantDetailScreen}
        options={{
          title: 'Restaurant Detail'
        }} />
      <Stack.Screen
        name='CreateRestaurantScreen'
        component={CreateRestaurantScreen}
        options={{
          title: 'Create Restaurant'
        }} />
```

```
      <Stack.Screen
      name='CreateProductScreen'
      component={CreateProductScreen}
      options={{
        title: 'Create Product'
      }} />
      <Stack.Screen
      name='EditRestaurantScreen'
      component={EditRestaurantScreen}
      options={{
        title: 'Edit Restaurant'
      }} />
      <Stack.Screen
      name='EditProductScreen'
      component={EditProductScreen}
      options={{
        title: 'Edit Product'
      }} />
      <Stack.Screen
      name='OrdersScreen'
      component={OrdersScreen}
      options={{
        title: 'Restaurant Orders'
      }} />
      <Stack.Screen
      name='EditOrdersScreen'
      component={EditOrderScreen}
      options={{
        title: 'Edit order'
      }} />

    </Stack.Navigator>
  )
}
```