

FrontendMayo25Schedules.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**

Ayudas hasta el 40%

MÁSTER EN

**Inteligencia Artificial
y Ciencia de Datos**

ONLINE

Estudia el máster líder en inteligencia
artificial y ciencia de datos

**¡ÚLTIMAS
PLAZAS!**

EOI Escuela de
organización
industrial

Info y descuentos



GUÍA TUS MÚSCULOS. Y TUS NEURONAS.

EL FITNESS TAMBIÉN ES MENTAL.

FrontendMayo25Schedules

Examen DeliverUS - Modelo B - Horarios de Productos (Schedules)

Recuerde que DeliverUS está descrito en: <https://github.com/IISII2-IS-2025>

Enunciado del examen

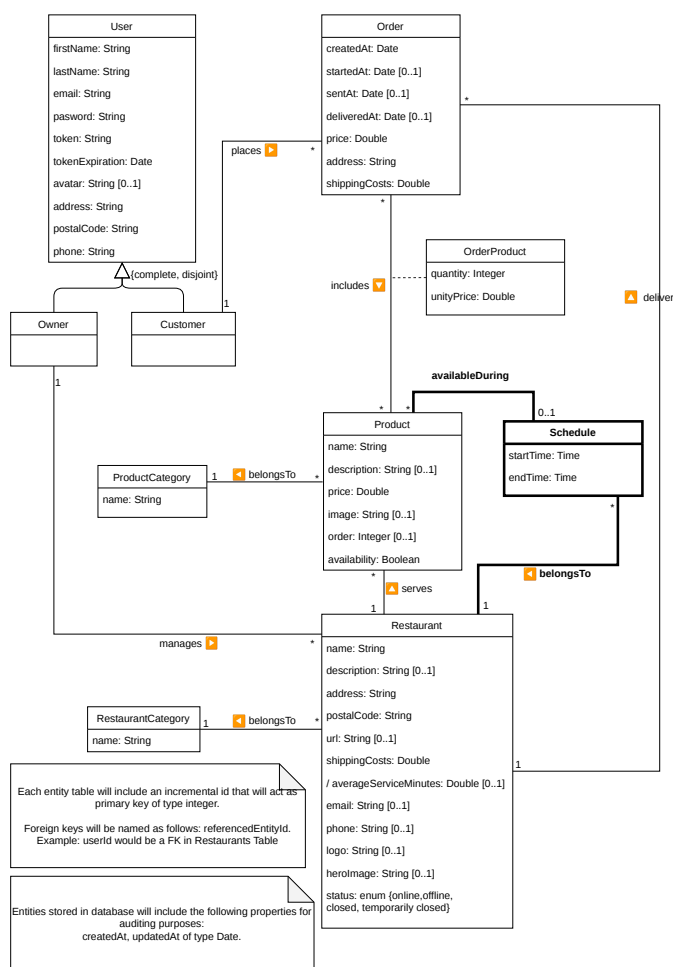
Se ha de implementar la interfaz gráfica de algunos requisitos funcionales de los propietarios, en concreto relacionados con la **gestión de horarios de los productos de los restaurantes**.

¿Qué son los horarios (Schedules)?

En DeliverUS, los horarios (Schedules) representan intervalos concretos de tiempo (con hora de inicio y hora de fin en formato HH:mm:ss) en los que ciertos productos de un restaurante están disponibles para pedidos. Esto permite definir claramente cuándo se pueden realizar pedidos de ciertos productos.

Los horarios son útiles, por ejemplo, para ofrecer menús específicos de desayuno, almuerzo o cena, y para establecer horas concretas en las que ciertos productos pueden estar disponibles.

Cada propietario de restaurante podrá definir los horarios para cada uno de sus restaurantes, y por lo tanto, cada producto se podrá asociar a uno de los horarios registrados para el restaurante al que pertenece. Se nos ha proporcionado el siguiente modelado conceptual:



Se le proporciona un backend con los endpoints siguientes ya implementados:



ENTRENA TU MENTE

Obtener el detalle de restaurante (incluye los productos con su horario)

- `GET /restaurants/:restaurantId`

Ejemplo de respuesta:

```
{
  id: 1,
  name: "Casa Félix",
  // ... resto de propiedades de restaurante
  products:
  [
    {
      id: 1,
      name: "Ensaladilla",
      // ... resto de propiedades de product
      schedule: {
        id: 2,
        startTime: "12:00:00",
        endTime: "15:00:00",
        restaurantId: 1,
        createdAt: "2025-05-14T08:48:12.000Z",
        updatedAt: "2025-05-14T08:48:12.000Z"
      }
    },
    {
      id: 2,
      name: "Olives",
      description: "Homemade",
      // ... resto de propiedades de product
      schedule: {
        id: 2,
        startTime: "12:00:00",
        endTime: "15:00:00",
        restaurantId: 1,
        createdAt: "2025-05-14T08:48:12.000Z",
        updatedAt: "2025-05-14T08:48:12.000Z"
      }
    }
  ]
  // ... otros productos del restaurante
}
```

Obtener todos los horarios de un restaurante (cada horario incluye los productos ya asociados a él)

- `GET /restaurants/:restaurantId/schedules`

Ejemplo de respuesta:

```
[
  {
    id: 1,
    startTime: "08:00:00",
    endTime: "11:00:00",
    restaurantId: 1,
    createdAt: "2025-05-14T08:48:12.000Z",
    updatedAt: "2025-05-14T08:48:12.000Z",
    products: [
      {
        id: 6,
```

```

      name: "Coffee",
      // ... resto de propiedades del producto
    },
    {
      id: 8,
      name: "Water",
      // ... resto de propiedades del producto
    }
  ]
  // ... resto de productos asociados al horario
]
},
{
  id: 2,
  startTime: "12:00:00",
  endTime: "15:00:00",
  restaurantId: 1,
  createdAt: "2025-05-14T08:48:12.000Z",
  updatedAt: "2025-05-14T08:48:12.000Z",
  products: [
    {
      id: 1,
      name: "Ensaladilla",
      // ... resto de propiedades del producto
    },
    {
      id: 2,
      name: "Olives",
      // ... resto de propiedades del producto
    }
  ]
  // ... resto de productos asociados al horario
]
}
// ... resto de horarios del restaurante
]

```

Actualizar un horario existente

- `PUT /restaurants/:restaurantId/schedules/:scheduleId`

Recibe un objeto json en el cuerpo de la petición con la forma del siguiente ejemplo:

```

{
  startTime: '08:00:00',
  stopTime: '12:00:00'
}

```

Eliminar un horario existente

- `DELETE /restaurants/:restaurantId/schedules/:scheduleId`

Actualizar un producto (asignar o desasignar horario)

- `PUT /products/:productId`

Recibe un objeto json en el cuerpo de la petición con la forma del siguiente ejemplo:

```

{
  name: "Ensaladilla",
  description: "Tuna salad with mayonnaise",
  price: 2.5,

```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins?

Plan Turbo: barato

Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

```
order: 1,  
productCategoryId: 1,  
availability: true,  
scheduleId: 2 // null para desasignar  
}
```

Es necesaria la implementación de la parte de frontend de los siguientes requisitos funcionales:

RF.01 – Visualización de productos con sus horarios asignados

Como

propietario,

Quiero

ver un listado con todos los productos de uno de mis restaurantes, incluyendo el horario que tengan asignado (si lo hubiera),

Para

conocer fácilmente su disponibilidad en función del horario.

Pruebas de aceptación

- El listado de productos se muestra en `RestaurantDetailScreen.js`.
- Para cada producto se muestra la hora de inicio y de fin del horario asignado.
- Si un producto no tiene horario asignado, debe mostrarse "Not scheduled".

RF.02 – Visualización de horarios del restaurante y número de productos asociados

Como

propietario,

Quiero

ver un listado de todos los horarios definidos para un restaurante, junto con el número de productos que están asignados a cada horario,

Para

gestionar mejor las franjas horarias en función del uso real que tienen.

Pruebas de aceptación

- La pantalla `RestaurantSchedulesScreen.js` muestra los horarios del restaurante con:
 - Hora de inicio
 - Hora de fin
 - Número de productos asociados
- Cada horario incluye un botón para editar y otro para eliminarlo.
- Existe un botón en `RestaurantDetailScreen.js` para acceder a esta pantalla.

RF.03 – Borrado de horario de restaurante

Como

propietario,

Quiero

borrar los horarios de mis restaurantes,

Para

ajustar los horarios disponibles asignables a los productos de mis restaurantes.

Pruebas de aceptación

- Se muestra un botón de eliminación en el listado de horarios `RestaurantSchedulesScreen.js` para cada horario.
- Si se desea eliminar un horario, se muestra una alerta de confirmación.

WUOLAH

- Tras eliminar un horario, se navega a `RestaurantDetailScreen.js` con el listado de productos actualizado.
-

RF.04 – Creación de horario para restaurante (se le proporciona ya implementado)

Como

propietario,

Quiero

crear un nuevo horario indicando hora de inicio y fin,

Para

establecer la disponibilidad de productos en mi restaurante.

Pruebas de aceptación

- El formulario está implementado en `CreateScheduleScreen.js`.
 - Los campos son obligatorios y deben cumplir:
 - Hora de inicio y fin en formato `HH:mm:ss`.
 - Si algún campo no cumple la validación, se muestra un mensaje de error adecuado.
 - Al guardar correctamente, se redirige al listado de horarios y se muestra un mensaje de éxito.
 - Tras la creación de un horario, se navega a `RestaurantDetailScreen.js`.
-

RF.05 – Edición de un horario existente

Como

propietario,

Quiero

poder editar un horario previamente creado en el restaurante,

Para

corregir errores o ajustar la disponibilidad.

Pruebas de aceptación

- El formulario está implementado en `EditScheduleScreen.js`.
 - Los campos se precargan con los valores actuales del horario.
 - Se validan los mismos criterios que en la creación.
 - Si la validación falla, se muestra el error correspondiente.
 - Tras editar un horario, se navega a `RestaurantDetailScreen.js` con el listado de productos actualizado.
-

RF.06 – Asignación o desasignar de horario en productos

Como

propietario,

Quiero

asignar uno de los horarios existentes a un producto, o eliminar un horario ya asignado,

Para

modificar la disponibilidad del producto en función de las necesidades.

Pruebas de aceptación

- La funcionalidad debe estar implementada en `EditProductScreen.js`.
- Se muestra una lista desplegable con los horarios disponibles del restaurante.
- Puede seleccionarse un horario existente o establecer que no haya horario asignado.

SALÓN DEL MOTOR DE SEVILLA

DEL **29 OCT**
AL **2 NOV**
FIBES

El futuro está en marcha

ORGANIZA:



PROMUEVEN:



FINANCIA:



COLABORA:



PATROCINAN:



salondelmotorsevilla.com

#salonmotor2025

- Tras editar un producto, se navega a `RestaurantDetailScreen.js` con el listado de productos actualizado.
-

Ejercicios

1. Listado de productos con horarios. 1,5 puntos

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

CUIDA TUS MÚSCULOS. Y TUS NEURONAS.

EL FITNESS TAMBIÉN ES MENTAL.

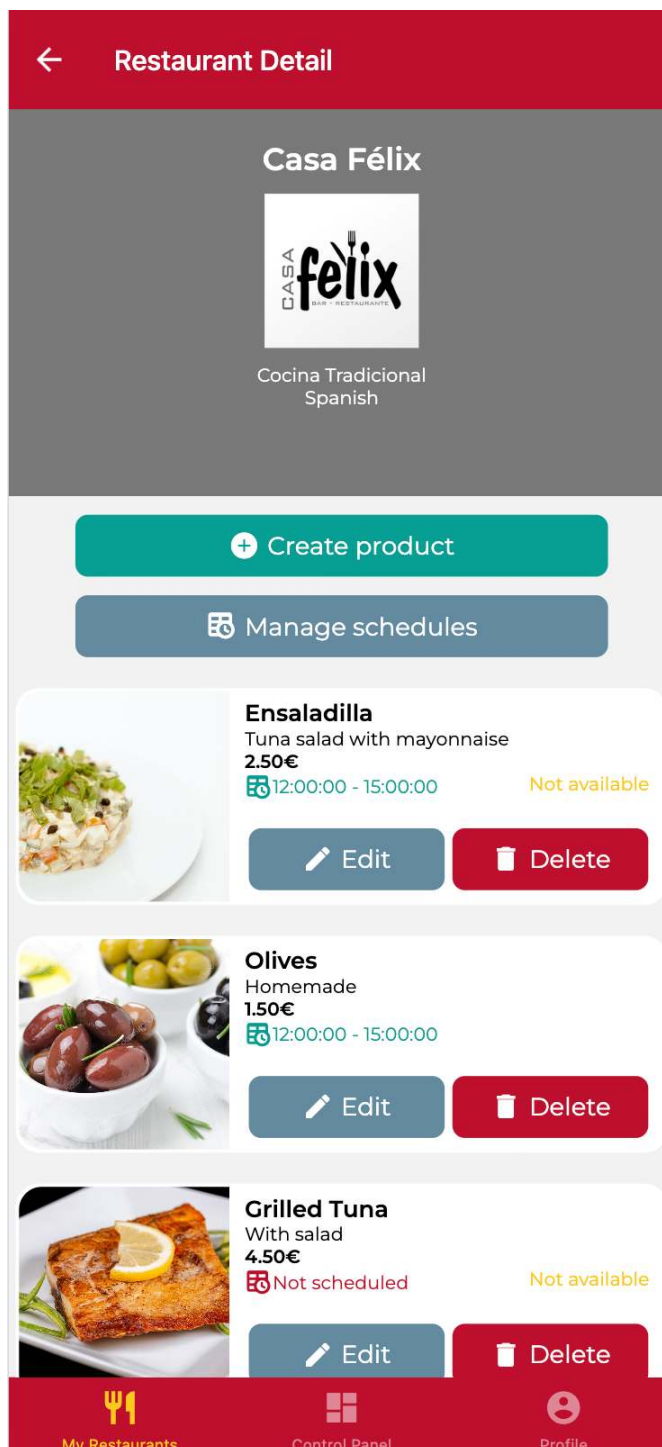


Figura 1: listado de productos con horarios

Aspectos a tener en cuenta:

- Modificar la pantalla existente para incluir el horario asignado (si lo hay) de cada producto.
- Mostrar el horario en formato legible: Hora de inicio - Hora de fin.
- Si el producto no tiene horario, mostrar el mensaje "Not Scheduled".



ENTRENA TU MENTE

FrontendMayo25Schedules

- El nombre del `MaterialCommunityIcon` del icono junto al horario (y del botón del próximo ejercicios) es `timetable`

2. Gestión de horarios. 2,5 puntos

Implemente el RF.02 en el fichero `./DeliverUS-Frontend-Owner/src/screens/restaurants/RestaurantSchedulesScreen.js` realizando todos los cambios necesarios para mostrar una interfaz como se muestra en la Figura 2.

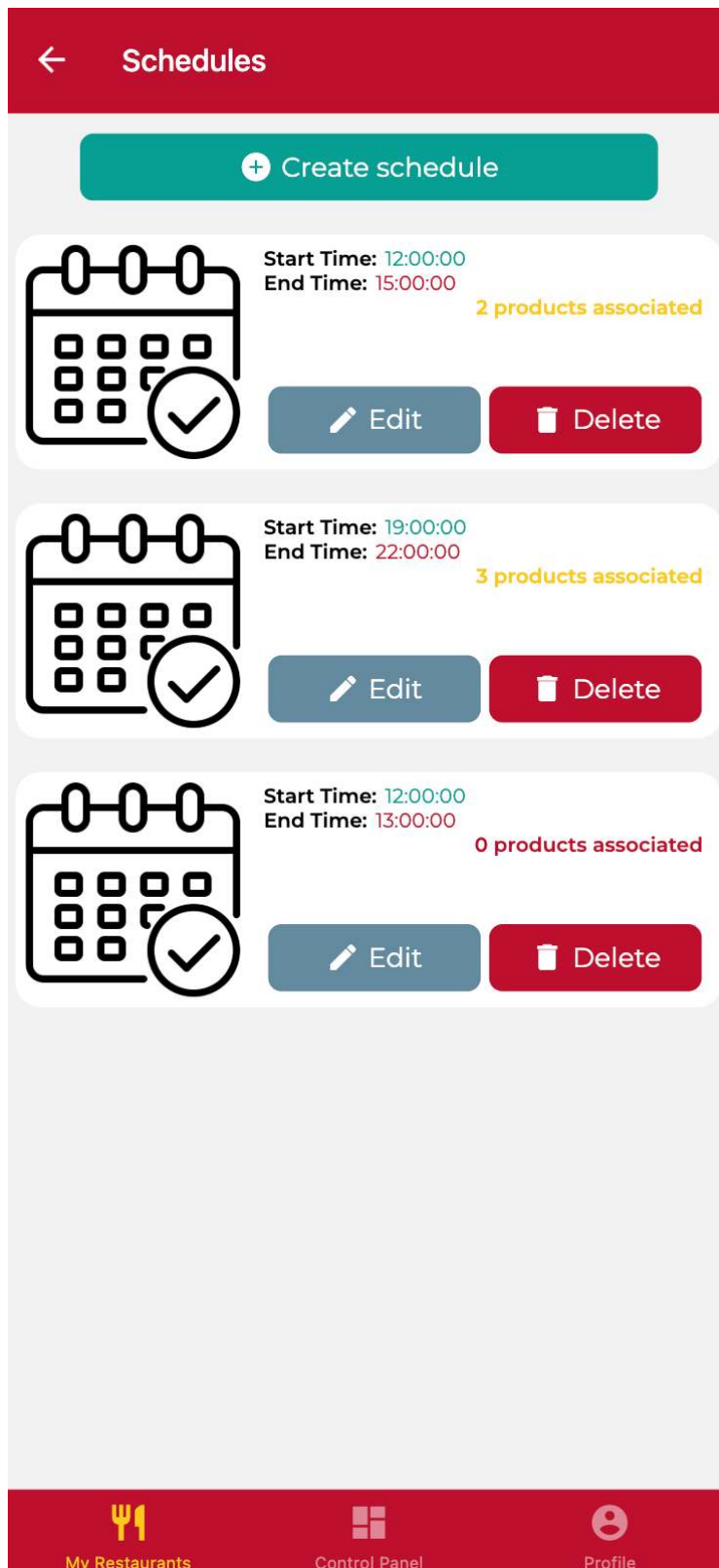


Figura 2: listado de horarios

Aspectos a tener en cuenta:

- Mostrar el listado de horarios con el número de productos asociados.
- Incorporar botones de edición y borrado para cada horario.
- El listado de horarios puede obtenerse a través de la función `getRestaurantSchedules (id)` que ya se le proporciona en `./DeliverUS-Frontend-Owner/src/api/RestaurantEndpoints.js` y que realiza la petición necesaria a backend.

3. Borrado de horarios. 1 punto

Implemente el RF.03 en el fichero `./DeliverUS-Frontend-Owner/src/screens/restaurants/RestaurantSchedulesScreen.js` realizando todos los cambios necesarios.

Aspectos a tener en cuenta:

- Use el componente `DeleteModal` para mostrar pedir confirmación de borrado.
- El borrado de horario puede completarse llamando a la función `removeSchedule (restaurantId, scheduleId)` que ya se le proporciona en `./DeliverUS-Frontend-Owner/src/api/RestaurantEndpoints.js` y que realiza la petición necesaria a backend.

4. Edición de horario. 2,5 puntos

Implemente el RF.05 en el fichero `./DeliverUS-Frontend-Owner/src/screens/restaurants/EditScheduleScreen.js` realizando todos los cambios necesarios para mostrar una interfaz como se muestra en la Figura 3.

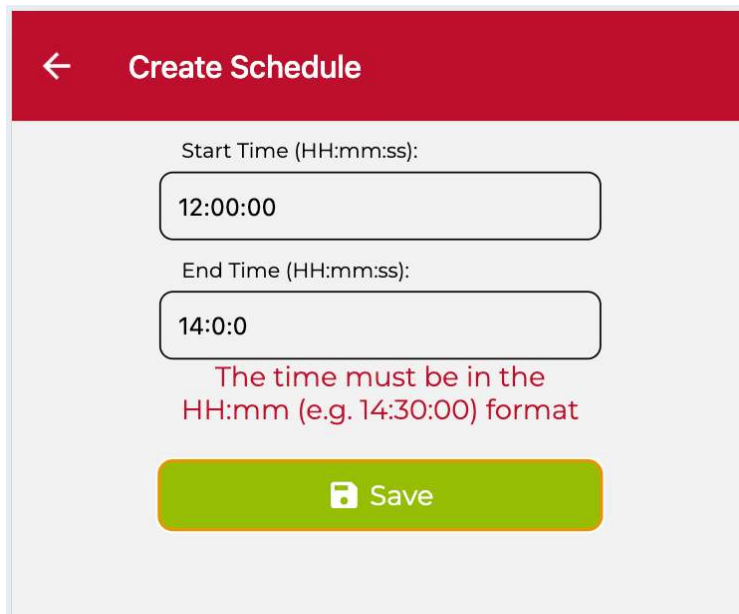


Figura 3: formulario de edición de horario

Aspectos a tener en cuenta:

- Precargar los valores actuales del horario.
- Debe incorporar lo necesario para habilitar la navegación desde la pantalla de gestión de horarios.
- La edición de horario puede realizarse llamando a la función `updateSchedule (restaurantId, scheduleId, data)` que ya se le proporciona en `./DeliverUS-Frontend-Owner/src/api/RestaurantEndpoints.js` y que realiza la petición necesaria a backend.

5. Asignar o desasignar horario a producto. 2,5 punto

Implemente el RF.06 en el fichero `./DeliverUS-Frontend-Owner/src/screens/restaurants/EditProductScreen.js` realizando todos los cambios necesarios para mostrar una interfaz como se muestra en la Figura 4.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito concentración

ali ali oohh
esto con 1 coin me
lo quito yo...

WUOLAH

← Edit Product

Name:
Ensaladilla

Description:
Tuna salad with mayonnaise

Price:
2.5

Order/position to be rendered:
1

Product category:
Starters

Schedule:
Not scheduled
Not scheduled
12:00:00 - 15:00:00
12:00:00 - 13:00:00

Save

Figura 4: formulario de edición de producto incorporando asignación de horario

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

WUOLAH

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:linux
```

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 desees depurar, y usa las herramientas de depuración del navegador.

SOLUCIONES:

El proyecto base utilizado: <https://github.com/IISSI2-IS-2025/ExLab-Frontend-Curso-Schedules>

EJERCICIO1:

En `RestaurantDetailScreen.js` añadimos la información de los horarios en cada producto dentro de la función `renderProduct`, antes de los botones:

```
<View style={{ flexDirection: 'row', justifyContent: 'space-between' }}>
  {item.schedule
    ? <View style={{ flexDirection: 'row' }}>
      <MaterialCommunityIcons name='timetable' color={GlobalStyles.brandGreen} size={20}/>
      <TextRegular textStyle={{ color: GlobalStyles.brandGreen }}>{item.schedule.startTime} - {item.schedule.endTime}</TextRegular>
    </View>
    : <View style={{ flexDirection: 'row' }}>
      <MaterialCommunityIcons name='timetable' color={GlobalStyles.brandPrimary} size={20}/>
      <TextRegular textStyle={{ color: GlobalStyles.brandPrimary }}>Not Scheduled</TextRegular>
    </View>
  }
</View>
```



```

    {item.availability &&
    <TextRegular textStyle={styles.availability}>Not available</TextRegular>
    }
  </View>

```

EJERCICIO2:

En `RestaurantSchedulesScreen.js` añadimos la información de cada horario como se muestra en la figura:

Primero tenemos que hacer la función para obtener los diferentes horarios:

```

const fetchSchedules = async () => {
  try {
    const fetchedSchedules = await getRestaurantSchedules(route.params.id)
    setSchedules(fetchedSchedules)
  } catch (error) {
    showMessage({
      message: 'There was an error while retrieving schedules. ${error}',
      type: 'error',
      style: GlobalStyles.flashStyle,
      titleStyle: GlobalStyles.flashTextStyle
    })
  }
}

```

Completamos la función `renderSchedule` añadiendo las propiedades `startTime`, `endTime` y los botones

```

const renderSchedule = ({ item }) => {
  return (
    <ImageCard
      imageUrl={scheduleIcon}
      title={item.name}
      onPress={() => {
        navigation.navigate('EditScheduleScreen', { restaurantId: item.id, scheduleId: item.schedule.id })
      }}
    >
    { /* TODO: mostrar los datos del horario */ }
    <TextSemiBold>Start Time: <TextRegular textStyle={{ color: GlobalStyles.brandGreen }}>{item.startTime}</TextRegular></TextSemiBold>
    <TextSemiBold>End Time: <TextRegular textStyle={{ color: GlobalStyles.brandPrimary }}>{item.endTime}</TextRegular></TextSemiBold>
    <View style={{ flex: 1, flexDirection: 'row', justifyContent: 'flex-end' }}>
      {item.products.length !== 0
        ? <TextSemiBold textStyle={{ color: GlobalStyles.brandSecondary }}>{item.products.length} products associated</TextSemiBold>
        : <TextSemiBold textStyle={{ color: GlobalStyles.brandPrimary }}>{item.products.length} products associated</TextSemiBold>}
    </View>

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

```

CUIDA TUS MÚSCULOS. Y TUS NEURONAS.

EL FITNESS TAMBIÉN ES MENTAL.

Promoción válida hasta el 31/12/2025 en Dreamfit Sevilla presentando esta imagen. No acumulable a otras promociones. Regalo aplicable a la cuota de inscripción.

```

        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>

<Pressable
  onPress={() => { setScheduleToBeDeleted(item) }}
  style={{ pressed }} => [
    {
      backgroundColor: pressed
      ? GlobalStyles.brandPrimaryTap
      : GlobalStyles.brandPrimary
    },
    styles.actionButton
  ]])>
<View style={{ flex: 1, flexDirection: 'row', justifyContent: 'center' }}>
  <MaterialCommunityIcons name='delete' color='white' size={20}/>
  <TextRegular textStyle={styles.text}>
    Delete
  </TextRegular>
</View>
</Pressable>
</View>

</ImageCard>
)
}

```

EJERCICIO 3

A través del botón que hemos creado antes de Delete hacemos que se pueda borrar dicho error, para ello:

Creamos la constante:

```
const [scheduleToBeDeleted, setScheduleToBeDeleted] = useState(null)
```

Y la función, que llama a la función ya creada en endpoints:

```

const remove = async (schedule) => {
  try {
    await removeSchedule(schedule.restaurantId, schedule.id)
    await fetchSchedules()
    setScheduleToBeDeleted(null)
    showMessage({
      message: `Shedule ${schedule.id} succesfully removed`,

```



ENTRENA TU MENTE

```

    type: 'success',
    style: GlobalStyles.flashStyle,
    titleStyle: GlobalStyles.flashTextStyle
  })
} catch (error) {
  console.log(error)
  setScheduleToBeDeleted(null)
  showMessage({
    message: `Restaurant ${schedule.id} could not be removed.`,
    type: 'error',
    style: GlobalStyles.flashStyle,
    titleStyle: GlobalStyles.flashTextStyle
  })
}
}
}

```

Y por último añadimos el delete Modal:

```

return (
  <>
  <FlatList
    style={styles.container}
    data={schedules}
    renderItem={renderSchedule}
    keyExtractor={item => item.id.toString()}
    ListHeaderComponent={renderHeader}
    ListEmptyComponent={renderEmptySchedulesList}
  />
  <DeleteModal
    isVisible={scheduleToBeDeleted !== null}
    onCancel={() => setScheduleToBeDeleted(null)}
    onConfirm={() => remove(scheduleToBeDeleted)}>
    <TextRegular>The shedule will be deleted </TextRegular>
  </DeleteModal>
</>
)
}

```

Todo esto podemos copiar el formato de RestaurantScreen.

Ejercicio4

Trabajamos en `EditScheduleScreen` pero antes tenemos que añadirla al stack:

```

<Stack.Screen
  name='EditScheduleScreen'
  component={EditScheduleScreen}
  options={{
    title: 'Edit Schedule'
  }} />

```

Para hacer `EditScheduleScreen` tomamos referencia de `CreateScheduleScreen` y también de otra pantalla de edición como `EditProductScreen`

Añadimos las constantes:

```
const [backendErrors, setBackendErrors] = useState()
const [schedule, setSchedule] = useState()
const [initialScheduleValues, setInitialScheduleValues] = useState({ startTime: null, endTime: null })
const validationSchema = yup.object().shape({
  startTime: yup
    .string()
    .required('Start time is required')
    .matches(
      /^[01]\d|2[0-3]:([0-5]\d):([0-5]\d)$/,
      'The time must be in the HH:mm (e.g. 14:30:00) format'
    ),
  endTime: yup
    .string()
    .required('End time is required')
    .matches(
      /^[01]\d|2[0-3]:([0-5]\d):([0-5]\d)$/,
      'The time must be in the HH:mm (e.g. 14:30:00) format'
    )
})
```

El useEffect:

```
useEffect(() => {
  async function fetchScheduleDetail () {
    try {
      const fetchedSchedule = await getRestaurantSchedules(route.params.restaurantId, route.params.scheduleId)
      setSchedule(fetchedSchedule)
      const initialValues = buildInitialValues(fetchedSchedule, initialScheduleValues)
      setInitialScheduleValues(initialValues)
    } catch (error) {
      showMessage({
        message: `There was an error while retrieving schedule details (id ${route.params.id}). ${error}`,
        type: 'error',
        style: GlobalStyles.flashStyle,
        titleStyle: GlobalStyles.flashTextStyle
      })
    }
  }
  fetchScheduleDetail()
}, [route])
```

La función update:

```
const update = async (values) => {
  setBackendErrors([])
  try {
    const updatedSchedule = await updateSchedule(schedule.restaurantId, schedule.id, values)
    showMessage({
      message: `Schedule ${updatedSchedule.startTime} - ${updatedSchedule.endTime} succesfully updated`,
      type: 'success',
      style: GlobalStyles.flashStyle,
      titleStyle: GlobalStyles.flashTextStyle
    })
  }
}
```

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato
→ Planes pro: más coins

pierdo espacio



Necesito
concentración

ali ali ooh
esto con 1 coin me
lo quito yo...

WUOLAH

```
})  
navigation.navigate('RestaurantDetailScreen', { id: schedule.restaurantId })  
} catch (error) {  
  console.log(error)  
  setBackendErrors(error.errors)  
}  
}
```

Y el return:

```
return (<>  
  <Formik  
    enableReinitialize  
    validationSchema={validationSchema}  
    initialValues={initialScheduleValues}  
    onSubmit={update}>  
    {{{ handleSubmit, setFieldValue, values }} => (  
      <ScrollView>  
        <View style={{ alignItems: 'center' }}>  
          <View style={{ width: '60%' }}>  
            <InputItem  
              name='startTime'  
              label='Start Time (HH:mm:ss)'  
            />  
            <InputItem  
              name='endTime'  
              label='End Time (HH:mm:ss)'  
            />  
  
            {backendErrors &&  
              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>  
        </ScrollView>  
      )  
    </Formik>  
  </>)
```


EJERCICIO 5

Por último primero hacemos un cambio en los endpoints:

```
const getRestaurantSchedule = async (restaurantId, scheduleId) => {  
  return (await getRestaurantSchedules(restaurantId)).find(schedule => schedule.id === scheduleId)  
}
```

Y seguimos con `EditProductScreen.js` para añadir el `DropDownPicker`

Añadimos estas constantes:

```
const [openSchedule, setOpenSchedule] = useState(false)
```

```
const [scheduleOptions, setScheduleOptions] = useState([])
```

Añadimos `scheduleId` a los valores iniciales:

```
const [initialProductValues, setInitialProductValues] = useState({ name: null, description: null, price: null, order: null, productCategoryId: null, availability: null, image: null, scheduleId: null })
```

```
scheduleId: yup  
  .number()  
  .nullable()  
  .optional()  
  .positive()
```

El `useEffect` para los schedules:

```
useEffect(() => {  
  async function fetchRestaurantSchedules () {  
    try {  
      const fetchedRestaurantSchedules = await getRestaurantSchedules(product.restaurantId)  
      const fetchedProductSchedulesOptions = fetchedRestaurantSchedules.map((schedule) => {  
        return {  
          label: `${schedule.startTime} - ${schedule.endTime}`,  
          value: schedule.id  
        }  
      })  
      setScheduleOptions(fetchedProductSchedulesOptions)  
    } catch (error) {  
      showMessage({  
        message: `There was an error while retrieving restaurant schedules. ${error}`,  
        type: 'error',  
        style: GlobalStyles.flashStyle,  
        titleStyle: GlobalStyles.flashTextStyle  
      })  
    }  
  }  
})
```

```
if (product) { fetchRestaurantSchedules() }  
}, [product])
```

Y el dropdown

```
<TextRegular textStyle={styles.textLabel}>Schedule: </TextRegular>  
<DropDownPicker  
  open={openSchedule}  
  value={values.scheduleId}  
  items={[  
    { label: 'Not scheduled', value: null },  
    ...scheduleOptions  
  ]}  
  setOpen={setOpenSchedule}  
  onSelectItem={item => {  
    setFieldValue('scheduleId', item.value)  
  }}  
  setItems={setScheduleOptions}  
  placeholder="Not scheduled"  
  containerStyle={{ height: 40, marginBottom: 10 }}  
  style={{ backgroundColor: GlobalStyles.brandBackground }}  
  dropDownStyle={{ backgroundColor: '#fafafa' }}  
</>  
<ErrorMessage name={'scheduleId'} render={msg => <TextError>{msg}</TextError> }/>
```