

AC7.1 – aplicación de comida rápida con Redux Toolkit

Mónica Garrigós Sánchez

Sección 1: Redux Toolkit configurado

En esta sección se muestra que Redux Toolkit fue configurado correctamente con configureStore, y que el store fue conectado globalmente usando el componente Provider.

```
store.ts M × main.tsx M menuSlice.ts 1, U firebaseService.ts U orderSlice.ts 1, U
sesion7 > AC7.1 - Redux Toolkit > src > redux > store.ts > ...
1 import { configureStore } from '@reduxjs/toolkit'
2 import menuReducer from './menuSlice'
3 import orderReducer from './orderSlice'
4
5 // * Se configura el store global con Redux Toolkit usando configureStore.
6 // * Aquí se registran los reducers del menú y de los pedidos.
7
8 export const store = configureStore({
9   reducer: {
10     menu: menuReducer,
11     orders: orderReducer
12   }
13 })
14
15 export type RootState = ReturnType<typeof store.getState>
16 export type AppDispatch = typeof store.dispatch
17 |
```

```
store.ts M main.tsx M × menuSlice.ts 1, U firebaseService.ts U orderSlice.ts 1, U
sesion7 > AC7.1 - Redux Toolkit > src > main.tsx
1 import './index.css'
2 import App from './App.tsx'
3 import { Provider } from 'react-redux'
4 import { store } from './redux/store'
5
6 // * Se conecta Redux Toolkit a la aplicación con <Provider store={store}>
7
8 <Provider store={store}>
9   <App />
10 </Provider>
11
```

Sección 2: Slices de Redux implementados

Aquí se muestran los slices *menuSlice* y *orderSlice*. *MenuSlice* gestiona los elementos del menú (actualización de cantidades). *OrderSlice* gestiona los pedidos usando *createAsyncThunk*, incluyendo *addOrder* y *getOrders*.

```
session7 > AC7.1 - Redux Toolkit > src > redux > menuSlice.ts > default
1 import { createSlice, PayloadAction } from '@reduxjs/toolkit'
2 import { MenuItem } from '../entities/entities'
3
4 // * Slice para el menú con acciones setMenu y updateQuantity
5
6 const initialState: MenuItem[] = []
7 const menuSlice = createSlice({
8   name: 'menu',
9   initialState,
10  reducers: {
11    setMenu: (state, action: PayloadAction<MenuItem[]>) => { 'state' is declared but its
12      return action.payload
13    },
14    updateQuantity: (state, action: PayloadAction<{ id: number; quantity: number }>) => {
15      const item = state.find(i => i.id === action.payload.id)
16      if (item) {
17        item.quantity = action.payload.quantity
18      }
19    }
20  }
21 })
22
23 export const { setMenu, updateQuantity } = menuSlice.actions
24 export default menuSlice.reducer
```

```
session7 > AC7.1 - Redux Toolkit > src > redux > orderSlice.ts > ...
1 import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'
2 import { saveOrder, getOrders as fetchOrders } from '../services/firebaseService'
3
4 // * Slice para manejar pedidos (orders)
5 // * Usa createAsyncThunk para manejar peticiones asincronas a Firebase.
6 // * Se gestionan los estados: loading, succeeded, failed
7
8 export interface Order {
9   id?: string
10  name: string
11  phone: string
12  quantity: number
13  totalPrice: number
14  date: string
15  customerName: string
16  product: number
17 }
18
19 interface OrderState {
20   list: Order[]
21   status: 'idle' | 'loading' | 'succeeded' | 'failed'
22   error: string | null
23 }
24
25 const initialState: OrderState = {
26   list: [],
27   status: 'idle',
28   error: null
29 }
30
31 // Guardar un pedido
32 export const addOrder = createAsyncThunk(
33   'orders/addOrder',
34   async (order: Order, thunkAPI) => { 'thunkAPI' is declared but its value is ne
35     const id = await saveOrder(order)
```

Sección 3: Firebase como backend (persistencia)

Las operaciones CRUD se conectan a Firebase Firestore. Se usa `saveOrder()` para guardar pedidos y `getOrders()` para recuperarlos y mostrarlos.

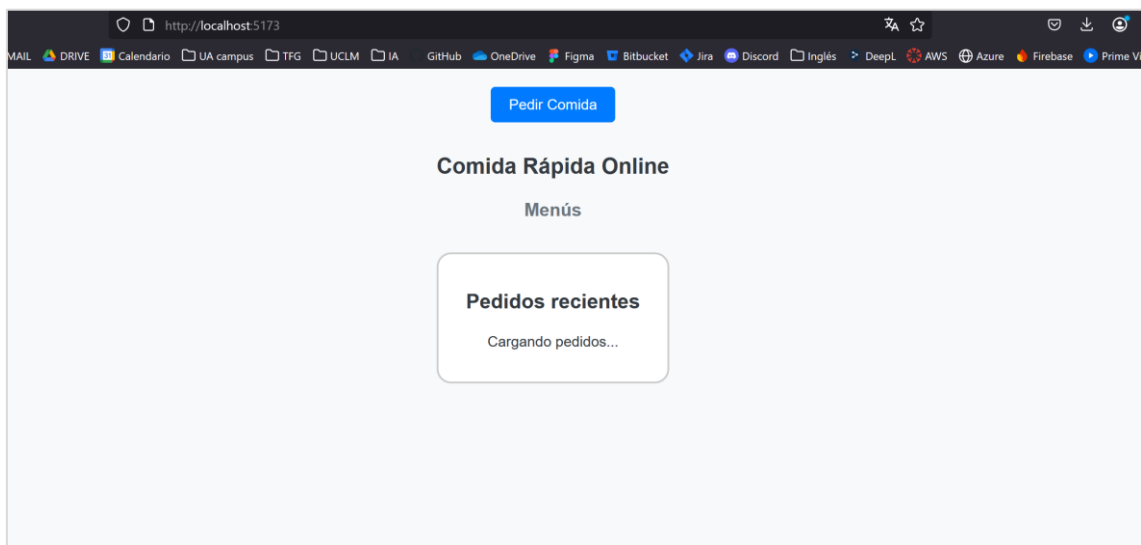
```

store.ts M  main.tsx M  menuSlice.ts 1, U  orderSlice.ts 1, U  firebaseService.ts U X
sesion7 > AC7.1 - Redux Toolkit > src > services > firebaseService.ts > ...
1  import { getFirestore, collection, addDoc, getDocs } from "firebase/firestore"
2  import { app } from "../firebaseConfig"
3  import { DocumentData } from 'firebase/firestore'
4  import { Order } from '../redux/orderSlice'
5
6  // * Integración de Firebase para persistencia usando Firestore
7  // * Funciones: saveOrder() y getOrders()
8
9  const db = getFirestore(app)
10 const ordersCollection = collection(db, "orders")
11
12 // Guardar un pedido
13 export async function saveOrder(order: any) {
14   try {
15     const docRef = await addDoc(ordersCollection, order)
16     return docRef.id
17   } catch (error) {
18     console.error("Error al guardar el pedido:", error)
19     throw error
20   }
21 }
22
23 // Obtener todos los pedidos
24 export async function getOrders(): Promise<Order[]> {
25   const querySnapshot = await getDocs(ordersCollection)
26   return querySnapshot.docs.map(doc => {
27     const data = doc.data() as DocumentData
28     return {
29       id: doc.id,
30       name: data.name,
31       phone: data.phone,
32       quantity: data.quantity,
33       totalPrice: data.totalPrice,
34       date: data.date,
35       customerName: data.customerName,
36       product: data.product
37     }
38   })
39 }
40
41

```

Sección 4: Guardado de pedidos con feedback

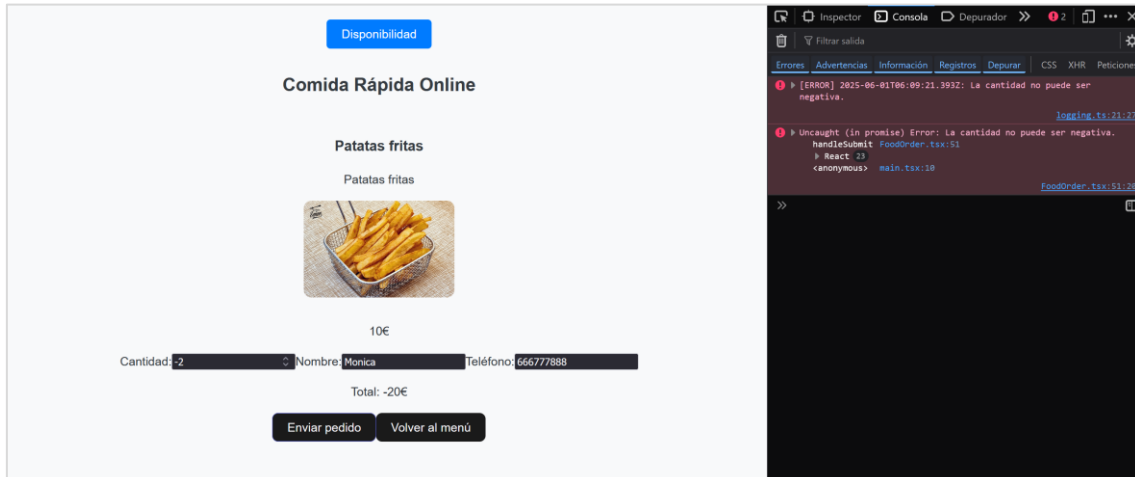
Al enviar un pedido, se muestra retroalimentación visual usando el estado *orderStatus*. El pedido se guarda con *dispatch(addOrder(...))* desde Redux.



Sección 5: Error y Logging

Se implementa un sistema de logging para registrar distintos niveles de eventos (debug, info, warning, error). Además, se usó un componente *ErrorBoundary* para capturar errores de ejecución y evitar que la aplicación se rompa.

En la siguiente captura de pantalla podemos observar que se lanza un error al intentar hacer un pedido con una cantidad negativa, este error se queda registrado en consola con un mensaje personalizado.



Sección 6: Visualización de pedidos

Los pedidos recientes se muestran en la sección de disponibilidad, accediendo a la lista *orders* del estado Redux. Esta lista se carga desde Firebase con *getOrders* y se renderiza en el componente *OrderLoss.tsx*.

