# Definiciones y componentes

**Redux 🡪** Es una biblioteca para la gestión del estado de aplicaciones en React. Proporciona la arquitectura para gestionar el estado de la aplicación.

**Redux toolkit 🡪** Es un conjunto de utilizades que simplifican el desarrollo con Redux. Ofrece abstracciones y herramientas que reducen la cantidad de código necesario para configurar y utilizar Redux

**Diagrama

El contenido generado por IA puede ser incorrecto.**

1. **Action**: un objeto que tiene intención de cambiar el estado. Tiene un tipo (type) que describe el cambio y, opcionalmente, datos adicionales (payload)
2. **Dispatch**: una función que se utiliza para enviar una acción al store de Redux
3. **Reducer**: una función que recibe el estado actual y una acción, y devuelve un nuevo estado. Especifican cómo se debe actualizar el estado en respuesta a las acciones
4. **Store**: un objeto que contiene el estado global de la aplicación. Es el único lugar donde está la realidad del estado
5. **Componentes**: los componentes de react se conectan al store para acceder al estado y despachar acciones. Cuando el estado cambia, los componentes se vuelven a renderizar para reflejar los cambios.

main(){

int a = f();

a += 5;

}

a+=5; 🡪 reducer

Int a 🡪 store

f() 🡪 dispatcher

**Redux Toolkit simplifica ese flujo gracias a:**

* ***createSlice***🡪 genera automáticamente los reducers y las acciones correspondientes para un slice del estado. Un slice representa una parte del estado global (ej: estado de los pedidos)
* ***createAsyncThunk* 🡪** simplifica el manejo de acciones asíncronas como las llamadas a APIs o la persistencia en Firebase. Crea acciones que representan los diferentes estados de una operación asíncrona (pendiente, exitosa, fallida) y maneja la lógica de actualización del estado en cada caso.

# Pasos a seguir práctica

1. ***fetchMenuItems* (Async Thunk) 🡪** maneja los estados pending, fulfilled y rejected automaticamente. Retorna una promesa con los datos del menu.
2. ***menuItemSlice* 🡪** define el estado inicial (ítems, loading, error).

* updateMenuItemQuantity: Reducer para actualizar la cantidad de un menú individual.
* extraReducers: Maneja las acciones de fetchMenuItems:
* pending: Establece loading: true.
* fulfilled: Establece loading: false y actualiza items con los datos.
* rejected: Establece loading: false y guarda el error en error.

1. Store 🡪 configura el store de Redux con el reductor de menuItems
2. App (componente principal) 🡪

# Cambio de Redux a Redux Toolkit

## Reconfiguración del store

Uno de los cambios clave al migrar a Redux Toolkit es la reconfiguración del store. En lugar de utilizar createStore de Redux, usarás configureStore de Redux Toolkit.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

## Refactorización de reducers con createSlice

Uno de los mayores beneficios de Redux Toolkit es la función **createSlice**, que **combina la lógica del reducer y las acciones en un solo lugar**.

Texto

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En el store cambiaria así:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

## Migración de Thunks a createAsyncThunk

Si tu aplicación utiliza thunks para **manejar operaciones asincrónicas**, Redux Toolkit ofrece *createAsyncThunk*, una forma más sencilla y estructurada de definir estas acciones.

*createAsyncThunk* gestiona automáticamente los ciclos de vida de la acción (pendiente, cumplida, rechazada) y actualiza el estado en consecuencia.

Texto

El contenido generado por IA puede ser incorrecto.

* Instalar dependencia de Redux 🡪 *@reduxjs/toolkit y react-redux*
* La función ***createAsyncThunk*** crea una acción asíncrona que se encarga de guardar el pedido en Firebase. Recibe dos argumentos
  + Nombre de la acción
  + Función que realiza la operación asíncrona (guardar en Firebase). Esta función recibe los datos del pedido y un objeto con utilizades como *rejectWithValue* para poder manejar los errores.
* ***createSlice*** crea un “slice” del store que contiene la lógica para manejar los pedidos. Define el estado inicial, los reducers y el nombre del slice.
* *extraReducers*: este campo se utiliza para añadir reducers para acciones que se definen fuera del slice, como las acciones asíncronas creadas con *createAsyncThunk*. En este caso, se manejan los estados pending, fulfilled y rejected de la promesa.
* ***useDispatch***: es un hook que permite obtener la función *dispatch* para despachar acciones.
* *useSelector*: es un hook que permite obtener partes del estado del store