Explicación de la UT2A4 – CRUD con acceso a base de datos

Este documento explica los contenidos nuevos del PDF de teoría que deben aprender para realizar la actividad **UT2A4 – CRUD con base de datos**.

En esta actividad se pasa de trabajar con datos locales (en memoria del navegador o estado global) a realizar operaciones **CRUD (Crear, Leer, Actualizar, Borrar)** sobre una **base de datos real** a través de un **backend (servidor)**.

Los nuevos conceptos a introducir son:
• Arquitectura cliente-servidor: el frontend ya no guarda los datos, solo los muestra.
• API REST: el backend ofrece "endpoints" para acceder a la información.
• Conexión a base de datos MySQL desde Node.js.
• Operaciones CRUD con sentencias SQL (SELECT, INSERT, UPDATE, DELETE).
• Uso de Fetch o Axios en el frontend para comunicarse con la API.
• Actualización automática de la interfaz (tabla MUI) tras cada operación.
• Buenas prácticas de seguridad y accesibilidad.

Explicaciones básicas:

1.- Diferencia entre *frontend* y *backend* (arquitectura cliente-servidor)

Cuando trabajamos con aplicaciones web modernas, distinguimos **dos partes principales**:

- **Frontend (cliente):**
  Es la parte visible, lo que el usuario ve y con lo que interactúa (formularios, botones, menús, tablas, etc.).
  En nuestro caso, el *frontend* está hecho con **React**, y se encarga de mostrar la información que recibe y de enviar las peticiones al servidor.

- **Backend (servidor):**
  Es la parte invisible, la que se encarga de **procesar los datos, acceder a la base de datos** y responder al cliente.
  Aquí utilizamos **Node.js y Express** para crear una API que se conecta con **MySQL**.

Ejemplo:
Cuando el usuario rellena un formulario para insertar un registro, el frontend manda los datos al backend, y este los guarda en la base de datos.
Después, el backend devuelve una respuesta (por ejemplo, "registro insertado correctamente") y el frontend actualiza la tabla para mostrarlo.

2.- Qué es una API y qué significa *endpoint*

Una **API** (Application Programming Interface) es un **puente de comunicación** entre el frontend y el backend.
Permite que dos partes del sistema se entiendan usando un **lenguaje común (normalmente JSON)**.

Dentro de la API, cada operación (como listar, insertar o borrar datos) se define en una **ruta especial** llamada **endpoint**.

Ejemplo:

- GET /listarArticulos → obtiene todos los artículos.
- GET /agregarArticulo?nombre=Ratón&precio=12 → inserta un artículo nuevo.
- GET /eliminarArticulo?id=5 → elimina el artículo con id=5.

El frontend hace una llamada "fetch" a estos endpoints, y el backend responde con un JSON

3.- Cómo se conecta Node.js a MySQL (librería mysql2)

Para que el backend pueda guardar o leer datos de una base de datos, necesita una conexión.
Esto se logra con la librería **mysql2**, que permite ejecutar **consultas SQL** desde Node.js de forma sencilla y segura.

Los pasos básicos son:

Instalar la librería:

```
npm install mysql2
```

Configurar las credenciales en config.js:

```
const config = {
  db: { host: 'localhost', user: 'root', password: '', database: 'bdgestion' }
}
module.exports = config
```

Crear una función genérica para ejecutar consultas:

```
const mysql = require('mysql2/promise')

const config = require('../config')


async function query(sql, params) {

 const connection = await mysql.createConnection(config.db)

 const [results] = await connection.execute(sql, params)

 return results

}

//Con esto, cualquier archivo del backend puede ejecutar consultas con await db.query(...).
```

4.- Consultas SQL básicas: SELECT, INSERT, DELETE

SQL (Structured Query Language) es el lenguaje usado para comunicarse con la base de datos.
Las operaciones CRUD se traducen así:

Acción	Significado	Ejemplo SQL
CREATE	Insertar un registro nuevo	INSERT INTO articulos (nombre, precio) VALUES ('Ratón', 12)
READ	Leer datos existentes	SELECT * FROM articulos
UPDATE	Modificar datos	UPDATE articulos SET precio=15 WHERE id=1
DELETE	Eliminar un registro	DELETE FROM articulos WHERE id=1

El backend usa estas consultas dentro de funciones JavaScript y devuelve los resultados como JSON.

5.- *Fetch* y asincronía: promesas, await y manejo de errores

El frontend necesita comunicarse con el backend para enviar o recibir datos.
Esto se hace con la función fetch(), que lanza una petición HTTP a un endpoint**.**

```
// Ejemplo de uso de fetch con await

const respuesta = await fetch('http://localhost:3030/listarArticulos')

const datos = await respuesta.json()

console.log(datos)
```

La palabra **await** significa “esperar la respuesta antes de continuar”.
Esto es necesario porque las llamadas a una API son **asíncronas** (tardan un tiempo y no bloquean la aplicación).

También se puede usar un bloque **try/catch** para manejar errores de conexión:

```
try {

  const resp = await fetch('http://localhost:3030/agregarArticulo?nombre=Ratón')

  const json = await resp.json()

  console.log(json)

} catch (err) {

  console.error('Error al conectar con la API:', err)

}
```

6.- Actualización de la tabla en React tras operaciones

Una vez que insertamos, borramos o modificamos un registro, debemos actualizar la interfaz para mostrar los datos más recientes.
Esto se logra de dos maneras:

1. Volver a hacer la consulta (SELECT) después de cada operación.
2. Actualizar directamente el estado local de React si solo cambia una fila.

Ejemplo con React:

```
async function insertarArticulo(){

  await fetch('http://localhost:3030/agregarArticulo?nombre=Teclado')

  listarArticulos() // función que hace un nuevo SELECT
```

}

Y dentro de useEffect() cargamos los datos iniciales:

```
useEffect(() => {
  listarArticulos()
}, [])
```

De esta forma, la tabla siempre muestra los datos más actuales de la base de datos.

7.- Protección de rutas y validaciones básicas

Si nuestra aplicación tiene inicio de sesión, debemos evitar que alguien entre directamente escribiendo la URL del panel sin estar autenticado.
Esto se hace verificando el estado global (Redux) y usando useEffect para redirigir al usuario.

Ejemplo:

```
import { useSelector } from 'react-redux'
import { useNavigate } from 'react-router-dom'

const { autenticado } = useSelector((s) => s.autenticador)
const navegar = useNavigate()

useEffect(() => {
  if (!autenticado) navegar('/') // redirige al login
}, [autenticado])
```

También debemos **validar los datos** antes de enviarlos:

- No dejar campos vacíos.
- Comprobar que los valores numéricos son correctos.
- Mostrar mensajes de error o advertencia si algo falta.

8.- Accesibilidad en tablas: aria-label y claves únicas

Una interfaz accesible permite que **cualquier persona**, incluso usando lectores de pantalla o teclado, pueda interactuar con la aplicación.

En las tablas de MUI (Material UI):

- **aria-label** describe la tabla para los lectores de pantalla.
- **key** (clave única) en cada fila ayuda a React a renderizar correctamente.

Ejemplo:

```
<Table aria-label="Tabla de artículos">
 <TableBody>
  {articulos.map((art) => (
   <TableRow key={art.id}>
    <TableCell>{art.nombre}</TableCell>
   </TableRow>
  ))}
 </TableBody>
</Table>
```

También es recomendable usar suficiente contraste de color y tamaño de letra legible.

9.- Buenas prácticas: separación de lógica, modularidad y seguridad

Para mantener un código limpio y profesional:

- Separar responsabilidades:
    - El frontend se encarga de la interfaz y las llamadas al servidor.
    - El backend gestiona la base de datos.
- Modularidad:
  Divide el código en archivos (por ejemplo, db.js, articulos.js, PanelArticulos.tsx).

- Evitar duplicar código:
  Si una función se repite, muévela a un archivo común.
- Seguridad básica:
  - No exponer contraseñas en el código del frontend.
  - Validar los datos que llegan al backend.
  - Evitar ejecutar consultas con texto sin sanitizar (usar parámetros ? en MySQL).

Después de las explicaciones básicas de lo que tienen en la teoría, paso a explicarles el programa básico que se debe desarrollar.

2. Arquitectura general

La aplicación ahora se divide en dos partes:

• **Frontend (React + MUI + Redux)**: maneja la interfaz y las peticiones HTTP.
• **Backend (Node.js + Express + MySQL)**: recibe las peticiones y ejecuta consultas a la base de datos.

El flujo completo de datos es el siguiente:
Formulario → fetch() → API → consulta SQL → respuesta JSON → renderización de tabla.

3. Estructura de carpetas del backend

```
backend/
│
├── index.js        → Servidor Express con endpoints de ejemplo.
├── config.js       → Configuración y credenciales de la base de datos.
├── helper.js       → Funciones auxiliares.
└── services/
    ├── db.js       → Conexión MySQL (pool).
    └── articulos.js → Consultas CRUD a la tabla de ejemplo.
```

4. Configuración mínima del backend (index.js)

```
// backend/index.js — Servidor Express básico
const express = require('express')      // Framework web para Node
```

```
const cors = require('cors')          // Permite peticiones desde el frontend
const app = express()                 // Creamos la app de Express

app.use(cors())                       // Habilita CORS
app.use(express.json())               // Permite recibir datos JSON

// Importamos las funciones CRUD del servicio articulos
const articulos = require('./services/articulos')

// Endpoint raíz (solo comprobación)
app.get('/', (req, res) => {
  res.json({ ok: true, mensaje: 'API de ejemplo conectada' })
})

// Listar artículos (GET)
app.get('/listarArticulos', async (req, res, next) => {
  try {
    const resultado = await articulos.obtenerListado()
    res.json(resultado)
  } catch (err) {
    console.error('Error al listar artículos', err.message)
    next(err)
  }
})

// Agregar artículo (GET por simplicidad)
app.get('/agregarArticulo', async (req, res, next) => {
  try {
    const filas = await articulos.insertarArticulo(req)
    res.json({ filasAfectadas: filas })
  } catch (err) {
    console.error('Error al insertar artículo', err.message)
    next(err)
  }
})

// Borrar artículo (GET con ?id=)
app.get('/eliminarArticulo', async (req, res, next) => {
  try {
    const filas = await articulos.eliminarArticulo(req)
    res.json({ filasAfectadas: filas })
  } catch (err) {
    console.error('Error al eliminar artículo', err.message)
```

```
    next(err)
  }
})

// Servidor escuchando en puerto 3030
const PUERTO = 3030
app.listen(PUERTO, () => {
  console.log(`API disponible en http://localhost:${PUERTO}`)
})
```

5. Conexión MySQL y consultas CRUD

```
// backend/services/db.js — Conexión a MySQL
const mysql = require('mysql2/promise')
const config = require('../config')

async function query(sql, params) {
  const connection = await mysql.createConnection(config.db)
  const [results] = await connection.execute(sql, params)
  return results
}
module.exports = { query }

// backend/config.js — Configuración BD
const config = {
  db: {
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'bdgestion',
  }
}
module.exports = config
```

6. Consultas CRUD – services/articulos.js

```
// backend/services/articulos.js — Consultas SQL de ejemplo
const db = require('./db')
const helper = require('../helper')
```

```
// Listar
async function obtenerListado () {
  const filas = await db.query('SELECT * FROM articulos', [])
  const data = helper.emptyOrRows(filas)
  return { data }
}

// Insertar
async function insertarArticulo (req) {
  const { nombre, marca, categoria, precio } = req.query
  const sql = 'INSERT INTO articulos (nombre, marca, categoria, precio) VALUES (?, ?, ?, ?)'
  const result = await db.query(sql, [nombre, marca, categoria, Number(precio)])
  return result.affectedRows
}

// Eliminar
async function eliminarArticulo (req) {
  const { id } = req.query
  const sql = 'DELETE FROM articulos WHERE id = ?'
  const result = await db.query(sql, [Number(id)])
  return result.affectedRows
}
module.exports = { obtenerListado, insertarArticulo, eliminarArticulo }
```

7. Frontend – ejemplo de Panel con formulario y tabla

```
// src/paginas/PanelArticulos.tsx — Ejemplo educativo
import React, { useState, useEffect } from 'react'
import { Box, Paper, TextField, Button, Table, TableHead, TableRow, TableCell, TableBody }
from '@mui/material'

export default function PanelArticulos(){
  const [articulo, setArticulo] = useState({ nombre: '', marca: '', categoria: '', precio: 0 })
  const [datos, setDatos] = useState([])

  // Cargar datos al iniciar
  useEffect(()=>{ listar() }, [])

  async function listar(){
    const resp = await fetch('http://localhost:3030/listarArticulos')
```

```
  const json = await resp.json()
  setDatos(json.data || [])
 }

 async function insertar(e){
  e.preventDefault()
  const qs = new URLSearchParams(articulo).toString()
  await fetch(`http://localhost:3030/agregarArticulo?${qs}`)
  listar() // refrescar
 }

 async function borrar(id){
  await fetch(`http://localhost:3030/eliminarArticulo?id=${id}`)
  listar()
 }

 return (
  <Box sx={{ p:2 }}>
   <Paper sx={{ p:2 }}>
    <Box component="form" onSubmit={insertar}>
     <TextField label="Nombre" value={articulo.nombre}
onChange={e=>setArticulo({...articulo, nombre:e.target.value})} fullWidth/>
     <TextField label="Marca" value={articulo.marca}
onChange={e=>setArticulo({...articulo, marca:e.target.value})} fullWidth/>
     <TextField label="Categoría" value={articulo.categoria}
onChange={e=>setArticulo({...articulo, categoria:e.target.value})} fullWidth/>
     <TextField label="Precio" type="number" value={articulo.precio}
onChange={e=>setArticulo({...articulo, precio:Number(e.target.value)})} fullWidth/>
     <Button type="submit" variant="contained" sx={{mt:2}}>Insertar</Button>
    </Box>
   </Paper>
   <Table aria-label="Tabla de artículos">

<TableHead><TableRow><TableCell>Acciones</TableCell><TableCell>Nombre</TableCell
><TableCell>Marca</TableCell><TableCell>Categoría</TableCell><TableCell>Precio</Tab
leCell></TableRow></TableHead>
    <TableBody>
     {datos.map(fila=>(
      <TableRow key={fila.id}>
       <TableCell><Button onClick={()=>borrar(fila.id)}>Borrar</Button></TableCell>
       <TableCell>{fila.nombre}</TableCell>
       <TableCell>{fila.marca}</TableCell>
       <TableCell>{fila.categoria}</TableCell>
```

```
          <TableCell>{fila.precio}</TableCell>
         </TableRow>
        ))}
      </TableBody>
    </Table>
   </Box>
  )
}
```

8. Protección de rutas y autenticación

```
// src/componentes/MenuGeneral.tsx — Redirige si el usuario no está autenticado
import React, { useEffect } from 'react'
import { useNavigate, Link } from 'react-router-dom'
import { useSelector } from 'react-redux'
import { RootState } from '../store'

export default function MenuGeneral(){
 const navegar = useNavigate()
 const { autenticado, nombreUsuario } = useSelector((s:RootState)=>s.autenticador)

 useEffect(()=>{
  if(!autenticado) navegar('/')
 }, [autenticado, navegar])

 return (
  <nav>
   <Link to="/panel">Panel</Link> | <Link to="/estadisticas">Estadísticas</Link>
   <span style={{float:'right'}}>Usuario: {nombreUsuario}</span>
  </nav>
 )
}
```

Recuerden que deben revisar bien el Código, esto es un ejemplo, no la actividad A4UT2 resuelta. Es para que entiendan lo que deben hacer.