**DOCUMENTACIÓN PROYECTO GESTOR DE TAREAS (TODO APP)**

## Resumen de la Funcionalidad

Esta aplicación web permite gestionar una lista de tareas personales o laborales mediante las siguientes funcionalidades clave:

* **Listar tareas:** Carga y muestra todas las tareas almacenadas en la base de datos, separándolas en tareas pendientes y tareas completadas.
* **Añadir tareas:** Permite al usuario crear nuevas tareas ingresando una descripción (y opcionalmente detalles).
* **Eliminar tareas:** Opción para eliminar tareas existentes, actualizando la lista de forma inmediata.
* **Marcar tareas como completadas:** Cambiar el estado de una tarea para indicar que ya ha sido realizada.

La aplicación sigue un modelo cliente-servidor, donde el frontend consume una API REST que expone los datos y operaciones sobre las tareas.

## Tecnologías Utilizadas

### Backend

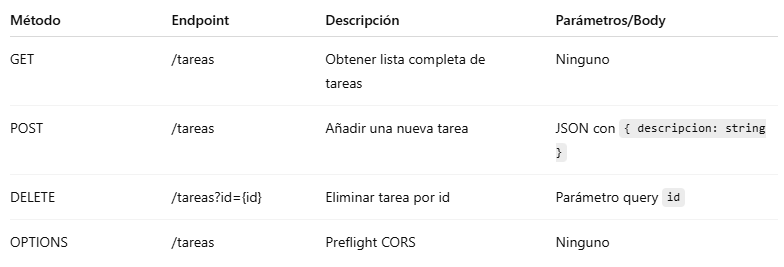
* **Java 11+**: Lenguaje principal para la lógica del servidor.
* **com.sun.net.httpserver.HttpServer**: Servidor HTTP embebido en Java para manejar peticiones.
* **Gson**: Biblioteca para convertir objetos Java a JSON y viceversa.
* **MySQL**: Base de datos relacional para almacenar las tareas.
* **JDBC**: API para conectarse y ejecutar consultas SQL sobre la base de datos.
* **Gestión de CORS:** Implementada manualmente en el servidor para permitir llamadas cross-origin desde el frontend.

### Frontend

* **HTML5, CSS3 y JavaScript (Vanilla JS)**: Para construir la interfaz de usuario y lógica de interacción.
* **Fetch API**: Para hacer peticiones HTTP (GET, POST, DELETE, OPTIONS) al backend.
* **Materialize CSS (o similar)**: Framework CSS para diseño responsivo y componentes UI (según clases CSS observadas).

## APIs y Endpoints

El backend expone la siguiente API REST para manejar las tareas:



Todos los endpoints responden con JSON y gestionan adecuadamente las cabeceras CORS para permitir peticiones desde el frontend.

## Arquitectura y Flujo

1. **Frontend**: Renderiza la interfaz y responde a interacciones del usuario (crear, eliminar, marcar tareas).
2. **Fetch API**: El frontend envía peticiones al backend según la acción.
3. **Backend Java**: Procesa las peticiones, realiza operaciones SQL en MySQL mediante JDBC, y devuelve respuestas JSON.
4. **Base de Datos MySQL**: Guarda la información persistente de las tareas.

Proyecto web completo sin frameworks avanzados como Spring o Servlets, solo con Java “puro”, HTML/CSS, JavaScript y MySQL. Vamos a trabajar con lo que ya sabes y usar **Material Design** para el front (con Materialize o Material Design Lite).

* Proyecto: Gestor de Tareas (ToDo App)
* Tecnologías a usar:
  + Frontend HTML, CSS, JS, Materialize CSS
  + Backend Java (con consola o programa simple tipo servidor)
  + Base de datos MySQL
  + Versionado Git + GitHub
* Estructura general del proyecto (carpetas)
  + todo-app/
    - ├── frontend/ │
      * ├── index.html │
      * ├── styles.css │
      * └── script.js
    - ├── backend/ │
      * └── build
      * └── lib
        + └── gson-2.10.jar
        + └── mysql-connector-j-9.3.0.jar
      * └── nbproject
      * └── src
        + └── backend

Main.java

ConexionBD.java

Tarea.java

TareaDAO.java

* + - * └── test
      * └── build.xml
    - ├── db/ │
      * └── schema.sql
    - └── README.md

**PASO A PASO**

1. Crea tu repositorio Git : utilizo Git Bash
2. Diseña la base de datos (db/schema.sql). Utilizo MySQL Workbench.
   1. BBDD todo\_app y tabla tareas
3. Frontend con HTML + CSS + JS (usando Materialize) : utilizo Visual Studio Code
4. Backend simple en Java: utilizo NetBeans IDE 18

Haz una pequeña aplicación Java que:

Lea tareas desde la base de datos

Inserte nuevas tareas (Opcional)

Exporte datos a un archivo o interfaz de consola

Clases creadas:

* Main.java
* ConexionBD.java
* Tarea.java
* TareaDAO.java

Estoy usando **Gson** para convertir objetos Java <-> JSON.

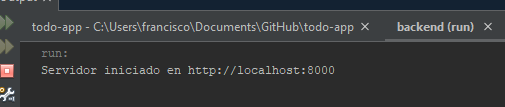
1. Testea por separado
   1. Primero: prueba solo el frontend con JS (sin backend).
   2. Luego: ejecuta el backend en consola y verifica inserciones en MySQL.
2. Conectar el JS al backend usando una API REST en Java

Enfoque: API REST simple con Java + HTTPServer. Para no usar frameworks como Spring o Servlets, vamos a usar la clase com.sun.net.httpserver.HttpServer, que viene incluida en el JDK y permite crear un servidor HTTP simple para manejar solicitudes.

**CÓMO PROBAR LA APLICACIÓN COMPLETA PASO A PASO**

**1. Asegúrate de que el backend esté corriendo**

* Ejecuta tu clase **Main.java** en NetBeans.
* Verifica en la consola que el servidor HTTP se haya iniciado, debería mostrar algo como:



Esto significa que la API REST está escuchando peticiones.

**2. Abre tu frontend HTML en el navegador**

* Abre el archivo index.html desde tu carpeta frontend (doble clic o desde el IDE).
* Importante: para evitar problemas de **CORS** (Cross-Origin Resource Sharing) cuando frontend y backend están en distintos puertos (tu frontend se abre en file:// y backend en http://localhost:8000), lo ideal es:
  + Servir tu frontend desde un servidor local (por ejemplo, con [Live Server](https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer) en VSCode), o
  + Abrir el navegador con configuración que permita CORS en modo desarrollo, o
  + (más sencillo) abrir el frontend también con un servidor local (netbeans, python -m http.server, etc).

Si abres directo el archivo HTML con file://, algunas funciones fetch pueden fallar.

**3. Probar la carga inicial de tareas**

* Al abrir el frontend, debería ejecutarse el método cargarTareas() (si seguiste el código que te di antes), que hace una petición GET a http://localhost:8000/tareas.
* Si el backend está bien y la DB tiene datos, deberías ver listadas las tareas.

**4. Probar añadir una nueva tarea**

* Escribe una tarea en el formulario y presiona el botón "Añadir tarea".
* El fetch hace un POST a la API REST enviando la nueva tarea.
* Si todo funciona, la tarea se insertará en la base de datos y la lista se actualizará automáticamente.

**5. Confirmar que la tarea fue añadida**

* Puedes volver a cargar la página para asegurarte que la tarea sigue ahí (la lee de la base de datos).
* O usa un cliente MySQL para consultar directamente la tabla tareas:

SELECT \* FROM tareas;

**6. Depurar posibles errores**

* Si no ves las tareas cargando o no se añade la tarea, abre la consola del navegador (F12 > Consola) para ver si hay errores JavaScript o de red (fetch).
* En el backend, revisa la consola donde ejecutas Java para ver si aparece algún error o excepción.

**ERRORES DETECTADOS:**

## Cómo habilitar CORS en tu backend Java simple

Como estás usando un servidor básico hecho en Java (probablemente con HttpServer o similar), debes agregar los headers CORS en las respuestas para que el navegador permita las peticiones.

### Ejemplo básico para agregar CORS en tu manejador de /tareas

Si usas com.sun.net.httpserver.HttpServer, en el handle(HttpExchange exchange) agrega estas líneas justo antes de enviar la respuesta:

// Permitir cualquier origen (para desarrollo)

exchange.getResponseHeaders().add("Access-Control-Allow-Origin", "\*");

exchange.getResponseHeaders().add("Access-Control-Allow-Methods", "GET, POST, OPTIONS");

exchange.getResponseHeaders().add("Access-Control-Allow-Headers", "Content-Type");

**ERROR java.sql.SQLException: Field 'nombre' doesn't have a default value**

java.sql.SQLException: Field 'nombre' doesn't have a default value

**¿Qué significa?**  
Que tu tabla en MySQL tiene una columna llamada nombre que está definida como NOT NULL y no tiene valor por defecto, y al hacer el INSERT no le estás pasando ningún valor para esa columna.

### ¿Cómo arreglarlo?

Tienes dos opciones:

### 1. Modificar la tabla en la base de datos para que nombre pueda ser NULL o tenga un valor por defecto

Ejemplo SQL para permitir NULL en la columna nombre:

ALTER TABLE tareas MODIFY nombre VARCHAR(255) NULL;

O para poner un valor por defecto (ejemplo vacío):

ALTER TABLE tareas MODIFY nombre VARCHAR(255) NOT NULL DEFAULT '';

### 2. Ajustar tu código Java para que al insertar tareas también proporcione el campo nombre

Por lo que muestras, tu clase Tarea y el JSON que envías solo tienen descripcion. Si la tabla espera nombre, entonces:

* Debes incluir nombre en tu objeto Tarea y en el JSON que envías desde el frontend.
* Modificar el método insert de TareaDAO para que también inserte el valor nombre.

### Cómo identificar el problema con más detalle

Revisa la estructura actual de tu tabla en MySQL con:

DESCRIBE tareas;

Ahí verás qué columnas hay, si nombre es obligatoria, etc.

### Para probar rápido:

Si tu tabla tiene nombre como NOT NULL y no tiene valor por defecto, y no quieres complicarte, cambia la tabla para que esa columna pueda ser NULL o tenga un default. Luego prueba tu app de nuevo y debería funcionar.

**MEJORAS**

* 1. Ocultar datos de usuario y contraseña en ConexiónBD.java utilizando un archivo externo db.properties
  2. Mejora de apariencia visual de tarjetas añadiendo datos en styles.css
  3. Añadir funcionalidad Drag & Drop con SortableJS en scripts.js para que las tarjetas puedan arrastrarse y añadirse a diferentes estados
  4. Nuevo estado IN PROGRESS añadido. Se elimina lógica antigua del campo Completada, y se añade nueva lógica creando campo Estado en BBDD.
  5. Nueva funcionalidad editar tarea