**ConnectionPool:** A connection pool in SQL Server is similar to a thread pool in programming. Without pooling, each database query opens a new connection, executes, and then closes it, which is slow and inefficient. With pooling, a fixed number of connections are pre-opened and reused, avoiding the overhead of repeatedly creating and destroying connections. This improves performance, resource efficiency, and query execution speed. It works like Socket.io's persistent connection, where the connection remains open until manually closed. The pool manages multiple active connections, allowing multiple queries to execute efficiently.

**Query Requests:** Takes actual data from the 3rd parameter (user input). Validates and processes it using the 2nd parameter (data type). Stores it in the 1st parameter (a named placeholder like "Name"). Uses this placeholder (@Name) in the query, so only clean and validated data is sent to SQL Server.

**OUTPUT INSERTED**.\*: is used in SQL Server to return the newly inserted row(s) from an INSERT statement.

**Pre-Flight Request (Options):** When your React, app sends certain types of requests (like POST with JSON or requests with custom headers), the browser first sends an OPTIONS request — this is called a preflight. “If the frontend from http://localhost:5173 sends an OPTIONS request, respond with the right headers to allow it.”

**app.options('http://localhost:5173', cors());**

**Query Requset Syntax:**

**const {sql, poolPromise} = require('../DB')**

const pool = await poolPromise;

let result = await pool.request()

.input("PropertyID", sql.Int, ID)

.input("MainImage", sql.VarChar(255), MainImage)

.input("SubImage1", sql.VarChar(255), SubImage1)

.input("SubImage2", sql.VarChar(255), SubImage2)

.query(`INSERT INTO Images (PropertyID, MainImage, SubImage1, SubImage2) OUTPUT INSERTED.\* VALUES (@PropertyID, @MainImage, @SubImage1, @SubImage2)`);

return result;