Chatthawan Sakhon

Part 1: Analyze Legacy Code

1. In your own words, what is the primary purpose of this function?
A Method to fetch customer information from the database using customer ID and return in a table form

2. Identify at least three distinct problems with this implementation. Consider aspects such as security, maintainability, and performance.

- **SQL injection**
  sensitive for ‘ OR 1 = 1 --
- **Hard-coded database connection string**
  bad for maintainability and security
- **Missing Error Handling**
  try-catch block for error
- **Select ***
  may show unwanted columns
- **Input validation**
  checking ID is it an empty string or in wrong format

3. For each problem identified, briefly propose a specific improvement.

- **SQL injection**
  using SqlCommand.Parameter for storing ID instead of string
- **Hard-coded database connection string**
  store connection string in Environment Variable or Secret Manager Tool
- **Missing Error Handling**
  try-catch code for an error when open connection or execute sql command
- **Select ***
  specific what column expected to be return
- **Input validation**
  checking id with .isNullOrEmpty or specific Regular expression for specific format

Part 2: Rewrite and Modernize

2.1 C# Dappa .NET 9

2.2 Construct the SQL query with the correct placeholders for parameters

```
var queryString = @"SELECT Item, Price, TypeOfPayment, Date FROM Customer
                    WHERE id = @id";
```

2.3 Demonstrate how you would pass parameters to the library's execution function to prevent SQL injection

```
SqlCommand command = new(queryString, connection);
command.Parameters.Add("@id", SqlDbType.Char, 10).Value = id;

connection.Open();
using var dataAdapter = new SqlDataAdapter(command);
```

Part 3: Extend with New Logic

3.1 Your modern, rewritten version from Part 2 (as complete code).

```
app.MapGet("/GetCustomerInfo/{id}", (
    ICustomerRepository repo,
    string id,
    DateOnly? startDate,
    DateOnly? endDate) =>
{
    var conString = builder.Configuration.GetConnectionString("BloggingDatabase") ??
     throw new InvalidOperationException("Connection string 'BloggingDatabase' not found.");

    var dataTable = new DataTable();
    try
    {

        using (var connection = new SqlConnection(conString))
        {
            if (startDate.HasValue && endDate.HasValue)
            {
                var queryString = @"
                SELECT Item, Price, TypeOfPayment, Date FROM Customer
                WHERE id = @id
                AND CreatedTime BETWEEN @startDate AND @endDate";

                using SqlCommand command = new(queryString, connection);
                command.Parameters.Add("@id", SqlDbType.Char, 10).Value = id;
                command.Parameters.Add("@startDate", SqlDbType.Date).Value = startDate;
                command.Parameters.Add("@endDate", SqlDbType.Date).Value = endDate;

                connection.Open();
                using var dataAdapter = new SqlDataAdapter(command);
                dataAdapter.Fill(dataTable);
            }
```

```
        else
        {
            var queryString = @"SELECT Item, Price, TypeOfPayment, Date FROM Customer
                                WHERE id = @id";
            SqlCommand command = new(queryString, connection);
            command.Parameters.Add("@id", SqlDbType.Char, 10).Value = id;

            connection.Open();
            using var dataAdapter = new SqlDataAdapter(command);
            dataAdapter.Fill(dataTable);

        }
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
return JsonConvert.SerializeObject(dataTable);
);
```

3.2. The original legacy C# code from Part 1 (pseudocode or a brief code snippet is sufficient).

```
public DataTable GetCustomerInfoWithDate(string id, DateOnly startDate, DateOnly endDate)
{
    var dt = new DataTable();
    using (var conn = new SqlConnection("..."))
    {
        conn.Open();

        string startDateString = startDate.ToString("yyyy-MM-dd");
        string endDateString = endDate.ToString("yyyy-MM-dd");

        var sql = $"SELECT * FROM Customer WHERE id = '{id}' "
            + $"AND DateColumn BETWEEN '{startDateString}' AND '{endDateString}'";

        using (var da = new SqlDataAdapter(sql, conn))
        {
            da.Fill(dt);
        }
    }
    return dt;
}
```

Part 4 (Optional): System and User Perspective

1. What input fields would a user (e.g., a customer service representative) need on the screen to search for customer data?
use an customer ID and range of date to filter for data

2. After the search is performed, what would the output look like on their screen? (e.g., a table, a profile card, a list of transactions)
List of transactions in JSON format can be parsed into objects for displaying each item. can be display into table

[
{
"Item":"Item1","Price":2000.0,"TypeOfPayment":"Credit","Date":"2025-07-29"},
{"Item":"Item2","Price":4000.0,"TypeOfPayment":"Credit","Date":"2025-07-30"},
{"Item":"Item3","Price":6000.0,"TypeOfPayment":"Credit","Date":"2025-07-31"},
{"Item":"Item4","Price":8000.0,"TypeOfPayment":"Credit","Date":"2025-08-01"},
{"Item":"Item5","Price":10000.0,"TypeOfPayment":"Credit","Date":"2025-08-02”
}
]

External help notes

using copilot , chatGPT , stackoverflow , C# documents , youtube chatGPT for generic overview and cross checking with stackoverflow or document and using copilot in Visual Studio for error code and warning.

I don’t have much experience in writing C# but I believe that if you can learn the hardest thing. You can learn anything else. So i choose C# over 2 other languages.

Changing from Kotlin to C# is quite challenging. Generative AI is helping me in the learning phase and mostly syntax or how C# is working in general. It summarizes basic concepts and reduces time from learning without it. Using it with other source like Blog , Document or Video is helping me to learn about C#