Interview – SQL Sanitize

This document contains the release notes of the "***BloopTestSolution***" application.

The Database:

This solution is built using MS SQL Server 2019 Express (using Visual Studio 2019).
The server for the database is defined as the local db server included in Visual Studio 2019 ("***(localdb)\mssqllocaldb***")
The database (called "**BloopServiceDB**"), contains one table:

```
CREATE TABLE [dbo].[Sensitivewords]
(
        [Id]                INT             IDENTITY(1,1)           NOT NULL,
        [WordDefinition]    VARCHAR(150)                            NOT NULL,
        [CreatedOn]         DATETIME        DEFAULT(GETDATE())      NOT NULL,
        [ModifiedOn]        DATETIME                                NULL,
        PRIMARY KEY CLUSTERED
        (
                [Id] ASC
        )
)
```

This table holds the company defined sensitive words to use during the word "**bloop**" when evaluating a user message.

The Solution:

The solution is split into four projects:

1. Data.Processing
2. Sensitivewords.API
3. Company.UI
4. Web.UI

1. Data.Processing:

This project holds the code responsible for the data processing logic of the solution.

Models:

1. **Sensitiveword** – To consume the "***Sensitivewords***" database table.
2. **BloopResponse** – To encapsulate the responses returned by the "***BloopService***".

Services:

1. **BloopService** – This service contains the logic for the CRUD operations.
    a. **IBloopService** – The interface exposing the service functions.
    b. **BloopService** – The function logic of the service.
2. **MessageService** – This service contains the message manipulation operations.
    a. **IMessageService** – The interface exposing the service functions.
    b. **MessageService** – The function logic of the service.

2. Sensitivewords.API(start-up project):

This project contains the API logic of the solution.

Attributes:

1. **ApiKeyAttribute** – This holds the Api Key authentication logic.

Controllers:

1. **BloopController** – Holds the external exposed endpoint logic of the API for message manipulation.
2. **BloopServiceController** – Holds the internal exposed endpoint logic, authenticated by an Api Key.

Additional:

The Api Key is defined in the appsettings.json file and is named "***X-API-Key***".
The BloopServiceController is decorated with the custom "**ApiKey**" attribute to enforce Api Key authentication.
This project contains a project reference to (and is depends on) the "**Data.Processing**" project.

Endpoints:

https://localhost/api/Bloop:
Message manipulation endpoint (external).
It consumes a "message" (string) as Json in the request body.

https://localhost/api/BloopService/Words:
Internal exposed endpoint to retrieve a list of sensitive words.
https://localhost/api/BloopService/Word/{id}:
Internal exposed endpoint to retrieve a specific sensitive word. (***The logic for this endpoint is not specifically implemented.***)
https://localhost/api/BloopService/Create:
Internal exposed endpoint to create/add a new sensitive word.
https://localhost/api/BloopService/Import:
Internal exposed endpoint to import a new list of sensitive words.
https://localhost/api/BloopService/Update:
Internal exposed endpoint to modify a specific sensitive word.
https://localhost/api/BloopService/Delete/{id}:
Internal exposed endpoint to remove/delete a specific sensitive word.

For additional information and examples of endpoints, please review the Sensitivewords API Swagger UI (set as a startup project of the solution)

3. Company.UI(start-up project):

This project serves as a dummy example company website to use the internal exposed endpoints.
This project has a reference and dependency on the "**Data.Processing**" project.

4. Web.UI(start-up project):

This project serves as a dummy example client website to show the use of the external exposed endpoint.

<u>Setup</u>

For the initial setup, please run the included SQL script “**Initial_database_script.sql**”.
The solution is built using Visual Studio 2019 and .Net Core (**.Net 5.0**) as the target framework.

Production deployment walkthrough

For real world, production deployment, this project would need some additional features.
For example, the client message application would possibly need to be expanded to include entities that define the "Message Sender" and "Message Receiver".
A proper message send/receive service would need to be added to allow message sending between sender and receiver.

Assuming the endpoints are used correctly and that all additional required development is done, deployment can be setup as follows:

1. Deploy the Data.Processing, the Sensitivewords.API and the Company.UI (company website) parts to the same production server. (To persist project dependency)
2. Deploy the database to any accessible, independent server (ensure that the appsettings.json of the Sensitivewords.API project points to the correct database and server).
3. Deploy/develop the Web.UI (client message website/application) to any accessible, independent server.

Simply ensure the use of the external bloop api endpoint between the message sent and message received communication parts (or before actual message sending).

For example:
A user selects a receiver from a contacts list to send a message to.
The user is then prompted to enter the message to send.
Once the user triggers the event to send the message to the receiver the message can be:

- Evaluated by the Sensitivewords API (using the external exposed endpoint).
- Replaced by the Sensitivewords API response.
- Continued to be sent by the send event.

Since the API will either replace matching words or not, the message will either be replaced by the "bloop" or by the original message returned.

Simply put, if the API is deployed and setup correctly and the "*https://{whatever}/api/Bloop*" endpoint can be successfully accessed by whatever application wishes to use it, and if that application sends a message as the request body (in the correct format), it will receive an evaluated, replaced string message as a result.