Full-Stack Architecture: Angular Frontend, .NET Core 8 Backend, and MongoDB

**1. Overview**

This document describes the architecture of a full-stack web application that utilizes:

* **Frontend**: Angular(Version :Latest 18)
* **Backend**: .NET Core (Version:8)
* **Database**: MongoDB (NoSQL database)

This architecture is suitable for modern web applications that require a decoupled and scalable system with a rich, interactive user interface on the client side and robust data handling on the server side.

**2.Frontend (Angular)**

**Responsibilities**:

* **User Interface (UI)**: Angular handles the presentation layer, providing a dynamic and responsive UI. The UI is built using components, templates, and styles.
* **Client-Side Logic**: Business logic for rendering views and managing user interactions is handled by Angular.
* **Communication with Backend**: HTTP requests (typically via HttpClient) are sent to the backend (ASP.NET Core API) for data fetching, user authentication, and other interactions.

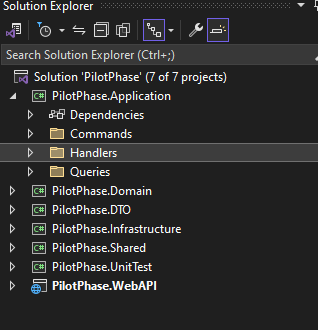
**Key Components**:

* **Angular Modules**: Organizes components, services, and directives.
* **Components**: Define views and handle the UI logic.
* **Services**: Handle business logic and data communication with the backend.
* **Routing**: Manages client-side navigation.

### Example Request Flow:

1. User clicks a button in the Angular UI.
2. The component calls an Angular service, which sends an HTTP request to the backend API.
3. The service handles the response and updates the UI.

**3.Backend (.NET Core 8 with CQRS Pattern)**



##### **3.1 .NET Core 8 Overview**

The backend is built using **.NET Core 8**, a cross-platform framework that provides scalability, security, and performance. The backend is structured around the **CQRS (Command and Query Responsibility Segregation)** pattern to separate read and write operations.

##### **3.2 CQRS Pattern**

The CQRS pattern divides the application logic into:

* **Commands**: For performing write operations (create, update, delete).
* **Queries**: For fetching read-only data.

Each command and query has its own handler. This separation allows better scalability and flexibility.

##### **3.3 Key Layers of the Backend**

* **Controller Layer (API Layer)**: Exposes RESTful APIs for the frontend. Controllers route incoming HTTP requests to appropriate handlers based on whether they are commands or queries.
* **Application Layer**:
  + **Command** : These handles request from controller to handler
  + **Handlers:** These handle operations like creating new users, updating data, etc.
  + **Query Handlers**: These are responsible for fetching data and sending it back to the client. For instance, fetching user information or transaction details.
* **Domain Layer**: Contains business logic. It also defines the core business entities and ensures that business rules are enforced.
* **Infrastructure Layer**: Handles external services like database access (MongoDB) and encryption/decryption utilities.

##### **3.4 Encryption/Decryption in Backend**

* **Data Encryption**: Sensitive data is encrypted before being stored in the MongoDB database using AES encryption algorithms.
* **Data Decryption**: Data is decrypted when read from the database or sent back to the frontend securely.

## 4. Communication Between Frontend and Backend

### HTTP API Calls

* **RESTful Endpoints**: The Angular app communicates with the backend using standard HTTP methods (GET, POST, PUT, DELETE).
* **JSON**: Data is typically exchanged in JSON format between the frontend and backend.

### Example Workflow:

1. Angular makes an HTTP GET request to /api/products.
2. The backend controller receives the request and processes it.
3. The backend queries the MongoDB database for product data.
4. The backend returns the data to the Angular client as JSON.
5. The Angular client displays the data to the user.

## ****5.**** Conclusion

This architecture is designed to handle scalability, security, and separation of concerns. Using Angular for the frontend, .NET Core 8 with CQRS for the backend, and MongoDB for the database ensures a modern, robust application architecture. The encryption mechanisms ensure data security, while CQRS offers clear separation between query and command responsibilities, making the application more maintainable and scalable.

This document can serve as a starting point for the development and design of the application. If needed, additional details on specific components (e.g., exact encryption algorithms or further database configurations) can be added based on project-specific requirements.

Task Completed:

**Task 1: Basic .NET/C# Project with Angular Interface and GDPR Considerations**

**Implementation Done:**

* 1. Product Entity
  2. CRUD Functionality
  3. Angular Front-End
  4. MongoDB Integration using compound index
  5. GDPR Considerations: Implemented Encryption in Contact Us Form for storing data in encrypted manner.

**Task 2: Binary String Analysis Function**

**Implementation Done:**

1. Function accepts a binary string as input
2. Check if the binary string is 'good' based on these conditions:

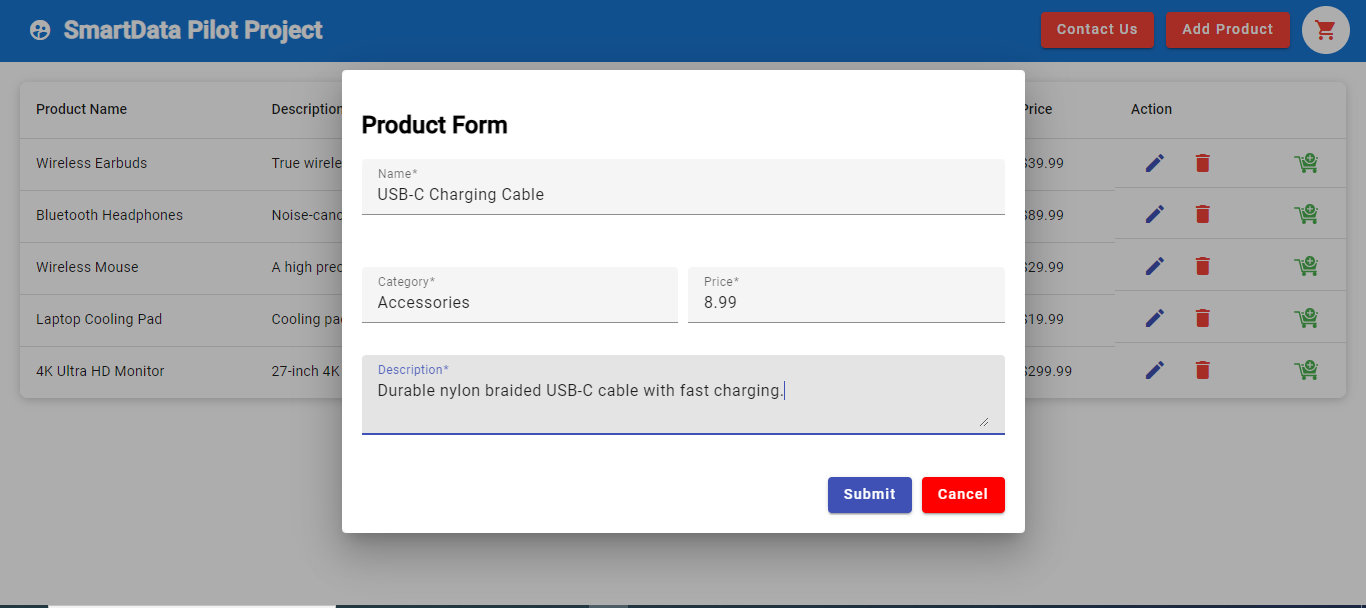
● Equal number of 0's and 1's.

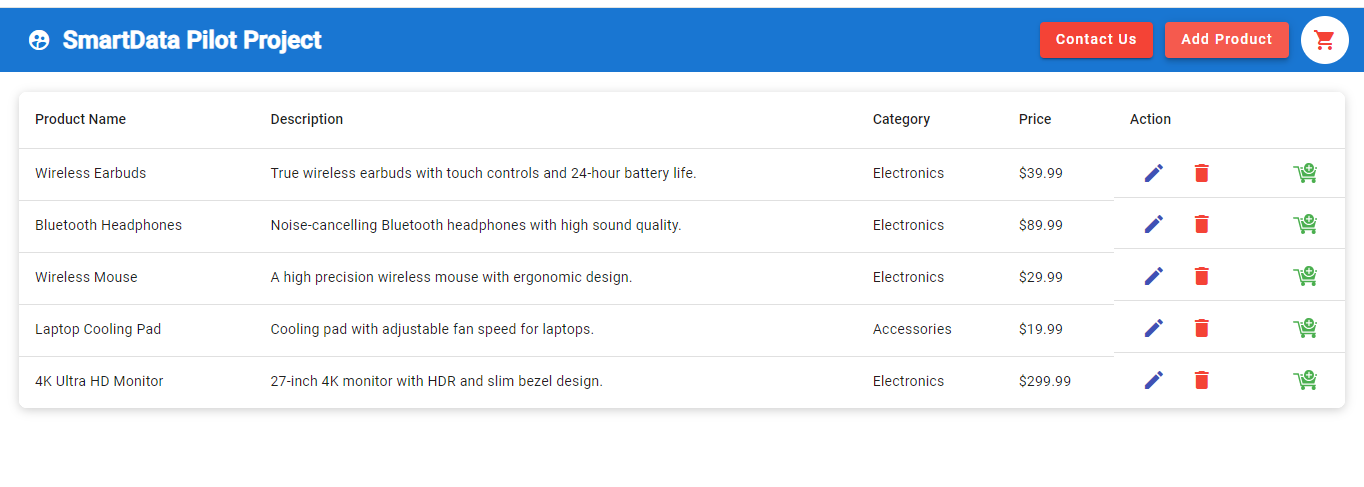
● For every prefix, the number of 1's is not less than the number of 0's

**Task 1: Basic .NET/C# Project with Angular Interface and GDPR Considerations**

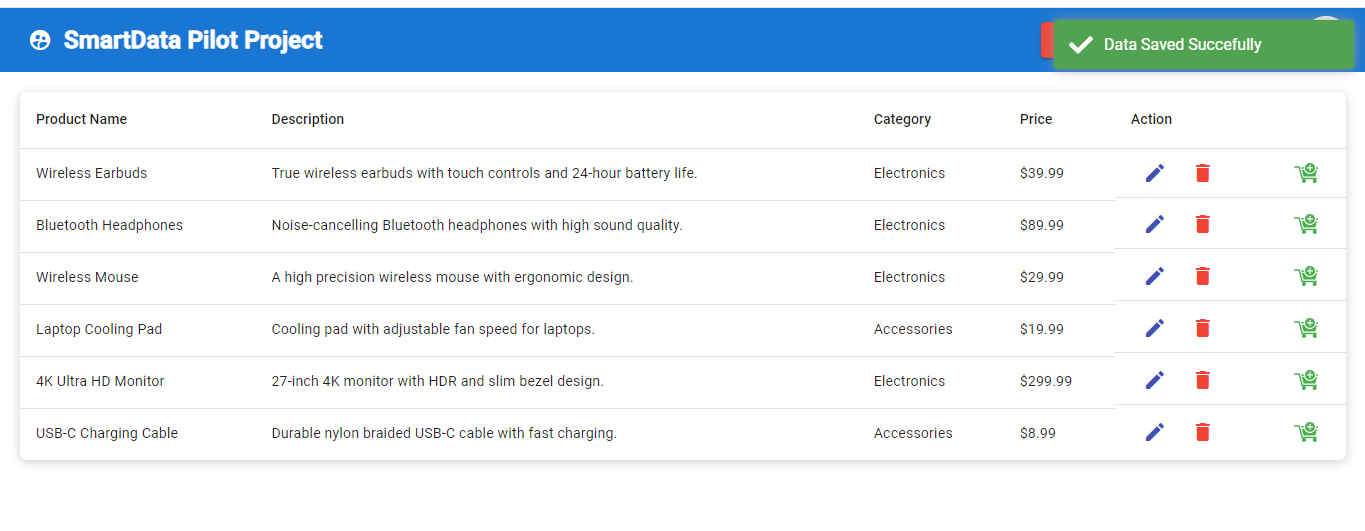
**Product Functionality: -**

**Product Add Page:**

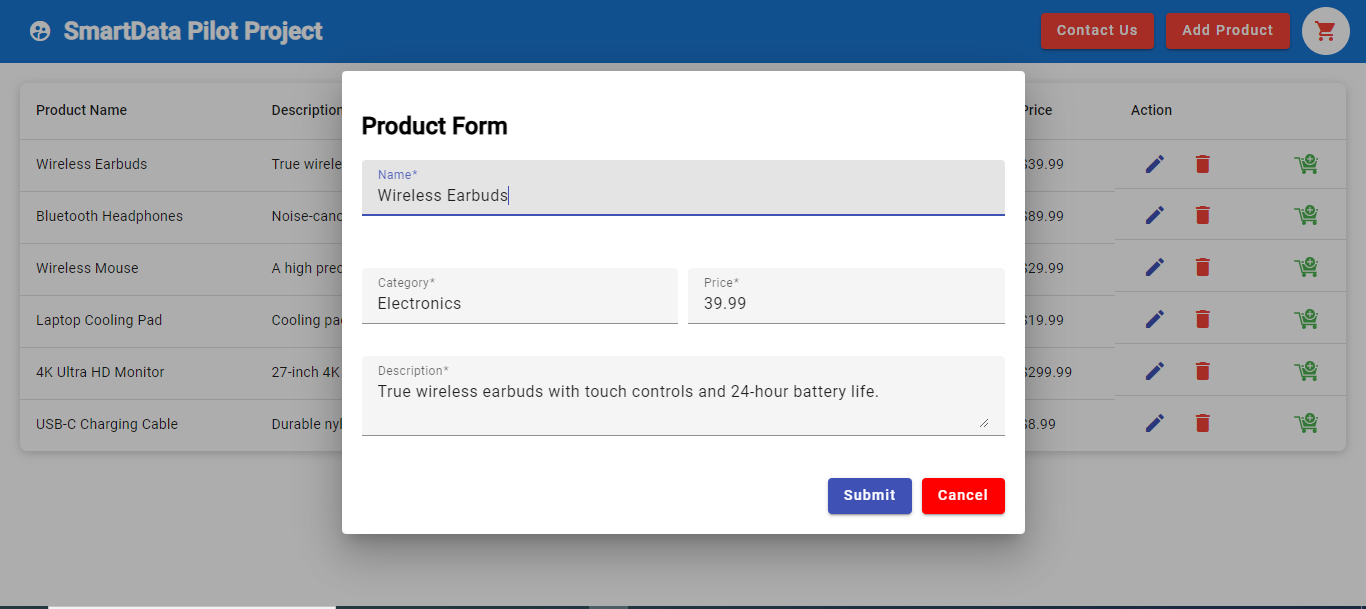


**Product List Page:** 

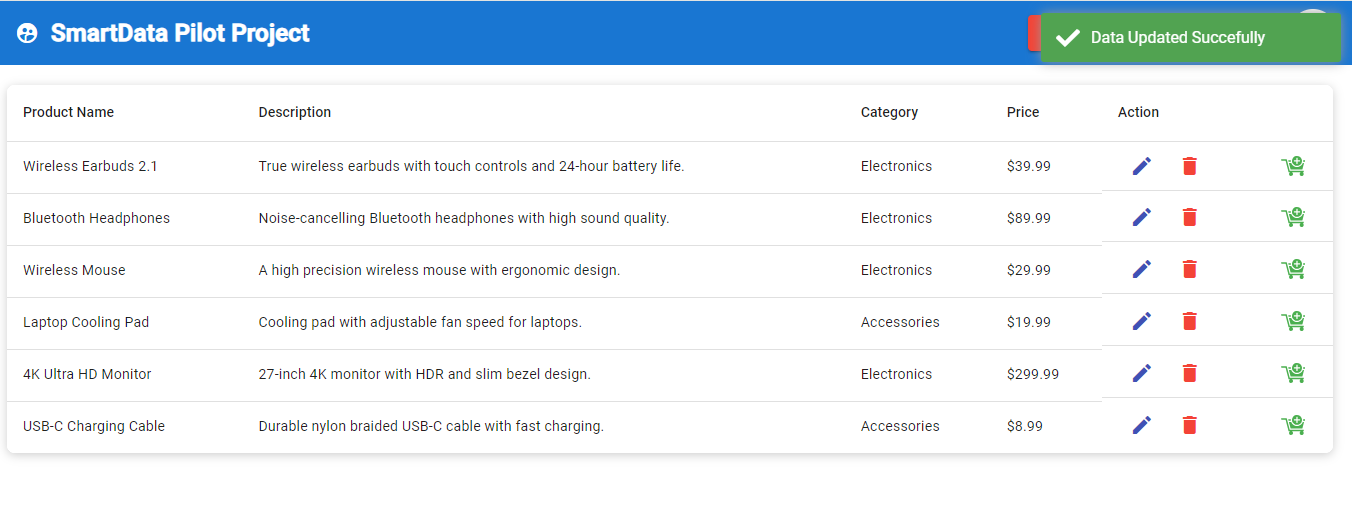
**Product Saving Taster :**



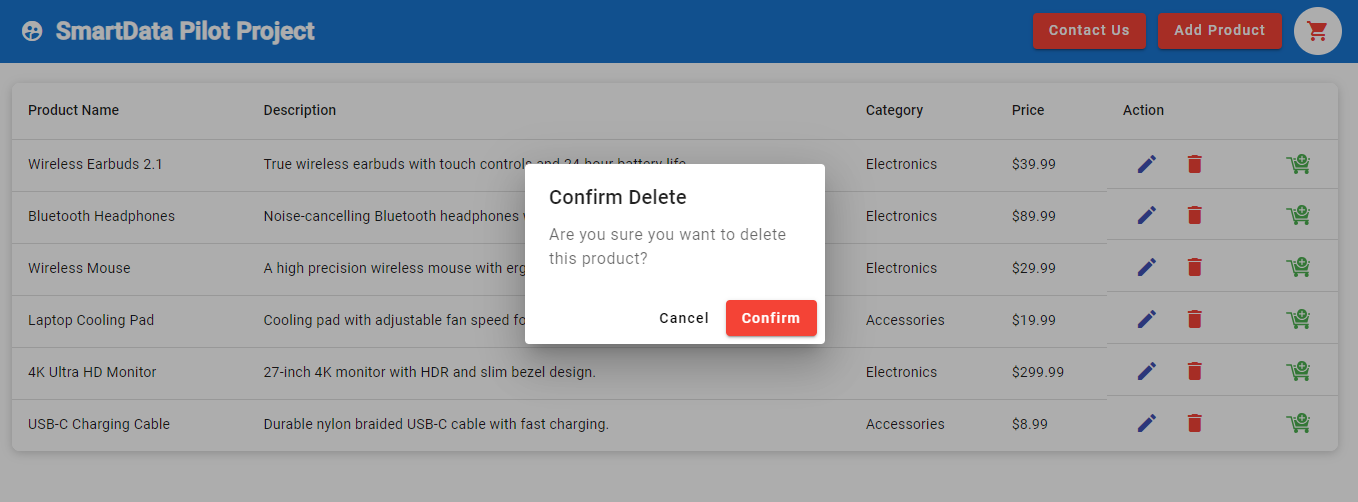
**Product Update Popup:**



**Product Update Toster :**

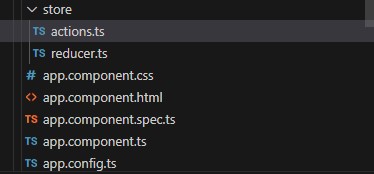


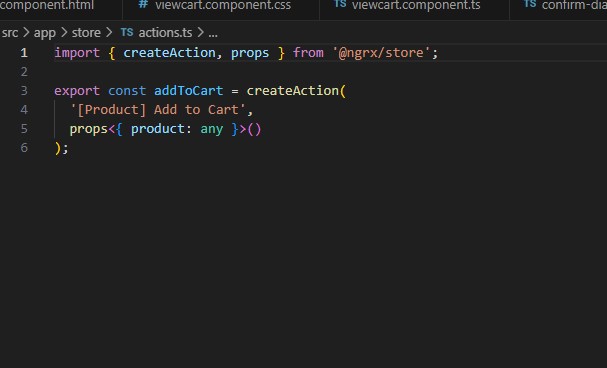
**Delete Confirmation:**

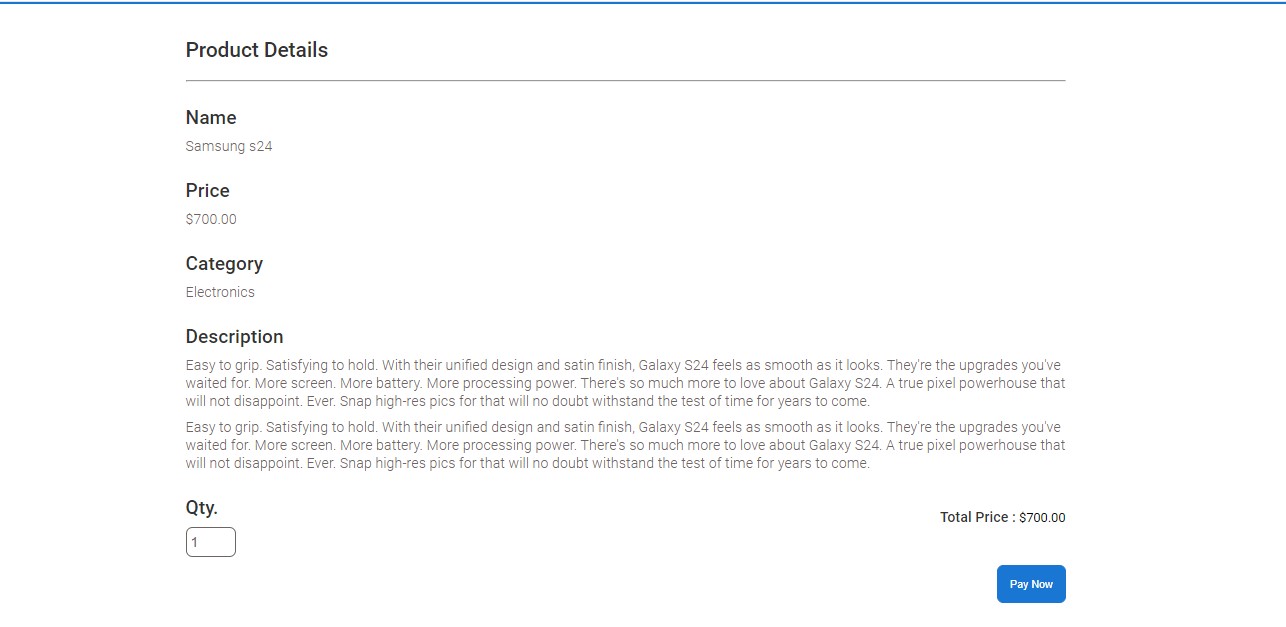


**Redux Implementation:**

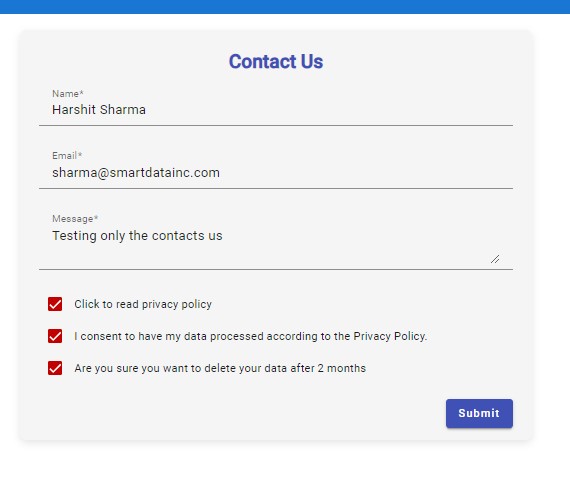
We have used redux in product cart page



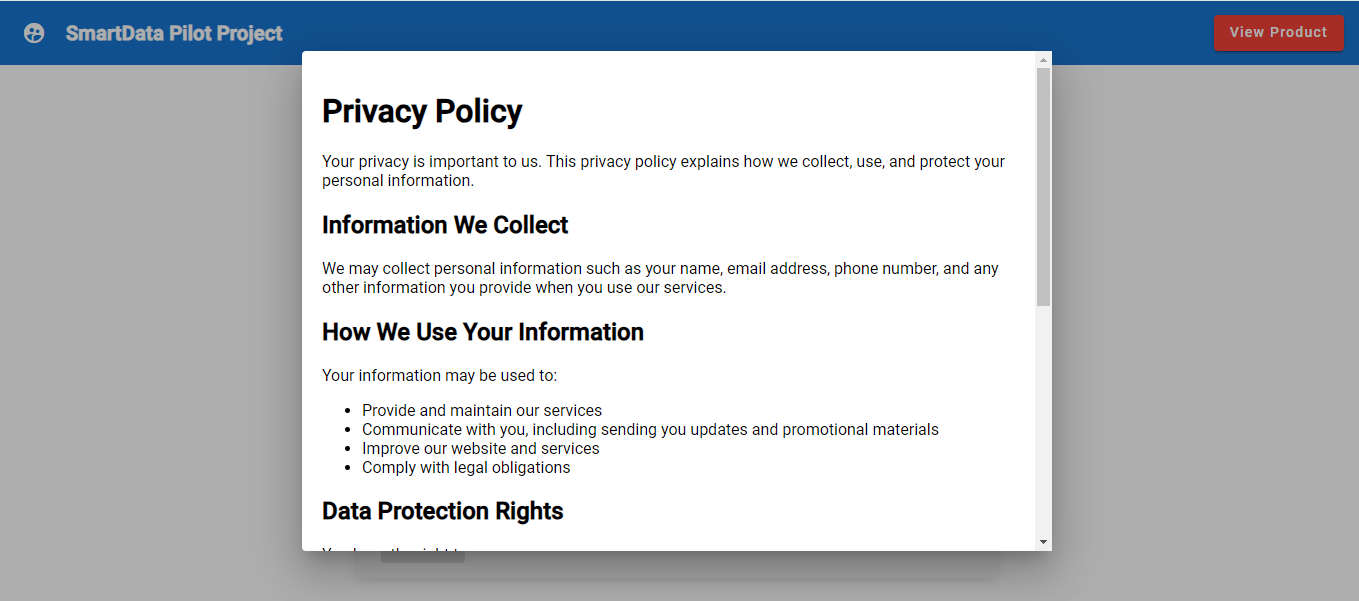




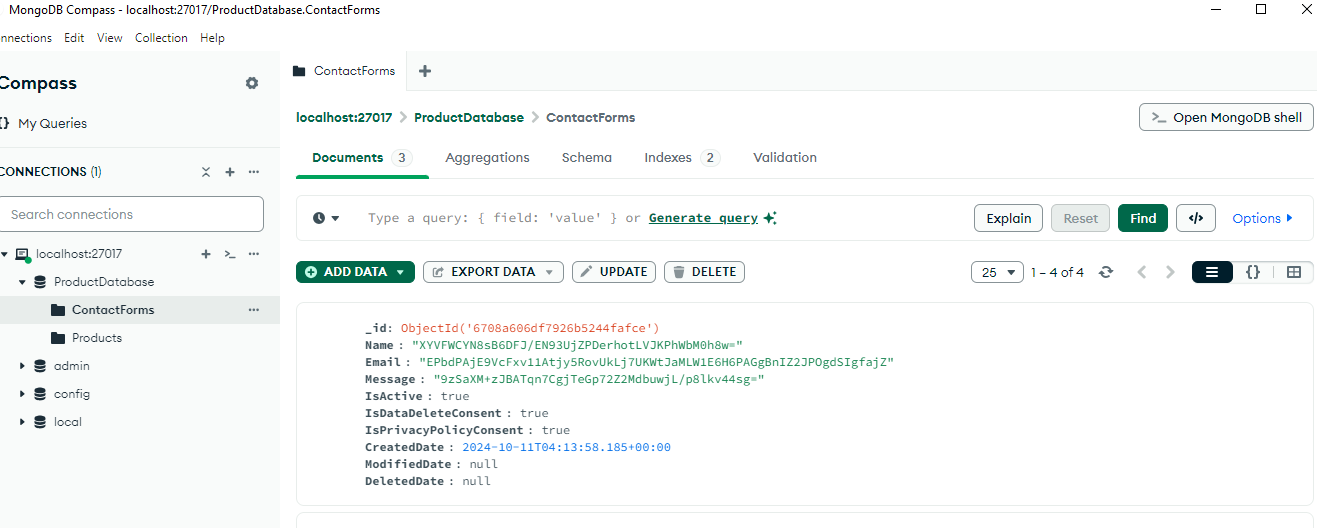
**Contact Us Form for GDPR:**



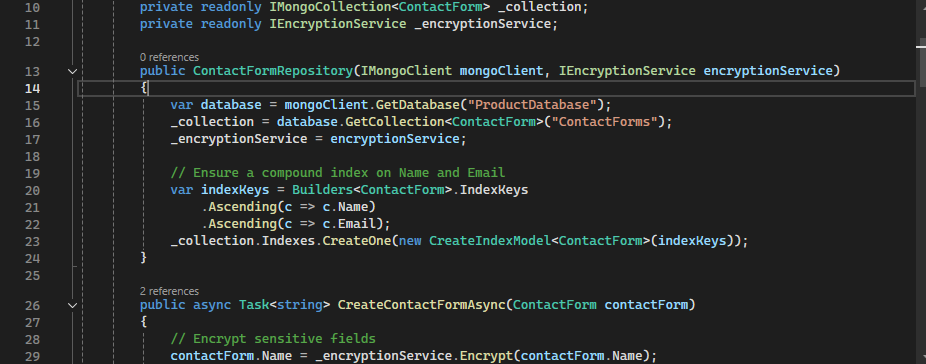
**Privacy Policy:**



**Consent form user data is stored in encrypted form in mongo db database.**



**Mongo DB Indexing used within the project**

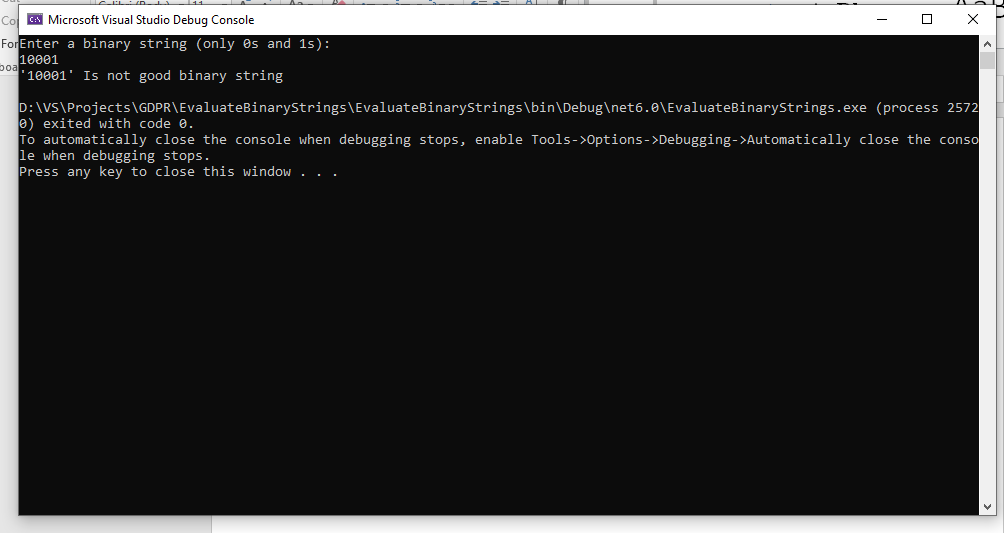


**Task 2: Binary String Analysis Function**

**Project Name : - EvaluateBinaryStrings**

**Input String: - 10001**

**Result: '10001' Is not good binary string**



**Input String: - 101010**

**Result: '101010' Is the good binary string**

