Code Implementation Report

Lab 07: Customer Segmentation and Development

Implementation Analysis

April 26, 2026

Contents

---

1 Introduction

This report details the implementation of the "Customer Segmentation and Development" module for Lab 07. The provided solution is built using the ASP.NET Web Forms framework, utilizing HTML/CSS for the presentation layer (`Segmentation.aspx`) and VB.NET for the backend logic (`Segmentation.aspx.vb`). The primary objective of this module is to allow administrators to dynamically analyze customer purchasing behavior and categorize them into distinct segments: Premium, Frequent, and Bulk buyers.

2 Summary of Pages

The implementation is cleanly separated into two distinct files following the standard ASP.NET Code-Behind model:

- `Segmentation.aspx` **(The Frontend View):** This file contains the HTML markup, ASP.NET Web Controls, and inline CSS. It defines the visual layout of the application, including input fields for dynamic thresholds, action buttons to trigger data queries, and data grids (`GridView` controls) to display the segmented customer lists.
- `Segmentation.aspx.vb` **(The Backend Logic):** This file contains the server-side Visual Basic code. It handles security (role-based access control), state management (saving user thresholds in Session variables), database connectivity (ADO.NET), and the execution of complex SQL queries to aggregate and filter customer data from the underlying `FurnitureDB` database.

3 Frontend Implementation: `Segmentation.aspx`

The frontend is designed with usability and clear data presentation in mind. It is divided into three main operational sections:

3.1 Threshold Configuration

Administrators are provided with a form to set custom threshold values that define what qualifies a customer for a specific segment.

- **Premium Threshold:** Defines the minimum total spend (in dollars).
- **Frequent Threshold:** Defines the minimum number of separate orders placed.
- **Bulk Threshold:** Defines the average quantity of items per order line.

These inputs use `<asp:TextBox>` controls and are submitted via an "Update Thresholds" button.

3.2 Quick Analysis Actions

A suite of `<asp:Button>` controls allows the user to trigger specific analysis routines. Buttons like "Load Premium Customers" or "Load Bulk Buyers" are styled using CSS classes (e.g., `btn btn-warning`) to distinguish their functions visually. A "Clear Results" button is also provided to reset the workspace.

3.3 Results Presentation

The results are displayed using multiple `<asp:Panel>` controls, which act as containers that can be hidden or shown dynamically from the backend. Inside these panels, `<asp:GridView>` controls are utilized to render the database tables. The `GridView` columns are explicitly bound to database fields (e.g., `CUSTOMER_NAME`, `Total_Spend`) and include currency and number formatting directly in the markup (e.g., `DataFormatString="{0:C}"`).

4 Backend Implementation: `Segmentation.aspx.vb`

The code-behind file dictates the behavior and data-fetching mechanisms of the application.

4.1 Security and State Management

Upon the `Page_Load` event, the system immediately checks the user's session variables.

- **Role-Based Access Control (RBAC):** The page ensures that `Session("User_Role")` is set to `"admin"`. If not, unauthorized users are redirected to `Login.aspx`.
- **Session State:** To ensure threshold configurations persist across multiple postbacks (button clicks), the application stores and retrieves the Premium, Frequent, and Bulk values using the `Session` object.

4.2 Database Integration and ADO.NET

The application connects to a SQL Server database named `FurnitureDB` via the `ConfigurationManager`. It uses the `SqlConnection`, `SqlCommand`, and `SqlDataAdapter` classes to securely execute parameterized queries, which protects the application from SQL injection attacks.

4.3 Analytical SQL Queries

The core of the lab lies in the SQL queries generated within the VB code. To perform customer segmentation, the queries execute complex joins and aggregations:

- **Data Joining:** Queries combine `customer_T`, `Order_T`, `Order_Line_T`, and `Product_T` to get a holistic view of customer transactions.
- **Aggregation:** The `SUM()` function calculates total spend, `COUNT(DISTINCT)` counts unique orders, and `AVG()` calculates the average items bought.
- **Filtering (`HAVING` Clause):** Unlike standard `WHERE` clauses, the `HAVING` clause is used to filter the aggregated data against the user-defined thresholds (e.g., `HAVING SUM(...) > @PremiumThreshold`).
- **Dynamic Classification (`CASE` Statement):** The "Load All Segments" function employs a SQL `CASE` statement to automatically tag every customer in the database as 'Premium', 'Frequent', 'Bulk', or 'Standard' in a single efficient query.

5 Conclusion

The provided code successfully demonstrates a robust implementation of dynamic customer segmentation. By cleanly separating the UI design in the `.aspx` file from the data access and business logic in the `.vb` file, the application remains maintainable. The use of parameterized SQL aggregation queries combined with ASP.NET Session state ensures both accuracy and an excellent user experience for database administrators.