

NIKE STORE



Session: 2023 – 2027

Submitted by:

Mian Saad Tahir 2023-CS-62

Supervised by:

Ma'am Maida Shahid

Sir Laeeq Khan Niazi

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Table of Contents

1.1. Overview:.....	3
1.2. Objectives:	3
1.3. Intended Functionality:	4
2. OOP-CONCEPTS:	4
2.1. Association:.....	5
2.2. Encapsulation:.....	5
2.3. Inheritance:.....	5
2.4. Polymorphism:.....	5
3. COMPARE WITH PROCEDURAL PROGRAMING:	5
3.1. Reusability and Code Sharing:	5
3.2. Maintainability and Scalability:.....	6
3.3. Code Readability and Understandability:	6
3.4. Data Security and Abstraction:	6
4. DESIGN PATTERN IMPLEMENTATION:	6
4.1. Business Logic (BL):.....	6
4.2. Data Access Layer (DL):.....	6
4.3. User Interface (UI):	7
5. Class Diagram:.....	7
6. Wireframes:	8
7. Complete Code:	15
7.1. Business Logic (BL):.....	15
7.2. Data Layer (DL):	17
7.3. User Interface (UI):	27
8. CONCLUSION:	30
8.1. Summarization and Achievements:	30
8.2. Challenges:.....	30
8.3 Lessons Learned:.....	30

1. INTRODUCTION

1.1. Overview:

Problem Statement: In the fast-paced and competitive retail industry, managing clothes store efficiently and effectively can be a complex undertaking. Owners often face challenges in keeping track of inventory, sales, and overall business operations. These difficulties create a demand for a comprehensive Clothing Management System that streamlines various processes and offers a centralized platform for shop owners to optimize their operations.

Solution: My Nike Store Management System is an innovative and efficient software solution designed to simplify the complexities of running a retail shop. It seeks to address the challenges faced by shop owners and customers, providing a centralized platform that optimizes inventory management, sales, customer relationships, supplier interactions, and reporting.

Significance: The introduction and adoption of my Nike Store immense significance for the retail industry and business management. By harnessing the power of technology, this system brings forth a host of advantages that can revolutionize how general shops operate, impacting various stakeholders positively.

1.2. Objectives:

Efficient Inventory Control: The main objective of the Nike Store is to implement efficient inventory management practices. By providing real-time updates on stock levels, automating stock replenishment, and categorizing products, the system aims to optimize inventory control. This objective helps reduce stockouts, minimize excess inventory, and ensure that the shop maintains an adequate supply of products to meet customer demand.

Streamlined Sales and Billing Processes: The system's objective is to streamline sales and billing processes for a smoother customer checkout experience. Through automated billing procedures, accurate calculations, and detailed invoicing, the system minimizes errors in transactions. This objective enhances customer satisfaction and reduces delays at the point of sale.

Enhanced Customer Relationship Management: The Nike Store Management System seeks to enhance customer relationship management by maintaining a centralized database of customer information. With access to purchase history,

preferences, and loyalty rewards, the system enables personalized interactions and targeted marketing efforts. This objective aims to improve customer retention and loyalty.

Data-Driven Decision Making: Another objective of the system is to enable data-driven decision-making for shop owners and managers. The reporting and analytics module generate real-time sales reports, inventory status, and customer feedbacks. This valuable data empowers users to make informed business decisions, identify trends, and optimize strategies for improved profitability and growth.

1.3. Intended Functionality:

User-Friendly Interface: This System features a user-friendly interface that is easy to navigate and operate. It is designed with simplicity in mind, ensuring that both shop owners and customers can quickly adapt to the system without the need for extensive training.

Real-Time Inventory Tracking: The system will offer real-time tracking of inventory levels. Users receive alerts and current stock status of products, and check the availability of specific items instantly. This functionality ensures that shops can promptly restock products and avoid stockouts.

Sales and Billing Automation: The system will automate the sales and billing processes to streamline transactions. It will calculate accurate totals, generate detailed invoices, and process various payment methods seamlessly, reducing manual errors and facilitating faster checkout.

Customer Relationship Management: The system will enable efficient customer relationship management by maintaining a centralized database of customer information. Users can access customer profiles, view purchase history, and track customer preferences to offer personalized services and promotions.

2. OOP-CONCEPTS:

In the context of the Nike Store, object-oriented programming (OOP) principles and concepts can be applied to enhance the design, organization, and functionality of the system. Here are some examples:

2.1. Association:

Association is a fundamental OOP concept that represents relationships between classes. In the Nike Store, association can be observed between different classes that interact with each other. For instance, there can be an association between the ProductBL class and the SuitBL class, where the SuitBL class accesses the ProductBL class to update product quantities after each sale.

2.2. Encapsulation:

Encapsulation involves bundling data and methods within a class, shielding the internal implementation details from outside access. In the Nike Store, encapsulation can be utilized to protect sensitive data and provide controlled access to class members.

2.3. Inheritance:

Inheritance is a key OOP concept that enables the creation of hierarchical relationships between classes. In the Nike Store, inheritance can be applied to establish relationships between various entities. For instance, there can be a base class named User, which is inherited by more specific classes like Admin or Customer.

2.4. Polymorphism:

Polymorphism is a powerful OOP concept that allows objects of different classes to be treated interchangeably. In the Nike Store, polymorphism can be employed to handle various types of products uniformly. For example, a method for calculating the total price of items in the shopping cart can accept objects of different product types.

3. COMPARE WITH PROCEDURAL PROGRAMING:

Following are the various aspects in which OOP is better than procedural programing:

3.1. Reusability and Code Sharing:

OOP promotes code reusability through the concept of inheritance. Inheritance allows classes to inherit properties and behaviors from other classes, reducing the need to rewrite code. This not only saves development time but also makes the codebase more efficient and easier to maintain. In procedural programming, reusing code requires copying and pasting or creating separate functions, which can lead to code redundancy and maintenance issues.

3.2. Maintainability and Scalability:

OOP promotes code maintainability and scalability. With encapsulation and modularity, making changes to a specific functionality or fixing issues becomes easier because the affected code is localized within the relevant class. Additionally, OOP's ability to extend existing classes through inheritance allows for the addition of new features without modifying the existing codebase extensively. In procedural programming, making changes or adding features often involves modifying multiple functions, increasing the likelihood of introducing errors and making maintenance more challenging.

3.3. Code Readability and Understandability:

OOP encourages a more natural representation of real-world entities and their relationships. This makes the codebase more readable and understandable, as classes and objects closely resemble the entities and interactions they model. Procedural programming, on the other hand, can lack this intuitive representation, making it harder to grasp the overall system design and the relationships between various parts of the code.

3.4. Data Security and Abstraction:

OOP allows for the implementation of data security through the concept of encapsulation. By hiding the internal implementation details of a class, OOP ensures that data can only be accessed and modified through designated methods, reducing the risk of unintended modifications or data corruption. Procedural programming typically lacks built-in mechanisms for data security and abstraction, making it more prone to data integrity issues and unauthorized access.

4. DESIGN PATTERN IMPLEMENTATION:

In the Nike Store the utilization of design patterns helps ensure modularity, separation of concerns, and maintainability. Here is how the project incorporates the Business Logic (BL), Data Access Layer (DL), and User Interface (UI) design patterns:

4.1. Business Logic (BL):

The BL design pattern focuses on encapsulating the business rules and logic of the application. It ensures that the core functionality and operations of the system are independent of the underlying data storage or user interface.

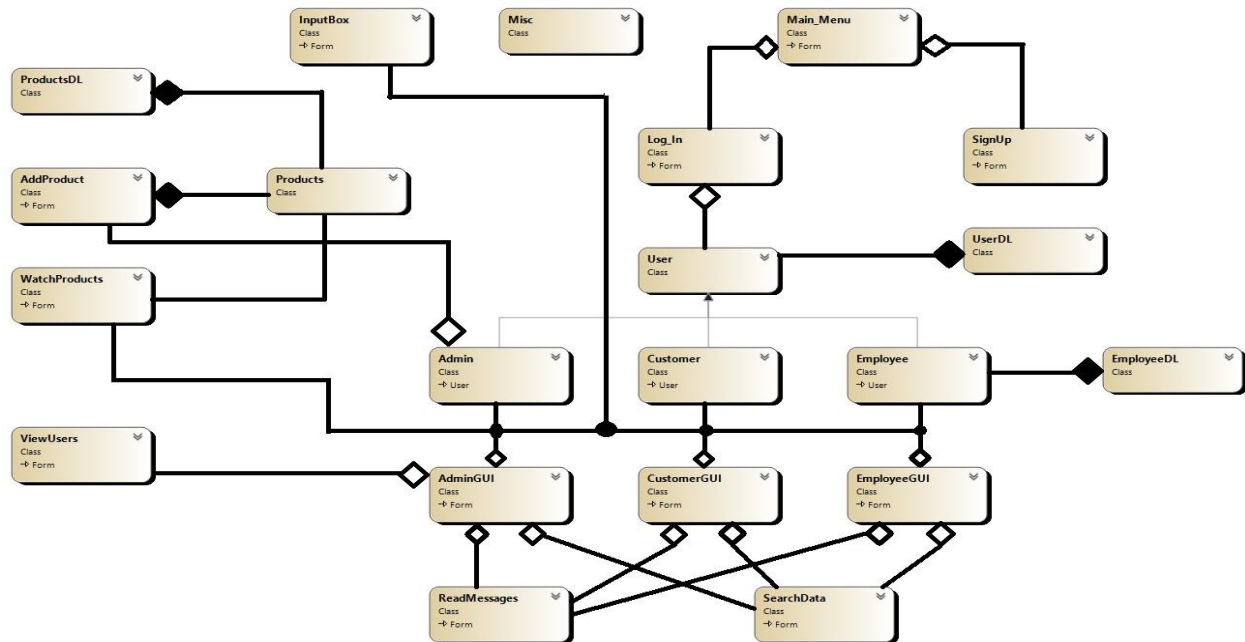
4.2. Data Access Layer (DL):

The DL design pattern focuses on managing the interaction between the application and the underlying data storage or database. It abstracts the data access operations, ensuring that the business logic is decoupled from the specifics of the data storage implementation.

4.3. User Interface (UI):

The UI design pattern focuses on structuring and organizing the user interface components to ensure a consistent and user-friendly experience. It separates the presentation layer from the underlying business logic and data access operations.

5. Class Diagram:



6. Wireframes:

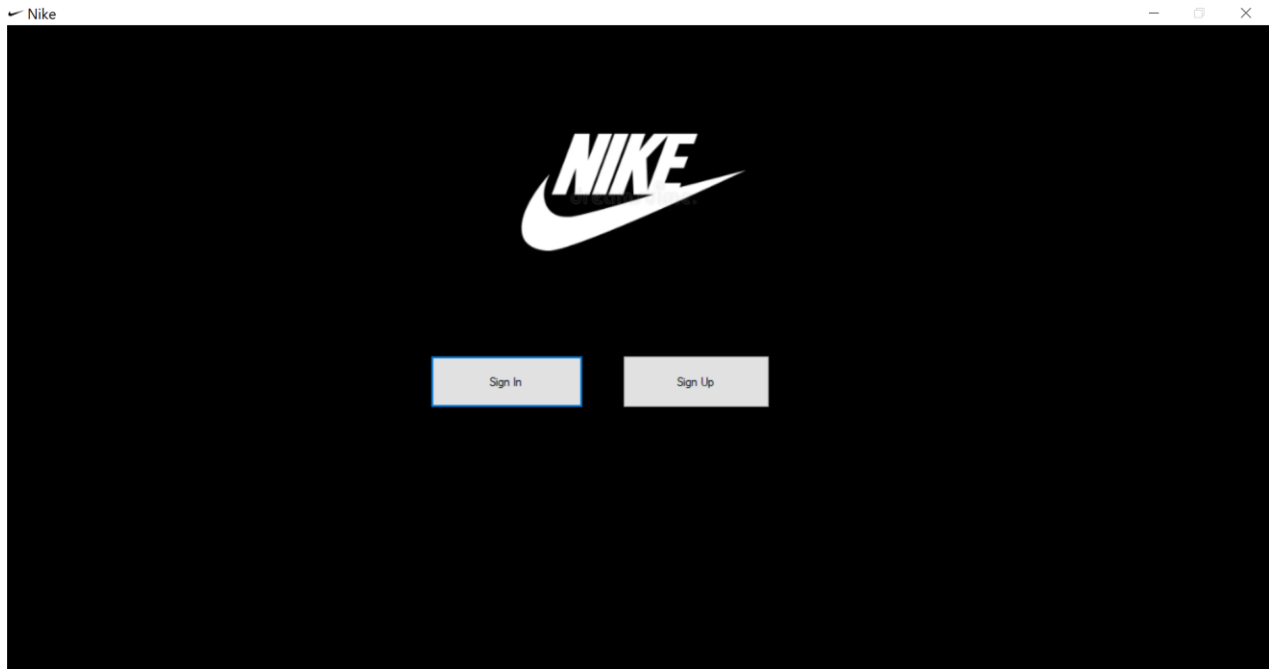


Fig.: Main Screen

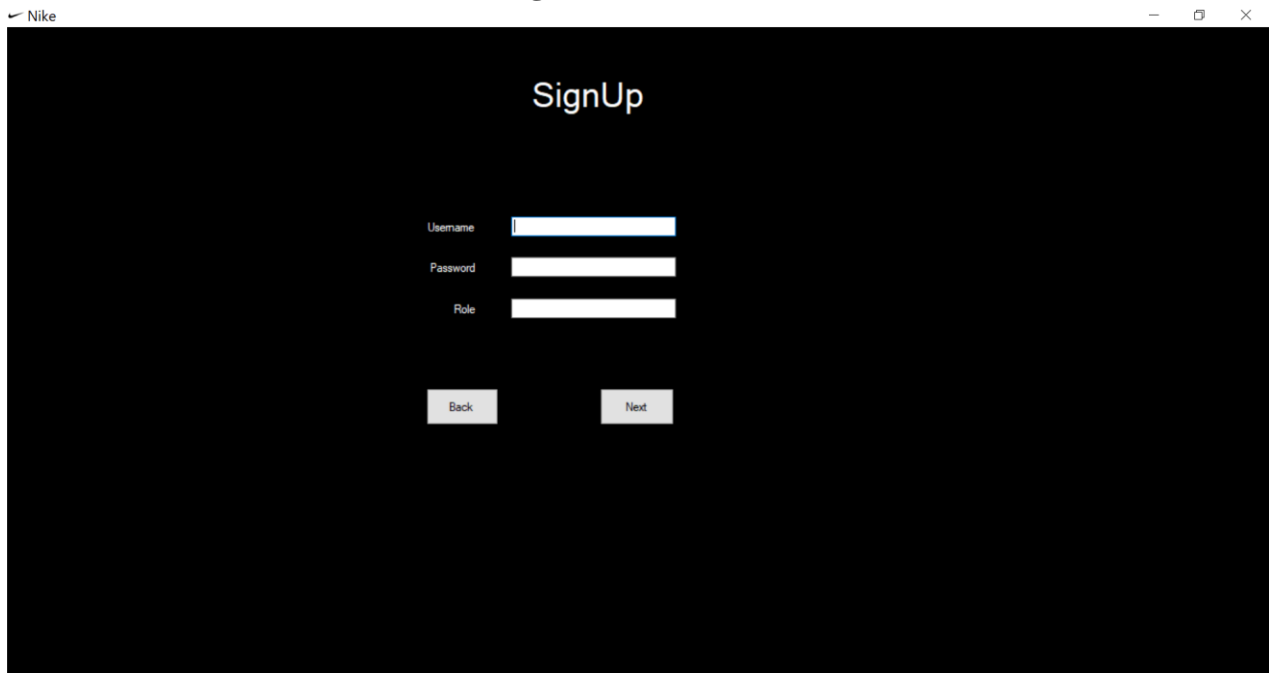
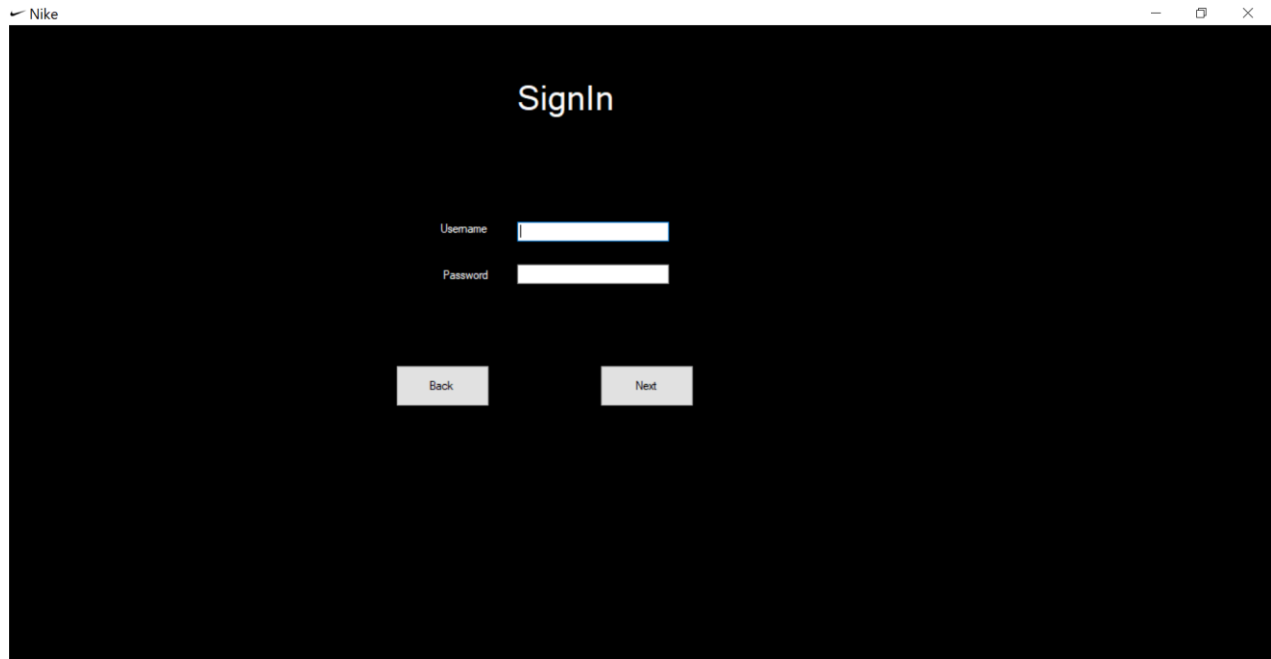


Fig.: Sign-Up



✓ Nike

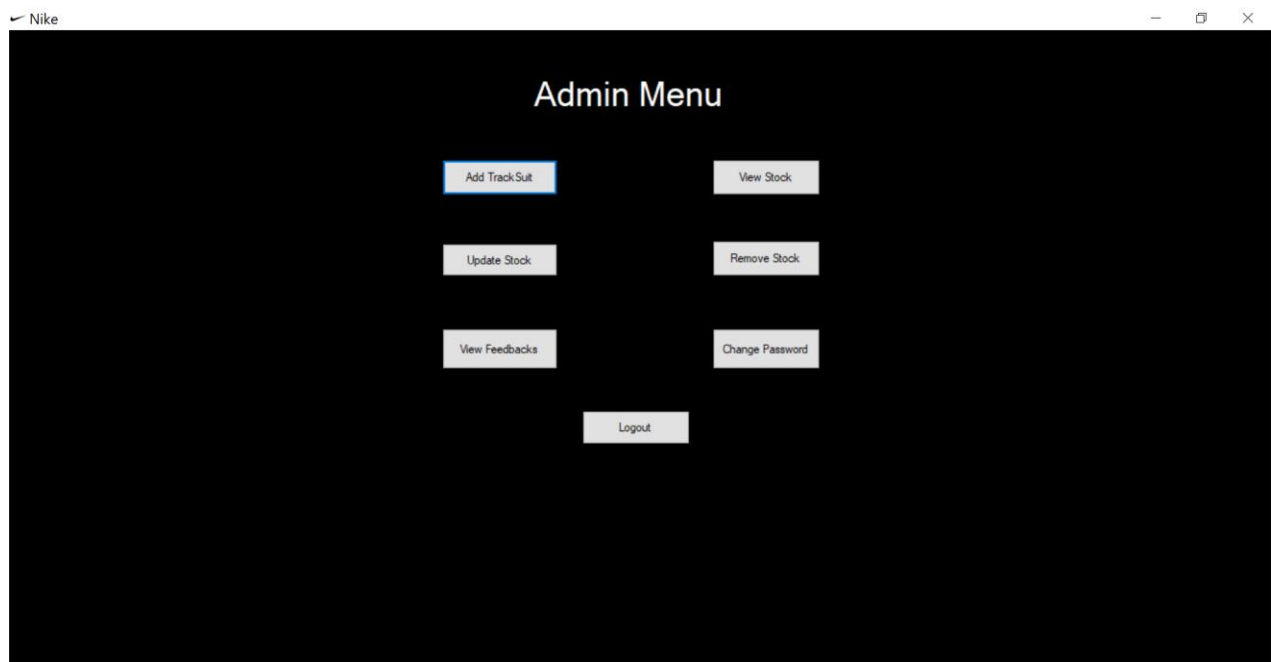
SignIn

Username

Password

Back Next

Fig.: Sign-In



✓ Nike

Admin Menu

Add Track Suit View Stock

Update Stock Remove Stock

View Feedbacks Change Password

Logout

Fig.: Admin Menu

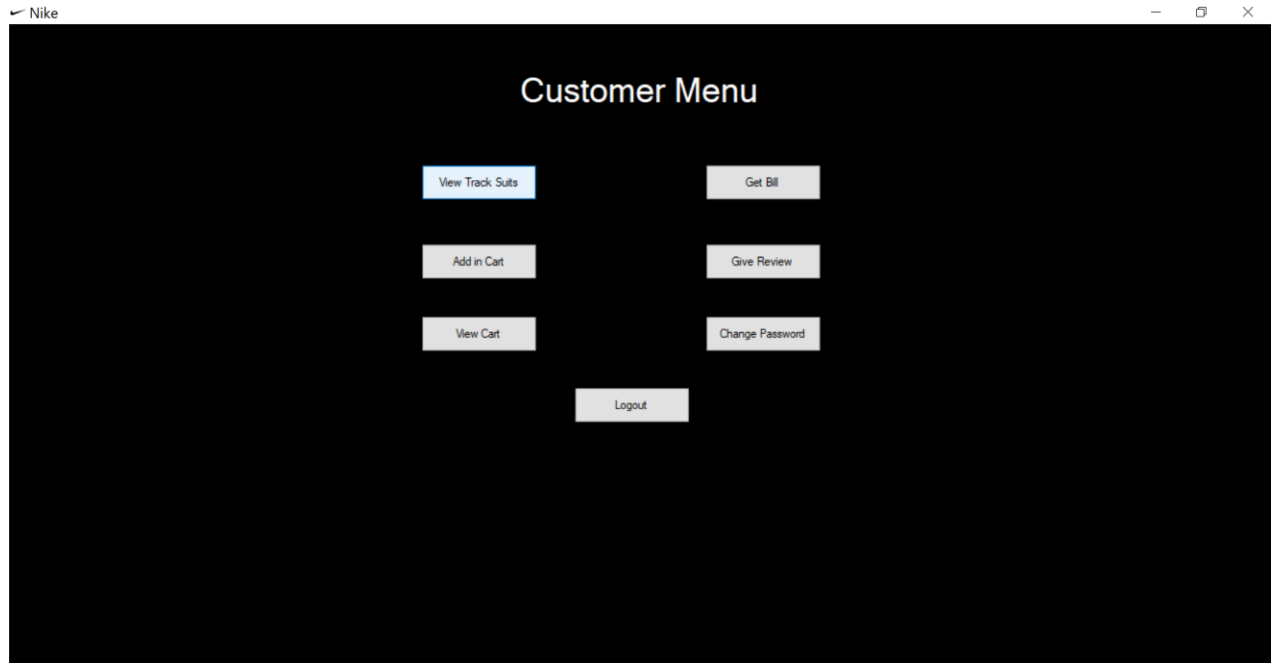


Fig.: Customer Menu

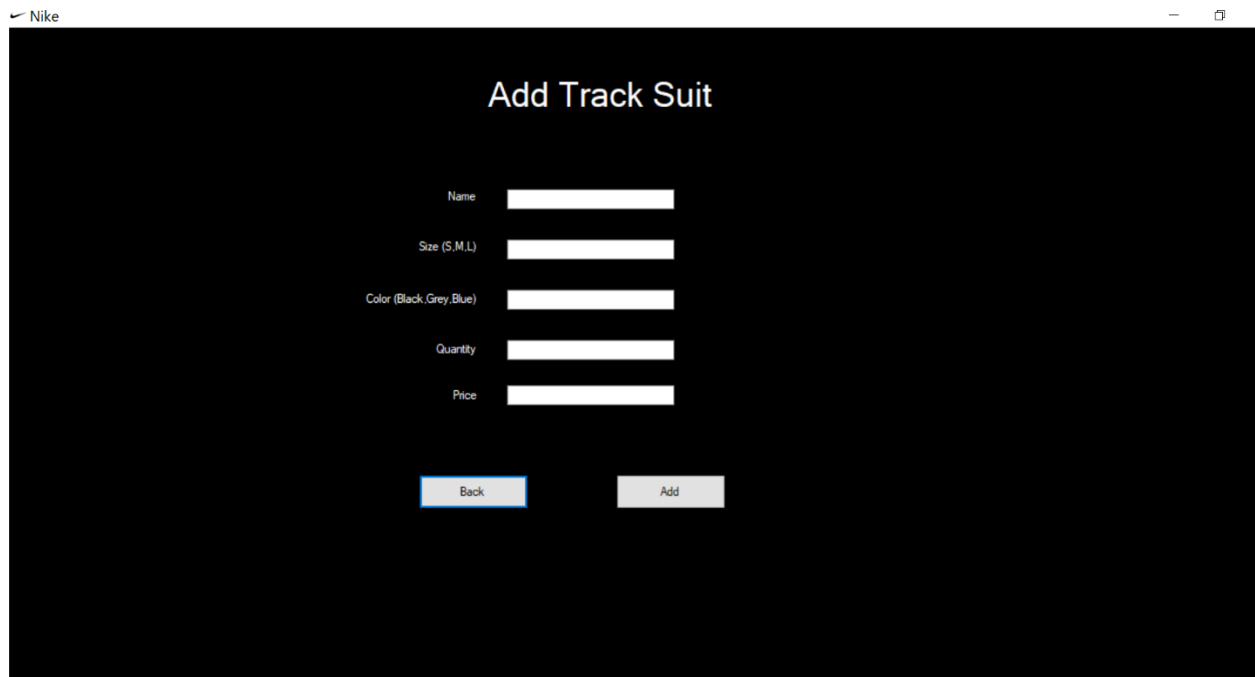


Fig.: Add Track-Suit

Nike Store

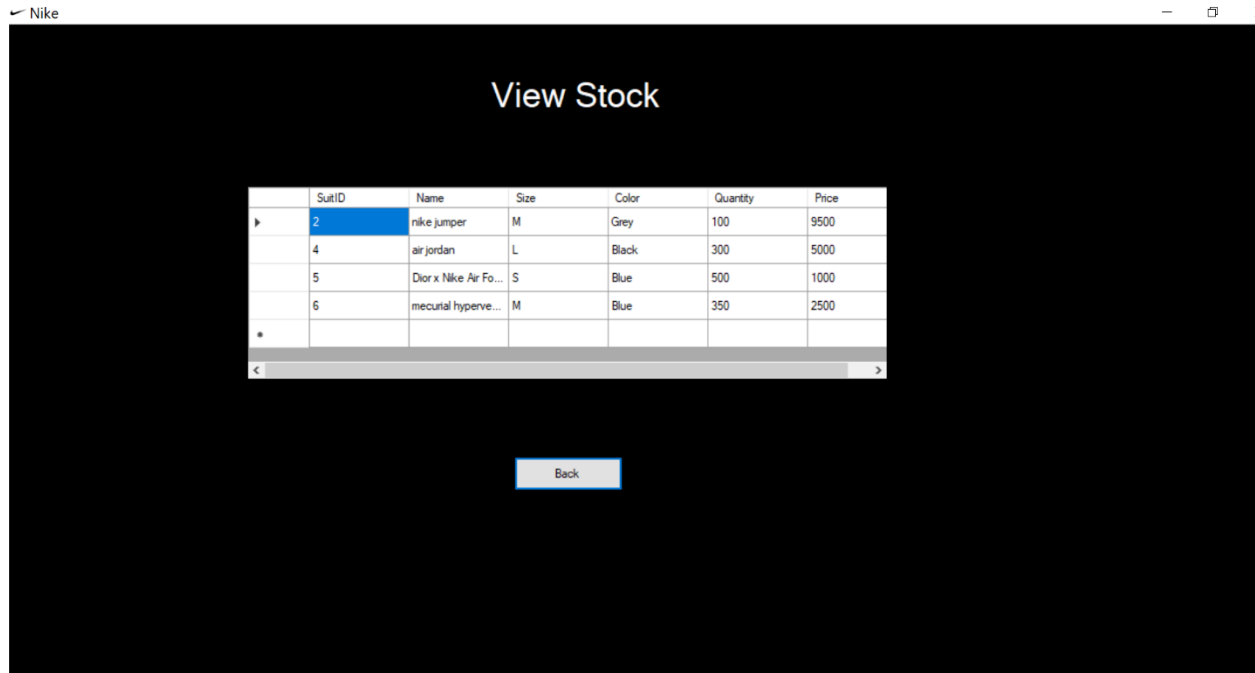


Fig.: View Stock

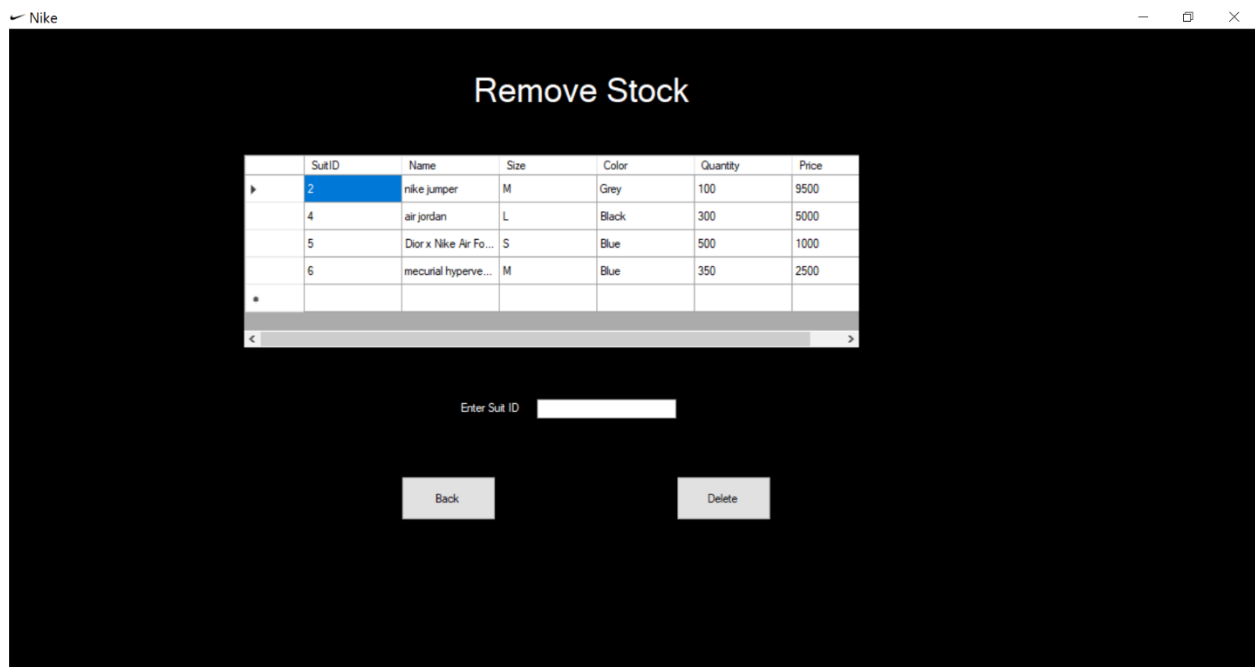


Fig.: Remove Stock

Update Stock

SuitID	Name	Size	Color	Quantity	Price
2	nike jumper	M	Grey	100	9500
4	air jordan	L	Black	300	5000
5	Dior x Nike Air Fo...	S	Blue	500	1000
6	mecurtal hyperve...	M	Blue	350	2500
*					

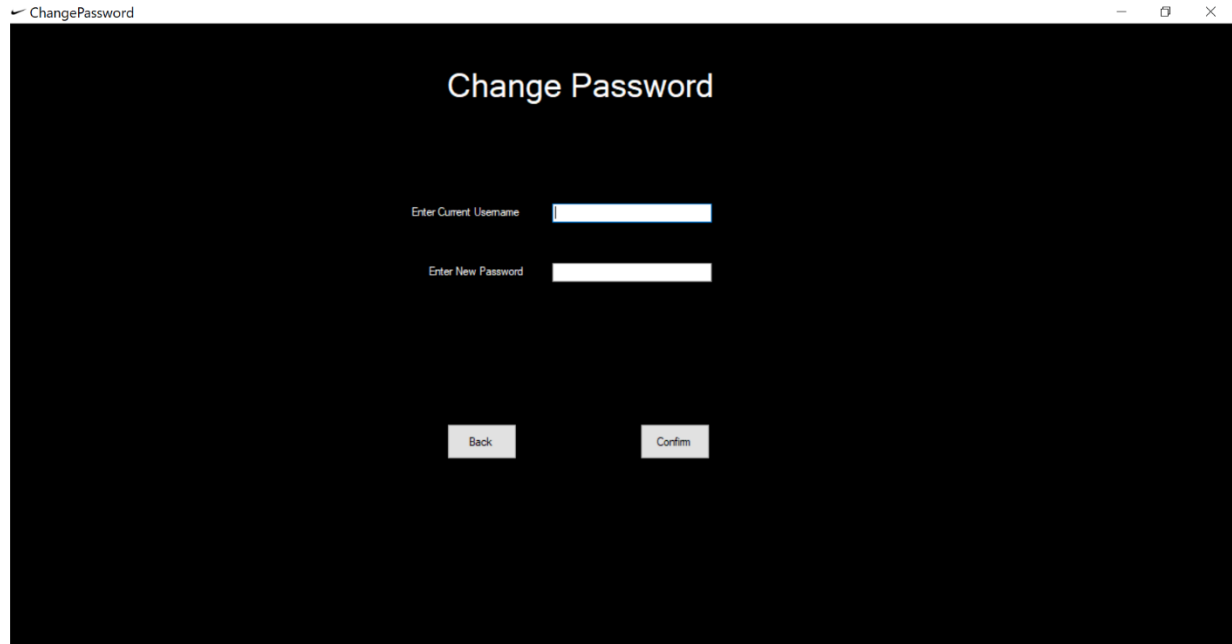
Enter Suit ID Enter New Price Enter New Quantity

Fig.: Update Stock

View Feedbacks

CustomerName	FeedbackText	Rating	DateSubmitted
abc	hello	1	4/23/2024
all	good quality fabric	4	4/23/2024
*			

Fig.: View Feedbacks



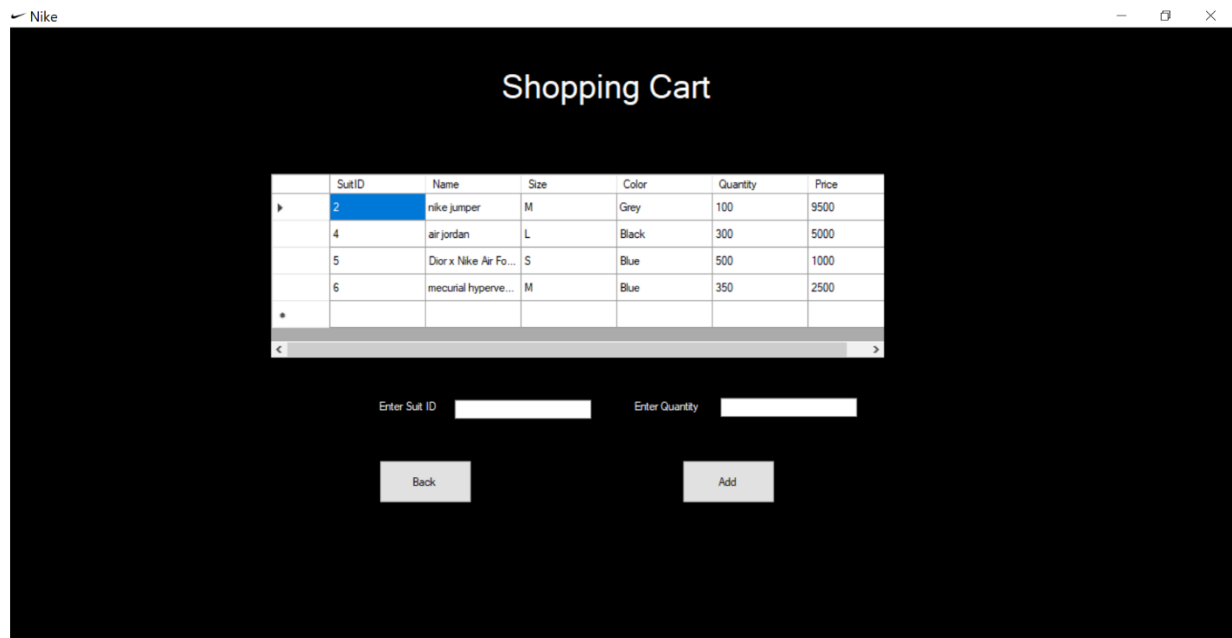
Change Password

Enter Current Username

Enter New Password

Back Confirm

Fig.: Change Password



Shopping Cart

SuitID	Name	Size	Color	Quantity	Price
2	nike jumper	M	Grey	100	9500
4	air jordan	L	Black	300	5000
5	Dior x Nike Air Fo...	S	Blue	500	1000
6	mecurial hyperven...	M	Blue	350	2500
*					

Enter Suit ID Enter Quantity

Back Add

Fig.: Add to Cart

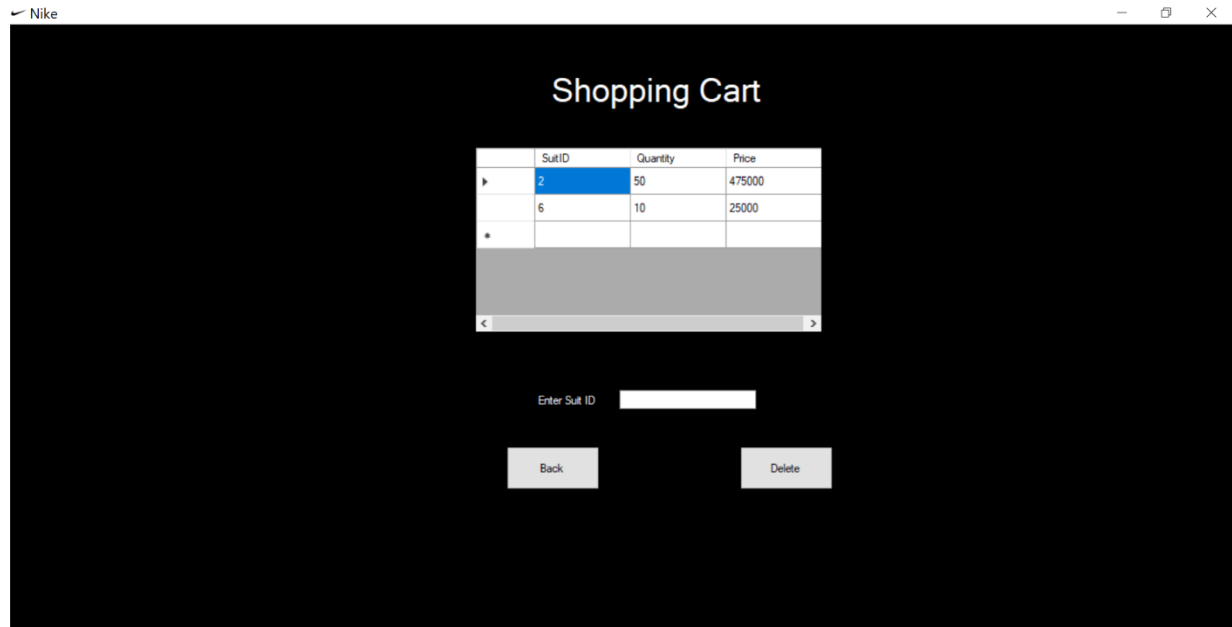


Fig.: Remove from Cart

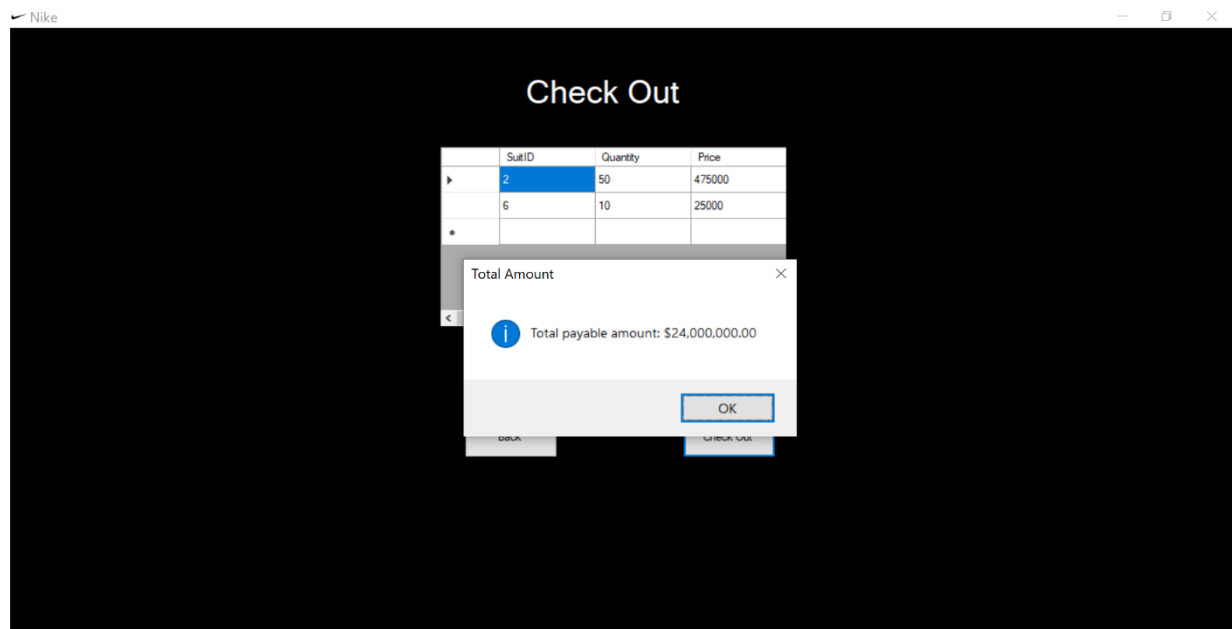
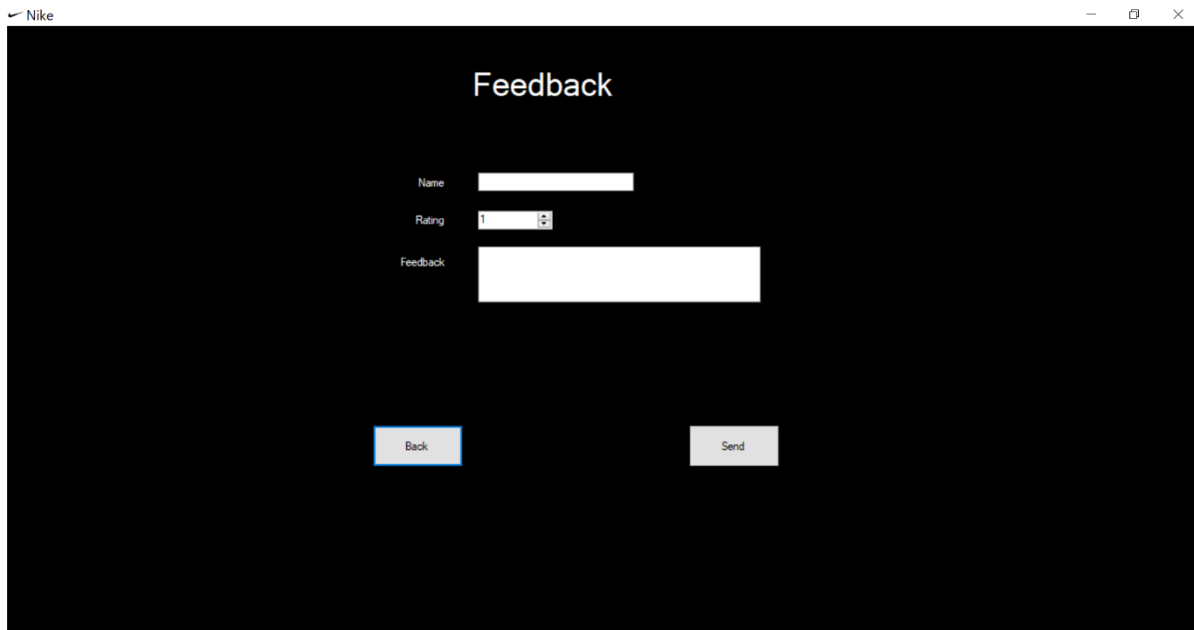


Fig.: Check-Out



The screenshot shows a 'Feedback' form with the following elements:

- Title:** Feedback
- Name:** A text input field.
- Rating:** A dropdown menu currently showing '1'.
- Feedback:** A larger text input field for comments.
- Buttons:** 'Back' and 'Send' buttons at the bottom.

Fig.: Give Review

7. Complete Code:

7.1. Business Logic (BL):

• ProductBL:

```
public class ProductBL
{
    private readonly ProductDB productDB;

    public ProductBL(ProductDB productDB)
    {
        this.productDB = productDB;
    }
    public int ProductId { get; set; }
    public string Name { get; set; }
    public string Size { get; set; }
    public string Color { get; set; }
    public int Quantity { get; set; }
    public decimal Price { get; set; }
    public ProductBL(string name, string color, string size, int quantity, decimal
price)
    {
        Name = name;
        Color = color;
        Size = size;
    }
}
```

```

        Quantity = quantity;
        Price = price;
    }
    public void AddSuit(string name, string size, string color, int quantity, decimal
price)
    {

    }

    public void ShowAllSuits()
    {
        // Call the corresponding function in the ProductDB class to retrieve all
shoes from the database
        productDB.ShowAllSuits();
    }
    public void UpdateSuit(int suitID, int quantity, decimal price)
    {
        productDB.UpdateSuit(suitID, quantity, price);
    }
    public void DeleteSuit(int suitID)
    {
        productDB.DeleteSuit(suitID);
    }
    public bool AddToSuitCart(int suitID, int quantity)
    {
        // Get the price of the suit
        decimal price = productDB.GetSuitPrice(suitID);

        if (price == -1)
        {
            // Suit ID not found
            return false;
        }

        // Calculate the total price
        decimal totalPrice = price * quantity;

        // Add the suit to the cart
        return productDB.AddToSuitCart(suitID, quantity, totalPrice);
    }
    public DataTable GetSuitCart()
    {
        return productDB.GetSuitCart();
    }
    public bool DeleteFromSuitCart(int suitID)
    {
        return productDB.DeleteFromSuitCart(suitID);
    }
    public decimal CalculateTotalPayableAmount()
    {
        return productDB.CalculateTotalPayableAmount();
    }
    public bool EmptyCartTables()
    {
        return productDB.EmptyCartTables();
    }
}

```



```

public bool AddFeedback(string customerName, string feedbackText, int rating)
{
    // Call the ProductDB method to add feedback
    return productDB.AddFeedback(customerName, feedbackText, rating);
}
public DataTable GetFeedback()
{
    DataTable feedbackTable = null;
    try
    {
        feedbackTable = productDB.GetFeedback();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while getting feedback:
{ex.Message}");
    }
    return feedbackTable;
}

```

7.2. Data Layer (DL):

• ProductDB:

```

public class ProductDB : IProductDL
{
    private string connectionString;

    // Constructor to initialize the connection string
    public ProductDB(string connectionString)
    {
        this.connectionString = connectionString;
    }
    private bool IsValidSuitColor(string color)
    {
        string[] validColors = { "black", "grey", "blue" };
        return Array.Exists(validColors, c => c.Equals(color,
StringComparison.OrdinalIgnoreCase));
    }
    private bool IsValidSuitSize(string size)
    {
        string[] validSize = { "S", "M", "L" };
        return Array.Exists(validSize, t => t.Equals(size,
StringComparison.OrdinalIgnoreCase));
    }
    public void AddSuit(string name, string size, string color, int
quantity, decimal price)
    {
        if (!IsValidSuitSize(size))
        {
            throw new ArgumentException("Invalid suit size (only
small,medium and large)");
        }
    }
}

```

```

        if (!IsValidSuitColor(color))
        {
            throw new ArgumentException("Invalid suit color (only black,
grey, blue.");
        }
        // SQL query to insert a new suit into the database

        string query = "INSERT INTO Suits (Name, Size, Color, Quantity,
Price) VALUES ( @Name, @Size, @Color, @Quantity, @Price)";

        // Open a connection to the database
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Create a SqlCommand object with the query and connection
            using (var command = new SqlCommand(query, connection))
            {
                // Add parameters to the query to prevent SQL injection
                command.Parameters.AddWithValue("@Name", name);
                command.Parameters.AddWithValue("@Size", size);
                command.Parameters.AddWithValue("@Color", color);
                command.Parameters.AddWithValue("@Quantity", quantity);
                command.Parameters.AddWithValue("@Price", price);

                // Execute the SQL command to insert the suit into the
database
                command.ExecuteNonQuery();
            }
        }
    }
    public DataTable ShowAllSuits()
    {
        DataTable suitData = new DataTable();

        try
        {
            using (var connection = new SqlConnection(connectionString))
            {
                connection.Open();

                // Query to select all suits from the database
                string query = "SELECT * FROM Suits";

                // Create a SqlCommand to execute the query
                SqlCommand command = new SqlCommand(query, connection);

                // Execute the query and fill the DataTable with the
retrieved data
                SqlDataAdapter adapter = new SqlDataAdapter(command);
                adapter.Fill(suitData);
            }
        }
    }

```

```

    }
    catch (Exception ex)
    {
        // Handle any exceptions that occur during database interaction
        Console.WriteLine($"An error occurred: {ex.Message}");
    }

    return suitData;
}
public void UpdateSuit(int suitID, int quantity, decimal price)
{
    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Query to update the quantity and price of the suit with
            the given ID
            string query = "UPDATE Suits SET Quantity = @Quantity,
            Price = @Price WHERE suitID = @suitID";

            // Create a SqlCommand to execute the query
            SqlCommand command = new SqlCommand(query, connection);

            // Set the parameters for the query
            command.Parameters.AddWithValue("@suitID", suitID);
            command.Parameters.AddWithValue("@Quantity", quantity);
            command.Parameters.AddWithValue("@Price", price);

            // Execute the query
            int rowsAffected = command.ExecuteNonQuery();

            // Check if any rows were affected by the update operation
            if (rowsAffected > 0)
            {
                Console.WriteLine("suit quantity and price updated
            successfully.");
            }
            else
            {
                Console.WriteLine("No suit found with the provided
            ID.");
            }
        }
    }
    catch (Exception ex)
    {
        // Handle any exceptions that occur during database interaction
        Console.WriteLine($"An error occurred: {ex.Message}");
    }
}
public void DeleteSuit(int suitID)

```

```

    {
        try
        {
            using (var connection = new SqlConnection(connectionString))
            {
                connection.Open();

                // Query to delete the suit with the given ID
                string query = "DELETE FROM suits WHERE suitID = @suitID";

                // Create a SqlCommand to execute the query
                SqlCommand command = new SqlCommand(query, connection);
                command.Parameters.AddWithValue("@suitID", suitID);

                // Execute the query
                int rowsAffected = command.ExecuteNonQuery();

                if (rowsAffected == 0)
                {
                    // If no rows were affected, the suit with the given
                    ID doesn't exist
                    throw new Exception($"No suit found with ID {suitID}.");
                }
            }
        }
        catch (Exception ex)
        {
            // Handle any exceptions that occur during the deletion
            operation
            throw new Exception("Failed to delete suit.", ex);
        }
    }

    public DataTable GetFeedback()
    {
        DataTable feedbackTable = new DataTable();

        try
        {
            using (SqlConnection connection = new
            SqlConnection(connectionString))
            {
                connection.Open();

                string query = "SELECT * FROM Feedbacks";

                using (SqlCommand command = new SqlCommand(query,
                connection))
                {
                    using (SqlDataAdapter adapter = new
                    SqlDataAdapter(command))
                    {

```

```

        adapter.Fill(feedbackTable);
    }
}
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred while retrieving
feedback: {ex.Message}");
}

return feedbackTable;
}
public bool AddToSuitCart(int SuitID, int quantity, decimal price)
{
    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Check if the suit with the given ID exists in the suits
            string checkQuery = "SELECT COUNT(*) FROM Suits WHERE
SuitID = @SuitID";
            SqlCommand checkCommand = new SqlCommand(checkQuery,
connection);
            checkCommand.Parameters.AddWithValue("@SuitID", SuitID);
            int SuitExists = (int)checkCommand.ExecuteScalar();

            if (SuitExists == 0)
            {
                throw new Exception("The Suit with the provided ID does
not exist.");
            }

            // Insert the Suit into the Cart table
            string insertQuery = "INSERT INTO Cart (SuitID, Quantity,
Price) VALUES (@SuitID, @Quantity, @Price)";
            SqlCommand insertCommand = new SqlCommand(insertQuery,
connection);
            insertCommand.Parameters.AddWithValue("@SuitID", SuitID);
            insertCommand.Parameters.AddWithValue("@Quantity",
quantity);
            insertCommand.Parameters.AddWithValue("@Price", price);

            int rowsAffected = insertCommand.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
    catch (Exception ex)
    {
        // Handle any exceptions that occur during database interaction
    }
}

```

```

        Console.WriteLine($"An error occurred: {ex.Message}");
        return false;
    }
}
public decimal GetSuitPrice(int SuitID)
{
    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Query to select the price of the Suit with the given ID
            string query = "SELECT Price FROM Suits WHERE SuitID =
@SuitID";

            // Create a SqlCommand to execute the query
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@SuitID", SuitID);

            // Execute the query and retrieve the price
            object result = command.ExecuteScalar();

            if (result != null && result != DBNull.Value)
            {
                return Convert.ToDecimal(result);
            }
            else
            {
                // Suit ID not found
                return -1;
            }
        }
    }
    catch (Exception ex)
    {
        // Handle any exceptions that occur during database interaction
        Console.WriteLine($"An error occurred: {ex.Message}");
        return -1;
    }
}
public DataTable GetSuitCart()
{
    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Query to select all rows from the SuitCart table
            string query = "SELECT * FROM Cart";

```

```

        // Create a SqlDataAdapter to fill a DataTable with the
results of the query
        SqlDataAdapter adapter = new SqlDataAdapter(query,
connection);

        // Create a new DataTable to store the results
        DataTable SuitCartTable = new DataTable();

        // Fill the DataTable with the results of the query
        adapter.Fill(SuitCartTable);

        return SuitCartTable;
    }
}
catch (Exception ex)
{
    // Handle any exceptions that occur during database interaction
    Console.WriteLine($"An error occurred: {ex.Message}");
    return null;
}
}
public bool DeleteFromSuitCart(int SuitID)
{
    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Check if the Suit with the given ID exists in the
SuitCart table
            string checkQuery = "SELECT COUNT(*) FROM Cart WHERE SuitID
= @SuitID";
            SqlCommand checkCommand = new SqlCommand(checkQuery,
connection);
            checkCommand.Parameters.AddWithValue("@SuitID", SuitID);
            int SuitExists = (int)checkCommand.ExecuteScalar();

            if (SuitExists == 0)
            {
                throw new Exception("The Suit with the provided ID does
not exist in the cart.");
            }

            // Delete the Suit from the SuitCart table
            string deleteQuery = "DELETE FROM Cart WHERE SuitID =
@SuitID";
            SqlCommand deleteCommand = new SqlCommand(deleteQuery,
connection);
            deleteCommand.Parameters.AddWithValue("@SuitID", SuitID);

            int rowsAffected = deleteCommand.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```

```

        }
    }
    catch (Exception ex)
    {
        // Handle any exceptions that occur during database interaction
        Console.WriteLine($"An error occurred: {ex.Message}");
        return false;
    }
}

public decimal CalculateTotalPayableAmount()
{
    decimal totalAmount = 0;

    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Calculate total amount from SuitCart
            string SuitCartQuery = "SELECT SUM(Price * Quantity) FROM
Cart";
            SqlCommand SuitCartCommand = new SqlCommand(SuitCartQuery,
connection);

            object SuitCartTotal = SuitCartCommand.ExecuteScalar();
            if (SuitCartTotal != null && SuitCartTotal != DBNull.Value)
            {
                totalAmount += Convert.ToDecimal(SuitCartTotal);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred: {ex.Message}");
    }

    return totalAmount;
}

public bool EmptyCartTables()
{
    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Empty SuitCart
            string emptySuitQuery = "DELETE FROM Cart";
            SqlCommand emptySuitCommand = new
SqlCommand(emptySuitQuery, connection);
            emptySuitCommand.ExecuteNonQuery();
        }
    }
}

```



```

        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred: {ex.Message}");
        return false;
    }
}

public bool AddFeedback(string name, string feedbackText, int rating)
{
    try
    {
        using (var connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Insert the feedback into the Feedback table
            string insertQuery = "INSERT INTO Feedbacks (CustomerName,
FeedbackText, Rating, DateSubmitted) " +
                                "VALUES (@CustomerName,
@FeedbackText, @Rating, GETDATE())";
            SqlCommand insertCommand = new SqlCommand(insertQuery,
connection);

            insertCommand.Parameters.AddWithValue("@CustomerName",
name);
            insertCommand.Parameters.AddWithValue("@FeedbackText",
feedbackText);
            insertCommand.Parameters.AddWithValue("@Rating", rating);

            int rowsAffected = insertCommand.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
    catch (Exception ex)
    {
        // Handle any exceptions that occur during database interaction
        Console.WriteLine($"An error occurred: {ex.Message}");
        return false;
    }
}
}

```

• ProductFH:

```

public class ProductFH : IProductDL
{
    string filePath = "tracksuits.txt";
    public ProductFH(string filePath)
    {
        this.filePath = filePath;
    }
}

```

```

private List<ProductBL> products = new List<ProductBL>();
public void WriteToFile()
{
    using (StreamWriter sw = new StreamWriter(filePath))
    {
        foreach (var product in products)
        {
            sw.WriteLine($"{product.Name},{product.Color},{product.Size},{product.Quantity},{product
.Price}");
        }
    }
    //Console.WriteLine("Products written to file successfully!");
}

public void ReadFromFile()
{
    if (File.Exists(filePath))
    {
        using (StreamReader sr = new StreamReader(filePath))
        {
            string line;
            while ((line = sr.ReadLine()) != null)
            {
                string[] parts = line.Split(',');
                if (parts.Length == 5)
                {
                    string name = parts[0];
                    string color = parts[1];
                    string size = parts[2];
                    int quantity;
                    if (int.TryParse(parts[3], out quantity))
                    {
                        decimal price;
                        if (decimal.TryParse(parts[4], out price))
                        {
                            products.Add(new ProductBL(name, color, size, quantity,
price));
                        }
                    }
                }
            }
        }
        //Console.WriteLine("Products read from file successfully!");
    }
    else
    {
        //Console.WriteLine("No products file found.");
    }
}

public void AddProduct(ProductBL product)
{
    products.Add(product);
    WriteToFile();
}

public List<ProductBL> GetProducts()
{
    return products;
}

```

Nike Store

```
    }
    public bool DeleteProductByName(string name)
    {
        ProductBL productToDelete = products.FirstOrDefault(p => p.Name.Equals(name,
StringComparison.OrdinalIgnoreCase));
        if (productToDelete != null)
        {
            products.Remove(productToDelete);
            WriteToFile(); // Update file after deleting the product
            return true;
        }
        else
        {
            return false;
        }
    }
    public bool UpdateProductQuantityByName(string name, int newQuantity)
    {
        ProductBL productToUpdate = products.FirstOrDefault(p => p.Name.Equals(name,
StringComparison.OrdinalIgnoreCase));
        if (productToUpdate != null)
        {
            productToUpdate.Quantity = newQuantity;
            WriteToFile(); // Write to file after updating the product quantity
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

7.3. User Interface (UI):

• ProductUI:

```
public class ProductUI
{
    private ProductFH productFH;

    public static int MainMenu()
    {
        Console.WriteLine("-----NIKE-----");
        Console.WriteLine();
        Console.WriteLine("1. Add TrackSuit");
        Console.WriteLine("2. View TrackSuits");
        Console.WriteLine("3. Update Stock");
        Console.WriteLine("4. Delete TrackSuit");
        Console.WriteLine("5. Exit");
        Console.WriteLine();
        Console.Write("Enter your choice: ");

        int choice;
        if (int.TryParse(Console.ReadLine(), out choice))
        {
            return choice;
        }
    }
}
```

```

        }
        else
        {
            return -1; // Invalid choice
        }
    }
}
public static void AddProduct(ProductFH productFH)
{
    Console.WriteLine("Enter TrackSuit name: ");
    string name = Console.ReadLine();

    Console.WriteLine("Enter color: ");
    string color = Console.ReadLine();

    Console.WriteLine("Enter size: ");
    string size = Console.ReadLine();

    int quantity;
    do
    {
        Console.WriteLine("Enter stock quantity: ");
    } while (!int.TryParse(Console.ReadLine(), out quantity));

    decimal price;
    do
    {
        Console.WriteLine("Enter price: ");
    } while (!decimal.TryParse(Console.ReadLine(), out price));

    productFH.AddProduct(new ProductBL(name, color, size, quantity, price));
    Console.WriteLine();
    Console.WriteLine("TrackSuit added successfully!");
}

public static void ViewProducts(ProductFH productFH)
{
    List<ProductBL> products = productFH.GetProducts();
    Console.WriteLine();
    if (products.Count == 0)
    {
        Console.WriteLine("No TrackSuit available.");
    }
    else
    {
        Console.WriteLine();
        Console.WriteLine("TrackSuit List:");
        Console.WriteLine();
        foreach (var product in products)
        {
            Console.WriteLine($"Name: {product.Name}, Color: {product.Color},
Size: {product.Size}, Quantity: {product.Quantity}, Price: {product.Price}");
            Console.WriteLine();
        }
    }
}

public static void DeleteProduct(ProductFH productFH)

```

Nike Store

```
{
    Console.WriteLine();
    Console.Write("Enter TrackSuit name to delete: ");
    string nameToDelete = Console.ReadLine();

    bool deleted = productFH.DeleteProductByName(nameToDelete);
    if (deleted)
    {
        Console.WriteLine();
        Console.WriteLine($"TrackSuit '{nameToDelete}' deleted successfully!");
    }
    else
    {
        Console.WriteLine();
        Console.WriteLine($"TrackSuit '{nameToDelete}' not found.");
    }
}

public static void UpdateProductQuantity(ProductFH productFH)
{
    Console.WriteLine();
    Console.Write("Enter TrackSuit name to update stock: ");
    string productName = Console.ReadLine();

    Console.Write("Enter new stock quantity: ");
    int newQuantity;
    while (!int.TryParse(Console.ReadLine(), out newQuantity))
    {
        Console.WriteLine();
        Console.WriteLine("Invalid quantity. Please enter a valid integer value.");
        Console.Write("Enter new stock quantity: ");
    }

    bool updated = productFH.UpdateProductQuantityByName(productName, newQuantity);
    if (updated)
    {
        Console.WriteLine();
        Console.WriteLine($"Quantity updated successfully for TrackSuit '{productName}'.");
    }
    else
    {
        Console.WriteLine();
        Console.WriteLine($"TrackSuit '{productName}' not found.");
    }
}
}
```

8. CONCLUSION:

In conclusion, I will summarize the Nike Store project, its achievements, the challenges faced during development, and the insights gained from this endeavor.

8.1. Summarization and Achievements:

Throughout the development of the project, several noteworthy achievements have been realized. The system offers a user-friendly interface that enables easy navigation and quick access to essential shop data. Real-time inventory tracking ensures that shop owners can efficiently manage stock levels, minimizing stockouts and maximizing profits. The system also incorporates features such as sales and purchase management, and data security measures, bolstering the system's overall functionality and reliability.

8.2. Challenges:

However, the project did encounter some challenges during its development journey. One significant challenge was ensuring seamless communication and integration between different modules within the system. Managing a vast amount of product data, sales records, and while maintaining data accuracy and consistency with database and file-handling also brought their own set of challenges. Additionally, implementing Library Project (DLL) techniques required careful consideration and attention to detail.

8.3 Lessons Learned:

Throughout the project, several valuable lessons were learned that have contributed to a better understanding of software development and management systems. Effective planning and thorough requirement gathering were vital for the successful development of a complex system like this. Emphasizing modularity and separation of concerns played a crucial role in achieving maintainability and adaptability. Additionally, continuous testing and quality assurance were indispensable for identifying and rectifying any issues early in the development process.