21 de abril de 2024

Natalia Andrea Herrera Espinosa

2232086

College LaSalle

420-DA3-AS\_Final\_Project

Multi-tier Applications Development

Contenido

[Project Description 4](#_Toc164571801)

[Project Design 5](#_Toc164571802)

[Database Design 5](#_Toc164571803)

[Relationships Between Tables 6](#_Toc164571804)

[Data Flow and Integrity 7](#_Toc164571805)

[Referential Integrity and Foreign Keys 8](#_Toc164571806)

[Design of Application Domain Classes 9](#_Toc164571807)

[Module 1: MIS Manager 10](#_Toc164571808)

[Module 2: Sales Manager 10](#_Toc164571809)

[Module 3: Inventory Controller 10](#_Toc164571810)

[Module 4: Order Clerks 11](#_Toc164571811)

[Class Diagram 11](#_Toc164571812)

[Design of GUI Classes 12](#_Toc164571813)

[Module 1: MIS Manager 12](#_Toc164571814)

[Module 2: Sales Manager 14](#_Toc164571815)

[Module 3: Inventory Controller 17](#_Toc164571816)

[Module 4: Order Clerks 27](#_Toc164571817)

[Design of Data Access Classes 33](#_Toc164571818)

[Module 1: MIS Manager 33](#_Toc164571819)

[Module 2: Sales Manager 36](#_Toc164571820)

[Module 3: Inventory Controller 36](#_Toc164571821)

[Module 4: Order Clerks 39](#_Toc164571822)

[Design of Validations 41](#_Toc164571823)

[Module 1: MIS Manager 41](#_Toc164571824)

[Module 2: Sales Manager 42](#_Toc164571825)

[Module 3: Inventory Controller 43](#_Toc164571826)

[Module 4: Order Clerks 45](#_Toc164571827)

[Project Implementation 47](#_Toc164571828)

[Module 1: MIS Manager (Connected Mode) 47](#_Toc164571829)

[Purpose and Functionality: 47](#_Toc164571830)

[Key Implementation Details: 47](#_Toc164571831)

[Module 2: Sales Manager (Disconnected Mode) 48](#_Toc164571832)

[Purpose and Functionality: 48](#_Toc164571833)

[Key Implementation Details: 48](#_Toc164571834)

[Module 3: Inventory Controller (Connected Mode) 48](#_Toc164571835)

[Purpose and Functionality: 48](#_Toc164571836)

[Key Implementation Details: 48](#_Toc164571837)

[Module 4: Order Clerks (Entity Framework) 49](#_Toc164571838)

[Purpose and Functionality: 49](#_Toc164571839)

[Key Implementation Details: 49](#_Toc164571840)

[User Interface Integration: 49](#_Toc164571841)

[Project Testing 50](#_Toc164571842)

[Conclusion 50](#_Toc164571843)

# Project Description

The Hi-Tech Distribution Inc. project, developed as a part of the Multi-tier Applications Development course, a challenging educational project that aims to combine the knowledge and abilities learned during the semester into a single, all-inclusive application. This project offers a realistic setting for using database and programming skills by simulating a business scenario for a made-up book distribution corporation.

The architecture of our project heavily relies on key ideas from the course, including LINQ, the Entity Framework, and Database Programming with ADO.NET in both Connected and Disconnected Modes. Our custom program is designed to handle a wide range of tasks for different types of users, including inventory controllers, sales managers, order clerks, and MIS managers.

Because the system is designed with CRUD operations in mind, users can easily add, read, update, and delete data. The program has strong display and search features and a user-friendly interface that makes complicated database interactions easier to understand. Our attention to these details is to emulate the dynamic workflows that a high-tech book distributor would normally use.

Our goal with the Hi-Tech Distribution Inc. project is to show off our technical expertise in database management and software development, as well as our comprehension of how these abilities relate to corporate logistics and management. This project demonstrates that we are prepared to face the challenges of contemporary business technology in the workplace, having acquired the necessary skills and expertise.

# Project Design

## Database Design

A crucial component of this multi-tier program is database architecture. Relationships between entities is the main focus of this design since they facilitate data organization and reveal how various system components interact with one another. Several columns in our project Hi-Tech Distribution database reflect different organizations that are a part of different forms like an order management system. Orders, Order Details, Books, Employees, Statuses, Customers, and other entities are examples of these entities. With distinct fields and keys for every table, the database can communicate with one another.

The initial phase of our database architecture was highly collaborative. Our team gathered in person to brainstorm and sketch the foundational structure based on the project requirements. This session was crucial as it allowed us to discuss the scope and objectives of the project in detail, guided by our course content on Multi-tier Application Development.

We began by thoroughly reading the project brief to identify necessary functionalities and the types of data to manage. Then sketching Core Tables initial tables such as Orders, Employee, UserAccount, Books, Author, Customer, and Publisher were outlined. After this first phase we stepped to Stage 2: Ideation and Relational Mapping, we did a brainstorming Relationships: We discussed the relationships between tables, categorizing them as one-to-one, one-to-many, and many-to-many to ensure a robust relational structure. Each table's role was aligned with user needs, focusing on CRUD operations and search functionalities. And then finally we refined and finalized Database Schema, adjustments were made to optimize performance and scalability, including setting up keys and indexing.

This streamlined approach ensured that our database design was closely aligned with both the project's requirements and the operational needs, laying a solid foundation for the subsequent development phases of the project. Even thought we make small changes in each module that we think it will work better.

### Relationships Between Tables

*One-to-Many Relationships:*

* Orders and Order Details: Each Order can include multiple Order Details, which mean that a single order can contain numerous items. Each entry in the Orders table is potentially linking to multiple entries in the Order Details table.
* Publishers and Books: A Publisher can release numerous Books, but each Book is published by only one Publisher. Here, the Publishers table is the parent, and the Books table is the child in a one-to-many relationship.
* Status and Orders: A Status can apply to multiple Orders, which means a single state like “In Process”, “Pending”, “Complete”, “Shipped” or “Canceled” can be associated with numerous individual orders.
* Books and Book Categories: A Category can have several Books, but each Book belongs to a single Category, denoting a one-to-many relationship.
* Employees and Jobs: Multiple Employees can hold the same Job Title, establishing a many-to-one relationship. Each record in the Employees table is associated with one record in the Jobs table, but a single Job Title in the Jobs table can correspond to many Employees.
* Customers and Orders: A single Customer can place multiple Orders, which means there's a one-to-many relationship from Customers to Orders. Each Customer can be associated with multiple entries (Orders). This scenario where a customer might make various purchases over time, each recorded as a separate order.

*Many-to-Many Relationships:*

* Books and Authors: Authors can write multiple Books, and each Book can be co-authored by multiple Authors. This is a many-to-many relationship typically represented using a pivot table like BooksAuthors to handle the associations between Books and Authors.

*One-to-One Relationships:*

* Employees and User Accounts: Each Employee has a corresponding User Account, which is used to manage system access and credentials. Each employee record in the Employees table is linked to a unique record in the User Accounts table.

Diagrama

Descripción generada automáticamente

### Data Flow and Integrity

Our created architecture maintains the data's integrity and progress. For example, a customer's order moves forward to the Orders database and is connected to different Order Details there. Because every Order Detail relates to a different book, the relational structure is mirrored, and data consistency and coherence are maintained across the order management system.

Every Book has a classification and is linked to a Publisher, defining a process from the Publishers to the Books they publish. Categories group several Books together under them, indicating how the system classifies them. Furthermore, numerous Employees are connected under shared job functions by the Job Titles from the Jobs table flowing into the Employees table.

A many-to-many relationship occurs when one or more authors write books, and authors may work on several books at once. The BooksAuthors table, an associative object that maintains the connections between Books and Authors, controls this intricate movement.

### Referential Integrity and Foreign Keys

A crucial component of our system's architecture is referential integrity, which guarantees logical and consistent links between tables. Foreign keys serve as references to primary keys in other tables and are used to enforce these relationships. We tried to make sure to this process that every Order Detail points to a certain Order, every Book is connected to a specific Publisher and Category, and every Order has a specific Customer and Status attached to it.

With foreign keys, employees are also mapped to the appropriate Job Titles and User Accounts. By ensuring that there are no isolated or unlinked entries, these links preserve the accuracy and dependability of the data.

## Design of Application Domain Classes

Texto

Descripción generada automáticamente con confianza media

The design of the application domain classes involves mapping out the core business entities and their relationships as they pertain to the functional requirements of our Hi-Tech Distribution application. For each module, we conceptualized classes that encapsulate the essential attributes and behaviors necessary for the application's business logic.

### Login

*UserAccount*: Represents and manages user credentials and authentication.

*LoginUserAccount*: Checks if the provided user ID and password match those in the database, returning true if the user is authenticated successfully.

### User Profile

*Purpose*: Represents and manages details for an employee within the system.

*Attributes*:

EmployeeId (int, private): Identifier for the employee.

FirstName, LastName (string, private): Name of the employee.

Email (string, private): Contact email of the employee.

JobTitle (string, private): Title of the employee's job.

*StatusDescription*: Description of the employee's current status.

*Methods:*

*SearchEmployee*: Retrieves an employee's information based on the provided ID.

### Module 1: MIS Manager

*UserAccount*: This class is designed to represent the system's users with properties to manage identification, authentication, and status. It includes attributes like userId, password, and statusId. Methods in this class allow for creating, updating, and validating user information.

*Employee*: Mirroring the structure of the UserAccount, the Employee class holds information related to the company's employees, including their job details. It provides functionalities to manage employee records, reflecting operations like adding a new employee or updating existing employee data.

### Module 2: Sales Manager

*Customer*: A class that serves as a data container for customer information. Its design includes attributes such as CustomerId and CustomerName and operations that allow for customer data manipulation, intended to be used in disconnected mode to support batch processing of customer data.

### Module 3: Inventory Controller

*AuthorItem*: This class is a simple representation of an author, used primarily in UI elements to list and select authors in the context of associating them with books.

Authors: It includes the details of Authors and is linked with books to establish the author of which book.

*Publisher*: It includes the details of book publishers and is linked with books to establish who publishes which book.

*BooksAuthors*: A junction class representing the many-to-many relationships between books and authors.

*BookCategory*: Designed to categorize books, this class includes a unique categoryId and a descriptive attribute to outline the genre or type.

*Book*: At the core of the inventory, the Book class includes information such as ISBN, title, and price. It also includes methods for CRUD operations and search functionality within the inventory system.

### Module 4: Order Clerks

*Status*: It encapsulates the possible states of an order, allowing the system to track and update the progress of customer orders.

Customer: A class that serves as a data container for customer information. It's mainly used to display the current customers in the database.

*OrdersDetail*: A class representing the line items within an order, linking the ordered books with quantities and prices to individual orders.

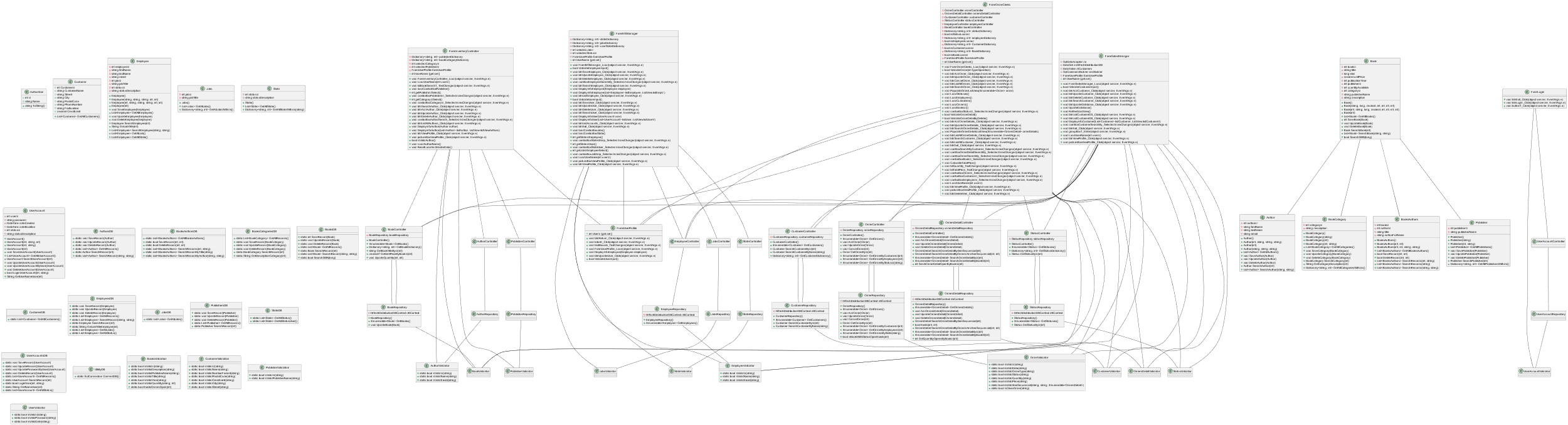
*Order*: This class acts as a container for customer orders, binding customer, employee, and order details in a coherent structure.

Each class was designed with careful consideration of how it would be used within the application, ensuring encapsulation of the necessary data and behaviors. The relationships between classes are established to mimic real-world interactions, such as authors writing multiple books or orders containing several order details.

### Class Diagram

A comprehensive class diagram was created to visualize the relationships between the classes described above. The diagram illustrates the attributes and methods of each class and the associations, such as inheritance and aggregations, which guide the development process and inform the creation of the database schema.

(The original class Diagram is attached to the zip)



## Design of GUI Classes

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

### Login

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

*Purpose:* Provides a user interface for the user to input their credentials and gain access to the application.

*Components:*

Text boxes for user input: txtUserId and txtPassword.

Buttons for actions: btnLogin, btnCancel, and btnExit.

*Behavior:*

btnLogin\_Click: Validates the input user ID and password using UserValidator and, if valid, calls LoginUserAccount to attempt authentication. Successful login leads to navigating to a user-specific form based on their job title.

### UserProfile

*Purpose*: Allows the user to view and update their profile information.

*Components*:

Text fields for displaying employee information: txtEmployeeId, txtEmail, txtFirstName, txtLastName, textBoxJob, and textBoxStatus.

Buttons for actions: btnUpdateUser and btnReturn.

*Behavior*:

FormUserProfile\_Load: Populates the form fields with the current employee's information upon loading.

btnUpdateUser\_Click: Validates new password input and, if valid, updates the user's password in the system.

### Module 1: MIS Manager

Interfaz de usuario gráfica

Descripción generada automáticamente

The FormMISManager class is the user interface for the MIS Manager module. It is designed with meticulous attention to detail, featuring text boxes for a easily user input, combo boxes for selection from pre-loaded lists, and action buttons for CRUD operations. The initialization of these controls and the pre-population of necessary data is managed when the form is loaded, ensuring the UI is ready for interaction.

**Employee Management Panel:**

Input Fields: Text boxes are provided for entering employee names and emails, with validation logic enforced to ensure data correctness.

Job and Status Selection: Combo boxes linked to jobsDictionary and stateDictionary allow for the selection of an employee's job and status from system-maintained lists, enhancing data consistency.

Action Buttons: The 'Add', 'Update', and 'Delete' buttons facilitate employee data management, invoking event handlers like btnSaveEmployee\_Click that communicate with the BLL after passing input validation.

**User Account Management Panel:**

Account Details: Fields for the User ID and Password capture essential account information. Password requirements are specified to guide the MIS Manager in setting secure credentials.

Status Dropdown: A combo box sourced from userStateDictionary provides selectable account statuses to maintain uniformity in user state management.

List Views: Both panels are anchored by ListViews that display summaries of employee and user accounts, creating an accessible data overview for management purposes.

**GUI Design and Interaction:**

Feedback and Validation: Dialog messages confirm successful operations or alert to issues, ensuring constant communication with the user. The EmployeeValidator and UserValidator ensure that all user inputs meet established business rules before proceeding with any action.

Data Display and Search: Methods such as DisplayInfoEmployee and event handlers for search functions allow for easy data retrieval and manipulation, making the user interface a powerful tool for data management.

Welcome and Profile: The top of the form greets the user with a welcome message and provides access to the user profile through the 'View Profile' button that direct to the User profile with the information and the password changing.

Exit Strategy: An 'Exit' button is provided as a clear means to safely close the application, demonstrating thoughtful design in application flow and user exit scenarios.

Consistent Interface Design: Across the form, a consistent design language is used. This consistency extends to color schemes, control placement, and interaction patterns, ensuring that the user can learn and navigate the system with ease.

### Module 2: Sales Manager

Interfaz de usuario gráfica, Escala de tiempo

Descripción generada automáticamente con confianza media

The FormSalesManager class is crafted to be the central interface for the Sales Manager module, facilitating the management of customer information within the system. This form mirrors the attention to detail seen in the MIS Manager module thinking of the components to the sales domain's specific needs.

**Customer Management Panel:**

Input Fields: The customer management section is equipped with input fields for capturing comprehensive customer details such as Name, Street, City, Postal Code, Phone Number, Fax Number, and Credit Limit. These fields validate the user's input to ensure accuracy and adherence to business rules.

Auto-Generated Customer ID: The Customer ID field is designed to be automatically populated during the customer creation process, signifying a unique identifier for each customer.

Action Buttons: Located beneath the input fields are 'Add', 'Update', and 'Delete' buttons that serve as the interface for performing CRUD operations on customer data. They trigger the appropriate data manipulation events in the BLL after rigorous validation checks.

Search Functionality: A search feature, accompanied by a dropdown for selecting search criteria and a search box for query input, enables the Sales Manager to retrieve customer records based on specific parameters.

**GUI Design and Interaction:**

Feedback and Validation: Dialog messages provide immediate and clear feedback to the user regarding the success or issues encountered during operations, facilitated by classes like CustomerValidation.

Data Presentation: A ListView control is prominently placed at the bottom of the form, displaying a detailed list of customers, offering the Sales Manager a quick reference to the entire customer base and the ability to select records for editing.

Welcome and Profile: The UI greets the Sales Manager with their name at the top and provides a direct link to their user profile via the 'View Profile' button, ensuring a personalized experience.

Consistent Layout: The design employs a consistent layout strategy with a cohesive color scheme and logical control placement, aligning with the rest of the system's user interface for a seamless user experience.

**DataSet Integration:**

Disconnected Mode Operations: The form operates in a disconnected mode, utilizing a DataSet to hold customer data, which is then synchronized with the database using an SqlDataAdapter and command builders.

Live Data Refresh: Methods like RefreshListView and btnListCustomerDB\_Click ensure the ListView is always up-to-date with the latest information from the database.

**Navigation and Accessibility:**

Exit Protocol: An 'Exit' button is strategically placed for easy navigation, allowing the Sales Manager to close the application safely.

Profile Management: The form includes the ability to access and manage the Sales Manager's profile, further personalizing the user experience within the system.

### Module 3: Inventory Controller

Interfaz de usuario gráfica

Descripción generada automáticamente

The FormInventoryController class is a finely crafted interface within the application dedicated to Inventory Controllers. It provides a seamless and comprehensive environment for inventory management with an emphasis on user-friendliness and functionality.

**Books Tab - Book Inventory Panel:**

Input Fields: This section includes input fields for Book ID (auto-generated), Title, ISBN (with a format guideline), Unit Price, Publication Year, Quantity Available, and selection dropdowns for Publisher ID and Category ID, emphasizing data integrity and ease of entry.

Author Association: A CheckedListBox allows for the association of one or more authors with a new book, reinforcing the relational aspect of book data.

Action Buttons: 'Add', 'Update', and 'Delete' buttons control the inventory records' life cycle, with event handlers executing the logic after input validation is passed.

Search and Listing Features: A comprehensive search function aids in locating books based on various attributes. The 'List All Books' button populates a ListView with the complete book inventory for quick access and management.

**GUI Design and Interaction:**

Feedback and Interaction: The form employs dialogue messages for user feedback and validation logic to maintain data consistency.

Data Display: The ListView is an integral part of the UI, presenting book data in an organized manner, with capabilities for selection and modification.

Welcome and Profile Access: Echoing the personalized touch, the UI welcomes the Inventory Controller by name and allows profile viewing and editing through a distinct 'View Profile' button.

Consistency in Design: Adhering to the unified UI scheme, the design language across the form remains consistent, ensuring intuitive navigation.

**Additional Tabs for Extended Functionality:**

Authors, Categories, Publishers, BooksAuthors Tabs: These tabs house specific functionalities for managing associated book details, each with its dedicated input fields and list views, thus streamlining the complex data management process.

**Navigation and Control:**

Logical Flow and Exit Protocol: The 'Exit' button provides a clear path to exit the application, while the design ensures the logical flow of operations from creation to deletion.

Interfaz de usuario gráfica

Descripción generada automáticamente

**Authors Tab**

Input Fields: The Authors tab facilitates the management of author details within the inventory system. It contains fields for Author ID, which is automatically generated, as well as input fields for First Name, Last Name, and Email, ensuring all relevant details are captured for each author record.

Action Buttons: This section houses the 'Add', 'Update', and 'Delete' buttons, which serve as the control points for managing author records within the inventory. Each button is integrated with event handlers that, upon activation, validate the input data to ensure accuracy before any database operation is executed.

Search and Listing Features: Equipped with a dropdown to refine search criteria and a 'Search' button, this feature simplifies the process of finding authors based on ID or name. The 'List All Authors' button generates a comprehensive list of authors in the ListView, providing a quick overview and access to the full scope of author records.

**GUI Design and Interaction:**

Feedback and Interaction: The form is constructed to give instant feedback via dialogue messages, creating an interactive experience that keeps the user informed of the outcome of their actions.

Data Display: A ListView displays author details in a well-organized format, offering functionalities such as selection and editing, which facilitates efficient data handling.

Welcome and Profile Access: The user is greeted by name upon login, fostering a personalized experience. The 'View Profile' button is prominently placed, offering easy navigation to the user's profile management section.

Consistency in Design: The uniform design language ensures that users can navigate through different sections of the application with ease and familiarity.

**Navigation and Control:**

Logical Flow and Exit Protocol: The design of the Authors tab is intuitive, guiding the user through a logical sequence from data input to final actions. An 'Exit' button is intuitively positioned for safe and easy navigation out of the application.

Interfaz de usuario gráfica

Descripción generada automáticamente

**Books Categories Tab:**

Input Fields: The Books Categories tab is designed for the organization and management of book categories within the inventory system.

It contains an input field for the Category ID, which is automatically generated to ensure a unique identifier for each category, and a field for Description to detail the category's focus.

Action Buttons: 'Add', 'Update', and 'Delete' buttons enable the Inventory Controller to manage the category records seamlessly. These buttons have associated event handlers that trigger after validating the data to ensure the integrity and accuracy of the information.

Search and Listing Features: A search function, including a field for Category ID and a 'Search' button, allows for the efficient location of categories within the inventory.

The 'List All Categories' button fills a ListView with all categories, offering a snapshot view for easy access and reference.

**GUI Design and Interaction:**

Feedback and Interaction: The system provides real-time dialogue messages for user interactions, offering direct feedback on the success or failure of user actions.

Data Display: The ListView component displays the categories in a structured and readable format, supporting interactive features such as review and edit operations.

Welcome and Profile Access: True to the application's user-centric approach, this tab greets the Inventory Controller by name and facilitates easy access to the profile management through the 'View Profile' button.

Consistency in Design: The tab maintains a consistent design language with the rest of the system, enhancing the user's ability to navigate and use the application effectively.

**Navigation and Control:**

Logical Flow and Exit Protocol: The 'Exit' button is thoughtfully placed to allow users to close the application safely, reinforcing a user-friendly flow from the beginning to the end of any operation within the Books Categories tab.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

**Publishers Tab:**

Input Fields: The Publishers tab features dedicated fields for creating and maintaining publisher records within the system.

An automatic ID generation system is in place for Publisher ID, guaranteeing unique identification for each publisher.

The Publisher Name field is provided for inputting the name of the book publishers, ensuring that book records can be accurately linked to publishers.

Action Buttons: The inclusion of 'Add', 'Update', and 'Delete' buttons allows for comprehensive management of publisher data. The underlying event handlers activate upon user interaction after necessary validations are conducted.

These actions support the dynamic and ever-evolving nature of publishing records within the inventory system.

Search and Listing Features: A targeted search function is made available, which allows users to find publishers using the Publisher ID. This feature streamlines the process of managing a potentially large list of publishers.

'List All Publishers' is a key feature that presents the entire list of publishers in a ListView, facilitating easy access and administrative tasks.

**GUI Design and Interaction:**

Feedback and Interaction: Consistent with the application's user interface, dialog messages are used to provide feedback on user actions, enhancing the interactive experience.

Data Display: The ListView is central to the user interface, offering a neatly organized presentation of publisher data. It affords the capability for selection, modification, and deletion directly from the ListView.

Welcome and Profile Access: Users are greeted by name for a personalized experience and provided with the convenience of profile access through the 'View Profile' button for any necessary adjustments.

Consistency in Design: The design remains in harmony with the overall application, with a focus on intuitive navigation and a uniform aesthetic across the entire system.

**Navigation and Control:**

Logical Flow and Exit Protocol: The well-placed 'Exit' button exemplifies the user-friendly design philosophy, ensuring a clear and straightforward method to safely conclude the use of the application.

The Publishers tab, with its functional layout and comprehensive controls, exemplifies the system's commitment to efficiency and usability in inventory management.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**BooksAuthors Tab:**

Search Interface: This tab provides a specialized interface for querying the relationship between books and authors, illustrating the interconnected data within the inventory system.

Users can search by multiple criteria, including Book ID, Book Title, Author ID, or Author Name, showcasing the system’s flexibility and depth.

Listing Features: The 'List All Authors and Books' button is a standout feature, offering a quick way to view all books and their associated authors in one place. This can be particularly useful for inventory audits and catalog management.

**GUI Design and Interaction:**

Search Flexibility: Dropdown menus allow for dynamic search options, and the dual 'Search' buttons enable quick access to the records based on the selected criteria.

Data Display: The ListView component displays the search results, linking books to authors in an easily digestible format. Users can thus understand the author-book relationships at a glance.

Intuitive Navigation: The tab continues the design philosophy of intuitive navigation and user-friendly interactions that define the overall application experience.

**Control and Accessibility:**

Efficiency and Control: This tab places a significant emphasis on efficient data retrieval, granting users full control over how they view and interact with book-author relationships.

Consistent User Experience: The design consistency ensures that users who are familiar with other sections of the system can easily adapt to the functionalities presented in this tab.

Welcome and Profile Access: As with other tabs, the 'View Profile' button offers personalized access to user settings, maintaining a consistent access point across the system.

**Logical Flow and Exit Protocol:**

The system's logical flow is maintained within this tab, ensuring that users can perform searches and view lists without disrupting their workflow. The 'Exit' button is ever-present, providing a clear and consistent way to safely exit the application at any time.

This BooksAuthors tab exemplifies the system’s commitment to providing specialized tools for managing complex data relationships, delivering a seamless and powerful experience for inventory controllers.

### Module 4: Order Clerks

Interfaz de usuario gráfica

Descripción generada automáticamente

The FormOrderClerks class is the interface tailored to the Order Clerks role, concentrating on managing customer orders effectively and intuitively.

**Orders Panel:**

Input Fields: Incorporates fields for Order ID (auto-generated), Customer ID, Employee ID, with dropdowns for selection, alongside input for Order Date (with date format guidance), Order Type, and Status ID, ensuring precise data capture.

Action Buttons: 'Add', 'Update', and 'Cancel' buttons to govern the lifecycle of an order, invoking appropriate logic post-input verification.

Search and Listing Features: A powerful search utility aids in pinpointing orders by various criteria, while 'List All Orders' compiles and displays a full listing of orders in a ListView, streamlining order management tasks.

**GUI Design and Interaction:**

Feedback and Interaction: Consistent with the system's approach, the form uses dialogue messages for real-time feedback and incorporates validation checks for data reliability.

Data Presentation: The ListView becomes a central component, structuring order information for easy interpretation and action, reinforcing the clerks' efficiency.

Personalized User Experience: True to form, the interface greets the Order Clerk by name and offers profile access for personal settings through 'View Profile'.

Unified Design Approach: Mirroring the system-wide design language, the interface maintains a consistent aesthetic and operational pattern, ensuring a frictionless user journey.

**Extended Functionality in Additional Tabs:**

Order Details, Customers, and Other Related Tabs: Each of these tabs is equipped with specific functionalities pertinent to order management, from fine-grained details to broader customer interactions, all furnished with their input fields and list views for comprehensive data administration.

**Navigation and System Control:**

Systematic Workflow and Exit Path: The 'Exit' button provides an immediate route to close the application safely, emphasizing the thoughtful navigation structure from beginning to end of the user's session.

Interfaz de usuario gráfica

Descripción generada automáticamente

**Order Details Tab**

Input Fields:

Order ID: A dropdown that lists existing orders, enabling users to select the order to which the item belongs.

Item Sequential: A field guiding users to input the next sequential number for the current order, ensuring organized tracking of order items.

Book ID: A dropdown menu populated with book identifiers, allowing for the selection of a specific book for the order item.

Quantity: An input field for specifying the number of books ordered, crucial for inventory and order size management.

Unit Price and Total Price: Fields displaying the cost per unit and the calculated total price for the quantity ordered, crucial for financial oversight.

Action Buttons: 'Add', 'Update', and 'Delete Item' buttons: These controls manage the lifecycle of an order's items, from addition to deletion, with validation to ensure data accuracy before any operation is performed.

Search and Listing Features: A dropdown to select the search criterion and an adjacent search field enable users to search for specific order details. Results are displayed in the ListView below.

'List All Orders Details' button: Fills the ListView with all order items, offering a complete overview and allowing for easy access to specific details.

**GUI Design and Interaction:**

User Feedback: Consistent interaction is maintained through dialogue messages that inform the user of successful operations or provide error warnings when data does not meet validation criteria.

Data Presentation: A ListView displays order details, including the sequential item number, book ID, quantity, unit price, and total price, facilitating quick review and modification actions.

Welcome and Profile Access: As with other tabs, the user is greeted by name, fostering a personal connection. The 'View Profile' button is ever-present for quick profile edits or reviews.

Design Consistency: The design is consistent with the application's overall aesthetic and functional standards, ensuring a seamless user experience across different modules.

**Navigation and System Control:**

Adherence to logical workflow is demonstrated through the layout and functionality, with the 'Exit' button clearly positioned to terminate the application session securely.

Interfaz de usuario gráfica

Descripción generada automáticamente

**Customer Tab**

Search Functionality: A dropdown to choose the search parameter, either by Customer ID or Name, paired with a search field to enter the corresponding data.

'Search' button: Triggers the search operation, with results displayed in the ListView.

'List All Customers' button: Presents every customer record in the ListView, facilitating quick navigation through customer data.

ListView Display: Details such as Customer ID, Name, Street, City, Postal Code, Phone Number, Fax Number, and Credit Limit are listed in a tabulated format, offering a clear and detailed overview of customer information.

**GUI Design and Interaction:**

User Experience: Consistency in the dialogue messages reinforces a responsive interface, guiding users through successful processes or alerting them to issues that may require attention.

The ListView serves as a central element of the UI, structured to allow users to easily read and understand customer data.

Welcoming and Personalized Access: Users are greeted by their name, creating a welcoming atmosphere, and are provided with quick access to their profile through the 'View Profile' button for any necessary modifications.

Design Consistency: The design and layout of the Customers Tab align with the rest of the application, ensuring users experience a coherent and familiar environment across different modules.

**Navigation and Control:**

The logical flow of user interaction is well-designed, with clear entry points for searches and a straightforward exit route through the 'Exit' button, adhering to the application's overall navigational structure.

## Design of Data Access Classes

Texto

Descripción generada automáticamente con confianza media

### Login

Purpose: Handles the database operations specifically for user account management.

Interaction with the database: Uses ADO.NET connected mode to execute direct SQL queries for authentication and user management.

### User Profile

**EmployeesDB Class:**

Purpose: Interacts with the database to retrieve and update employee information.

Methods:

UpdateRecord(Employee emp): Updates the employee's record in the database.

**UserAccountsDB Class:**

Purpose: Updates user account details like passwords in the database.

Methods:

UpdatePasswordByUser(UserAccount user): Updates the password for the user account in the database.

### Module 1: MIS Manager

The DAL classes for the MIS Manager module primarily focus on handling database operations related to user accounts and employees within the HiTech Distribution application. They provide an abstraction layer for the Business Logic Layer (BLL), enabling secure and efficient interaction with the database.

***UtilityDB Class:***Provides a ConnectDB method that establishes a connection to the database using the connection string configured in the application settings, encapsulating the database connectivity logic.

***UserAccountsDB Class:*** Contains static methods to perform CRUD operations on the UserAccounts table:

SaveRecord(UserAccount user): Inserts a new user record into the database.

UpdateRecord(UserAccount user): Updates an existing user record.

DeleteRecord(UserAccount user): Deletes a user record from the database.

GetAllRecords(): Retrieves a list of all user records.

SearchRecord(int userId): Searches for a user by their ID.

LoginUser(int userId, string password): Authenticates a user.

GetNameUser(int userId): Gets the name of a user based on their ID.

***EmployeesDB Class:***Facilitates CRUD operations on the Employees table:

SaveRecord(Employee emp): Adds a new employee record.

UpdateRecord(Employee emp): Modifies an existing employee record.

DeleteRecord(Employee emp): Removes an employee record.

GetAllRecords(): Fetches all employee records, joining with the Jobs and Status tables to include job titles and status descriptions.

SearchRecord(string input1, string input2): Performs a search based on first and last names.

SearchRecord(int inputEmployeeId): Retrieves a single employee's details by their ID.

GetJobTitleEmployee(int inputEmployeeId): Obtains the job title of an employee.

GetAllJobs(): Gathers all available job titles from the Jobs table.

GetAllStatus(): Collects all statuses from the Status table.

***StateDB Class:*** The StateDB class contains methods to interact with the Status table in the database.

GetAllStatus(): Retrieves a list of all status records, converting them into State objects. This method is useful for displaying status options throughout the application.

GetAllStatusUser(): Specifically fetches statuses relevant to users, such as active or inactive. This method could be used to filter user account records based on their status.

***JobsDB Class:*** The JobsDB class manages interactions with the Jobs table.

GetAllJobs(): Obtains all job records from the database, which are then converted into Jobs objects. It is crucial for populating job-related information, possibly for employee records or for use within HR-related modules.

These DAL classes ensure data consistency and integrity by managing the database operations and providing a secure interface for the BLL to interact with the database without exposing the underlying SQL commands and structure.

### Module 2: Sales Manager

The DAL classes for Module 2 manage the database operations related to customer data in the HiTech Distribution application.

***UtilityDB Class:*** Identical to the UtilityDB class in Module 1, it offers a ConnectDB method to establish and open a database connection.

***CustomerDB Class:*** Handles operations for the Customers table with the following method:

GetAllCustomers(): Retrieves a list of all customers from the database, converting the data into Customer objects and adding them to a list. This method provides a complete view of the customer base, including details such as customer names, contact information, and credit limits.

This DAL class ensures seamless access to customer data, allowing other application modules to perform actions like displaying customer lists, searching for customers, and integrating customer information into order processing and inventory management. It abstracts the underlying SQL logic, allowing the Business Logic Layer (BLL) to work with customer data without dealing directly with the database layer.

### Module 3: Inventory Controller

For Module 3 of the HiTech Distribution Application, the DAL classes are focused on managing the database operations for publishers, books, book categories, book authors, and authors. Here's a brief overview of each class and its methods:

***PublishersDB Class:*** Manages CRUD operations for publishers in the database.

SaveRecord(): Inserts a new publisher record.

UpdateRecord(): Updates an existing publisher record.

DeleteRecord(): Deletes a publisher record.

GetAllRecords(): Retrieves all publisher records.

SearchRecord(): Finds a specific publisher by ID.

***BooksDB Class:*** Handles CRUD operations for books.

SaveRecord(): Inserts a new book and returns the autogenerated book ID.

UpdateRecord(): Updates an existing book record.

DeleteRecord(): Deletes a book record.

GetAllRecords(): Fetches all book records with joined publisher and category information.

SearchRecord(): Overloaded method that can search for books by different criteria.

SearchISBN(): Checks if a book with a given ISBN already exists.

***BooksCategoriesDB Class:*** Manages CRUD for book categories.

GetAllRecords(): Retrieves all book category records.

SaveRecord(): Inserts a new book category.

UpdateRecord(): Updates an existing book category.

DeleteRecord(): Deletes a book category.

SearchRecord(): Finds a specific book category by ID.

GetDescriptionCategory(): Retrieves the description of a category by its ID.

***BooksAuthorsDB Class:*** Deals with the relationship between books and authors.

GetAllBooksAuthors(): Fetches all book-author pairings with book titles and author names.

SaveRecord(): Creates a new association between a book and an author.

DeleteRecord(): Removes an association between a book and an author.

SearchRecords(): Searches for book-author pairings based on book or author ID.

SearchRecordByTitle(): Searches for book-author pairings based on book title.

SearchRecordByAuthor(): Searches for book-author pairings based on author name.

***AuthorsDB Class:*** Manages CRUD for authors.

SaveRecord(): Inserts a new author record.

UpdateRecord(): Updates an existing author record.

DeleteRecord(): Deletes an author record.

GetAllRecords(): Retrieves all author records.

SearchRecord(): Overloaded method that can find authors by ID or by name.

***UtilityDB Class:*** Provides a common method to establish a database connection.

Together, these classes form a comprehensive data access layer for managing the book-related data of the HiTech Distribution Application, encompassing all necessary operations to support the application's business logic and workflows. Each class is designed to handle specific database interactions, ensuring separation of concerns and making the code more maintainable and scalable.

### Module 4: Order Clerks

For Module 4, the Data Access Layer (DAL) of the HiTech Distribution Application leverages Entity Framework, which provides a more abstract and higher-level API for database interactions compared to traditional SQL queries. This module contains repository classes that perform CRUD operations and business logic directly related to the entity models.

***StatusRepository:*** Wraps the operations that can be performed on the Status entity.

GetStatuses(): Retrieves all statuses.

GetStatusById(): Finds a specific status by its ID.

***OrdersDetailRepository:*** Manages the details of orders in the system.

GetOrdersDetails(): Fetches all order details.

AddOrdersDetail(): Adds a new order detail record.

UpdateOrdersDetail(): Updates an existing order detail.

DeleteOrdersDetail(): Removes an order detail from the system.

SearchOrderDetailByItemSequencial(): Finds order detail by its sequential item number.

SearchOrderDetailByOrderIdAndItemSequencial(): Searches for an order detail by order ID and item sequential.

SearchOrderDetailById(): Retrieves multiple order details based on order ID.

SearchOrderDetailByBookID(): Finds order details for a particular book.

GetQuantityOpensByBookId(): Gets the quantity of open orders for a specific book.

***OrderRepository:*** Handles operations for the Order entity.

GetOrders(): Obtains a list of all orders.

AddOrder(): Adds a new order to the database.

UpdateOrder(): Updates the details of an existing order.

CancelOrder(): Cancels an order and updates its status accordingly.

GetOrderById(): Retrieves a single order by its ID.

GetOrderByCustomerId(), GetOrderByEmployeeId(): Fetches orders filtered by customer or employee ID.

GetOrdersByState(): Gets orders based on their status state.

IsBookWithStatusOpenExists(): Checks if there is an open order for a specific book.

***EmployeeRepository:*** Manages operations for the Employee entity.

GetEmployees(): Retrieves all employees from the database.

***CustomerRepository:*** Manages operations for the Customer entity.

GetCustomers(): Fetches all customers.

SearchCustomerById(): Finds a customer by ID.

SearchCustomerByName(): Searches for a customer by name.

***BookRepository:*** Contains methods for managing Book entities.

GetBooks(): Retrieves all books from the database.

UpdateBook(): Updates the quantity available for a book.

Each repository uses the HiTechDistributionDBContext to interact with the database via Entity Framework's DbContext, which manages entity objects during run time, including support for change tracking, and automatically mapping entities to database tables. These repositories abstract away the underlying database queries, allowing for cleaner, more maintainable code.

## Design of Validations

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

Descripción generada automáticamente

### Login

UserValidator : Ensures the input for user credentials is valid before attempting to interact with the BLL or database.

Methods:

IsValidId(string id): Checks if the user ID is a valid numeric string.

IsValidPassword(string password): Verifies that the password meets the specified complexity requirements.

### User Profile

UserValidator: Ensures that the updated password and other user inputs meet the system's requirements before updating in the database.

Methods:

IsValidPassword(string password): Ensures the new password adheres to the defined security standards.

### Module 1: MIS Manager

The validations ensure that any operation involving these entities, such as creating or updating records, is performed with proper and valid data.

***User Account Validations:***

IsValidId: Verifies that the user ID consists solely of digits, indicating a valid numerical identifier.

IsValidPassword: Checks the password to ensure it meets the complexity requirements, such as a minimum length of 8 characters, and includes at least one uppercase letter, one lowercase letter, one digit, and one special character. It also ensures the password does not exceed the maximum length specified, which is 50 characters.

IsValidDate: Although not directly linked to UserAccount, this validation could be used when handling date fields related to the user, such as the creation date or last modification date, ensuring they follow the yyyy-mm-dd format.

***Employee Validations:***

IsValidId: Ensures that the employee ID is a numeric value, similar to the user ID validation.

IsValidName: Validates the names (first and last) of the employee to confirm they are within the appropriate length (not exceeding 50 characters) and contain only alphabet characters or whitespace.

IsValidEmail: Checks if the email address provided adheres to a standard email format, which is essential for any communication or identification processes within the system.

These validations in the UserValidator and EmployeeValidator classes play a crucial role in the application's module that handles the management information system (MIS). They serve to verify that the information entered for user accounts and employee records is correct before the application proceeds with database transactions, thus preserving data integrity and consistency.

### Module 2: Sales Manager

For Module 2, which pertains to the Sales Manager role in the HiTech Distribution Application, the focus is on managing customer-related data. Here, the validations from the CustomerValidation class are crucial. The main aspects to validate are the customer ID, name, phone number, postal code, credit limit, city, and street. Let's delve into the specifics of each:

***Customer Validations:***

IsValidId: Confirms that the customer ID is a numerical value.

IsValidName: Checks that the customer's name is within the appropriate length (not exceeding 50 characters) and consists only of alphabetic characters or whitespace. This applies to both individual and company names.

IsValidNumberFormat: Validates the phone number format, ensuring it complies with a typical North American style, such as (123)456-7890.

IsValidPostalCode: Ensures that the postal code matches the Canadian postal code format, which is A1A 1A1, where A represents a letter and 1 is a digit.

IsValidCreditLimit: Verifies that the credit limit is a numerical value, which can be a decimal but must adhere to a suitable format for monetary values.

IsValidCity: Checks the city name for the appropriate length and character content, similar to the customer name validation.

IsValidStreet: Ensures the street address is within 100 characters and does not contain invalid characters.

In the Sales Manager module, these validations would typically be triggered during the following scenarios:

Adding a new customer: All fields would need to pass their respective validations before the customer record is created.

Updating customer information: When modifying any of the customer details, the updated information must be validated to maintain data integrity.

Processing orders: While not directly involving customer creation, ensuring that customer-related data like ID and credit limit are valid is crucial when linking orders to customers.

Integrating these validations into the Sales Manager module helps prevent errors in customer data management and ensures that sales operations are conducted based on accurate and valid customer information. This is essential not only for maintaining a reliable customer database but also for ensuring smooth sales transactions and compliance with financial and legal standards.

### Module 3: Inventory Controller

For Module 3, the Inventory Controller's role holds the management of book inventory, which includes books, publishers, categories, and the association of books with authors. Here's how the validations from the respective classes apply:

***Book Validations:***

IsValidId: Verifies that book ID, publisher ID, and category ID are numerical.

IsValidDescription: Ensures the category description doesn't exceed 50 characters and contains only letters or whitespace.

IsValidPublisherName: Checks the publisher's name for the correct length (up to 100 characters) and valid characters (letters and whitespace).

IsValidTitle: Confirms the book title is within the allowable character limit (100 characters).

IsValidYear: Checks that the publication year is reasonable (not before 1950 and not in the future).

IsValidQuantity: Verifies the book's available quantity is a non-negative number and not less than the quantity of books already committed to open orders.

ExistsOrdersOpen: Checks if there are open orders for a book before allowing it to be deleted or the quantity to be reduced.

***Author Validations:***

IsValidName: Ensures the author's first and last names are within the correct length and contain only valid characters.

IsValidEmail: Checks the author's email for the correct format.

In the Inventory Controller module, these validations are triggered in the following instances:

Adding new books or categories: Validation ensures that all information entered meets the system's data requirements.

Updating existing book records: Any changes in the book details, such as price, title, or quantity, must be validated.

Associating books with authors: Ensures that the authors' names and details conform to the required format.

Managing publishers and categories: Validates the input for publishers and categories to maintain data integrity.

The Inventory Controller's functionality is heavily dependent on these validations to avoid discrepancies in inventory levels, which are critical for order fulfillment and stock management. Ensuring that the inventory data is valid and consistent helps the Inventory Controller in making informed decisions regarding stock ordering, sales analysis, and maintaining a diverse yet manageable catalog of books.

### Module 4: Order Clerks

For Module 4, focused on the Order Clerks, the primary interactions involve handling orders and their details. The validations ensure the integrity and accuracy of the order processing and management functions.

***Order Validations:***

IsValidId: Checks if the order ID, customer ID, and employee ID are numeric, ensuring that references to these entities are valid.

isValidDate: Ensures that dates, such as order dates, adhere to the correct format (YYYY-MM-DD), which is crucial for consistency in record-keeping and reporting.

isValidOrderType: Validates the order type (Email, Phone, Fax) to ensure that it matches one of the predefined categories, which helps in categorizing and processing orders based on their source.

isValidStatus: Confirms that the order status (In Process, Completed, Pending, Cancel) is recognized by the system, which is vital for tracking the progress and current state of orders.

isValidQuantity: Ensures that the quantity specified in order details is a positive number, critical for inventory management and avoiding negative inventory scenarios.

isValidPrice: Verifies that prices entered (unit price, total price) are valid monetary values, which is essential for accurate financial transactions and calculations.

isValidItemSequencial: Checks the validity of the item sequence in order details to ensure uniqueness within the same order, which helps prevent duplication and errors in order item listings.

IsCloseOrder: Determines if an order is closed (either completed or canceled) to prevent modifications to such orders, which maintains data integrity and complies with business rules regarding closed financial transactions.

Adding and updating orders: Ensures all data entered meets business rules before submission, thus preventing errors that could affect inventory and customer service.

Managing order details: Validates entries related to specific items within an order, such as quantities and pricing, which are critical for accurate invoicing and stock management.

Searching and listing orders: Employs validations to ensure search operations return correct and expected results, enhancing the usability and functionality of the order management system.

These validations help maintain a reliable and efficient order management process, facilitating tasks ranging from order entry to modification and finalization, all while ensuring compliance with organizational rules and standards.

# Project Implementation

For the Project Implementation of the HiTech Distribution App, each module has been carefully designed to meet specific needs within the system and the project given by the Professor, taking advantage of the most suitable technologies and architectures—connected mode, disconnected mode, and entity framework. This structured approach ensures optimal performance and user experience across different user roles within the system.

## Login

Authentication Process: The FormLogin GUI handles user input, where the entered user ID and password are validated using the UserValidator class. If the credentials are validated, the UserAccount BLL class method LoginUserAccount is called to authenticate the user against the database.

Secure User Login: The system enforces security through password complexity validation before authentication to minimize unauthorized access.

Error Handling: Appropriate error messages are displayed if the login attempt fails, ensuring the user is aware of invalid credentials or other login issues.

User Experience: The login form is designed for a straightforward experience, with clear fields for user ID and password and conspicuous buttons for login and exit actions.

Direct Database Interaction: Using ADO.NET in connected mode, UserAccountsDB establishes a continuous connection for the login check, which provides immediate feedback to the user.

## User Profile

Profile Information Display: The FormUserProfile GUI fetches and displays the current employee's information using the Employee BLL class, specifically through the SearchEmployee method.

Password Update: The form provides fields for inputting the current and new password. It uses UserValidator to confirm the new password's validity before calling UserAccountsDB to make the change in the database.

Confirmation Feedback: After a successful update, the user is informed through a confirmation message. If there's an issue, such as a password mismatch, an error message is displayed.

Secure Password Management: All password updates are checked for complexity and correctness to maintain security within the system.

Connected Database Operations: Similar to the login process, updating user profile information is performed using ADO.NET in connected mode, enabling immediate execution of updates on the database.

## Module 1: MIS Manager (Connected Mode)

### Purpose and Functionality:

Manages user accounts and employee records, providing functionalities for creating, updating, deleting, and searching for user and employee information.

### Key Implementation Details:

Persistent Database Connection: Utilizes ADO.NET in a connected mode for direct and continuous interaction with the database for immediate CRUD operations.

Enhanced Security Measures: Implements advanced validation for user IDs and passwords to ensure security standards are met, using regex patterns and conditional checks.

State and Job Management: Uses the State and Jobs classes to manage and validate statuses and job titles, ensuring that these elements are consistent and accurate across the system.

Comprehensive Validation: Incorporates detailed validation logic within the UserValidator and EmployeeValidator to enforce business rules and data integrity before database operations are performed.

## Module 2: Sales Manager (Disconnected Mode)

### Purpose and Functionality:

Focuses on managing customer data, providing tools for handling customer records efficiently without constant database connectivity.

### Key Implementation Details:

Efficient Data Handling: Uses DataSets and DataAdapters to manipulate customer data in a disconnected mode, allowing batch processing and offline data management.

Validation Integration: Leverages CustomerValidation to ensure all customer data adheres to predefined formats and constraints before synchronization with the database.

Optimized Data Synchronization: Implements efficient data reconciliation processes to update the database with changes made offline, ensuring data consistency and integrity.

## Module 3: Inventory Controller (Connected Mode)

### Purpose and Functionality:

Manages inventory related to books, publishers, categories, and authors, with extensive functionalities for linking books to authors and managing publisher information.

### Key Implementation Details:

Direct Database Transactions: Similar to Module 1, it maintains a continuous database connection for real-time updates and retrievals.

Complex Relationship Management: Handles many-to-many relationships between books and authors using the BooksAuthors class, which is crucial for maintaining accurate bibliographic data.

Dynamic Data Validation: Utilizes BooksValidation to ensure that all entries, such as ISBNs and book prices, meet specific criteria before they are committed to the database.

Robust Search Capabilities: Implements advanced search functionalities that allow users to find books by various attributes, enhancing usability and efficiency.

## Module 4: Order Clerks (Entity Framework)

### Purpose and Functionality:

Manages orders and order details using Entity Framework, facilitating complex data interactions and state management with ease.

### Key Implementation Details:

ORM Utilization: Employs Entity Framework for object-relational mapping, simplifying data manipulation and reducing the need for explicit SQL coding.

Automated Change Tracking: Entity Framework's change tracking mechanism automatically calculates the necessary SQL updates based on the changes made to the data model.

Comprehensive Order Management: Provides methods for adding, updating, and cancelling orders, with integrated status updates through the StatusRepository.

Advanced Query Capabilities: Leverages LINQ for expressive and readable data queries, enhancing maintainability and development speed.

### User Interface Integration:

Each module is tightly integrated with a user interface that provides specific functionalities tailored to the roles of MIS Managers, Sales Managers, Inventory Controllers, and Order Clerks. This UI is designed to be intuitive and user-friendly, offering comprehensive data views and easy navigation, which are essential for efficient daily operations.

This structured approach not only addresses the functional needs of each department within the HiTech Distribution system but also ensures that data integrity, user experience, and system performance are maintained at optimal levels across different interactions and processes.

# Project Testing

## Login

### Input test

* Henry - MIS Manager

User ID: 9 – Password: Pass123!

* Thomas - Sales Manager

User ID: 10 - Password: Pass123!

* Peter - Inventory Controller

User ID: 11 - Password: Pass123!

* Mary - Order Clerk

User ID: 12 - Password: Pass123!

* Jennifer - Order Clerk

User ID: 13 - Password: Pass123!

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

## User Profile

### Input test

## Module 1: MIS Manager

### Input test

First Name: Hamilton

Last Name: Gomez

Email: gomez@email.com

Job: Order Clerk

Status: Full-Time

### Add Employee Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Update Employee Button

Email: [hgomez@email.com](mailto:hgomez@email.com)

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Delete Employee Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Search by Employee Button and List All Employee

Search by EmployeeId 14

Interfaz de usuario gráfica

Descripción generada automáticamente

Search by full name Jennifer Bouchard

Interfaz de usuario gráfica

Descripción generada automáticamente

### Input test

UserID: 18

Password: Pass123!

Status: Active

### Add UserAccount Button

## Interfaz de usuario gráfica, Sitio web Descripción generada automáticamente

### Update UserAccount Button

Status: InActive

Interfaz de usuario gráfica

Descripción generada automáticamente

### Delete UserAccount Button

User ID: 18

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Search UserAccount Button

Interfaz de usuario gráfica

Descripción generada automáticamente

## Module 2: Sales Manager

### Input test

Customer name: McGill University

Street: 845 Sherbrooke St W

City: Montreal

Postal Code: H3A 0G4

Phone Number: (123)589-2569

Fax Number: (123)589-2569

Phone Number: (123)589-2569

Credit Limit: 50000

### Add Button

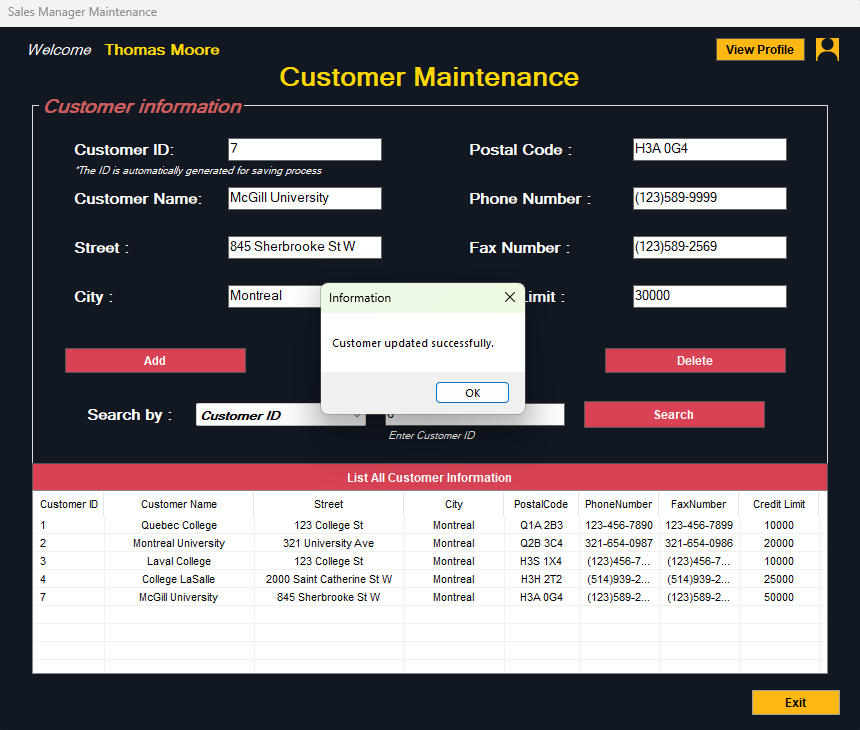
Una captura de pantalla de una computadora

Descripción generada automáticamente con confianza media

### Update Button

Phone Number: (123)589-9999

Credit Limit: 30000



### Delete Button

Captura de pantalla de computadora

Descripción generada automáticamente

### Search Button

Una captura de pantalla de una computadora

Descripción generada automáticamente con confianza media

## Module 3: Inventory Controller

### Input test

Title: The Pragmatic Programmer

ISBN: 369852147159

Unit Price: 39.99

Publication Year: 2006

Publisher ID: Premier Press

Quantity Available: 68

Category ID: Data Science

Author: Steve McConnell

### Add Book Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Update Book Button

Quantity: 90

Author Erich Gamma

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Search by ID and List All Buttons

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Input test

First Name: Martin

Last Name: Fowler

Email: mFowler@email.com

### Add Author Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Update Author Button

FirstName: Martin L

Captura de pantalla de un celular

Descripción generada automáticamente

Delete Author Button and Search By

Captura de pantalla de un celular

Descripción generada automáticamente

### List All Authors Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Input test

Description : web Development

### Add Book Category Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Update Book Category Button

Description : web Develop

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Delete Book Category Button

Interfaz de usuario gráfica

Descripción generada automáticamente

### List All and Search Book Category Button

Interfaz de usuario gráfica

Descripción generada automáticamente

### Input test

Publisher Name : New Code

### Add Publisher Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Update Publisher Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Cancel Publisher Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### List All and Search Publisher Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### List All and Search Author X Book

Interfaz de usuario gráfica

Descripción generada automáticamente

## Module 4: Order Clerks

### Input test

Customer: McGuill University

Employee: Jennifer Bouchard

Order Date: 2024-04-18

Order Type: Phone

Status: Pending

### Add Order Button

Interfaz de usuario gráfica

Descripción generada automáticamente

### Update Order Button

Order Type: Email

Interfaz de usuario gráfica

Descripción generada automáticamente

### Cancel Order Button

Interfaz de usuario gráfica

Descripción generada automáticamente Interfaz de usuario gráfica

Descripción generada automáticamente

### List All Order Button

Interfaz de usuario gráfica

Descripción generada automáticamente

### Input test

Order Id : 6

Item Sequential: 1

Book Id: The pragmatic Programming

Quantity: 5

### Add Publisher Button

Interfaz de usuario gráfica

Descripción generada automáticamente

If the order is cancelled

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### Update Publisher Button

Interfaz de usuario gráfica

Descripción generada automáticamente

### Cancel Publisher Button

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

### List All Publisher Button

Interfaz de usuario gráfica

Descripción generada automáticamente

### List All Customer Button

### Interfaz de usuario gráfica Descripción generada automáticamente

### Search by Customer Button

Interfaz de usuario gráfica

Descripción generada automáticamente

### User Profile

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

*Conclusion and Learnings*

The HiTech Distribution App is designed to manage different aspects of a distribution company, broken down into four main modules. Each module caters to different roles within the company and utilizes specific technologies to handle data effectively:

*MIS Manager and Inventory Controller*: These modules work with ADO.NET in a connected mode, which means they keep a constant connection to the database for real-time data updates—important for tasks that need immediate data reflection like managing user accounts and inventory.

*Sales Manager:* This module uses ADO.NET in a disconnected mode, allowing it to handle data offline and then sync changes in bulk. This approach is useful for managing large datasets efficiently without needing constant database connectivity.

*Order Clerks:* This module uses Entity Framework, an advanced technology that simplifies data operations and automatically manages relationships between data records, making it easier to work with complex data.

### What I Learned

Using Different Data Handling Techniques: I’ve learned when to use connected vs. disconnected database access. Connected access is great for real-time interactions with the database, while disconnected access is useful for situations where you can work offline and sync later.

Entity Framework Benefits: Using Entity Framework helps in reducing the amount of manual code needed to interact with the database, making the development process faster and less error-prone.

Layered Architecture Importance: Structuring a project into layers (Business Logic, Data Access, Validation, GUI) helps keep it organized, making it easier to manage and update. Each layer has its specific role, from handling business rules to managing data interactions and user interface.

### How This Applies to Real Projects

Adaptability: Knowing how to choose and apply the right technology based on the project's needs is crucial. This flexibility helps in building systems that are efficient and scalable.

Project Structure: Maintaining a clean and organized project structure makes the system easier to understand and maintain, which is crucial for long-term project success.

Critical Thinking: Deciding on the best architecture and technologies requires careful thought about the project’s specific needs and challenges, enhancing problem-solving skills.

### Conclusion

Building the HiTech Distribution App was an excellent practice in practicing development techniques and architectures. It highlighted the importance of selecting appropriate technologies based on specific needs and provided a deep understanding of managing data effectively. These learnings are invaluable for any software development project, ensuring systems are not only robust and efficient but also scalable and easy to maintain. This experience has greatly enhanced my ability to design and implement complex systems in real-world scenarios.