Hi-Tech Order Management System

Documentation

Author: Paula Miano

# Project Description

Here is the description of the project.

# Project Design

## Database Design

### List of tables

1. Status Table:
   * StatusID (PK)
   * State
   * StateDescription
2. Categories Table:
   * CategoryID (PK)
   * CategoryName
3. Publishers Table:
   * PublisherID (PK)
   * Name
   * StatusID (FK – Status)
4. Authors Table:
   * AuthorID (PK)
   * FirstName
   * LastName
   * Email
   * StatusID (FK – Status)
5. Books Table:
   * ISBN (PK)
   * Title
   * Price
   * YearPublished
   * QOH
   * CategoryID (FK – Categories)
   * PublisherID (FK - Publisher)
   * StatusID (FK – Status)
6. BooksAuthors Junction Table:
   * + (BookID, AuthorID) (composite PK)
   * BookID (FK – Books)
   * AuthorID (FK – Authors)
7. Positions Table:
   * PositionID (PK)
   * PositionTitle
   * PositionDescription
8. Employees Table:
   * EmployeeID (PK)
   * FirstName
   * LastName
   * Email
   * PhoneNumber
   * PositionID (FK – Positions)
   * Department
   * StatusID (FK – Status)
9. UserAccount Table:
   * UserID (PK)
   * EmployeeID (FK – Employees)
   * Username
   * Password (hashed)
   * Role
   * StatusID (FK – Status)
   * DateCreated
   * DateModified
   * StatusID (FK – Status)
10. EmployeesUserAccounts Junction Table:
    * (EmployeeID, UserID) (composite PK)
    * EmployeeID (FK - Employees)
    * UserID (FK – Users)
11. Customers Table:
    * CustomerID (PK)
    * Name
    * Street
    * StreetNumber
    * City
    * PostalCode
    * PhoneNumber
    * FaxNumber
    * CreditLimit
    * StatusID (FK – Status)
12. Orders Table:
    * OderID (PK)
    * CustomerID (FK – Customers)
    * OrderDate
    * TotalAmount
    * StatusID (FK – Status)
    * DateCreated
    * DateModified
13. OrderDetails Table:
    * OrderDetailID (PK)
    * OrderID (FK - Orders)
    * BookID (FK – Books)
    * Quantity
    * Price

### Relationships between tables

1. Status to UserAccount, Customers, Orders, Employees: One-to-Many

A single status can apply to multiple records in the UserAccount, Customers, Orders, and Employees tables. For instance, a status like "Active" could apply to many users, customers, orders, and employees.

Relationship: 1 (Status) : 0...\* (UserAccount, Customers, Orders, Employees)

1. Categories to Books: One-to-Many

Each category can include multiple books, but each book belongs to exactly one category.

Relationship: 1 (Categories) : 0...\* (Books)

1. Publishers to Books: One-to-Many

Each publisher can publish multiple books, but each book is published by exactly one publisher.

Relationship: 1 (Publishers) : 0...\* (Books)

1. Books to Authors (via BooksAuthors): Many-to-Many

Each book can have multiple authors, and each author can write multiple books. This is represented through the BooksAuthors junction table.

Relationship: \* (Books) : \* (Authors) via (BooksAuthors)

1. Positions to Employees: One-to-Many

Each position can be held by multiple employees, but each employee holds exactly one position at a time.

Relationship: 1 (Positions) : 0...\* (Employees)

1. Employees to UserAccount: One-to-Many

Each employee can have multiple user accounts, but each user account is associated with exactly one employee.

Relationship: 1 (Employees) : 0...\* (UserAccount)

1. Customers to Orders: One-to-Many

Each customer can place multiple orders, but each order is associated with exactly one customer.

Relationship: 1 (Customers) : 0...\* (Orders)

1. Orders to OrderDetails: One-to-Many

Each order can contain multiple order details (representing different books ordered), but each order detail line is associated with exactly one order.

Relationship: 1 (Orders) : 0...\* (OrderDetails)

1. Books to OrderDetails: One-to-Many

A single book can appear in multiple order details (across different orders), but each order detail line references exactly one book.

Relationship: 1 (Books) : 0...\* (OrderDetails)

### Tables Fields

1. Status Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| StatusID | INT | Primary key, auto-increments | 1 |
| State | NVARCHAR(50) | Describes the status name | 'Active' |
| StateDescription | NVARCHAR(255) | Detailed description of the state | 'The record is active and operational.' |

1. Categories Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| CategoryID | INT | Primary key, auto-increments | 1 |
| CategoryName | NVARCHAR(100) | Name of the book category | 'Science Fiction' |

1. Publishers Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| PublisherID | INT | Primary key, auto-increments | 1 |
| Name | NVARCHAR(255) | Name of the publisher | 'Penguin Random House' |
| StatusID | INT | Foreign key to Status table | 1 |

1. Authors Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| AuthorID | INT | Primary key, auto-increments | 1 |
| FirstName | NVARCHAR(100) | First name of the author | 'Jane' |
| LastName | NVARCHAR(100) | Last name of the author | 'Doe' |
| Email | NVARCHAR(255) | Email address of the author | 'jane.doe@example.com' |
| StatusID | INT | Foreign key to Status table | 1 |

1. Books Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| ISBN | VARCHAR(13) | Primary key, ISBN number of the book | '978-3-16-148410-0' |
| Title | NVARCHAR(255) | Title of the book | 'Introduction to Database Systems' |
| Price | DECIMAL(10,2) | Price of the book | 49.99 |
| YearPublished | INT | Year the book was published | 2020 |
| QOH | INT | Quantity on hand | 50 |
| CategoryID | INT | Foreign key to Categories table | 1 |
| PublisherID | INT | Foreign key to Publishers table | 1 |
| StatusID | INT | Foreign key to Status table | 1 |

1. Positions Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| PositionID | INT | Primary key, auto-increments | 1 |
| PositionTitle | NVARCHAR(100) | Title of the position | 'Manager' |
| PositionDescription | NVARCHAR(255) | Description of the position's duties | 'Oversees department operations' |

1. Employees Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| EmployeeID | INT | Primary key, auto-increments | 1 |
| FirstName | NVARCHAR(100) | First name of the employee | 'John' |
| LastName | NVARCHAR(100) | Last name of the employee | 'Doe' |
| Email | NVARCHAR(255) | Email address of the employee | 'john.doe@example.com' |
| PhoneNumber | NVARCHAR(15) | Contact phone number of the employee | '555-1234' |
| PositionID | INT | Foreign key to Positions table | 1 |
| Department | NVARCHAR(100) | Department where the employee works | 'IT' |
| StatusID | INT | Foreign key to Status table | 1 |

1. UserAccount Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| UserID | INT | Primary key, auto-increments | 1 |
| EmployeeID | INT | Foreign key to Employees table | 1 |
| Username | NVARCHAR(50) | Unique username for user login | 'johndoe123' |
| Password | NVARCHAR(255) | User's password (plain text for simplicity) | 'password123' |
| Role | NVARCHAR(50) | Role of the user in the system | 'Administrator' |
| DateCreated | DATETIME | Date and time the account was created | '2023-01-01 12:00:00' |
| DateModified | DATETIME | Date and time the account was last modified | '2023-01-02 12:00:00' |
| StatusID | INT | Foreign key to Status table | 1 |

1. Customers Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| CustomerID | INT | Primary key, auto-increments | 1 |
| Name | NVARCHAR(255) | Full name or company name of the customer | 'Jane Doe' |
| Street | NVARCHAR(255) | Street address of the customer | '123 Main St' |
| StreetNumber | NVARCHAR(10) | Street number, if applicable | '123' |
| City | NVARCHAR(100) | City where the customer resides | 'Anytown' |
| PostalCode | NVARCHAR(10) | Postal code of the customer's address | '12345' |
| PhoneNumber | NVARCHAR(15) | Contact phone number of the customer | '555-6789' |
| FaxNumber | NVARCHAR(15) | Fax number of the customer, if applicable | '555-6788' |
| CreditLimit | DECIMAL(10, 2) | Credit limit extended to the customer | 5000.00 |
| StatusID | INT | Foreign key to Status table | 1 |

1. Orders Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| OrderID | INT | Primary key, auto-increments | 1 |
| CustomerID | INT | Foreign key to Customers table | 1 |
| OrderDate | DATETIME | Date and time the order was placed | '2023-01-03 10:00:00' |
| TotalAmount | DECIMAL(10,2) | Total cost of the order | 99.99 |
| StatusID | INT | Foreign key to Status table | 1 |
| DateCreated | DATETIME | Date and time the order record was created | '2023-01-03 10:00:00' |
| DateModified | DATETIME | Date and time the order record was last modified | '2023-01-04 10:00:00' |

1. OrderDetails Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Design Notes** | **Example** |
| OrderDetailID | INT | Primary key, auto-increments | 1 |
| OrderID | INT | Foreign key to Orders table | 1 |
| BookID | VARCHAR(13) | Foreign key to Books table | '978-3-16-148410-0' |
| Quantity | INT | Number of copies of the book ordered | 2 |
| Price | DECIMAL(10, 2) | Price at which each book was sold | 49.99 |

### Database Diagram

## A screenshot of a computer generated diagram Description automatically generated

## A screenshot of a computer Description automatically generated

## A screenshot of a computer generated diagram Description automatically generated

## Class Diagram

To be done.

## Design of Application Domain Classes

## Employee Class

### Overview

The Employee class represents an employee in the system, encapsulating all relevant employee information and behaviors.

### Properties

EmployeeID (int): Unique identifier for the employee, assigned by the database.

FirstName (string): Employee's first name.

LastName (string): Employee's last name.

Email (string): Employee's email address.

PhoneNumber (string): Employee's phone number.

PositionID (int): Identifier for the employee's position, linking to a Position entity.

StatusID (int): Represents the current status of the employee (e.g., active, inactive).

### Constructors

Default constructor initializes the string properties to empty strings, PositionID to 0, and StatusID to 1 (active status).

Parameterized constructor allows setting initial values for FirstName, LastName, Email, PhoneNumber, and PositionID.

### Methods

SaveEmployee(Employee employee): Saves the given Employee instance to the database. Returns the EmployeeID of the new or updated record.

GetAllEmployees(): Retrieves a list of all employees from the database.

SearchEmployeeById(int employeeID): Searches for an employee by their EmployeeID.

SearchEmployeeByEmail(string email): Searches for an employee by their email address.

SearchEmployeesByPosition(int positionID): Retrieves a list of employees by their position.

SearchEmployees(string searchStr): Searches employees by first name, last name, or phone number.

SearchEmployees(string firstName, string lastName): Searches employees by both first and last names.

UpdateEmployee(Employee employee): Updates an existing employee's record in the database.

DeleteEmployee(int employeeID): Deletes an employee from the database using their EmployeeID.

## UserAccount Class

### Overview

The UserAccount class represents a user account in the system, encapsulating all relevant account information and behaviors.

### Properties

UserID (int): Unique identifier for the user account, assigned by the database.

EmployeeID (int): Identifier linking the user account to an Employee entity.

Username (string): The user's chosen username.

Password (string): The user's password.

UserRole (UserRole enum): The role assigned to the user, determining their permissions.

DateCreated (DateTime): The date when the account was created.

DateModified (DateTime?): The date when the account was last modified, nullable for accounts that haven't been modified.

StatusID (int): Represents the current status of the user account (e.g., active, inactive).

MustChangePassword (bool): Indicates whether the user is required to change their password upon next login.

### Constructors

Default constructor initializes Username and Password to empty strings, sets UserRole to default, StatusID to 1 (active status), and MustChangePassword to true.

Parameterized constructors allow initializing all properties with specific values, suitable for creating new user accounts or instantiating existing ones from database records.

### Methods

SaveUserAccount(UserAccount userAccount): Saves the given UserAccount instance to the database.

GetAllUserAccounts(): Retrieves a list of all user accounts from the database.

SearchUserAccountByEmployeeId(int id): Searches for a user account associated with a specific employee ID.

SearchUserAccountByUserAccountId(int id): Searches for a user account by its unique identifier.

SearchUserAccountByUsername(string username): Searches for a user account by the username.

SearchUserAccountByUserRole(UserRole userRole): Retrieves a list of user accounts with a specific role.

UpdateUserAccount(UserAccount user): Updates an existing user account in the database.

DeleteUserAccount(int userID): Deletes a user account from the database using the UserID.

GenerateRandomPassword(): Generates a random password string, used for initial password setup or resetting.

## Design of GUI Classes

## FormEmployee Class

### Overview

FormEmployee is a user interface for managing employee records. It allows users to add, update, search for, and delete employee records. It also provides functionality to link an employee to a user account.

### Key Controls and Functions

Employee Information Fields: Text boxes for entering and displaying employee details (ID, first name, last name, email, phone number).

Position ComboBox: A drop-down list for selecting the employee's position.

Add, Update, Delete Buttons: Controls for creating new employee records, updating existing ones, and deleting them.

Search Functionality: Allows users to search for employees by various criteria (ID, name, email, position).

Link to User Account: A button to open the FormUserAccount for the selected employee, allowing to either view or create a user account for them.

List All: Displays all employee records in a DataGridView.

### Workflow

Users can enter employee details and click the "Add" button to create a new record.

To update an employee, users can search for an employee, modify the details, and click the "Update" button.

Employees can be searched by ID, name, email, or position, with results displayed in the DataGridView.

The "Delete" button removes the selected employee from the database.

The "Link to User Account" button opens the user account form for the selected employee, facilitating account management.

## FormUserAccount Class

### Overview

FormUserAccount is a user interface for managing user accounts associated with employees. It allows for the creation, updating, searching, and deletion of user accounts.

### Key Controls and Functions

User Account Fields: Text boxes for entering and displaying user account details (ID, username).

UserRole ComboBox: A drop-down list for selecting the user's role.

Add, Update, Delete Buttons: Controls for creating new user accounts, updating existing ones, and deleting them.

Search Functionality: Allows users to search for user accounts by ID, username, or role.

Employee Information Display: Shows the associated employee's details for reference.

### Workflow

When accessed from the FormEmployee, the form pre-populates the associated employee's information.

Users can enter a username, select a role, and click the "Add" button to create a new user account.

To update a user account, users can search for an account by various criteria, modify the details, and click the "Update" button.

User accounts can be searched by ID, username, or role, with results displayed in the DataGridView.

The "Delete" button removes the selected user account from the database.

## Design of Data Access Classes

## UtilityDB Class

### Purpose

Provides a utility method to establish a database connection using the connection string defined in the application's configuration file.

### Method

GetDBConnection(): Returns an open SqlConnection object to be used in other DAL methods for database operations.

## EmployeeDB Class

### Purpose

Manages CRUD operations related to the Employee entity in the database.

### Methods

SaveRecord(Employee employee): Inserts a new employee record into the database and returns the auto-generated EmployeeID.

GetAllRecords(): Retrieves all employee records from the database and returns them as a list of Employee objects.

SearchById(int employeeID): Searches for an employee by their ID and returns the corresponding Employee object, or null if not found.

SearchByEmail(string email): Searches for an employee by their email and returns the corresponding Employee object, or null if not found.

SearchByPosition(int positionID): Retrieves all employees that match a given position ID and returns them as a list of Employee objects.

SearchByNameOrPhone(string searchStr): Searches for employees by their first name, last name, or phone number that match the given search string.

SearchByFirstAndLastName(string firstName, string lastName): Searches for employees by both first and last names.

UpdateRecord(Employee employee): Updates an existing employee record in the database.

DeleteRecord(int employeeID): Deletes an employee record from the database by their ID.

## UserAccountDB Class

### Purpose

Manages CRUD operations related to the UserAccount entity in the database.

### Methods

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

GetAllRecords(): Retrieves all user account records from the database and returns them as a list of UserAccount objects.

SearchByEmployeeId(int id): Searches for a user account associated with a given employee ID and returns the UserAccount object, or null if not found.

SearchByUserAccountId(int id): Searches for a user account by its ID and returns the UserAccount object, or null if not found.

SearchByUsername(string username): Searches for a user account by its username and returns the UserAccount object, or null if not found.

SearchByUserRole(UserRole userRole): Retrieves all user accounts that match a given user role and returns them as a list of UserAccount objects.

UpdateRecord(UserAccount userAccount): Updates an existing user account record in the database.

DeleteRecord(int userID): Deletes a user account record from the database by its ID.

# Project Implementation

## Problems Faced During Implementation

# Here's a summary of the main problems faced during the implementation:

# 1. Handling Null Reference Exceptions

# One of the recurrent issues encountered was related to null reference exceptions. This problem was particularly prevalent when dealing with GUI elements and data manipulation, highlighting the importance of thorough null checks and proper initialization of objects before use. Ensuring that all variables and objects were correctly instantiated before accessing their members was crucial in resolving these exceptions.

# 2. Database Connectivity and CRUD Operations

# Ensuring reliable and efficient database connectivity posed challenges, especially when implementing CRUD (Create, Read, Update, Delete) operations. Managing connections, executing SQL commands, and handling data retrieval and updates required meticulous attention to detail to prevent data inconsistency and ensure the integrity of transactions.

# 3. Data Conversion and Parsing Errors

# Errors related to data conversion and parsing were encountered, particularly when converting text input into numerical data types or parsing enum values. These issues underscored the need for robust data validation and error handling mechanisms to prevent runtime exceptions and ensure data integrity.

# 4. User Input Validation

# Validating user input to ensure it met the expected format and constraints was another challenge. This was crucial for maintaining data integrity and preventing SQL injection attacks. Implementing comprehensive validation logic, including regular expressions and custom validation rules, was essential for mitigating these issues.

# 5. Dynamic UI Updates Based on Data Changes

# Dynamically updating the user interface in response to data changes presented difficulties, especially in scenarios where user actions in one part of the application needed to reflect immediately in another. Ensuring a seamless and reactive user experience required careful design of event handlers and state management.

# 6. Enum Handling in GUI Elements

# Integrating enums with GUI elements, such as ComboBoxes, for roles or statuses, and ensuring their correct display and selection was challenging. This involved mapping enum values to user-friendly strings and vice versa, which required implementing utility methods for enum description handling.

# Project Testing

To be done.

# Project Development

This section is optional.

## Next Steps

Having completed Module 1 of the Hi-Tech Order Management System, I am now focusing on the next phases of development to enhance the system's capabilities and user experience.

### Module 2: Customer and Book Management

The next immediate step involves the development of customer and book management features, including the ability to add, update, and delete records, coupled with extensive search and listing functionalities. This module will be developed in disconnected mode, utilizing a dataset to ensure efficient data handling and manipulation without the need for a constant database connection.

### Module 3: Inventory Control

Developing an inventory control system is crucial for managing book stocks effectively. This phase will introduce functionalities for tracking and adjusting inventory levels to match actual stock, minimizing discrepancies and ensuring order accuracy.

### Module 4: Customer – Order Management

This phase will focus on implementing advanced order management functionalities using the Entity Framework for data access. This will include a comprehensive set of tools for managing customer orders, encompassing order creation, modification, cancellation, and review, all through an efficient and robust data access layer.

### GUI Design Enhancement

Improving the GUI design is a critical step in making the system more intuitive and user-friendly. I will focus on refining the interface to facilitate easy navigation and task execution, applying best practices in UI/UX design.

### Role-Based Permissions

Implementing role-based permissions is essential for maintaining system security and operational efficiency. This will ensure that users can access only those features and data that are relevant to their roles within the organization.

### Authentication Mechanisms

I plan to integrate robust authentication mechanisms, including secure login and password management functionalities. This will protect user accounts and align the system with industry-standard security practices.

### Rigorous Testing

Each development phase will include a thorough testing process to ensure the system's reliability and usability. Feedback from the initial module will guide the refinement of subsequent modules.

# Conclusion

The roadmap ahead involves the strategic implementation of development, design, security, and testing principles. By adhering to this plan and focusing on the technical requirements, including disconnected mode operations for Module 2 and the use of Entity Framework in Module 4, I aim to deliver a system that not only meets but also exceeds the operational needs of Hi-Tech Distribution Inc.