**presenter:**

Name: Itay Chabte
ID: 301839452
Object Oriented Programming - 20586

# SmartShop

**JUNE 13, 2021**

## Planning and design document

# Table of contents

## Overview of the system

The system is based on 3-Tier architecture, with an additional layer Which represents the Business entities (BE). The 3 Tiers are:

1. Frontend (UI Layer) -Implemented by Webforms ,a c# web application.
2. Backend (Server side) - A layer of C# Libraries that will be responsible for business logic, Database adapter ,Business entities and Data Validation Library .
3. Database - database layer storing data.

## Terminology

FrontEnd - the code that will be executed at the worker and their customers browser.

BackEnd - Server side, the code that executes on the server that is hosting the whole service, one instance per platform

## Conventions

1. Every Class will start with a capital letter and every meaningful word inside the class name will start with a capital letter.
2. Every Class name in the DAL/BLL libraries will start with the name of the class then bottom dash and then The name that represents the department
3. Every interface will start with “I” before the name.

# Backend

The server side is a composite of 4-Tier architecture.

- UI - Using the C # web interface divides the UI into Frontend and Backend
  Some controllers like GridView, SqlDataSource etc are used as Backend
  While buttons and links etc are used as a frontend

- BLL - Single instance of a class that will be responsible for business logic responding to worker/customer frontend requests.

- DAL - Database access layer.
- BE - Business entities, To store the data for each system user and for each purchase.

There is another class that is responsible for checking data validation and it is not part of the 4-Tier architecture, We will explain about it later

- DalidationByDataAnnotations - Data Annotations Validation Library for the Business entities specific to the implementation of the FrontEnd.

BLL, DAL and BE are libraries contain interfaces for each object which means we can make very big changes from changing the logical layer to changing the database to changing the objects themselves and everything according to the needs of the programmer
In addition, because we have libraries, it is possible to develop Phone app and Computer application without any additional programming other than user interface development

# Business Entities

**Item:** Describes each entity that is part of the product

| Property | Type | Description |
|---|---|---|
| ID | int | The ID number of the category/product |
| Description | string | Category/product description |
| AddDate | DateTime | Date of adding/change category/product |

**Category:** The data object to store each category that characterizes a product, inherits from Item

| Property | Type | Description |
|---|---|---|
| Title | string | Category title<br>Used to characterize a product |

**Product:** The data object to store the information in a single product , inherits from Item

| Property | Type | Description |
|---|---|---|
| Name | string | product name |
| Category | string | The category assigned to the product |
| Price | decimal | The price of the product |

| | | |
|---|---|---|
| Quantity | int | The quantity of the product |
| Product_Img_Link | string | The image of the product |

**Purchase:** Describes each entity that is part of the purchase

| **Property** | **Type** | **Description** |
|---|---|---|
| OrderID | string | The Order ID per purchase |
| Purchase_Time | DateTime | Time of purchase |

**Orders:** The data object to store the information of all products for each single purchase, inherits from Purchase

| **Property** | **Type** | **Description** |
|---|---|---|
| ProductId | int | ID number of the purchased product |
| ProductName | string | Name of the product purchased |
| Price | decimal | Price of the purchased product |
| Quantity | int | Quantity of product purchased |

**PurchaseDetails:**The data object to store the information for each single purchase, inherits from Purchase

| Property | Type | Description |
|---|---|---|
| ID_User | int | ID number of the user who made the purchase |
| Grand_Total | decimal | Total purchase amount |
| CardNumber | int | Credit card number |
| CVV | int | CVV of the credit card |
| CardExpirationDate | string | The expiration date of the credit card |
| PurchaseComplete | bool | Indication of completion of purchase |

**Person:**Describes each entity that is part of the person

| Property | Type | Description |
|---|---|---|
| ID | int | The ID of the store for the user\worker |
| FirstName | string | user\worker first name |
| LastName | string | user\worker last name |
| Password | string | user\worker password to enter the operating area |
| Phone | string | user\worker phone |

| | | |
|---|---|---|
| Address | string | user\worker address |
| BuisnessWorker | bool | Indication of whether it worker or user |
| AddDate | DateTime | Date added user\worker |

**User:**The data object to store the information for each single User, inherits from Person

| **Property** | **Type** | **Description** |
|---|---|---|
| Email | string | User email<br><br>To send an email and enter the personal area |

**Worker:**The data object to store the information for each single Worker in the store, inherits from Person

| **Property** | **Type** | **Description** |
|---|---|---|
| PersonalID | int | The personal ID of the worker<br><br>Through it he entered authorized areas |
| Authorization | string | Authorization for each worker |

# Database access layer

The Database access layer is responsible for working with a database to store new users(customers/workers), categories, products, store information, etc.
This layer is responsible for the connection between the database and Business logical
The selected database is Microsoft SQL server.
Each class starts with a name DAL_(NAME OF CLASS), Which contains the methods for inserting, deleting, adding, etc. to each class
Each class uses System.Configuration, System.Data, System.Data.SqlClient that contains the method:
**ConfigurationManager.ConnectionStrings -** Contains the address string of the database, the ConnectionStrings defined in the file “Web.config”.
**SqlConnection -** Establishes an object for contacting the database.
**Open() -** Opens  connection to the database.
**Close() -** Closes connection to the database.
**SqlCommand -** Represents a Transact-SQL statement or stored procedure to execute against a SQL Server database.
**ExecuteNonQuery() -** Executes an SQL statement against the Connection and returns the number of rows affected.
The library includes a Factory Design pattern So that we can create classes without knowing the internal classes, currently the project contains only implementation of  sql server under the interface implemented ,such implementation allows changing/shifting or switching between database providers without changes in other components.

## Database Access layer description:

This is the interface that every future database implementation must implement, the business logic layer will call those functions without considering the underlying implementation of the database.

# Interfaces:

## IDAL_DATABASE.cs

This interface represents any object in the system which also uses as an object stored in the database
This interface allows you to insert, delete, update, view, search, search by ID and return a requested object

Defining methods:

```
bool Add(Object NewObject);
bool Delete(Object NewObject);
bool Update(Object NewObject);
DataTable Select();
DataTable Search(string KeyWord);
DataTable SearchById(string id);
object FromIdToObject(string id);
```

## IDAL_Orders.cs

This interface inherits from IDAL_PurchaseDetails and represents any order object in the system ,This interface allows you all All inherited methods in addition to get the data for product order between two dates

Defining methods:

```
DataTable SelectProductByDate(string startDate, string endDate, string ProductName);
```

## IDAL_PurchaseDetails.cs

This interface inherits from IDAL_DATABASE and represents any purchaseDetails object in the system ,This interface allows you all All inherited methods in addition to get the data for purchase between two dates

Defining methods:

DataTable SelectByDate(string startDate, string endDate);

### IDAL_User.cs

This interface inherits from IDAL_DATABASE and represents any User's object in the system ,This interface allows you all All inherited methods in addition to check the user's connection

Defining methods:

bool LoginCheck(Object NewObject);

## Classes:

### DAL_config.cs

This class contains all the Entities responsible for communicating with the database

attributes:

```
public static string MyConnString =   ConfigurationManager.ConnectionStrings[Connection String of database].ConnectionString;
        public string command { get; set; }
        public SqlConnection connect { get; set; }
        public SqlCommand cmd { get; set; }
        public SqlDataAdapter adapter { get; set; }
        public bool AddSuccess { get; set; }
```

### DAL_Orders.cs

This class inherits from DAL_config and implements the interface IDAL_Orders (which inherits from IDAL_DATABASE ).

This class implements IDAL_DATABASE methods and in addition returns information about specific products and ordering products between two dates.

Defining methods:

public DataTable SelectByDate(string startDate, string endDate)
public DataTable SelectProductByDate(string startDate, string endDate, string ProductName)

### DAL_PurchaseDetails.cs

This class inherit from DAL_config and implement the interface IDAL_PurchaseDetails (which inherit from IDAL_DATABASE )
This class implements IDAL_DATABASE methods and in addition returns information about Purchases between two dates.

Defining methods:

public DataTable SelectByDate(string startDate, string endDate)

### DAL_Worker.cs/DAL_User.cs

This class inherit from DAL_config and implement the interface IDAL_User (which inherit from IDAL_DATABASE )
This class implements IDAL_DATABASE methods and in addition performs user authentication.

Defining methods:

public bool LoginCheck(Object NewObject).

### Repetitive methods

All classes:
DAL_Category.cs /DAL_Orders.cs/ DAL_Product.cs/ DAL_PurchaseDetails.cs/ DAL_StoreSet.cs/ DAL_Worker.cs/ DAL_User.cs

inherit from DAL_config and implement the interface IDAL_DATABASE therefore all classes have an implementation for adding, deleting, searching, searching by ID, updating, selecting and receiving the object of the same class object

Methods:

public bool Add(object NewObject)
public bool Delete(object NewObject)
public object FromIdToObject(string id)
public DataTable Search(string KeyWord)
public DataTable SearchById(string id)
public DataTable Select()
public bool Update(object NewObject)

## FactoryDal.cs

This class implements the Factory design pattern that will be explained later (in Design Pattern section)

Defining methods:

public static IDAL_User GetDalUser().
public static IDAL_User GetDalWorker().
public static IDAL_DATABASE GetDalCategory().
public static IDAL_DATABASE GetDalProduct().
public static IDAL_Orders GetDalOrders().
public static IDAL_PurchaseDetails GetPurchaseDetails().
public static IDAL_DATABASE GetStoreSet().

# Business logical layer

This layer is responsible for the connection between the UI and the Database access layer
The layer will prevent unauthorized users from making restricted changes to the platform and the database
The library includes a Factory Design pattern In addition to Singleton Design pattern,So we will create a class at most once ,and we will not have to know the internal structure of that class and thus we add a layer of protection and increase system performance ,such implementation allows changing/shifting or switching between Existing classes to others without changes in other components.

## Guiding principles

Encapsulation - Saving information in classes and extracting it using the attributes only
SRP (single responsibility principle) - each department receives only one responsibility
Design patterns - that will enable efficient use of the system and meet additional engineering objectives

## Interfaces:

### IBLL.cs

This interface represents any object in the system which also uses as an object stored in the database
This interface allows you to insert, delete, update, view, search, search by ID and return a requested object

Defining methods:

```
bool Add(Object NewObject);
bool Delete(Object NewObject);
bool Update(Object NewObject);
DataTable Select();
```

DataTable Search(string KeyWord);
DataTable SearchById(string id);
object FromIdToObject(string id);

### IBLL_Orders.cs

This interface inherits from IBLL and represents any order object in the system ,This interface allows you all inherited methods in addition to get the data for a single product or all products between two dates and

Defining methods:

DataTable SelectByDate(DateTime startDate, DateTime endDate)
DataTable SelectProductByDate(DateTime startDate, DateTime endDate, string ProductName)

### IBLL_Product.cs

This interface inherits from IBLL and represents any product object in the system ,This interface allows you all inherited methods in addition to get the data for all empty products

Defining methods:

DataTable ProductInventoryZero()

### IBLL_PurchaseDetails.cs

This interface inherits from IBLL and represents any purchaseDetails object in the system ,This interface allows you all All inherited methods in addition to get the data for purchase between two dates and sending mail for each purchase

Defining methods:

DataTable SelectByDate(string startDate, string endDate)
void SendEmail(string mailToSend, string orderID)

### IBLL_User.cs

This interface inherits from IBLL and represents any Users object in the system ,This interface allows you all All inherited methods in addition to check the user's connection

Defining methods:

bool LoginCheck(Object NewObject)

### IBLL_StoreSet.cs

This interface inherits from IBLL and represents any Store object in the system ,This interface allows you all All inherited methods in addition to sending mail for each new User

Defining methods:

void SendEmail(string mailToSend, string pass)

## Classes:

### BLL_Orders.cs

This class implements the interface IBLL_Orders(which inherits from IBLL).
This class implements IBLL methods and in addition returns information about specific products and ordering products between two dates.

methods:

public DataTable SelectByDate(string startDate, string endDate)
public DataTable SelectProductByDate(string startDate, string endDate, string ProductName)

### BLL_Product.cs

This class implements the interface IBLL_Product(which inherits from IBLL).
This class implements IBLL methods and in addition returns a table of all empty products.

methods:

public DataTable ProductInventoryZero()

### BLL_PurchaseDetails.cs

This class implements the interface IBLL_PurchaseDetails(which inherits from IBLL).
This class implements IBLL methods and in addition to get the data for purchase between two dates and sending mail for each purchase.

methods:

DataTable SelectByDate(string startDate, string endDate)
void SendEmail(string mailToSend, string orderID)

### BLL_StoreSet.cs

This class implements the interface IBLL_StoreSet(which inherits from IBLL).
This class implements IBLL methods and in addition to sending mail for each new User.

methods:

void SendEmail(string mailToSend, string orderID)

### BLL_User.cs/BLL_Worker

This class implements the interface IBLL_User(which inherits from IBLL).
This class implements IBLL methods and in addition to performs user authentication.

Defining methods:

public bool LoginCheck(Object NewObject).

## Repetitive methods

All classes:
BLL_Category.cs / BLL_Orders.cs / BLL_Product.cs / BLL_PurchaseDetails.cs / BLL_StoreSet.cs / BLL_User.cs / BLL_Worker.cs

Implement the interface IBLL therefore all classes have an implementation for adding, deleting, searching, searching by ID, updating, selecting and receiving the object of the same class object

Methods:

public bool Add(object NewObject)
public bool Delete(object NewObject)
public object FromIdToObject(string id)
public DataTable Search(string KeyWord)
public DataTable SearchById(string id)
public DataTable Select()
public bool Update(object NewObject)

In addition each class has a constructor that creates an object from FactoryDal class

| class | constructor |
|---|---|
| BLL_Category | IDAL_DATABASE newCategory = FactoryDal.GetDalCategory() |
| BLL_Orders | IDAL_Orders newOrder = FactoryDal.GetDalOrders() |
| BLL_Product | IDAL_DATABASE newProduct = FactoryDal.GetDalProduct() |
| BLL_PurchaseDetails | StoreSet newStore = new StoreSet()<br>IDAL_DATABASE newSet = FactoryDal.GetStoreSet()<br>IDAL_PurchaseDetails newPurchase = FactoryDal.GetPurchaseDetails() |
| BLL_StoreSet | StoreSet newStore = new StoreSet() |

| | |
|---|---|
| | IDAL_DATABASE newSet = FactoryDal.GetStoreSet(); |
| BLL_User | IDAL_User newUser = FactoryDal.GetDalUser() |
| BLL_Worker | IDAL_User newWorker = FactoryDal.GetDalWorker() |

### FactoryBLL_User.cs

This class implements the Factory and Singleton design pattern that will be explained later (in Design Pattern section)

Attributes:

static IBLL_User user
static IBLL_User worker
static IBLL_category
static IBLL_Product product
static IBLL_StoreSet storeSet
static IBLL_Orders order
static IBLL_PurchaseDetails Purchase

Defining methods:

public static IBLL_User GetBllUser()
public static IBLL_User GetBllWorker()
public static IBLL GetBllCategory()
public static IBLL_Product GetBllProduct()
public static IBLL_StoreSet GetBllStoreSet()
public static IBLL_Orders GetBllOrder()
public static IBLL_PurchaseDetails GetBllPurchase()

# Validation By DataAnnotations

This library is responsible for verifying all the Textbox  in each form in the FrontEnd according to the objects to which it is linked by Data Annotations
This library is not part of Business logical layer because of its direct link to the FrontEnd validation controller ,but because of the use of Data Annotations that implementation allows changing/shifting or switching between validation methode without changes in other components

## Classes:

### DataValidation.cs

This class inherits the abstract class BaseValidator , we override the EvaluateIsValid() method to find the object name and Data Annotation and check the information sent to it accurately

Attributes:

public string PropertyName { get; set; }
public string SourceType { get; set; }

methods:

protected override bool EvaluateIsValid()

# Frontend

This layer presents a user interface that will allow him to interact with the system
This layer interfaces with the logic layer to get services from it (which it itself provides through the database layer)
The frontend is built according to asp.net webforms and it contains the following extension:
Bootstrap - Adding a design to a website
DataTabels - Adds formatted tables
JavaScript, jquery - Short snippets of code appear throughout the site to support element design
ELMAH (Error Logging Modules and Handlers) - an application-wide error logging facility that is completely pluggable

## Guiding principles

Convenient interface- The interface is easy and convenient to operate.
Minimum operation - Minimal operation for both system employees and users

### Pages (Components)

#### AdminManagmentCategory.aspx.cs

This class is responsible for managing the categories
You can delete / add / update / search a category
When entering the data, a data validation check will be performed. If they are correct, the data will be added, If not a list with errors will appear
The categories will then appear on the product management page and search options on the main page

#### AdminManagmentProduct.aspx.cs

This class is responsible for product management
You can delete / add / update / search the product

When entering the data, a data validation check will be performed. If they are correct, the data will be added, If not a list with errors will appear
If there is no product, a list will be filled up
The product will then appear on the main page and search options on the main page

**AdminManagmentPurchaseDetails.aspx.cs**

This class is responsible for managing purchases from the site
You can search / view purchase details and update order status

**AdminManagmentUser.aspx.cs**

This class is responsible for managing users / buyers
You can delete / add / view details / perform a search / update of the user
When entering the data, a data validation check will be performed. If they are correct, the data will be added, If not a list with errors will appear
Each user gets a unique ID site

**AdminManagmentWorkers.aspx.cs**

This class is responsible for managing employees / managers
You can delete / add / view details / search / add permissions / update an employee
When entering the data, a data validation check will be performed. If they are correct, the data will be added, If not a list with errors will appear
Each employee receives a unique ID for the site

**DataAnalyst.aspx.cs**

This class is responsible for data analysis
You can add a start and end date and get a sales detail analysis of the store

In addition, it is possible to compare sales between 2 products on these dates

### LoginWorker.aspx.cs

This class is responsible for verifying data of the employee / manager
The department will perform an input validity check for a personal ID card and password
If they are correct, the employee will only be able to enter the places where he is authorized

### StoreInfo.aspx.cs

This class is responsible for managing store details
Store details can be deleted / added / updated
When entering the data, a data validation check will be performed. If they are correct, the data will be added, If not a list with errors will appear
The details will be updated automatically on the website / invoices
The email details are intended for sending an email to the user

### Cart.aspx.cs

This class is responsible for placing orders
When selecting a product from the home page, if the product exists and it does not exceed the threshold of 20 items, it will be added to the list
You can add / delete items from the list
You can make a purchase if the user is registered
If the user is not registered, he will be taken to the login or registration page and will immediately return to fill in the card details.
If everything is in order, the purchase will be made, an email will be send to the user's address with the order number for tracking and the list will be deleted automatically

**Home.aspx.cs**

This class is responsible for presenting products
Each product will appear with product details and ordering capability
You can also search by category or keyword
If more than 20 items are ordered for the product, an error message will be received

**LoginUser.aspx.cs**

This class is responsible for performing data verification of the buyer / user
If the user is not registered, he will be able to register by going to the registration page
The class will perform an input validity check for email and password
If they are correct, the user will be able to make purchases from the site and enter a personal area on the site

**PdfGenerator.aspx.cs**

This class is responsible for creating an invoice page
After making a purchase we will go directly to this page and we will be able to see the purchase details and buyer details
We can download a PDF file to the invoice

**Site1.Master.cs**

This class is responsible for initializing all pages

This class initializes the footer In addition it is responsible for managing the navigation bar and details

If a user enters / buys the navigation bar will change to the user name and login to a personal area and exit the system

If an employee enters, the navigation bar will change to enter pages according to the permissions given to the employee

The footer will change according to the store's data

**UserProfile.aspx.cs**

This class is responsible for personal management for each user / buyer

When entering the system, access to the personal area will be allowed

In this class you can view / search purchase details

In addition, it is possible to change details and change the password individually

**userSignUp.aspx.cs**

This class is responsible for adding a new user / buyer

When entering the data, a data validation check will be performed. If they are correct, the user will be added and an email will send to the user address with the user details, If not a list with errors will appear

**ErrorPageHandler.aspx.cs**

This class is responsible for managing all Exceptions

When Exception occurs The user receives an error message to contacting the administrator

The administrator receives the error details

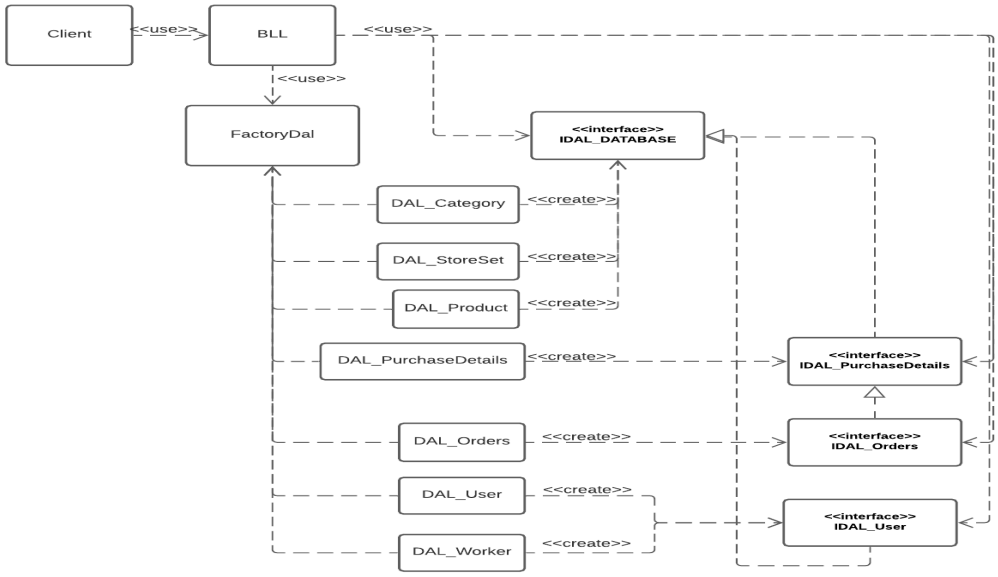
# Design patterns

## Factory

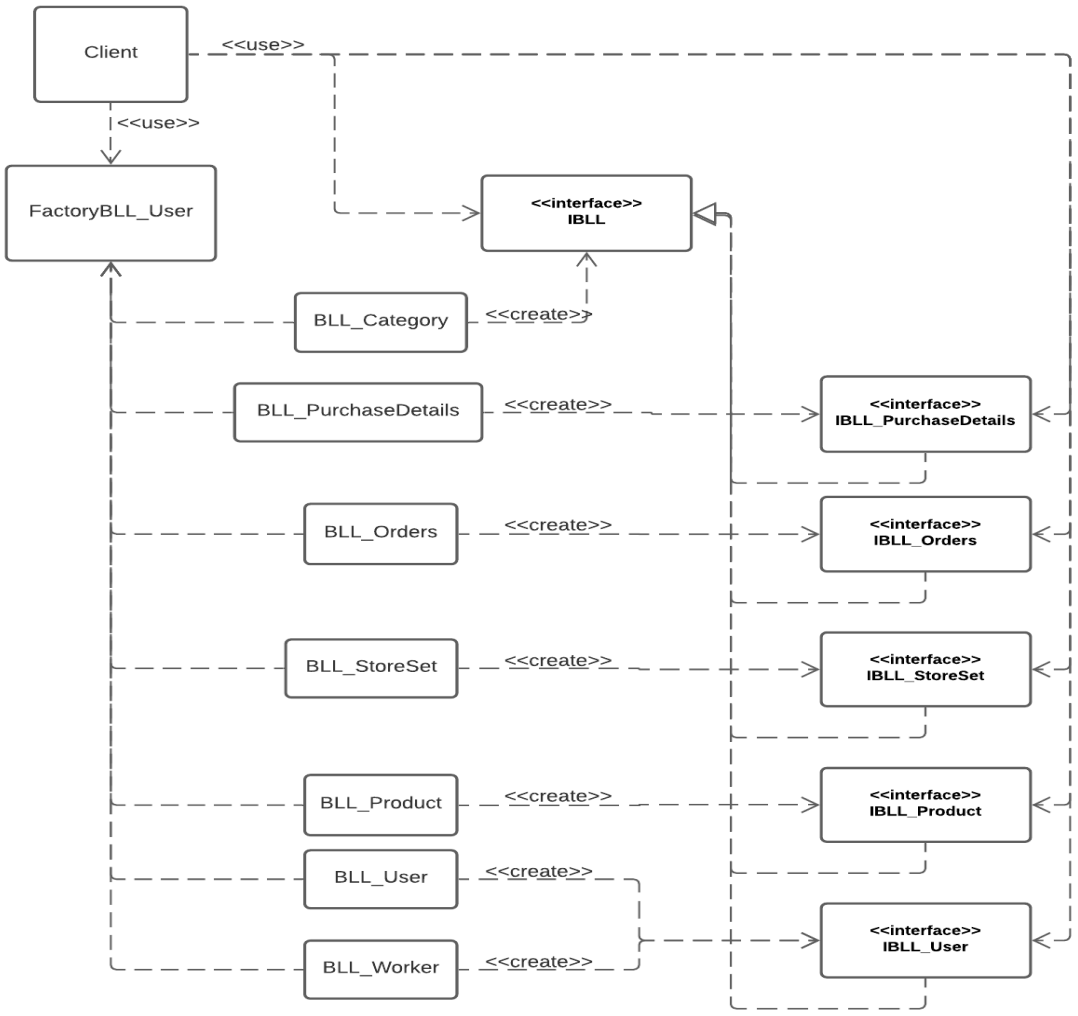
The purpose of Factory pattern is to create objects without exposing the creation logic to the client and refer to newly created objects using a common interface.

In this project we used this design pattern in the following classes:

1. FactoryDal.cs - This class represents a buffer between the DAL layer and the BLL layer, this is a uniform interface without knowing their class .
   The template is based on defining an independent method for creating objects.
   The structure of the class looks like this:



2. FactoryBLL_User.cs - This class represents a buffer between the UI layer and the BLL layer, this is a uniform interface without knowing their class .
   The template is based on defining an independent method for creating objects.
   The structure of the class looks like this:



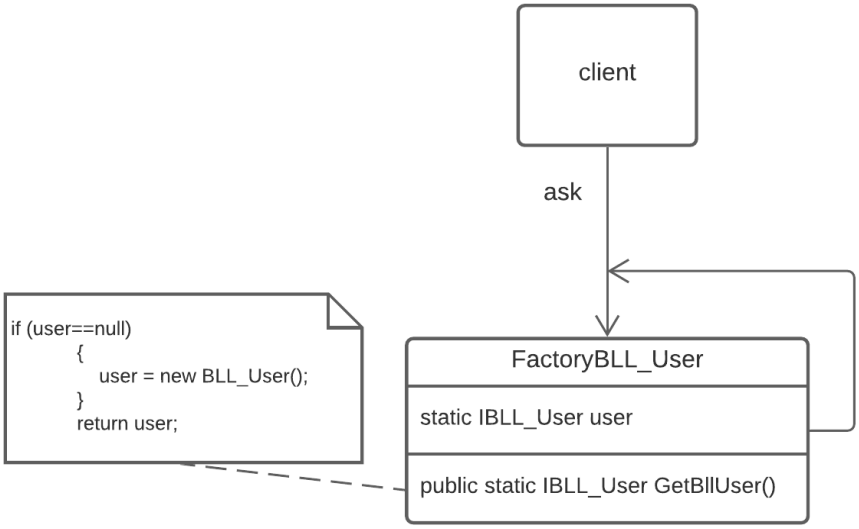

Note: These two diagrams are in detail on the Attachment page

## Singleton

A design pattern whose job is to make sure that the class has a single show
Using this pattern will prevent the formation of logical errors caused by the creation of several instances for classes that require at most one instance

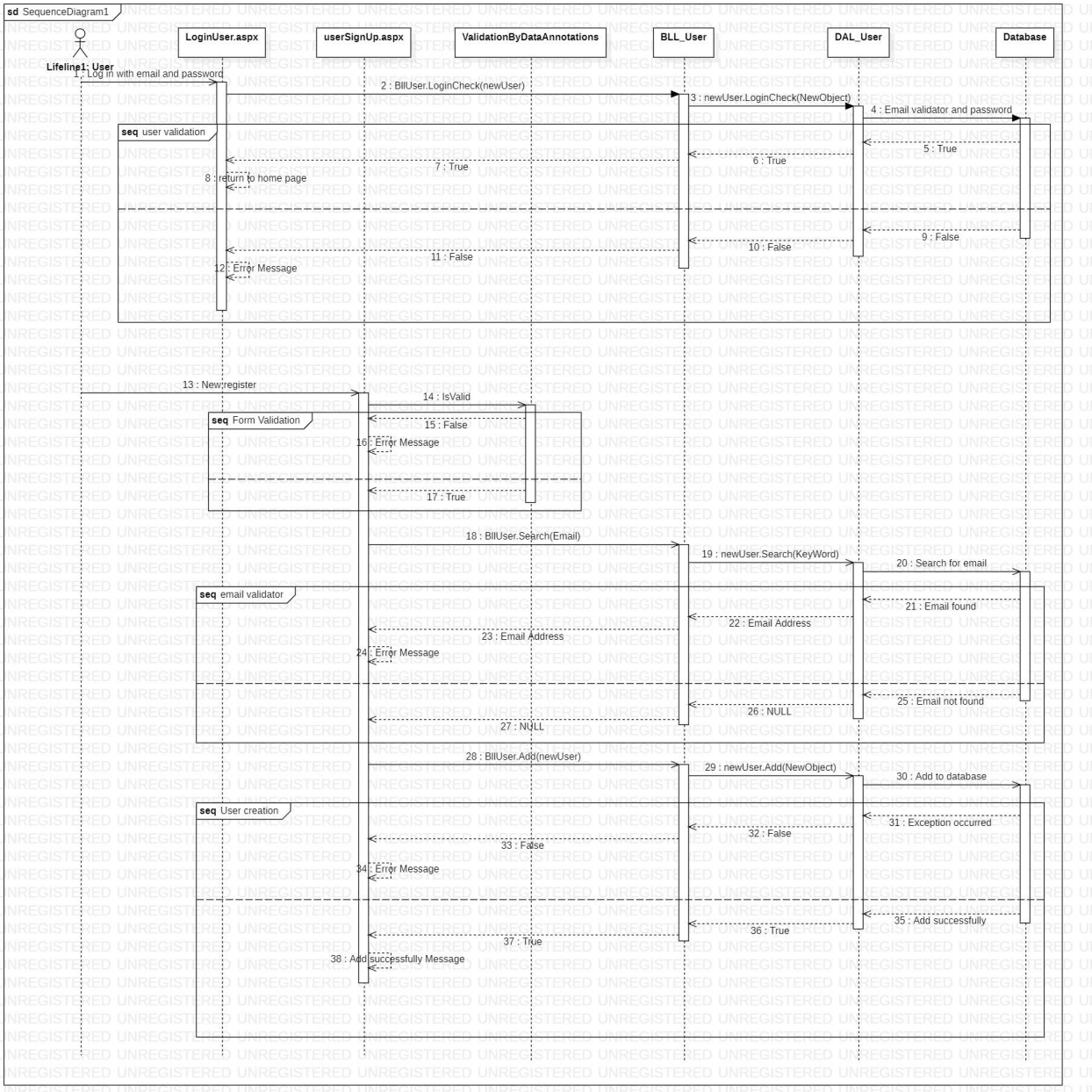
In this project the Singleton is integrated with the Factory method and is only in the BLL class so every time we create an object linked to the database we will create at most one object thus saving running time and system resources on which the system will run

The structure of the class looks like this:

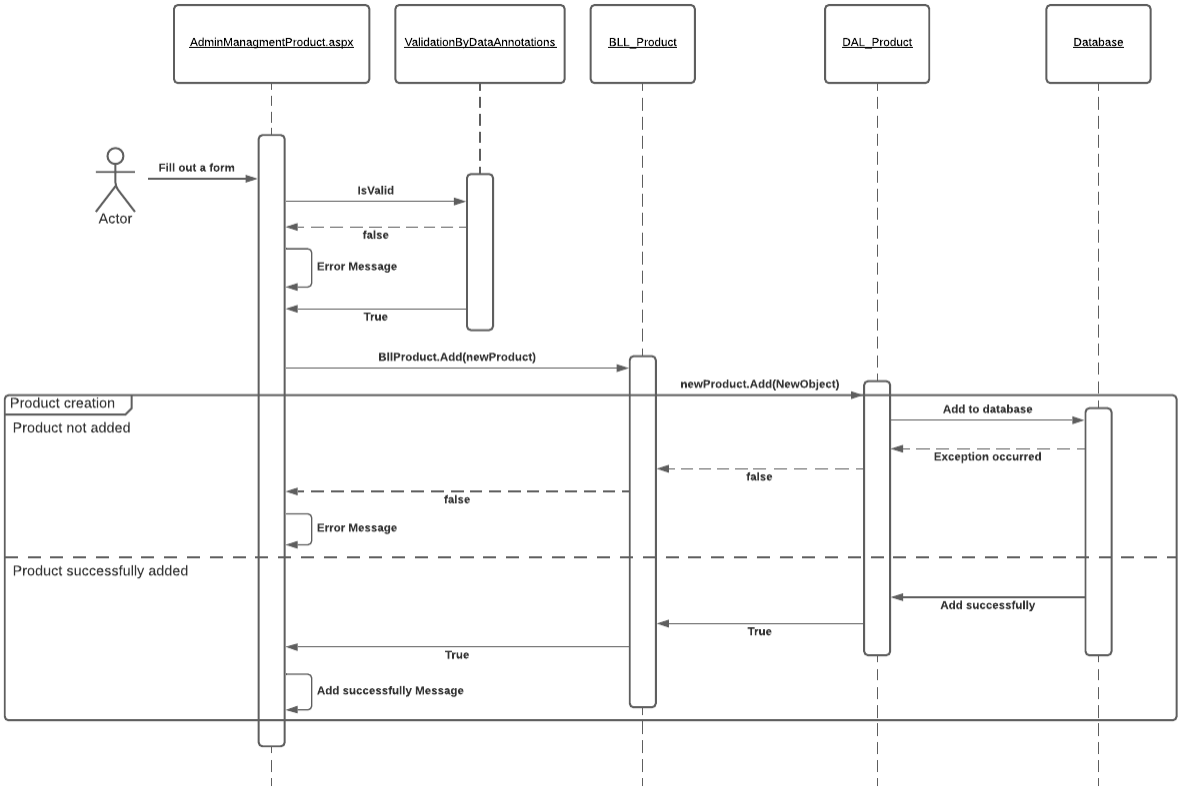


# Sequence diagram

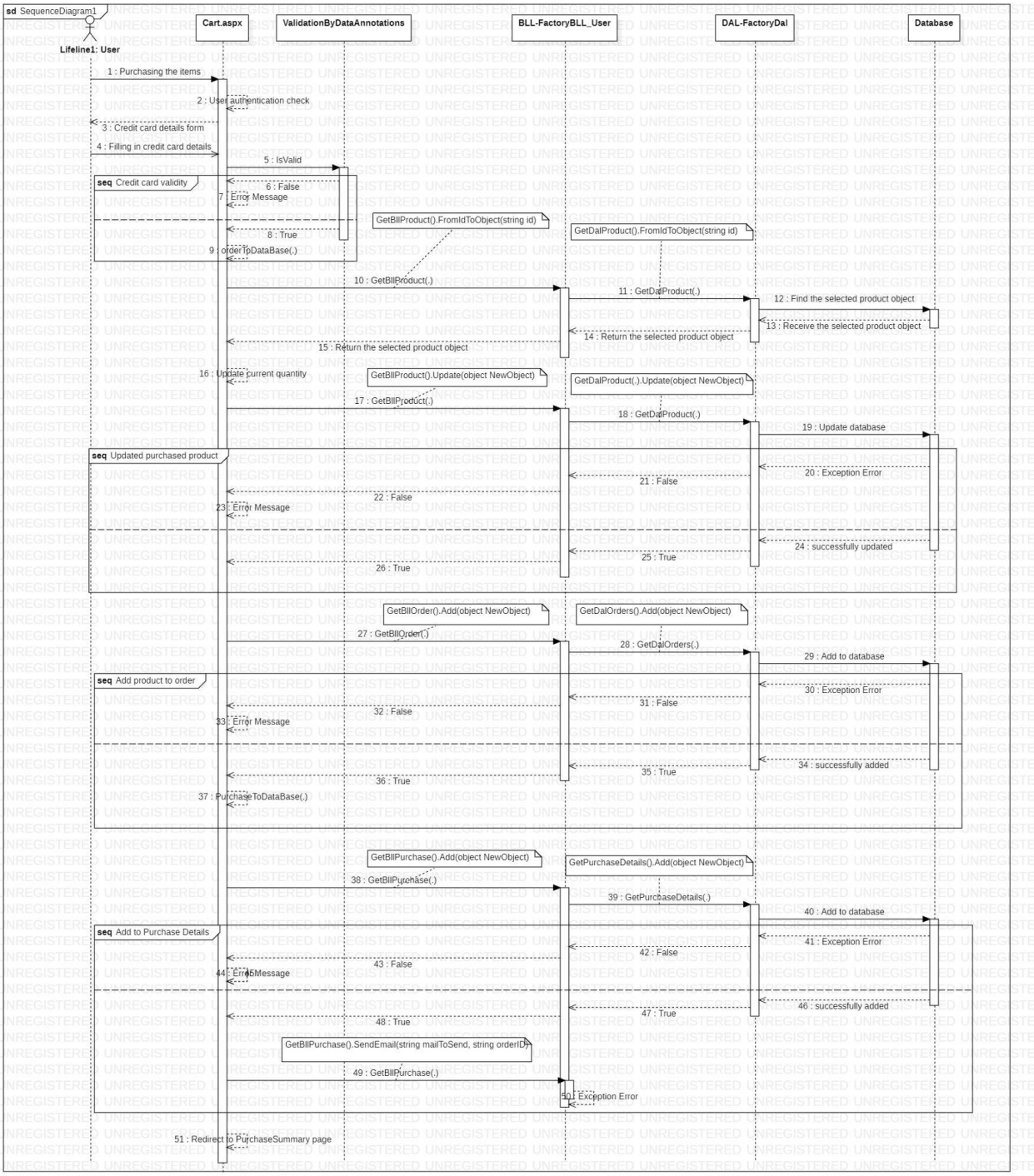
## Sequence diagram for user login and registration



## Sequence diagram for adding a product



# Sequence diagram for product purchase



# Future developments

## Multi platform

The 2 layers (BLL, DAL) are libraries so we can change the interface or add an interface

For example we could use these libraries and build a xamarin or WPF application almost effortlessly and make a connection to the database by a server and thus we created 3 interfaces running on one platform

Of course we can also change the entire interface to react.js or any technology that is convenient for us to use, which means that any developer can develop the interface according to his tools

## N-tier architecture

Using the N-tier architecture model we can change:

The database and the DAL layer - we can change to a faster and larger database and change all the add / delete / search logic, etc., so we can achieve better performance and we will not have to change the other layers because each piece of information has a different interface and everything is bound by Factory method

BLL layer - we can make changes in the logic layer, such as making input

We can change the whole layer because here too we use a separate interface for each piece of information when we use the Factory method to bind everything and the Singleton method to ensure a high level of performance

In addition, it is also possible to change the Factory method to abstract, thus adding flexibility to the system and its performance

UI layer - we can use different platforms from C # to implement easily or heavily depending on the programmer's requirements

# Setup Guide

The project contains all the extensions and packages for development

Only required to install Visual Studio : visual studio

In case of malfunction / lack of coordination, it is possible to download all extensions / packages again:

Bootstrap - Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.

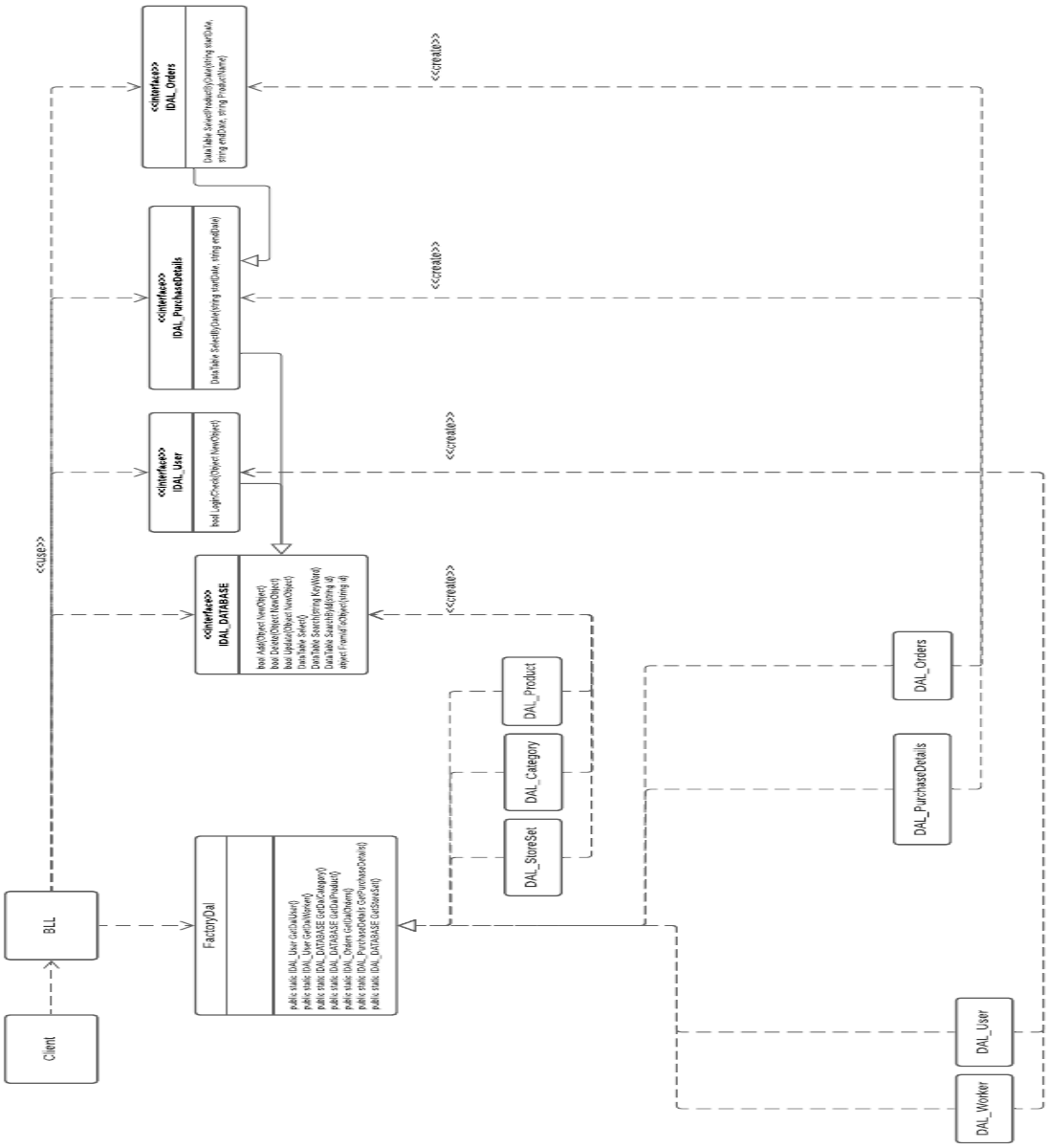
font awesome - vector icons and social logos on your website with Font Awesome, the web's most popular icon set and toolkit

Data Tables -advanced interaction controls to your HTML tables

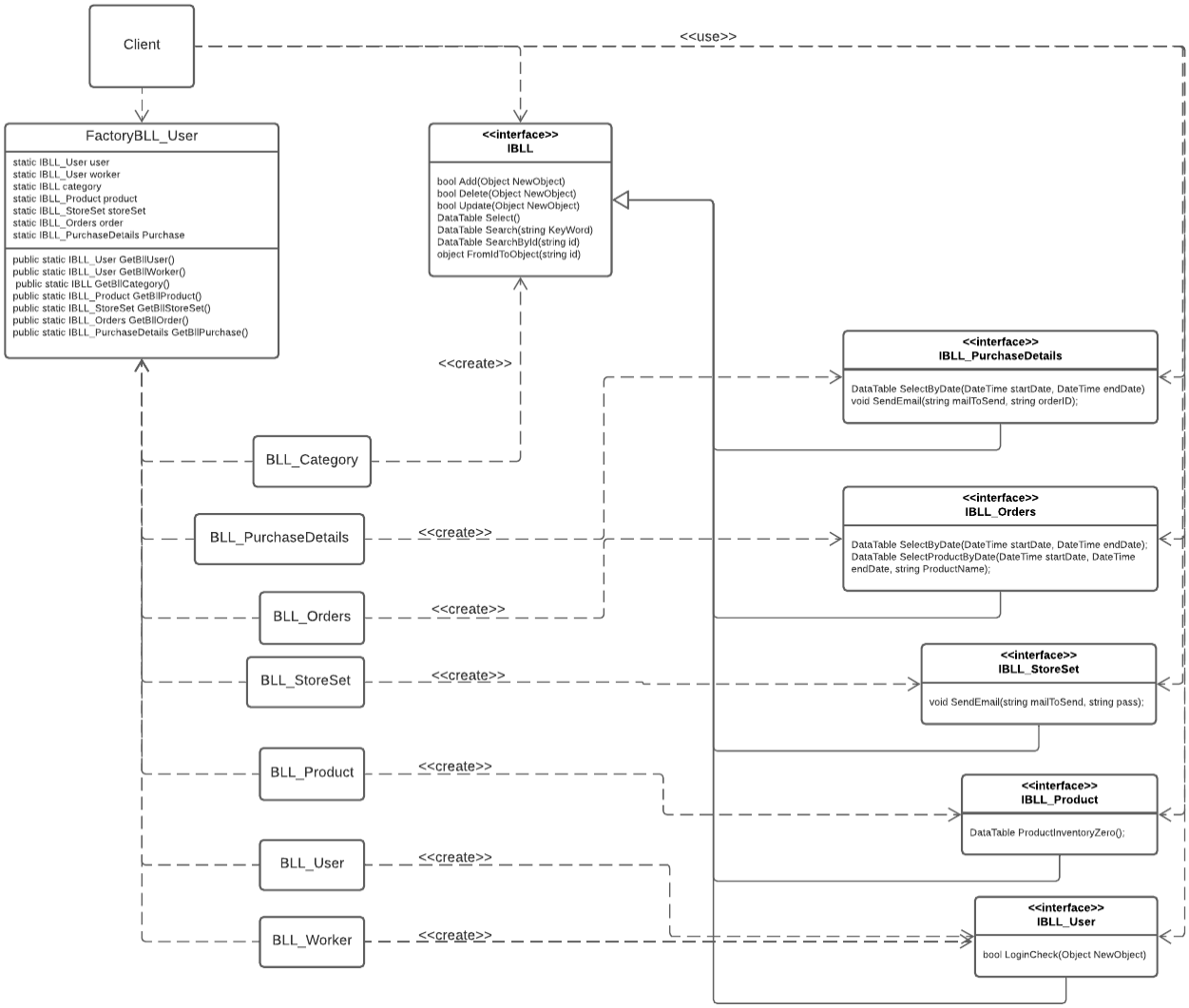
ELMAH- ELMAH (Error Logging Modules and Handlers) is an application-wide error logging facility that is completely pluggable. It can be dynamically added to a running ASP.NET web application, or even all ASP.NET web applications on a machine, without any need for re-compilation or re-deployment.

# Attachment

## FactoryDal.cs Diagram



## FactoryBLL_User.cs Diagram



## Data structure tables