# Pharmacy Management System


Authors:

Souheir Al-Khatib

Hawraa Zein

Instructor:

Dr. Mohammad Dbouk

# Contents

## Introduction and Product Description:

Our mini project is a comfortable and time saving application that is used by pharmacies. It displays the available products at the pharmacy and enables the customer to make orders online after signing up.

It also enables the admin to check latest updates about customers, commodities and suppliers.

The app was built with the specialty of ease to use and the appropriate storage of data using specific and studied database schemas.

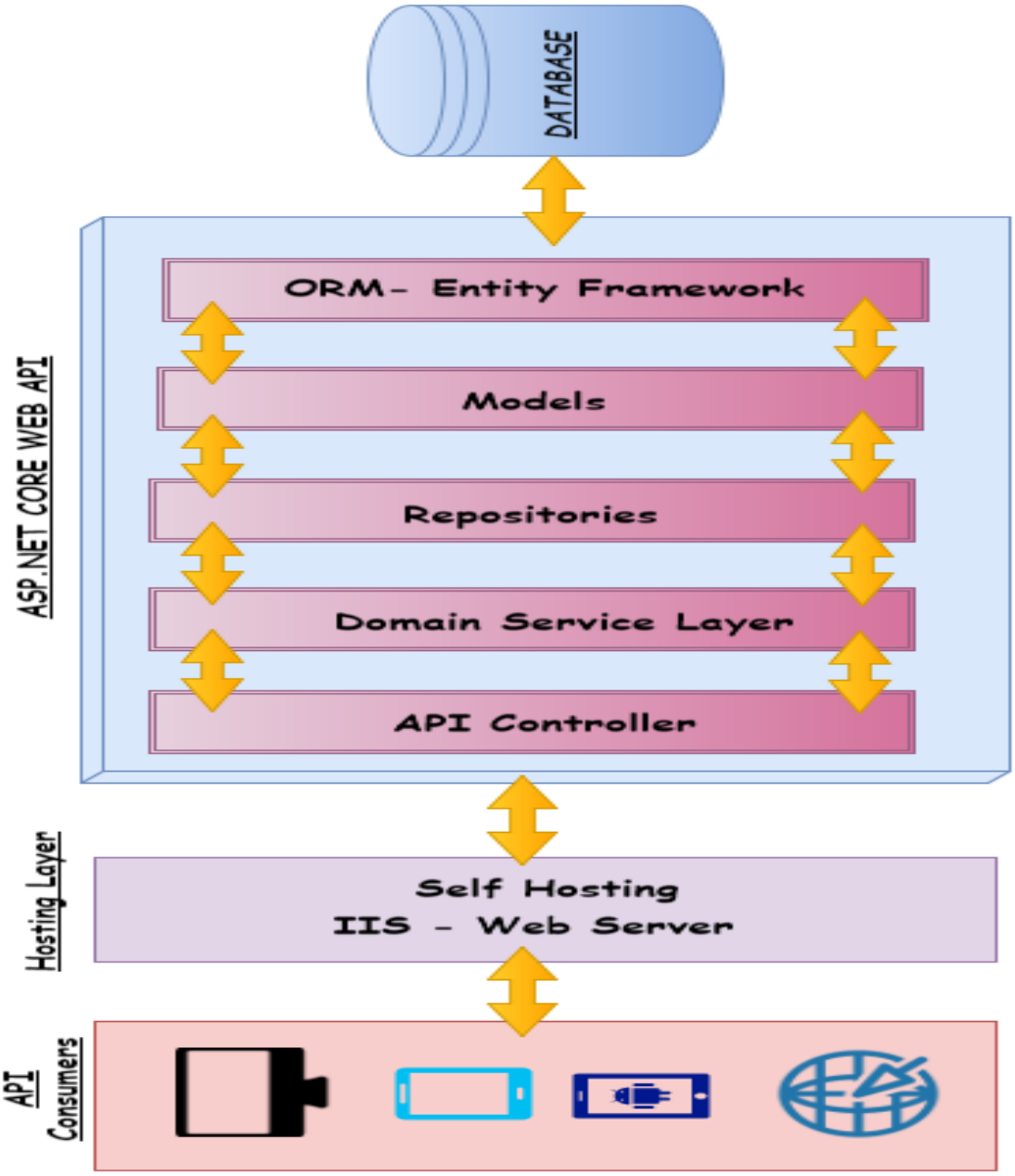
## Requirements:

- Database ( using sql server )
- Interfaces ( using c# )
- UML Diagram ( using power designer )

## Coverage:

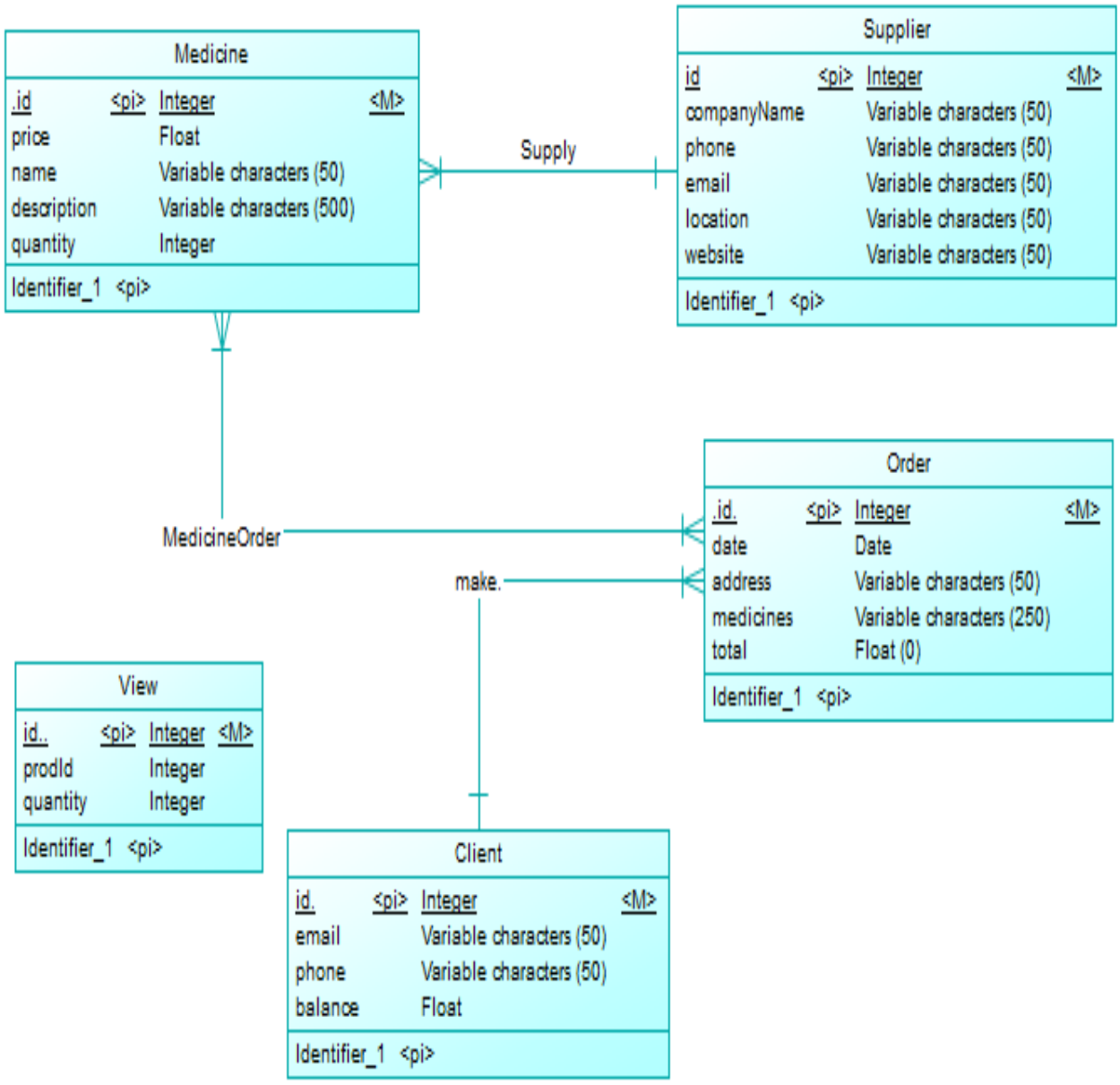
The following project covers the main concepts of Database course where indices, procedures, triggers, and transactions were used to make an efficient product.

# Chapter 1: Application Architecture

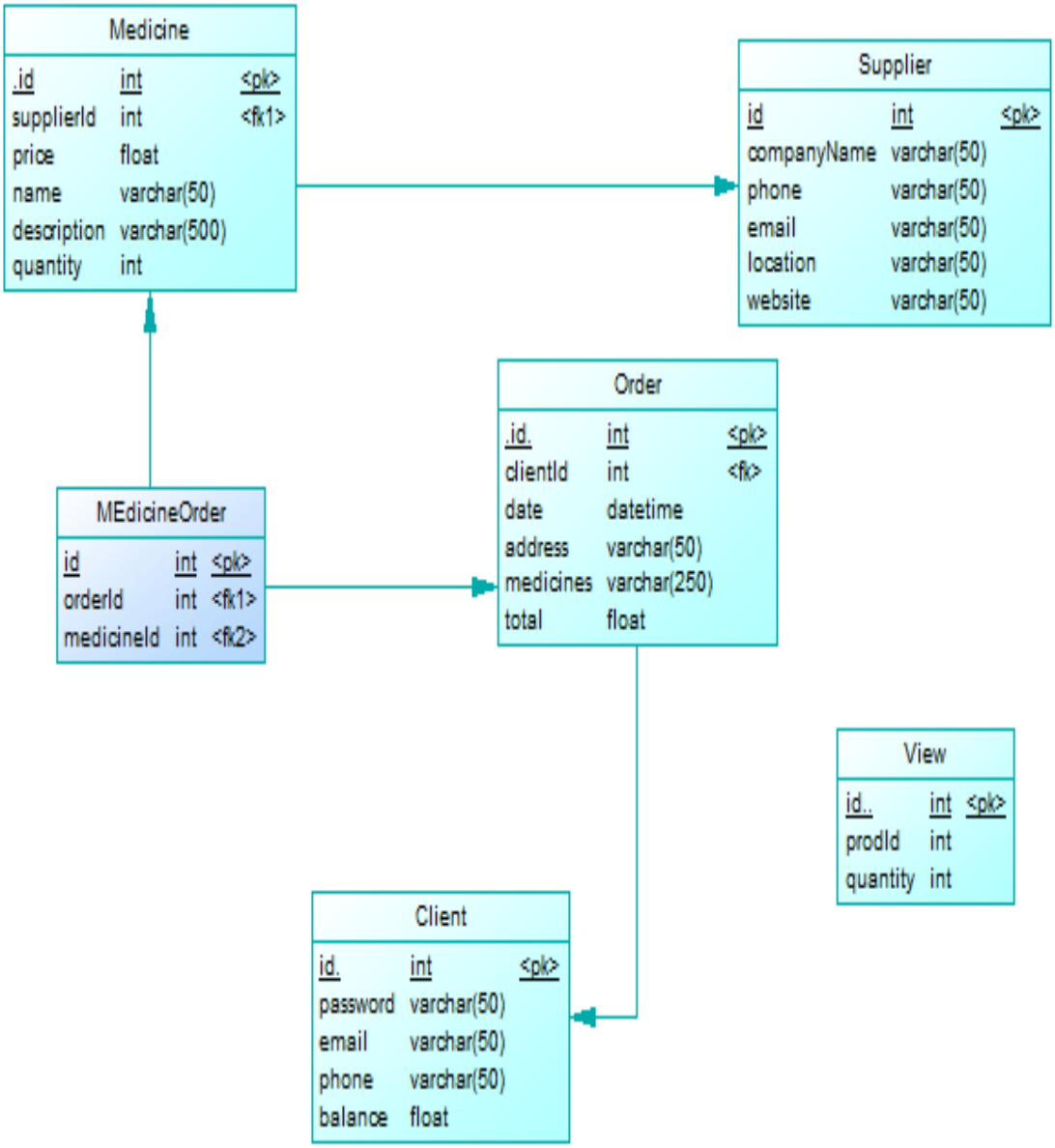


# Chapter 2: Database Modeling

## 2.1 – Conceptual Model:



## 2.2 – Physical Model:



## 2.3 – Tables Details:

| Table | Description |
|---|---|
| Client | This table is used to store information about clients including their ids, emails, passwords, phone number and balance in order to help in login and make order processes. |
| Medicine | This table is used to store information about medical products including their ids, suppliers, name, price, description and quantity in order to help in display and make order processes. |
| Order | This table is used to store information about when the order was made and by whom and what are the line items found in this order, in addition to the address of the client and bill. This table helps in generating reports process. |
| Supplier | This table is used to store information about suppliers |
| MedicineOrder | This table is the result of many to many relationship between Medicine and Order where both medicine and order id are stored. |
| View | This table is considered as a simple view. It was created in order to help in updating medicine's quantity whenever an order is placed. |

## 2.4 – Relationships and multiplicity:

- Each client can make several orders. An order is made by only one client. Therefore, there is a one-to-many relationship between ‘Client’ and ‘Order’ tables. This is translated in physical model by a foreign key in ‘Order’ table.
- A medicine has a specific supplier. Supplier can supply many medicines. Therefore, there is one-to-many relationship between ‘Medicine’ and ‘Supplier’ tables. This is translated in physical model by a foreign key in ‘Medicine’ table.
- An order can contain several products. A specific type of medicine can be ordered many times. Therefore, there is a many-to-many relationship between ‘Medicine’ and ‘Order’ tables. This is translated in physical model by a new table ‘MedicineOrder’ that has foreign keys to both tables.

## 2.5 – Stored Procedures:

### 2.5.1 – ‘usp_addOrder’:

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_addOrder]    Script Date:
2/12/2018 12:15:11 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[usp_addOrder]
@id int,@address nvarchar(50),@medicines nvarchar(250),@total float
as
begin
insert into [Order] values(@id,GETDATE(),@address,@medicines,@total);
end
```

This procedure was made to add an order to table ‘Order’. It takes as parameters : client’s id, client’s address, list of bought products and total bill.

### 2.5.2 – ‘usp_addProduct’:

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_addProduct]    Script
Date: 2/12/2018 12:21:36 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [dbo].[usp_addProduct]
@supplier nvarchar(50),
@price float,
@quantity int,
@name nvarchar(50),
@desc nvarchar(50)
AS
declare @id int;
BEGIN
set @id=(select id From Supplier where companyName=@supplier);
insert into Medicine values(@id,@price,@name,@desc,@quantity);
END
```

This procedure was made to add a product to table ‘Product’. It takes as parameters supplier’s company name, price of product, quantity of product, name of product and product’s description.

### 2.5.3 – ‘usp_addSupplier’:

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_addSupplier]    Script
Date: 2/12/2018 12:24:21 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
ALTER procedure [dbo].[usp_addSupplier]
@name nvarchar(50),
@email nvarchar(50),
@location nvarchar(50),
@website nvarchar(50),
@phone nvarchar(50)
AS
BEGIN
insert into Supplier values(@name,@phone,@email,@location,@website)
END
```

This procedure was made to add a supplier to table 'Supplier'. It takes as parameters supplier's name, email, location, website and phone number.

### 2.5.4– 'usp_addToTable':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_addToTable]    Script
Date: 2/12/2018 12:28:31 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[usp_addToTable]
@prodId int,@quantity int
AS
BEGIN
insert into [View] values(@prodId,@quantity);
END
```

This procedure was made to add product id and quantity ordered to table 'View' in order to help us update a specific product's quantity.

### 2.5.5– 'usp_addUser':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_addUser]    Script Date:
2/12/2018 12:31:35 AM ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [dbo].[usp_addUser]
@email     nvarchar(50),
@pass nvarchar(50),
@phone nvarchar(50),
@balance FLOAT
AS
BEGIN
insert into Client values(@pass,@email,@phone,@balance)
END
```

This procedure was made to add a user to table ‘Client’ by taking client’s email, password, phone number and balance.

### 2.5.6– ‘usp_checkQuantity’:

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_checkQuantity]    Script
Date: 2/12/2018 12:35:50 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[usp_checkQuantity]
@quantity int,@prodId int
AS
declare @totalQuan int
BEGIN
set @totalQuan=(select quantity from Medicine where id=@prodId);
if(@quantity<=@totalQuan)
      return 1;
else
      return -1;
END
```

This procedure was made to check the quantity of a specific medicine by taking its id. The purpose behind that is to display error message in case the client ordered more than available quantity.

### 2.5.7– 'usp_checkUser':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_checkUser]    Script Date:
2/12/2018 12:38:05 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [dbo].[usp_checkUser]
@email NVARCHAR(50)
AS
BEGIN
SELECT * FROM [client] where email = @email;
END
```

This procedure was made to check if a user is found in database by taking his email. The main purpose behind that is to check user in case of login, or to display error message in case of registration if email already exists.

### 2.5.8 – 'usp_deleteSupp':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_deleteSupp]    Script
Date: 2/12/2018 12:41:05 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[usp_deleteSupp]
@supplierId INT
AS
```

```
BEGIN
DELETE FROM Medicine WHERE supplierId=@supplierId;
DELETE FROM Supplier WHERE id=@supplierId;
END
```

This procedure was made to delete all medicines supplied by the supplier we want to delete and then delete the supplier. It takes supplier's id as parameter.

### 2.5.9 – 'usp_emptyTableView':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_emptyTableView]    Script
Date: 2/12/2018 12:46:23 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[usp_emptyTableView]
AS
BEGIN
DELETE FROM [View];
END
```

This procedure was made to empty table view. It is used by a trigger that we will talk about later.

### 2.5.10 – 'usp_totalOrders':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_totalOrders]    Script
Date: 2/12/2018 12:48:26 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[usp_totalOrders]
AS
declare @count int
BEGIN
```

```
set @count=(select count(id) from [Order]);
return @count;
END
```

This procedure was made to help us generate reports by getting to total number of orders in 'Order' table.

### 2.5.11 – 'usp_viewMedicine':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_viewMedicine]    Script
Date: 2/12/2018 12:50:18 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [dbo].[usp_viewMedicine]
AS
BEGIN
SELECT * FROM Medicine WITH (Nolock, index = IX_Medicine)
END
```

This procedure was made to help us display all medicines in database.

### 2.5.12 – 'usp_viewSuppliers':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_viewSuppliers]    Script
Date: 2/12/2018 12:52:16 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [dbo].[usp_viewSuppliers]
AS
BEGIN
SELECT * FROM Supplier WITH (Nolock, index = IX_Supplier)
END
```

This procedure was made to help us display all suppliers to the admin.

### 2.5.13 – 'usp_viewUsers':

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_viewUsers]    Script Date:
2/12/2018 12:53:55 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [dbo].[usp_viewUsers]
AS
BEGIN
SELECT * FROM Client WITH (Nolock, index = IX_Client)
END
```

This procedure was made to help us display all users to the admin.

---

### 2.6 – Triggers:

```
CREATE TRIGGER trgAfterInsert ON [Order]
FOR INSERT
AS
DECLARE @orderId INT,@id INT,@quantity INT;
SELECT @orderId=i.id FROM INSERTED as i;
declare cur_emp cursor static for
select prodId,quantity from [View];
open cur_emp
if(@@CURSOR_ROWS>0)
begin
FETCH NEXT from cur_emp
into @id,@quantity;
while(@@fetch_status=0)
begin
insert into MedicineOrder values(@orderId,@id);
update Medicine set quantity=quantity-@quantity where id=@id;
fetch next from cur_emp into @id,@quantity;
end
end
exec usp_emptyTableView;
```

```
close cur_emp
deallocate cur_emp
```

This trigger was made on table ‘Order’ inorder to insert needed data in table ‘MedicineOrder’ and update medicine’s quantity whenever a record is inserted in table ‘Order’.

## 2.7 – Transactions:

```
USE [Pharmacy]
GO
/****** Object:  StoredProcedure [dbo].[usp_checkBalance]    Script
Date: 2/12/2018 1:01:00 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[usp_checkBalance]
@email NVARCHAR(50),@amount float
AS
declare @id int,@balance float,@return int;
BEGIN
set @id=(SELECT id FROM Client WHERE email=@email);
set @balance=(SELECT balance from Client where id=@id);
BEGIN TRANSACTION
      IF(@balance>=@amount)
            BEGIN
            update Client set balance=balance-@amount where id=@id;
            commit;
            set @return=1;
            return @return;
            end
      ELSE
            begin
            rollback;
            set @return=-1;
            return @return;
            end
END
```

This transaction is nested in a procedure that takes client's email and amount of money he must pay if he added more products. The purpose behind it is to disable the user from adding more products to his cart if the amount to pay was greater than his balance, otherwise update his balance by subtracting it from the amount to pay.

## 2.8 – Indices:

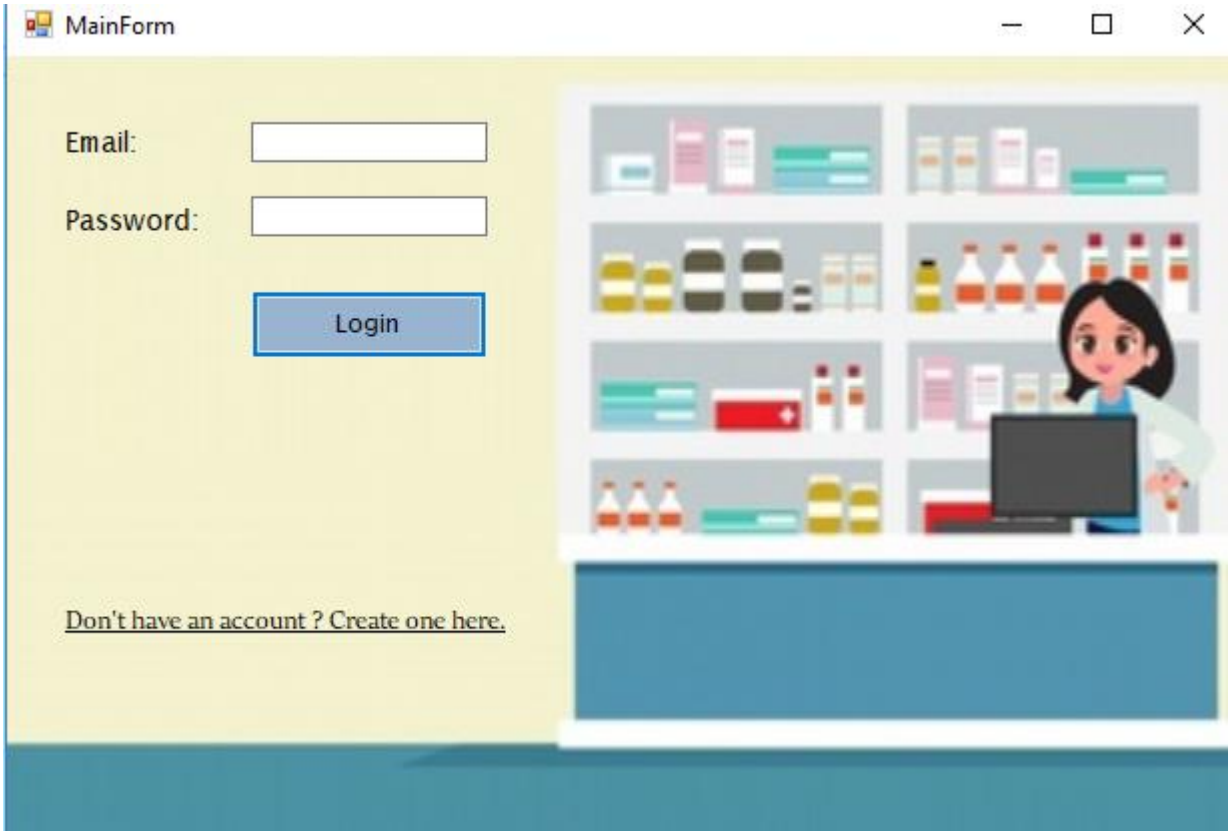
```
CREATE INDEX IX_Medicine ON Medicine ([name] ASC);
CREATE INDEX IX_Supplier ON Supplier ([companyName] ASC);
CREATE INDEX IX_Client ON Client ([email] ASC);
```

These indices were made to help the user find what he wants in a shorter time.

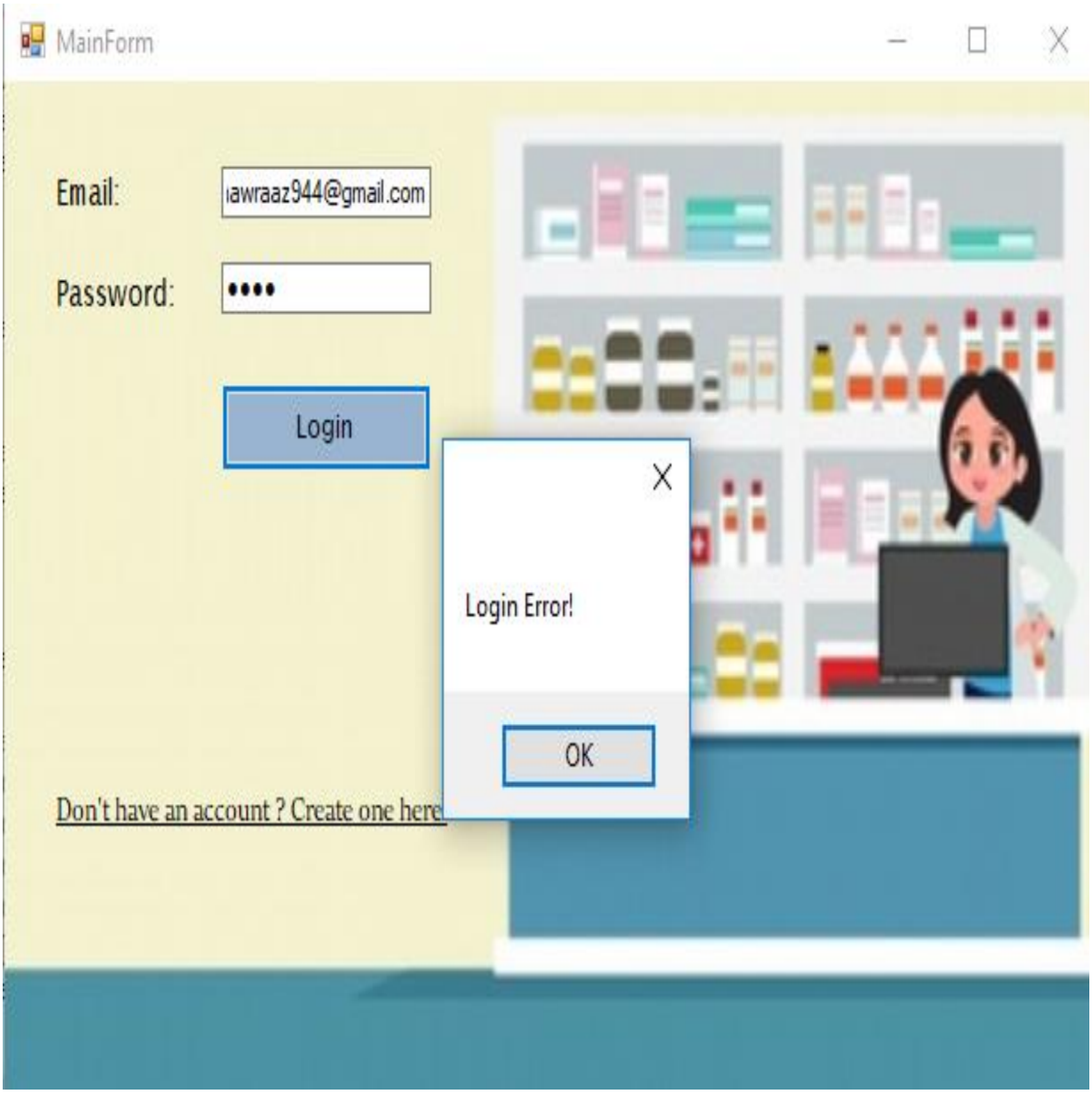
## Chapter 3: Services and Interface modeling:

In this chapter we will show you different forms, and talk about the specialty of each form.

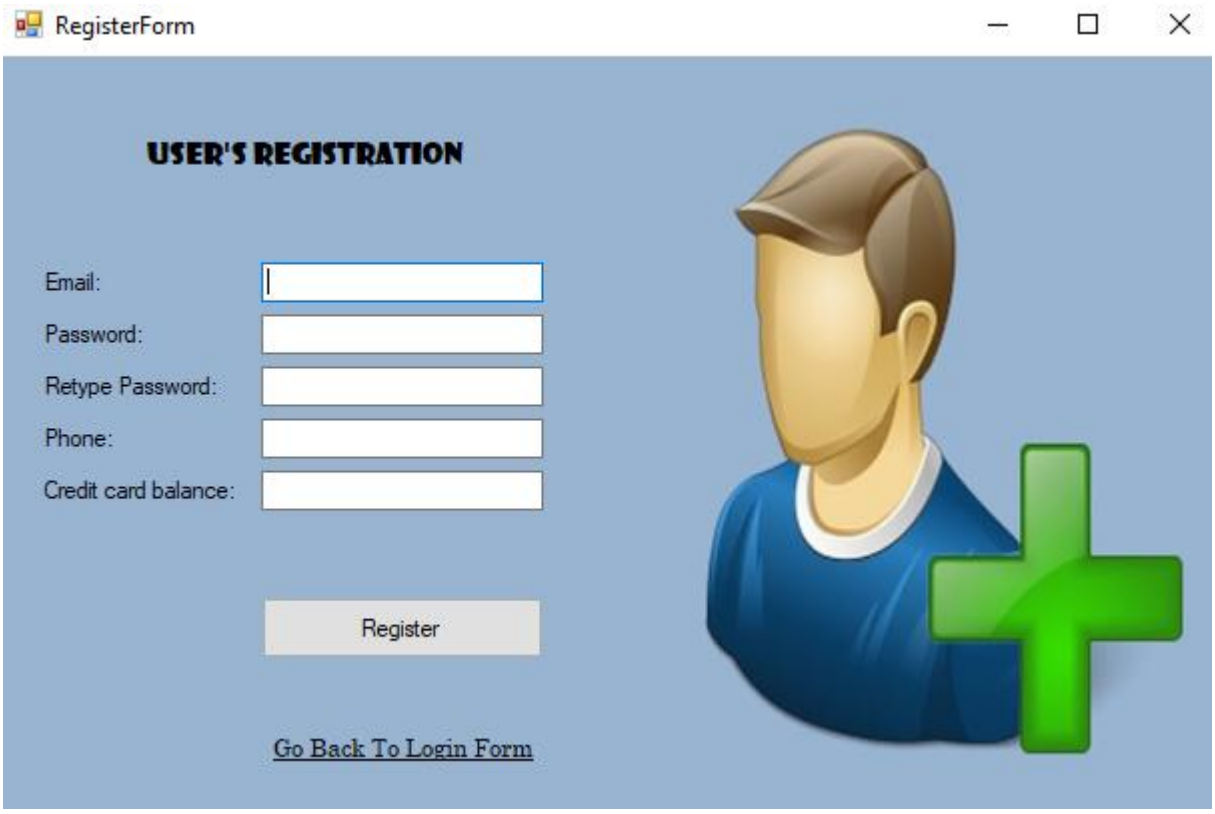
This is the main form that will appear when we run the project.




This form takes as input user's email and password and display error in case the user was not found.



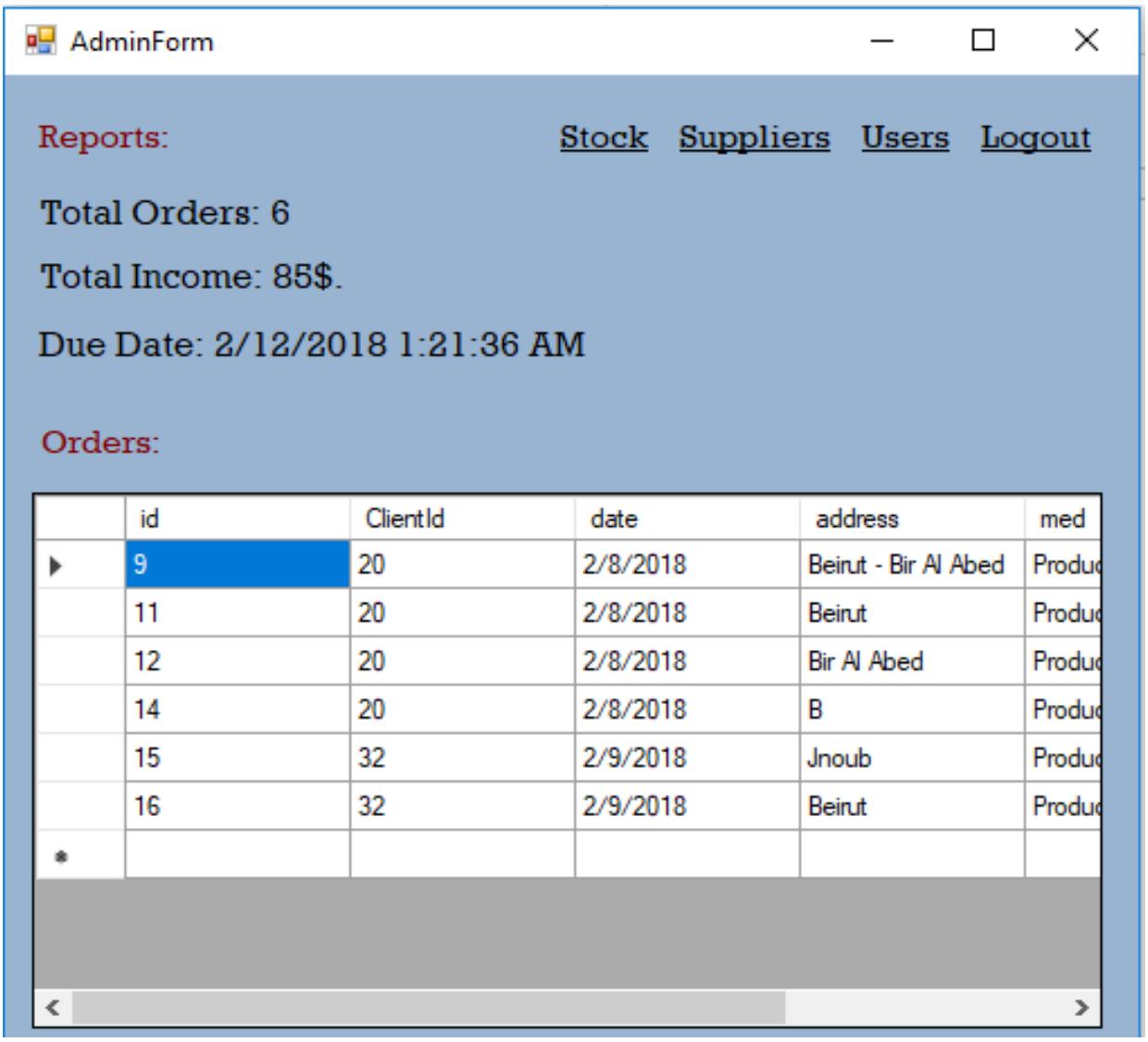

In case the user doesn’t have an account he can join us by pressing the link at the bottom of the form.




This is the registration form. This from displays an error in case the email already exists. Regex expressions were used here to make our application more secure.

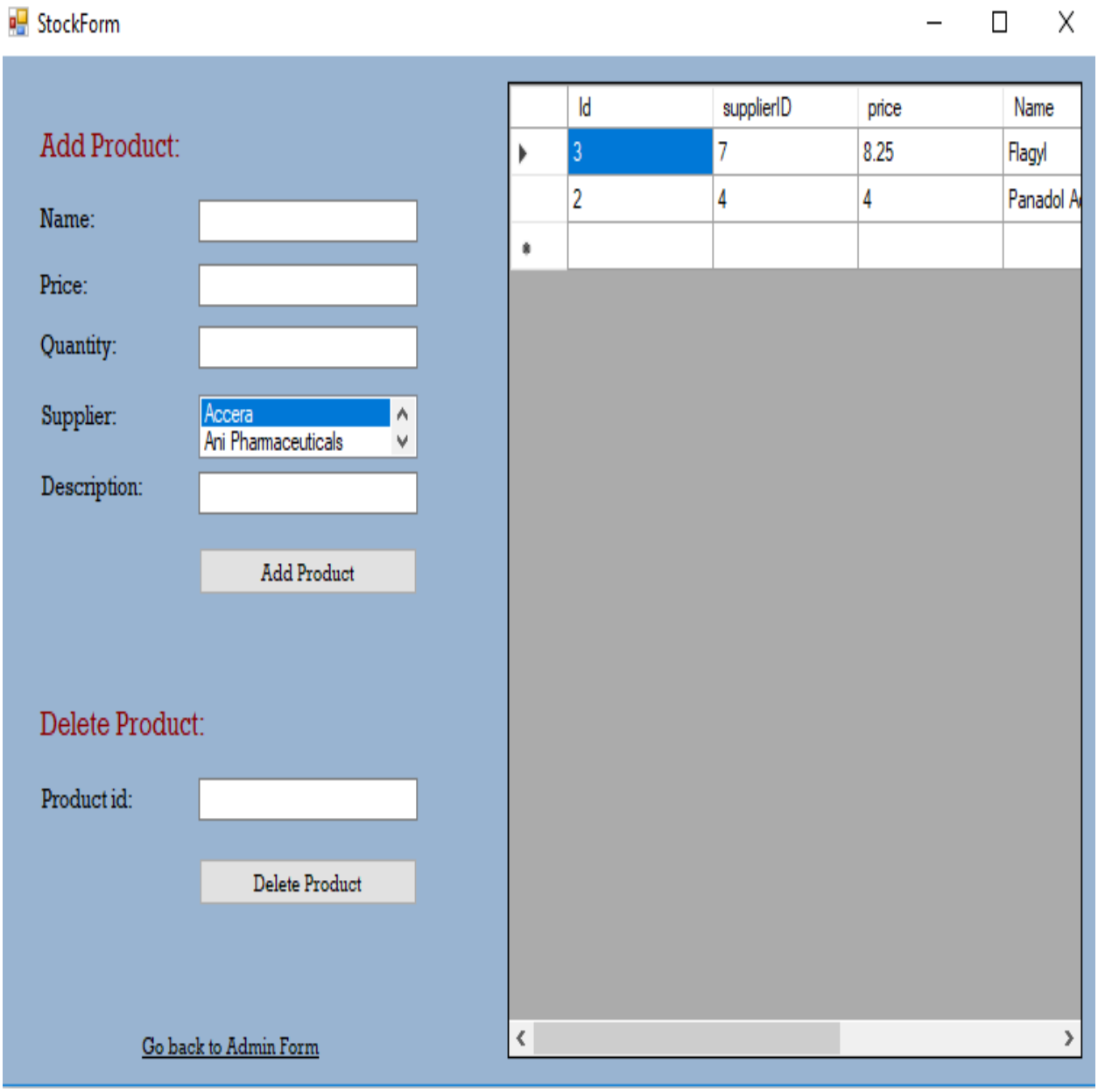
If the user is already registered, we have two cases. Case where the user is the admin and case where the user is the client.

If the admin signs in the following form will appear:



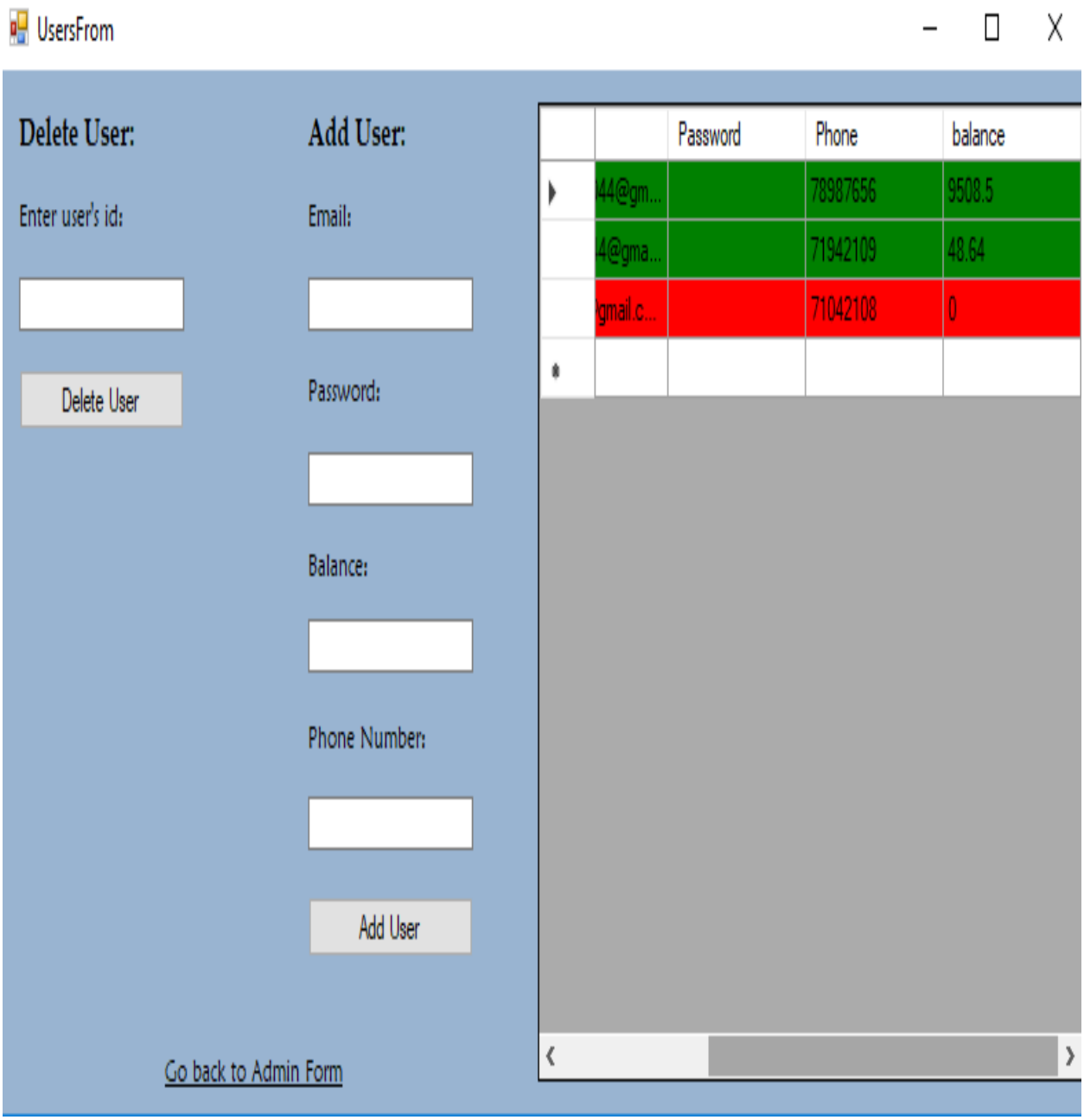

As we see at the top of the form, reports are generated. We also have several links and each link takes the admin to a different form.

For instance, if the admin pressed stock link, the following form will appear:

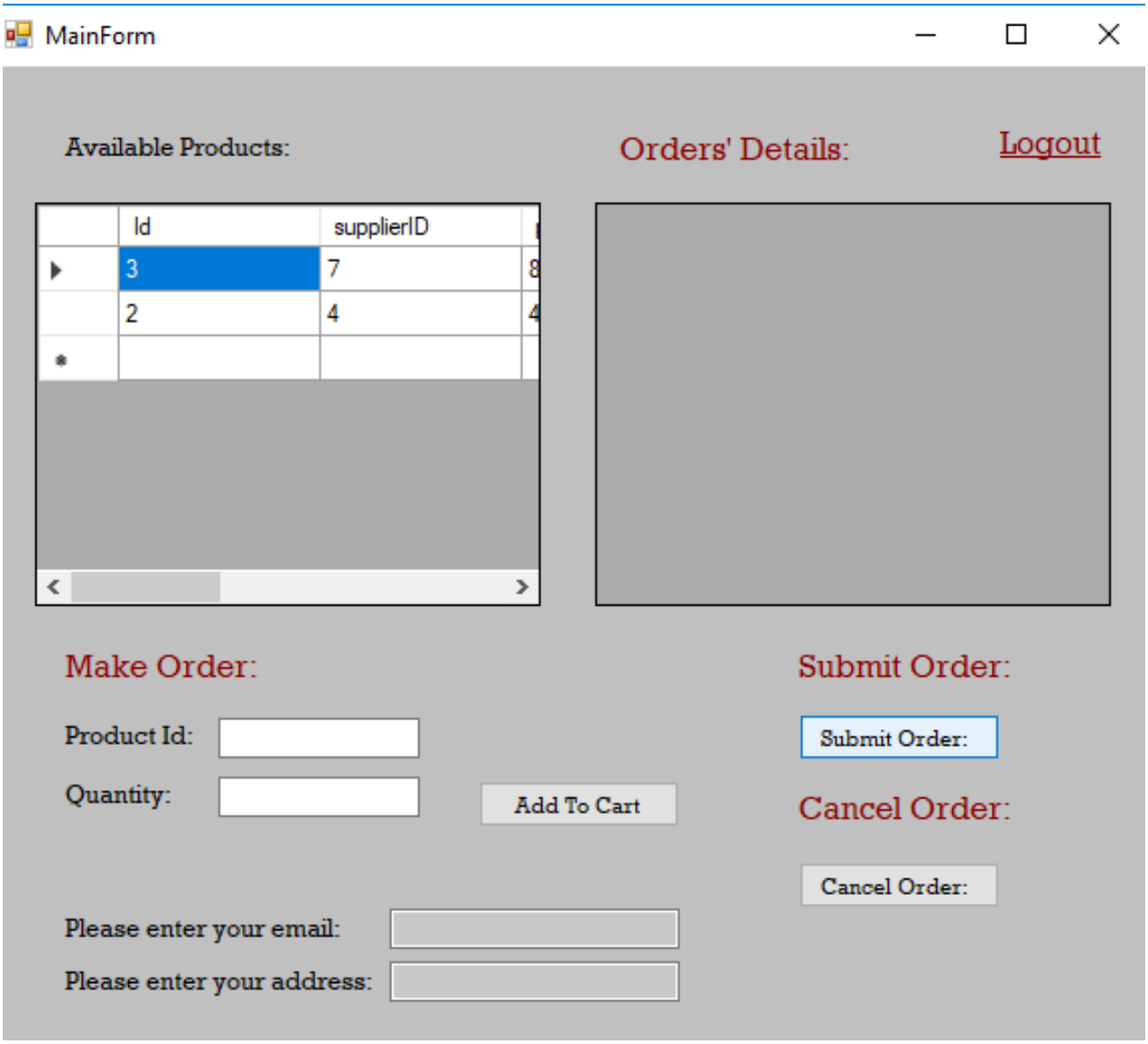



In this form, all medicines are displayed and the admin can add or delete a specific product by entering necessary information.

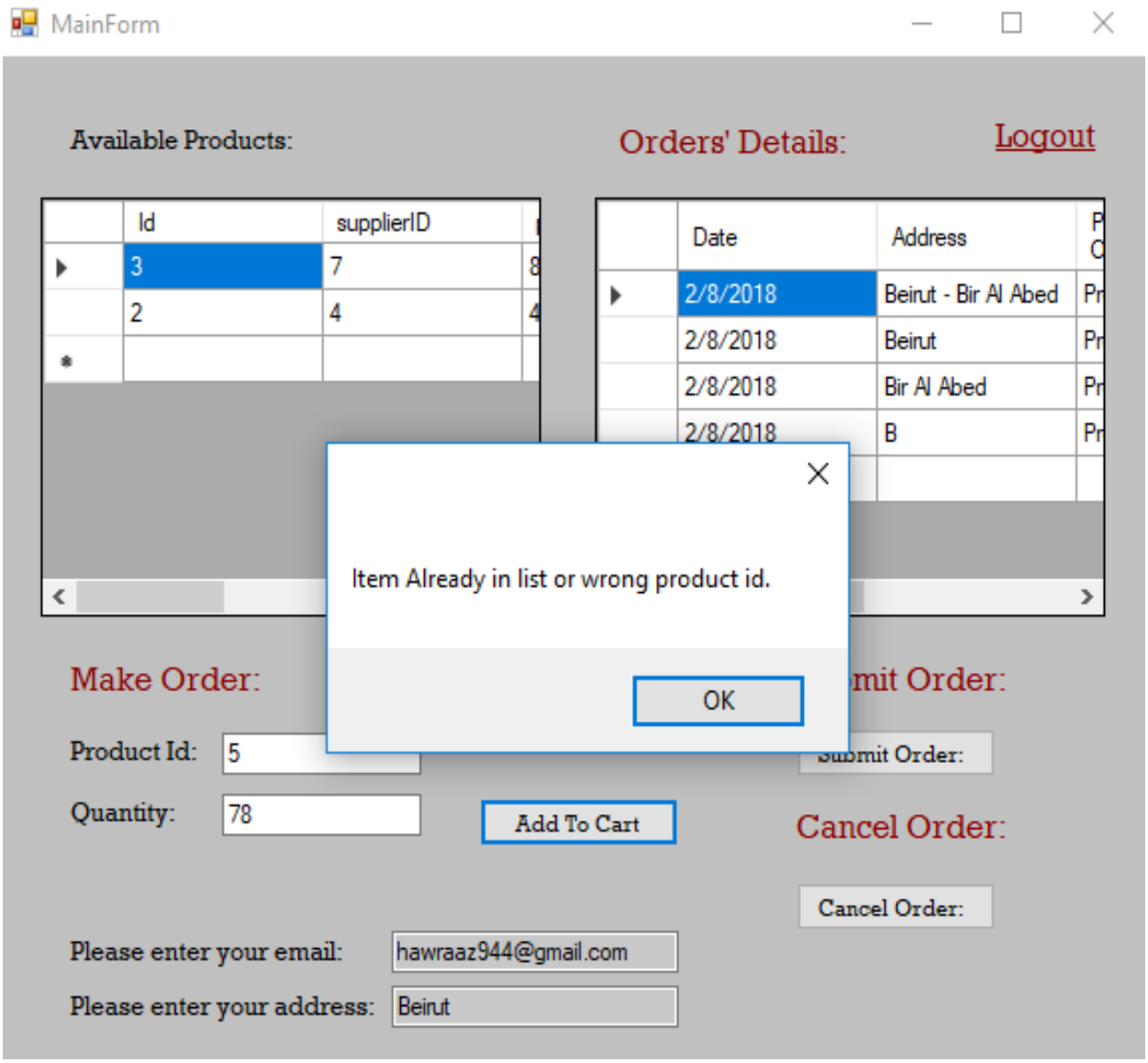
If the admin pressed users link the following form will appear. Note that users with zero balance are marked by red. This form also enables the admin to add or delete users by entering necessary information.



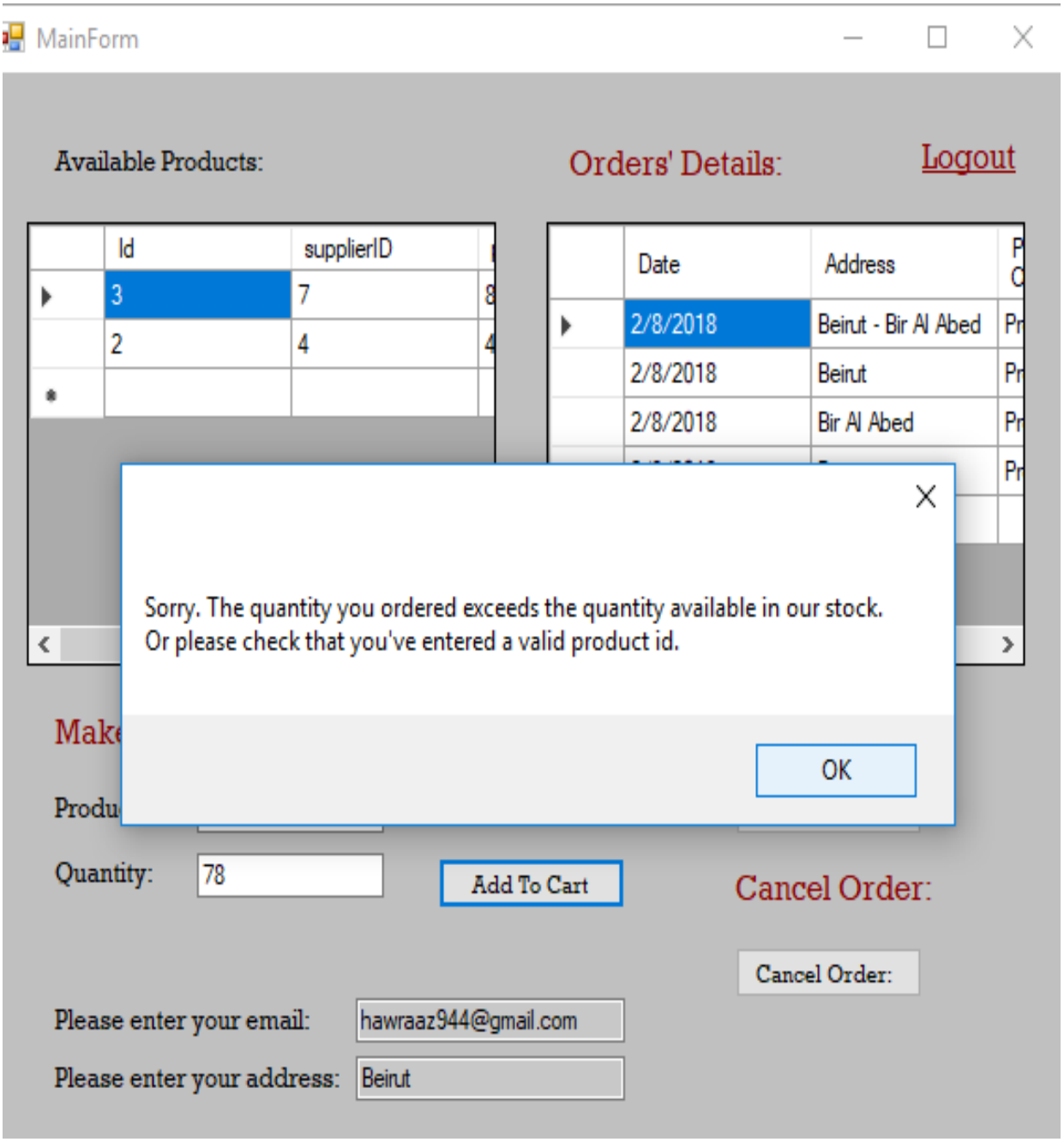
Now, in case a client signs in, the following form appears. Note that the grid view is initially empty.



Error messages are displayed in case the client entered wrong product id, ordered a quantity which is greater than available quantity, tried to add an item which is already in cart, his balance is less than the amount he must pay or in case he tried to cancel an order which is already placed.



Note that when the client enters his email, all orders he've orders previously appear in the grid view.



## Chapter 4: Implementation and Testing

Here are some functions we used to implement our project.

```
public Client CheckLogin2(string email)
        {
            var query = "usp_checkUser";
            // this will create dynamic object, wll be used to store the paramters that
corresponds
            // to the stored proc
            DynamicParameters param = new DynamicParameters();

            // the parameters to add
            param.Add("@email", email);

            // execute the query
            var result = ExecuteProc(query,param);

            Client client = new Client();

            // loop on the query result and bind the data, to the object(s)
            foreach (var row in result)
            {
                client.Id = row.id; // row.id means that we have id column
                                    // returned by the query, and we are taking its value
                client.Name = row.email;
                client.Password = row.password;
            }
            return client;
        }
```

This function returns an object of type Client. This function is found in Repositories.

```
public bool CheckLogin(string email, string password)
        {
            // get the client from database that have the "email" variable value
            var client = _clientRepository.CheckLogin2(email);
            return password == client.Password;

        }
```

This function calls the function found in Repository and return true or false based on if the client is found or not.

```
private void RegButton_Click(object sender, EventArgs e)
        {
            string email = txtEmail.Text;
            string pass = txtPass.Text;
            string phone = txtPhone.Text;
            string balance = txtCredit.Text;
            string password2 = pass2.Text;

            if (email == "" || pass == "" || phone == "" || balance == "")
            {
                MessageBox.Show("Please fill all information needed.");
            }


            else
            {

                if (password2 != pass)
                {
                    MessageBox.Show("Password don't match.");
                }

                else
                {
                    Match matchPhone = Regex.Match(phone, @"(03|70|71|76|78|05|01){1}[0-
9]{6}");
                    Match matchEmail = Regex.Match(email, @"[a-zA-Z]+[a-zA-Z0-9]*([-
_.]?\w)*@(gmail)\.(com)");
                    Match matchBalance = Regex.Match(balance, @"[0-9]+([.][1-9]+)?");
                    if (matchPhone.Success && matchEmail.Success && matchBalance.Success)
                    {

                        if (_clientService.AddUser(email, pass, phone, balance))
                            MessageBox.Show("You've been registered and you can now use
our system");
                        else
                            MessageBox.Show("Email already exist!");
                    }

                    else
                    {
                        if (!matchPhone.Success)
                            MessageBox.Show("The phone number should be of form : \n
03|70|71|76|78|05|01 followed by 6 digits");
                        else
                            if (!matchEmail.Success)
                            MessageBox.Show("Enter a valid email.");
                            else
                            MessageBox.Show("Enter a valid balance.");
                    }
                }
            }
        }
```

This function is called when Register button in register form is clicked. It checks if the textboxes are empty. If yes, error message is displayed. It then checks that the two passwords entered by the user match. If yes, it checks that the user enter the information in correct form using regex expressions.

Testing:

The features of the project were tested thoroughly.

## Chapter 5: Conclusion

This mini application was a good practice for us. We almost used all the techniques we learned triggers, transactions, procedures and more to make our project as efficient as possible. We worked with real life data models which was a good push for us in this domain. Future considerations are also taken into consideration as we can make our application more advanced and wide by adding more features to it.

Some of future considerations: use Bit Coin as a payment system and enable the client to search for products and more yet to come..