MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 400

Summer Practice Report

Name of Student :	Umut BAYBECE
ID Number :	2385201
Name of Company :	Adopen Plastik ve İnşaat Sanayi A.Ş.
Project Title :	Inventory Management System
Date of Submission :	24.12.2022

ABSTRACT

This report delivers the details of the assignments and the solutions I have accomplished during my internship at Adopen Plastik ve İnşaat Sanayi A.Ş. Utilized methods and approaches were primarily planned to improve my programming skills and gain a professional working life background. During this term, I have practiced entity framework implementations, ADO.NET(ActiveX Data Object) implementations, PostgreSQL object-relational database system implementations, CRUD operations, and dynamic product code generation mechanisms with SKU generators and windows form application structures. To expand, I created a stock management desktop application using the ADO.NET windows forms and connected it with the PostgreSQL object-relational database system. While completing my stock management program, I created an admin page to allow the admin to change the structure and elements of the departmental pages. Therefore, I saw the practical aspects of two critical concepts, software testing, and design. In addition, I have observed a collaborative working environment and permission-granting mechanisms for high-level security mechanisms.

In conclusion, my internship helped me in various ways beyond simply getting to use software testing and development; it also guided me to appreciate the value of a collaborative and secure work environment.

TABLE OF CONTENTS

LIST OF FIGURES

1. ADOPEN PLASTİK VE İNŞAAT SANAYİ A.Ş.

In 1997, Adopen, a producer of PVC profiles on a global scale, was founded in Antalya. The company is built on top of Çağlar Plastics Ind. Çağlar Plastics Ind. was one of Turkey's first window manufacturers.

Adopen's mission is to become the "world's largest PVC profile producer" by offering its clients the highest-quality goods and most effective assistance. Therefore, Adopen wants to deliver high-tech goods and top-notch solutions to the globe with complete awareness of all cultures.

Adopen is a part of many other installations of ADO Group. It is building is located at Organize Sanayi Bölgesi 2. Kısım Döşemealtı, Antalya, Turkey. It has several departments: the Department of Finance, Information Technologies, Accounting, Human Resources, Production, Importation, Exportation, and Management. It has over a hundred workers, eight of whom work in the Information Technologies Department.

The products made by the company include floor materials, PVC windows, and doors. They produce all relative accessories of these components in their side company, Accado. The company also provides technique services.

2. INTRODUCTION

I have decided to complete my second compulsory internship in ADOPEN since they have offered many benefits for my career. They follow worldwide standards for production facilities and implement advanced database protocols to manage data. Also, their experienced engineers work in the R&D center of the main factory. They use permission-granting mechanisms via their databases to ensure the security of their data flow between departments.

Furthermore, they use the quality management systems of Netsis to provide several user interfaces for their vast data distribution. Since they have several facilities and each facility produces different products, they use advanced data cataloging systems. They also store every error report and renew processes for later uses. All these data management protocols appealed to me since I want to become a data analyst in the future.

My supervisor Ibrahim Kuş was an experienced computer engineer. He was responsible for solving error requests related to the company's database and data flow. He guided me in my

projects and decided on the project complexity level. He gave me some tutorials related to entity frameworks and WinForms application structures. With his guidance, I could improve on these topics and started to create my main project.

My main task was creating a stock management system for my company as a WinForms desktop application connected to a database. To achieve this, I utilized PostgreSQL and ADO.NET framework. I also created several functions in C# for CRUD operations and stock code generators.

In conclusion, during my internship, I gained practical knowledge in numerous aspects of the working life of a computer engineer. It helped me to experience professional working life with disciplined and diligent coworkers.

3. PROBLEM STATEMENT

My supervisor asked me to work on a specific problem related to the company's stock management system.

Firstly, my supervisor asked me to research entity frameworks and their usage with databases. I was given several exercises to practice my knowledge and gain experience on these concepts. My tutorials also aimed to make me familiar with the C# language and WinForms. I have observed their database structure and data flow, an permission granting mechanism.

Secondly, I am asked to develop a login page for my application, allowing workers to reach different data for their depertmants. Aditionally, I am asked to create an admin page and admin login privileges. My supervisor asked me to store all data for dropdowns and all selectable information of products in a database that is controlled by only the admin.

Finally, my supervisor requested that I create two pages for different company departments that allow workers to enter a new product record, update existing reports, or delete them. I am asked to develop algorithms to produce unique stock codes for each product record. For these reasons, I applied basic CRUID operations. My supervisor guided me through this term and suggested several solutions to my problems.

4. DEVELOPMENT TOOLS' ARCHITECTURES AND SOLUTION IMPLEMENTATIONS

I will describe how I utilized tools to create my project in this report section as a part of the solution, along with their structural and behavioral aspects. In line with the company security regulations, real data and data types are not shared in this report. All the data and data types are representatives of the actual ones.

4.1. POSTGRESQL

PostgreSQL is a sufficiently stable open-source object-relational database system. It allows for maintaining and managing complex data workloads. It supports components of ACID (atomicity, consistency, isolation, durability) and preserves data integrity. My company was working with Microsoft SQL Server Management Studio, but I asked them to work with PostgreSQL to learn other technology, and they allowed me to work with it.

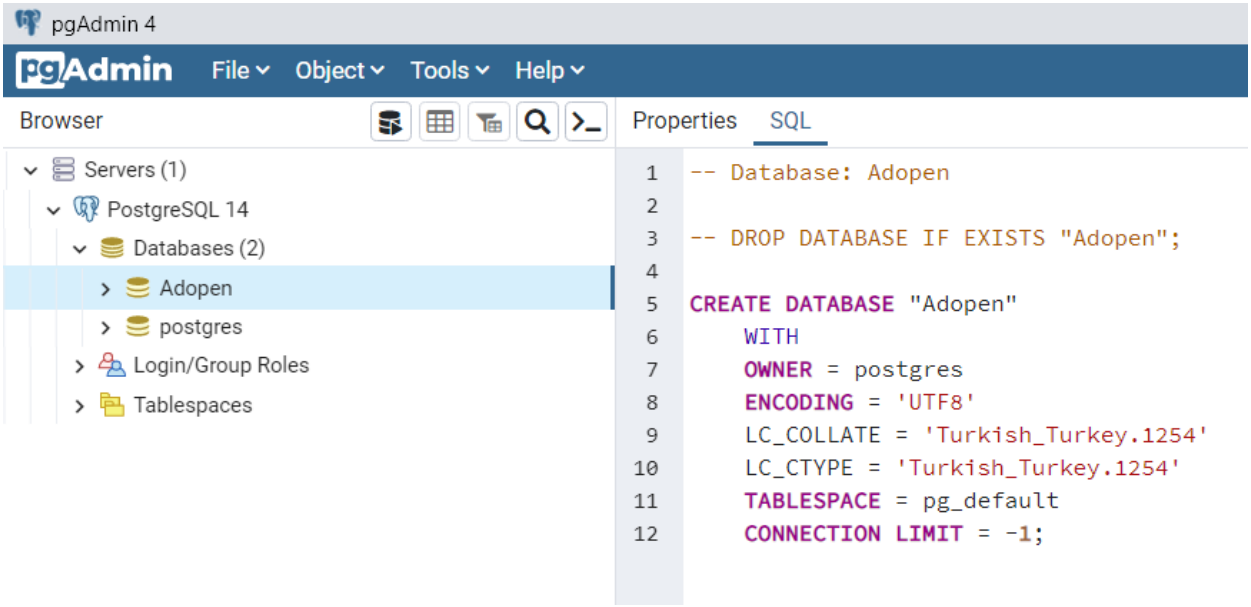


Figure 1: PostgreSQL Database

As you can see in the Figure 1, I have created Adopen database for my application. After that I have decided to create several tables based on my Entity Relationship (ER) Diagram.

4.1.1. DATABASE DESIGN

This database is designed to store information about various types of products, including doors, floors, and windows. It includes four tables: Products, Doors, Floors, and Windows.

The Products table serves as the parent table for the other three tables, and it contains fields for storing general information about the products, such as the product status, name, color, type, and size. The Products table is designed to store information that is common to all products, regardless of their type.

The Doors, Floors, and Windows tables are child tables that are related to the Products table via foreign key constraints. These tables contain fields that store specific information about the products they represent, such as the door sill and doorknob for doors, or the parquet pattern and thickness for floors. Each record in these child tables is linked to a corresponding record in the Products table, which allows for easy querying and retrieval of information about the products.

This database is designed to store and organize information about a variety of products, and to allow for easy querying and retrieval of this information. It is structured in such a way that it is easy to find and retrieve information about specific products, and to see how different types of products are related to one another.

Properties SQL

```sql
-- Creates the Products table
CREATE TABLE Products (
  id INTEGER PRIMARY KEY,
  productstatus VARCHAR(100),
  productname VARCHAR(100),
  productcolor VARCHAR(100),
  producttype VARCHAR(100),
  productsize VARCHAR(100)
);

-- Create the Doors table
CREATE TABLE Doors (
  id INTEGER PRIMARY KEY,
  door_sill VARCHAR(100) NOT NULL,
  doorknob VARCHAR(100) NOT NULL,
  filling VARCHAR(100) NOT NULL,
  stockcode VARCHAR(100) NOT NULL,
  FOREIGN KEY (id) REFERENCES Products(id)
);

-- Create the Floors table
CREATE TABLE Floors (
  id INTEGER PRIMARY KEY,
  parquet_pattern VARCHAR(100) NOT NULL,
  parquet_click VARCHAR(100) NOT NULL,
  thickness VARCHAR(100) NOT NULL,
  stockcode VARCHAR(100) NOT NULL,
  FOREIGN KEY (id) REFERENCES Products(id)
);

-- Create the Windows table
CREATE TABLE Windows (
  id INTEGER PRIMARY KEY,
  frame_material VARCHAR(100) NOT NULL,
  glass_type VARCHAR(100) NOT NULL,
  opening_mechanism VARCHAR(100) NOT NULL,
  stockcode VARCHAR(100) NOT NULL,
  FOREIGN KEY (id) REFERENCES Products(id)
);
```

Figure 2: SQL Queries of The Core SQL Tables

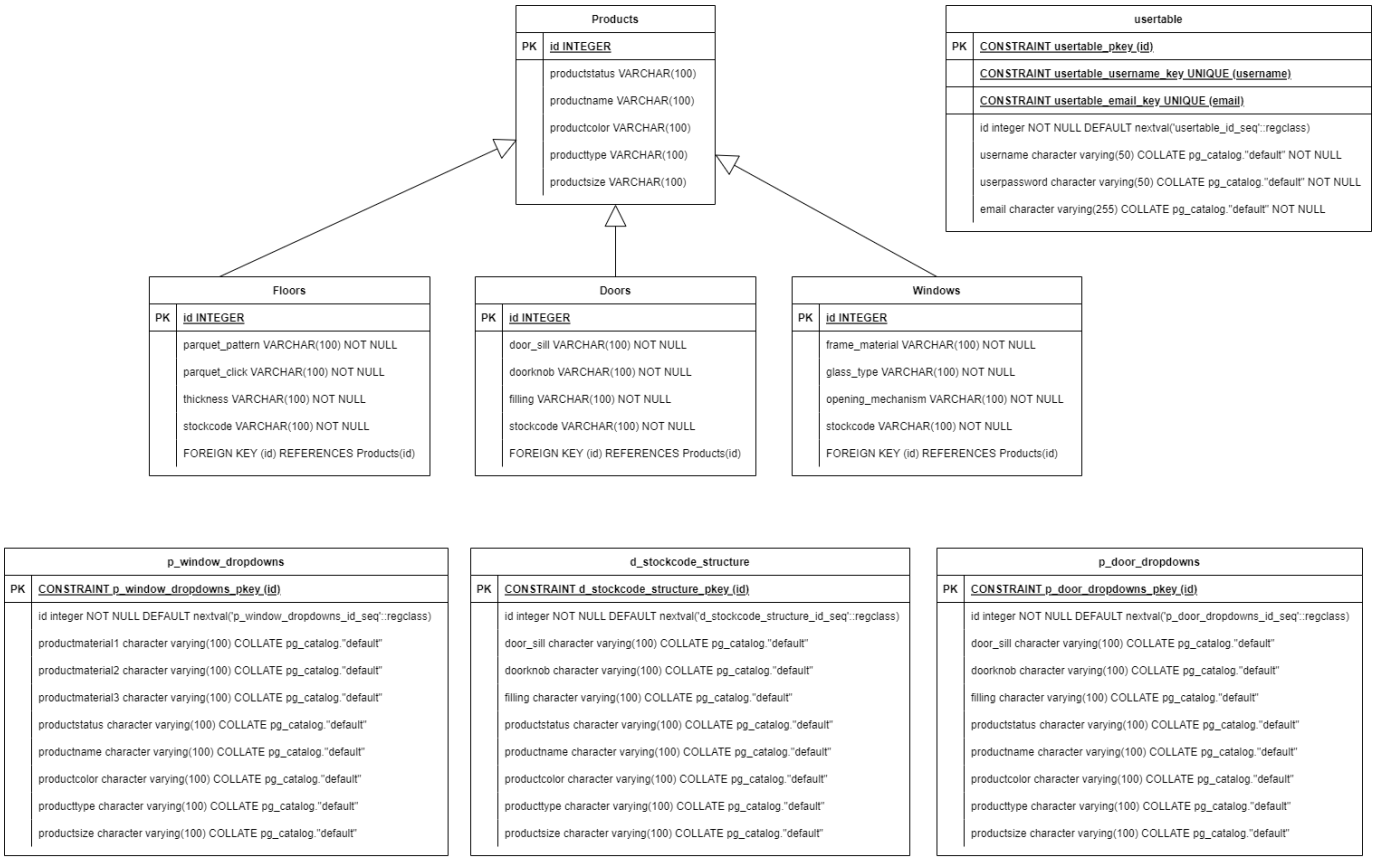


Figure 3: Complete Entity–relationship Model Diagram of The Database Tables

As shown in figure 2, SQL queries of the main product tables are created via these queries, based on the diagram given in figure 3. Furthermore, as you can see the three other database tables are created to store information on dropdown contents so that the admin can update the contents of user interfaces accordingly via the admin page.

In addition to the main product tables, the database schema also includes three other tables that are used to store dropdown content. Dropdown menus are a common UI element that allows users to select from a list of options. These tables store the options that are available in the dropdown menus, and they can be updated by an administrator via an admin page. This allows the administrator to modify the options that are available to users, which can be useful for customizing the user interface or making updates to the system.

Furthermore, the user table stores user information to allow the user to enter applications according to their department and store user details.

The d_stockcode_structure table is used to create and manage unique stock codes for the products in the database. A stock code is a unique identifier that is assigned to each product, and it is used to track and manage the inventory. The d_stockcode_structure table stores information about the structure of the stock code, including the format and any rules or requirements that need to be followed. This information can be used to ensure that new stock codes are created correctly and that they are unique within the database. It also allows the administrator to modify the structure of the stock code if needed, for example, to add or remove certain elements or to change the format.

To store data from each department separately, I created three tables. However, since the project started to be repetitive, I dropped the window table later. I made another table for the login page to store user information named as user table. Later, I created three additional tables to store dropdowns' contents to provide a control mechanism for the admin page. The structure of my database allows my program to work modularly and dynamically in line with the requirements.

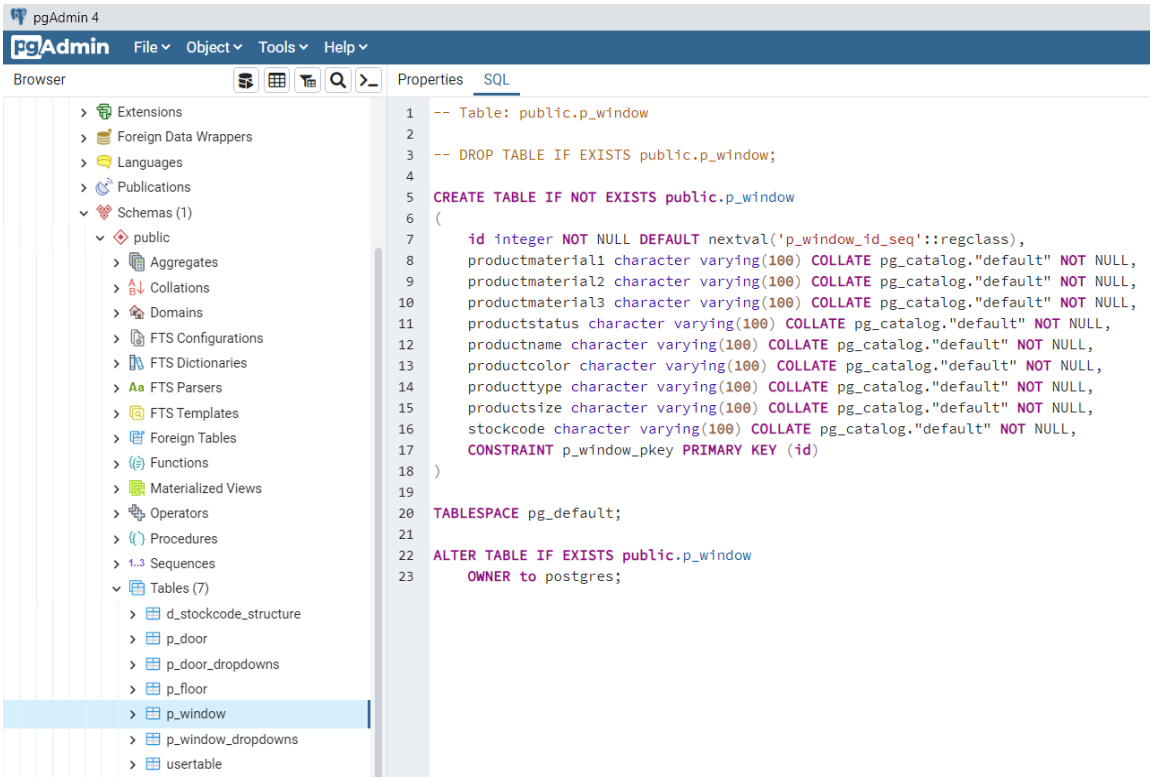


Figure 4: Database Tables Structure For Dropdown Implementations

As you can see in the figure above, I have created a variety of tables to store different kinds of information related to each department of the company. Also, I have created tables named "p_departmentname_dropdowns," allowing the admin to change the contents of the department interfaces via the admin page.

Each table contains specific product details such as “ProductStatus,” “ProductName,” “ProductColor,” “ProductType,” and so on. They have different constraints and primary keys for different types of products.

4.2. WINDOWS FORMS APP IN MICROSOFT VISUAL STUDIO WITH C#

To create a windows forms application with C#, I used Microsoft Visual Studio development environment. .NET Framework is very reliable and has ease of use in terms of coding and integration. As shown in the figure below, I have created several forms under the main application. The toolbox window is handy for navigating the different parts of the application. I have created an AdminForm for admin operations, such as changing the options served to the department users. Admin can add, update or delete products from data tables of dropdowns via the admin form. The following parts of the Login page, DoorForm and FloorForm are elaborated on in the following parts of the report.

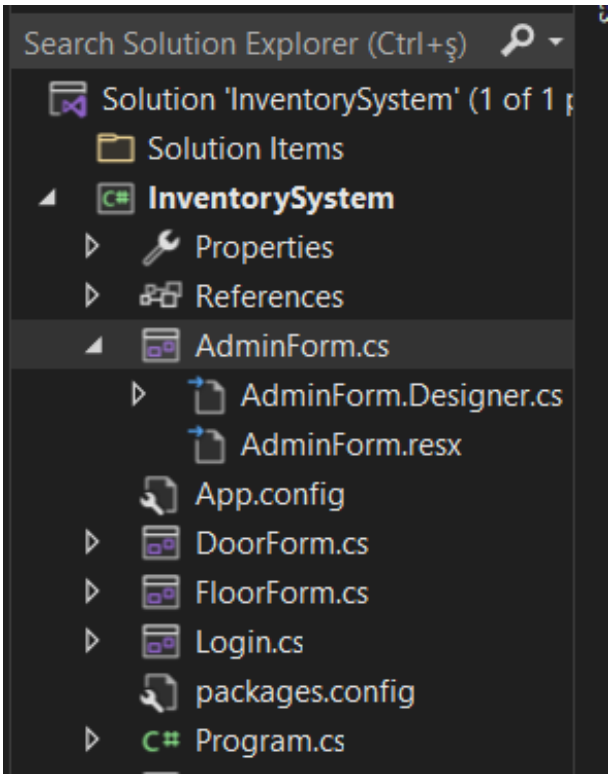


Figure 5: C# Forms

4.2.1. LOGIN PAGE

Firstly, I have created the design and interface of all the forms on the Visual Studio design environment (name_of_file.cs[design]). I started with the login form. The login form allows users to reach their departmental pages accordingly. The users enter their username and password and select the department name from the dropdown, as shown in the figures below. The login page is connected to the login data table and runs the following query to check user information. Later, I added the departmental check to the query and updated my table accordingly. Encryption details are not given in the report in line with the company contract.

```
Login.cs | Login.cs [Design] | FloorForm.cs [Design] | FloorForm.Designer.cs | DoorForm.cs | DoorForm.cs [Desi
InventorySystem                                  InventorySystem.Login
49
50          }
51
            1 reference
52          private void iconButtonLogin_Click(object sender, EventArgs e)
53          {
54              bool pass = false;
55              DataTable dt = new DataTable();
56              if (textBoxUserName.Text != "" && textBoxPassword.Text != "" && comboBoxDepartment.SelectedItem != null) {
57              string commandText = "SELECT username,userpassword FROM usertable WHERE username = @username and userpassword =@password;";
58                  conn.Open();
59                  NpgsqlCommand command = new NpgsqlCommand(commandText, conn);
60                  command.Parameters.AddWithValue("@username", textBoxUserName.Text);
61                  command.Parameters.AddWithValue("@password", textBoxPassword.Text);
62
63                  NpgsqlDataAdapter sda = new NpgsqlDataAdapter(command);
64                  sda.Fill(dt);
65                  conn.Close();
66                  if (dt.Rows.Count > 0)
67                  {
68                  if (comboBoxDepartment.SelectedIndex == 1) {
69                  WindowForm Check = new WindowForm();
70                  Check.Show();
71                  Hide();
72                  }
73                  else if (comboBoxDepartment.SelectedIndex == 0)
74                  {
75                      DoorForm Check = new DoorForm();
76                      Check.Show();
77                      Hide();
78                  }
79                  else if (comboBoxDepartment.SelectedIndex == 2)
80                  {
81                      FloorForm Check = new FloorForm();
82                      Check.Show();
83                      Hide();
84                  }
```

Figure 6: Login Page Queries

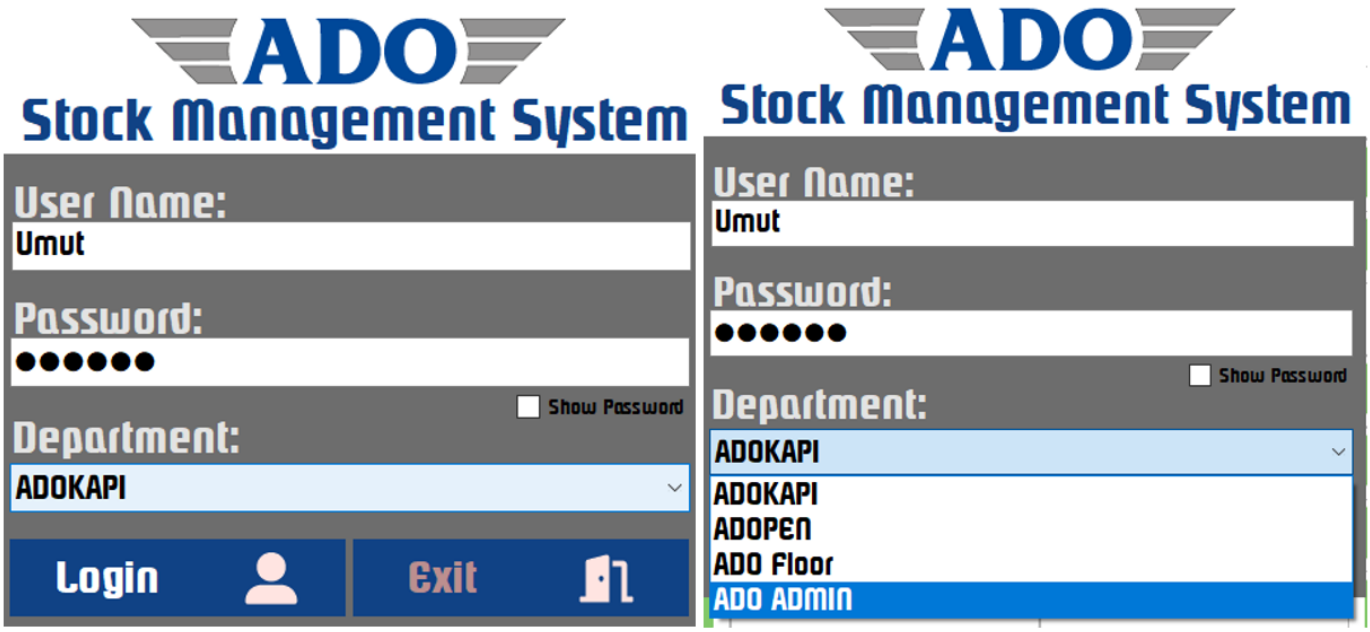
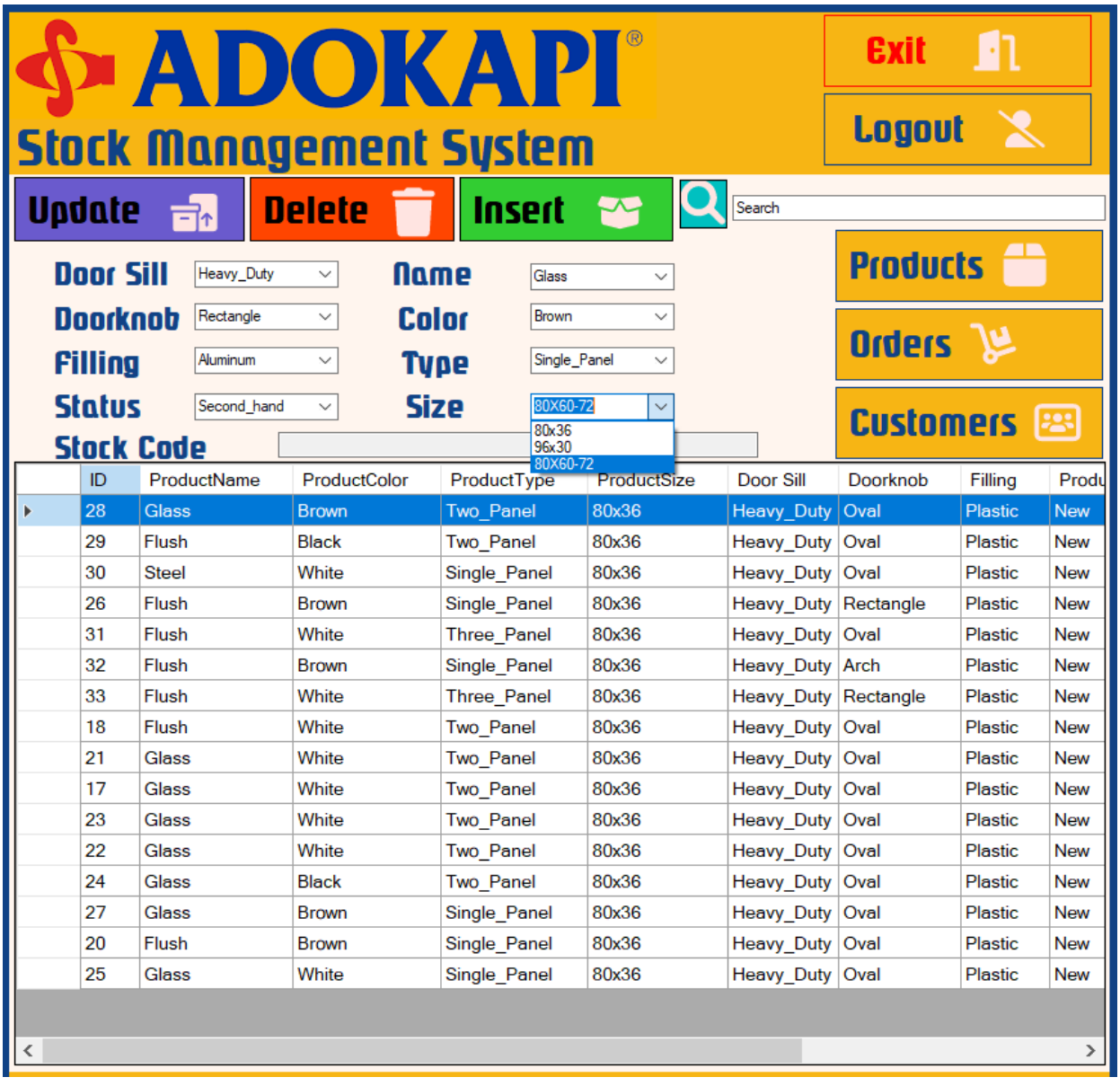


Figure 7: Login Page Design

4.2.2. DOOR AND FLOOR DEPARTMENTS WINFORMS

ADOKAPI is a department in ADO Grup companies producing doors for its customers. It offers many options to serve different style requests and needs. To achieve its abstraction in my application, I have created a table involving all the details of the specific stock. A stock has many features, such as door sill type, doorknob type, filling materials, size, etc. Also, a unique stock code is produced based on these features. Therefore, as shown in the figure below, I have implemented a grid table with CRUD operations. The table is connected to the database, and dropdowns are also connected to the admin data table, which allows the admin to change form entry options.



ID	ProductName	ProductColor	ProductType	ProductSize	Door Sill	Doorknob	Filling	Produ
28	Glass	Brown	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
29	Flush	Black	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
30	Steel	White	Single_Panel	80x36	Heavy_Duty	Oval	Plastic	New
26	Flush	Brown	Single_Panel	80x36	Heavy_Duty	Rectangle	Plastic	New
31	Flush	White	Three_Panel	80x36	Heavy_Duty	Oval	Plastic	New
32	Flush	Brown	Single_Panel	80x36	Heavy_Duty	Arch	Plastic	New
33	Flush	White	Three_Panel	80x36	Heavy_Duty	Rectangle	Plastic	New
18	Flush	White	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
21	Glass	White	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
17	Glass	White	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
23	Glass	White	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
22	Glass	White	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
24	Glass	Black	Two_Panel	80x36	Heavy_Duty	Oval	Plastic	New
27	Glass	Brown	Single_Panel	80x36	Heavy_Duty	Oval	Plastic	New
20	Flush	Brown	Single_Panel	80x36	Heavy_Duty	Oval	Plastic	New
25	Glass	White	Single_Panel	80x36	Heavy_Duty	Oval	Plastic	New

Figure 8: ADOKAPI Form

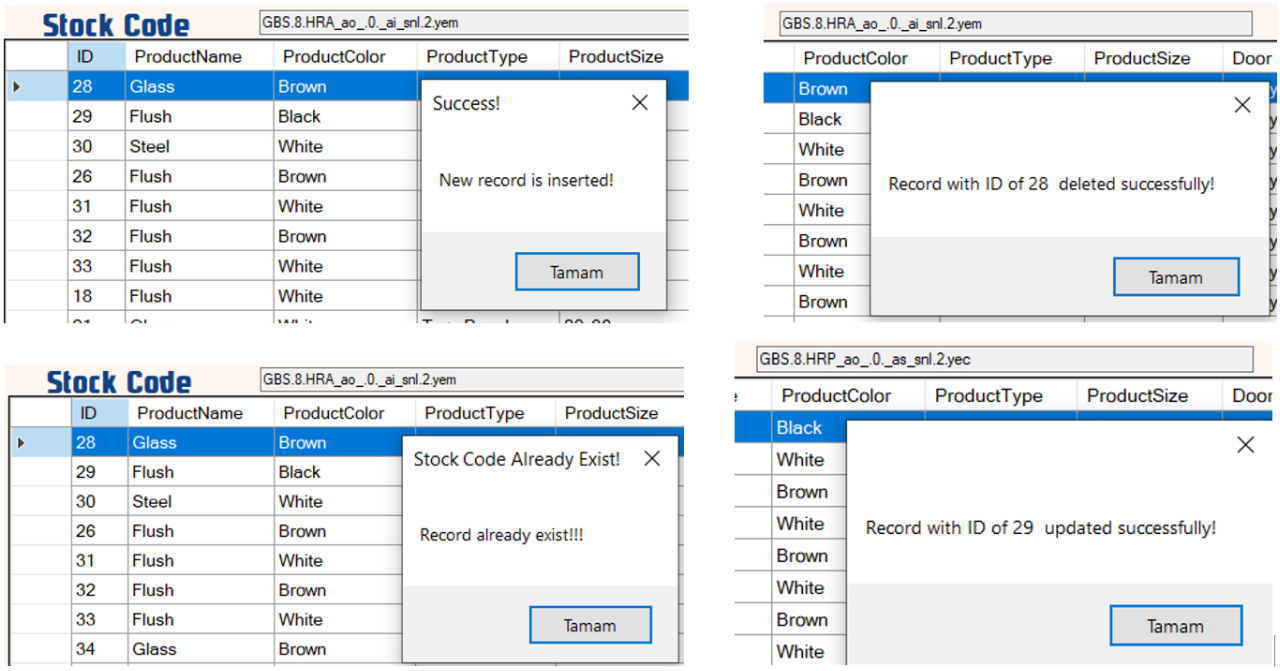


Figure 9: ADOKAPI CRUD Operations

As you can see in the figure above, all the CRUD operations are successfully implemented and tested. Some of the CRUD operation implementations are given in the figures below. They are implemented via C# methods and SQL queries.

```
private void iconButtonUpdate_Click(object sender, EventArgs e)
{
    if (dataGridViewProducts.Rows.Count == 0)
        MessageBox.Show("Table is empty");
    else
    {
        TempID = Convert.ToInt32(dataGridViewProducts.CurrentRow.Cells[0].Value);
        string commandText = "UPDATE p_door SET Door_sill =@Door_sill ,Doorknob=@Doorknob," +
            "Filling=@Filling,ProductStatus=@ProductStatus," +
            "ProductName=@ProductName,ProductColor=@ProductColor,ProductType=@ProductType,ProductSize=@ProductSize," +
            "StockCode=@StockCode WHERE ID = @TempID;";
        NpgsqlCommand command = new NpgsqlCommand(commandText, conn);
        string StockCode = StockCodeCreator();
        bool a = StockCodeDuplicateHandler(StockCode);
        if (a == true)
        {
            if (TempID > 0 && comboBoxDoor_sill.SelectedItem != null && comboBoxDoorknob.SelectedItem != null && comboBoxFilling.SelectedItem != null && comboBoxStatus.SelectedI
        && comboBoxName.SelectedItem != null && comboBoxColor.SelectedItem != null && comboBoxType.SelectedItem != null && comboBoxSize.SelectedItem != null)
            {
                conn.Open();
                command.Parameters.AddWithValue("@TempID", TempID);
                command.Parameters.AddWithValue("@Door_sill", comboBoxDoor_sill.SelectedItem.ToString());
                command.Parameters.AddWithValue("@Doorknob", comboBoxDoorknob.SelectedItem.ToString());
                command.Parameters.AddWithValue("@Filling", comboBoxFilling.SelectedItem.ToString());
                command.Parameters.AddWithValue("@ProductStatus", comboBoxStatus.SelectedItem.ToString());
                command.Parameters.AddWithValue("@ProductName", comboBoxName.SelectedItem.ToString());
                command.Parameters.AddWithValue("@ProductColor", comboBoxColor.SelectedItem.ToString());
                command.Parameters.AddWithValue("@ProductType", comboBoxType.SelectedItem.ToString());
                command.Parameters.AddWithValue("@ProductSize", comboBoxSize.SelectedItem.ToString());
                textBoxStockCode.Text = StockCodeCreator();
                command.Parameters.AddWithValue("@StockCode", textBoxStockCode.Text.ToString());
                command.ExecuteNonQuery();
                conn.Close();

                MessageBox.Show("Record with ID of " + TempID + "  updated successfully!");
                dataGridViewProducts.DataSource = GetData("SELECT * FROM p_door;");
            }
            else
                MessageBox.Show("Record could not be updated!!!");
        }
        else {
            string message = "Record already exist!!!";
            string title = "Stock Code Already Exist!";
            MessageBox.Show(message, title);
        }
```

Figure 10: Update Operation Implementation

```
private void iconButtonInsert_Click(object sender, EventArgs e)
{

    if (comboBoxDoor_sill.SelectedItem != null && comboBoxDoorknob.SelectedItem != null && comboBoxFilling.SelectedItem != null && comboBoxStatus.SelectedItem != null
        && comboBoxName.SelectedItem != null && comboBoxColor.SelectedItem != null && comboBoxType.SelectedItem != null && comboBoxSize.SelectedItem != null)
    {
        string StockCode = StockCodeCreator();
        bool a = StockCodeDuplicateHandler(StockCode);
        if (a == true)
        {
            textBoxStockCode.Text = StockCode;

            string commandText = "INSERT into p_door(Door_sill,Doorknob,Filling,ProductStatus,ProductName,ProductColor,ProductType,ProductSize,StockCode) " +
                "VALUES(@Door_sill,@Doorknob,@Filling,@ProductStatus,@ProductName,@ProductColor,@ProductType,@ProductSize,@StockCode);";

            NpgsqlCommand command = new NpgsqlCommand(commandText, conn);

            conn.Open();
            command.Parameters.AddWithValue("@Door_sill", comboBoxDoor_sill.SelectedItem.ToString());
            command.Parameters.AddWithValue("@Doorknob", comboBoxDoorknob.SelectedItem.ToString());
            command.Parameters.AddWithValue("@Filling", comboBoxFilling.SelectedItem.ToString());
            command.Parameters.AddWithValue("@ProductStatus", comboBoxStatus.SelectedItem.ToString());
            command.Parameters.AddWithValue("@ProductName", comboBoxName.SelectedItem.ToString());
            command.Parameters.AddWithValue("@ProductColor", comboBoxColor.SelectedItem.ToString());
            command.Parameters.AddWithValue("@ProductType", comboBoxType.SelectedItem.ToString());
            command.Parameters.AddWithValue("@ProductSize", comboBoxSize.SelectedItem.ToString());
            command.Parameters.AddWithValue("@StockCode", StockCode);
            command.ExecuteNonQuery();
            conn.Close();


            string message = "New record is inserted!";
            string title = "Success!";
            MessageBox.Show(message, title);
            dataGridViewProducts.DataSource = GetData("SELECT * FROM p_door;");
        }
        else
        {
            string message = "Record already exist!!!";
            string title = "Stock Code Already Exist!";
            MessageBox.Show(message, title);
        }
    }

    else
    {
        string message = "Fill all the necessary parts!!!";
        string title = "Error!";
        MessageBox.Show(message, title);
    }
}
```

Figure 11: Insert Operation Implementation

```
1 reference
227     private void iconButtonDelete_Click(object sender, EventArgs e)
228     {
229         if (dataGridViewProducts.Rows.Count == 0)
230             MessageBox.Show("Table is empty");
231         else
232         {
233             TempID = Convert.ToInt32(dataGridViewProducts.CurrentRow.Cells[0].Value);
234             string commandText = "DELETE FROM p_door WHERE ID = @TempID;";
235             NpgsqlCommand command = new NpgsqlCommand(commandText, conn);
236
237             if (TempID > 0)
238             {
239                 conn.Open();
240                 command.Parameters.AddWithValue("@TempID", TempID);
241                 command.ExecuteNonQuery();
242                 conn.Close();
243
244                 MessageBox.Show("Record with ID of " + TempID + "  deleted successfully!");
245                 dataGridViewProducts.DataSource = GetData("SELECT * FROM p_door;");
246             }
247             else
248                 MessageBox.Show("Record could not be deleted!!!");
249         }
250
251     }
```

Figure 12: Delete Operation Implementation

Similar approaches are followed while developing the floor form for ADOFLOOR. ADOFLOOR form has different variables other than ADOKAPI. It has the following parquet_pattern, parquet_click, and thickness, as shown in the figure below.

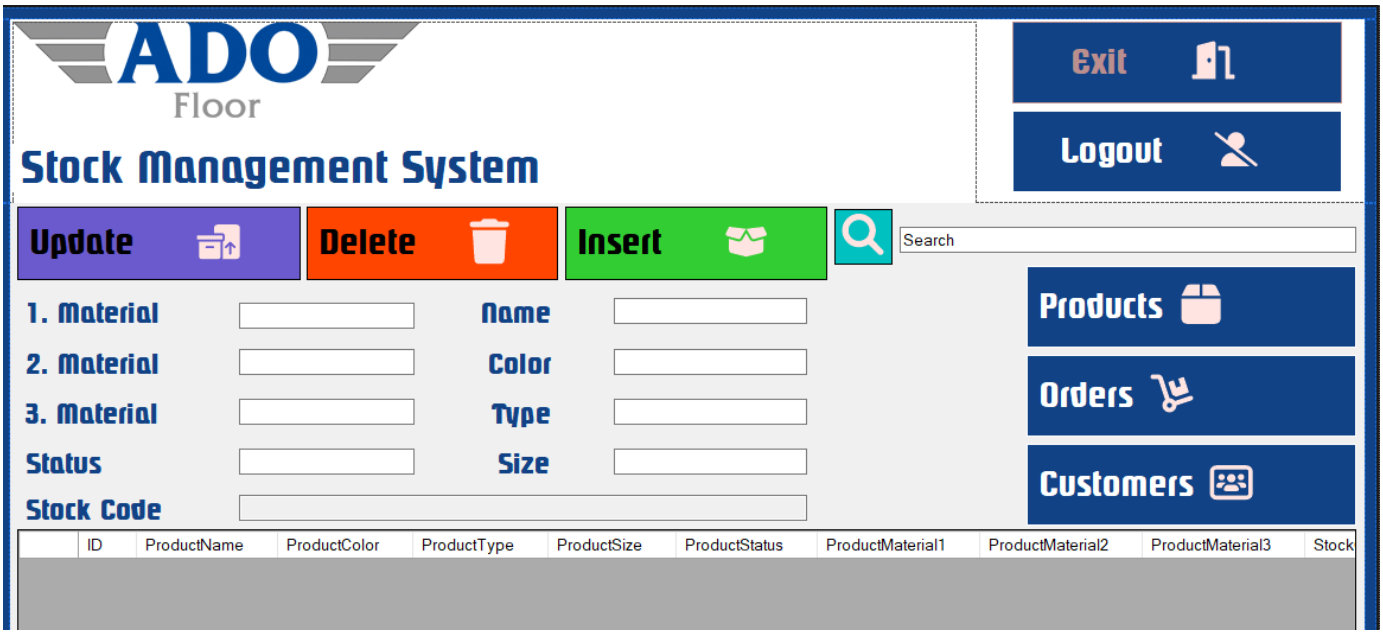


Figure 13: ADOFLOOR FORM

The needed information for an application to connect to a database server are stored in a connection string. While connecting forms with the PostgreSQL database the following connection string is used:

```
private static string connstring = "Server=127.0.0.1;Port=5432;Database=Adopen;User Id=postgres;Password=****;";
NpgsqlConnection conn = new NpgsqlConnection(connstring);
```

4.2.3. STOCK CODE GENERATION AND DUPLICATE STOCK CODE HANDLING

To create a stock code user needs to select from the dropdowns given in the forms. As shown in Figure 14, the stock code is created by different parts from the variables' names. I have done some string concatenation operations in C# to achieve that.

```
3 references
public string StockCodeCreator()//Creates unique stock code
{
    string StockCode = "";
    {
        string wholeText = comboBoxName.SelectedItem.ToString() + ' ' + comboBoxColor.SelectedItem.ToString() + ' '
            + comboBoxType.SelectedItem.ToString() +
        ' ' + '.' + ' ' + comboBoxSize.SelectedItem.ToString() + ' ' + '.'
        + ' ' + comboBoxDoor_sill.SelectedItem.ToString()+
        ' ' + comboBoxDoorknob.SelectedItem.ToString() + ' ' + comboBoxFilling.SelectedItem.ToString();
        StockCode = string.Join("", wholeText.Split(' ').Select(x => x[0]).ToArray())
            + '_' + string.Join("", wholeText.Split(' ').Select(x => x[x.Length / 2]).ToArray())
            + '_' + string.Join("", wholeText.Split(' ').Select(x => x[x.Length - 1]).ToArray());// Creating stock code
    }

    return StockCode; }
```

Figure 14: Stock Code Creator Code

To handle the duplication of stock codes, I have created a function shown below in Figure 15. It checks the database with a SELECT query and finds duplicate codes if there are any.

```
2 references
56  public bool StockCodeDuplicateHandler(string temp) //Stop user when they enter same product
57  {
58
59      if (dataGridViewProducts.Rows.Count != 0) {
60
61      string commandText = "SELECT COUNT(StockCode) FROM p_door where StockCode = @temp;";
62      NpgsqlCommand command = new NpgsqlCommand(commandText, conn);
63      conn.Open();
64      command.Parameters.AddWithValue("@temp", temp);
65
66      var result = command.ExecuteScalar().ToString();//if the product already exist returns 1
67
68      conn.Close();
69
70          if (result != "0")
71              result="1";  //if the product already exist result is 1
72
73          else
74              result ="0";
75
76
77          if (result != "0")
78                    return false;
79
80          else
81              return true;
82      }
83      return true;
84  }
```

Figure 15: Stock Code Duplicate Handler Function Code

4.2.3. ADMIN PAGE

For the admin of my company database, I have created an admin page. It allows the admin to add, delete or update dropdown values for each departmental page in the application. By using this page admin can add newly added products or delete expired ones. To achieve that, I designed several methods. As shown in Figure 16, I have a table in the database called "p_door_dropdowns." Also, "p_floor_dropdowns" and "p_window_dropdowns" store those departmental pages' dropdown information.

```
1 reference
private void buttonAddDoorSill_Click(object sender, EventArgs e)
{
    string commandText = "INSERT into p_door_dropdowns(Door_sill) VALUES(@Door_sill);";
    NpgsqlCommand command = new NpgsqlCommand(commandText, conn);
    if (textBoxNewDoorSill.Text != "") {
    conn.Open();
    command.Parameters.AddWithValue("@Door_sill", textBoxNewDoorSill.Text);
    command.ExecuteNonQuery();
    conn.Close();
    show_dropdowns();
    }
    else
        MessageBox.Show("No value entered!");
}
```

Figure 16: Code For Dropdown Insertion

As shown in the following Figure 17 and Figure 18, the admin page has many fields for each dropdown in the departmental pages of the application. As shown in Figure 17, a simple error detection mechanism is created to prevent null values in the database. As shown in the figures below, the entered amount of the material with the name can be removed or added to the dropdown database tables.

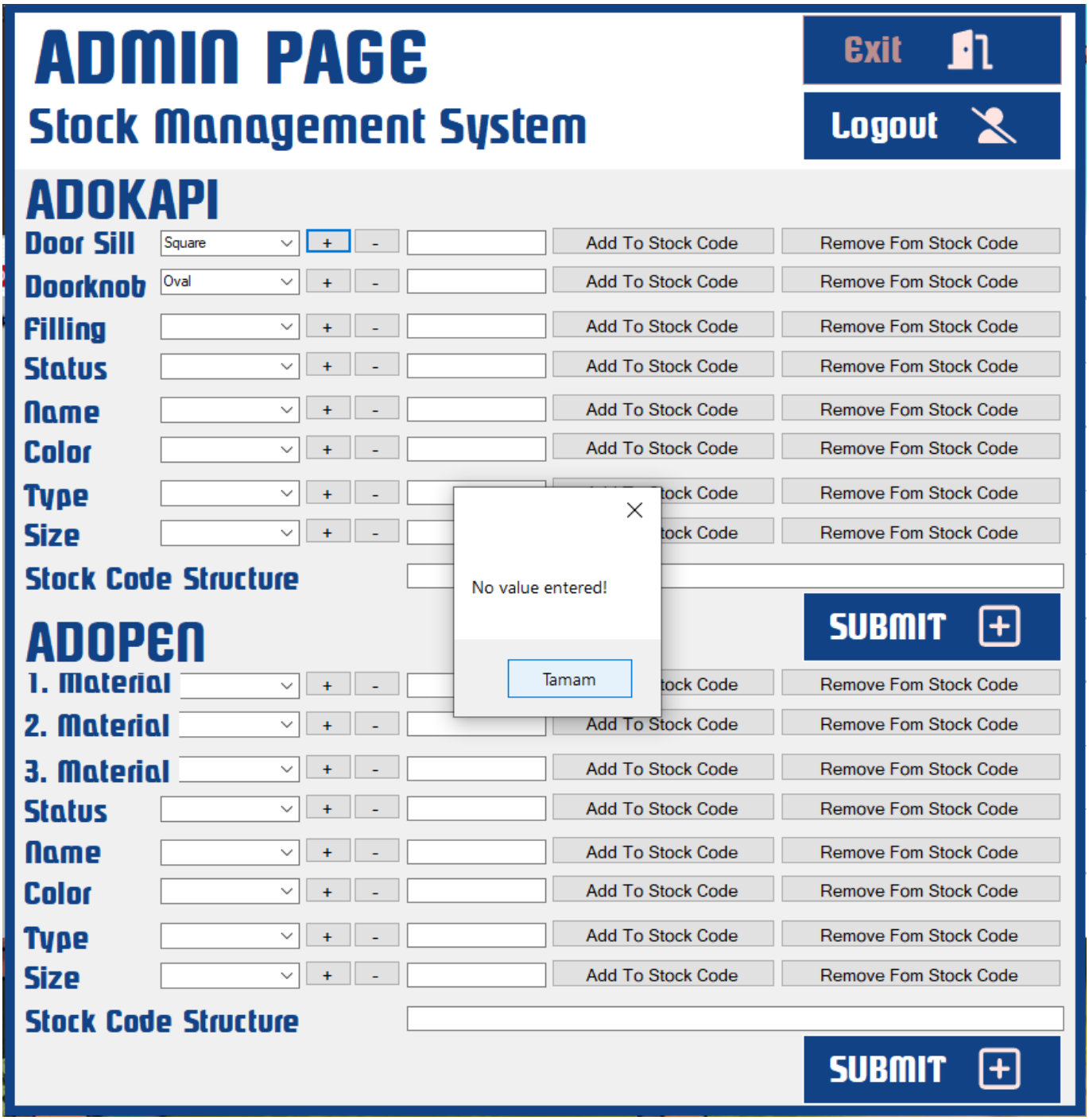


Figure 17: Admin Page Error

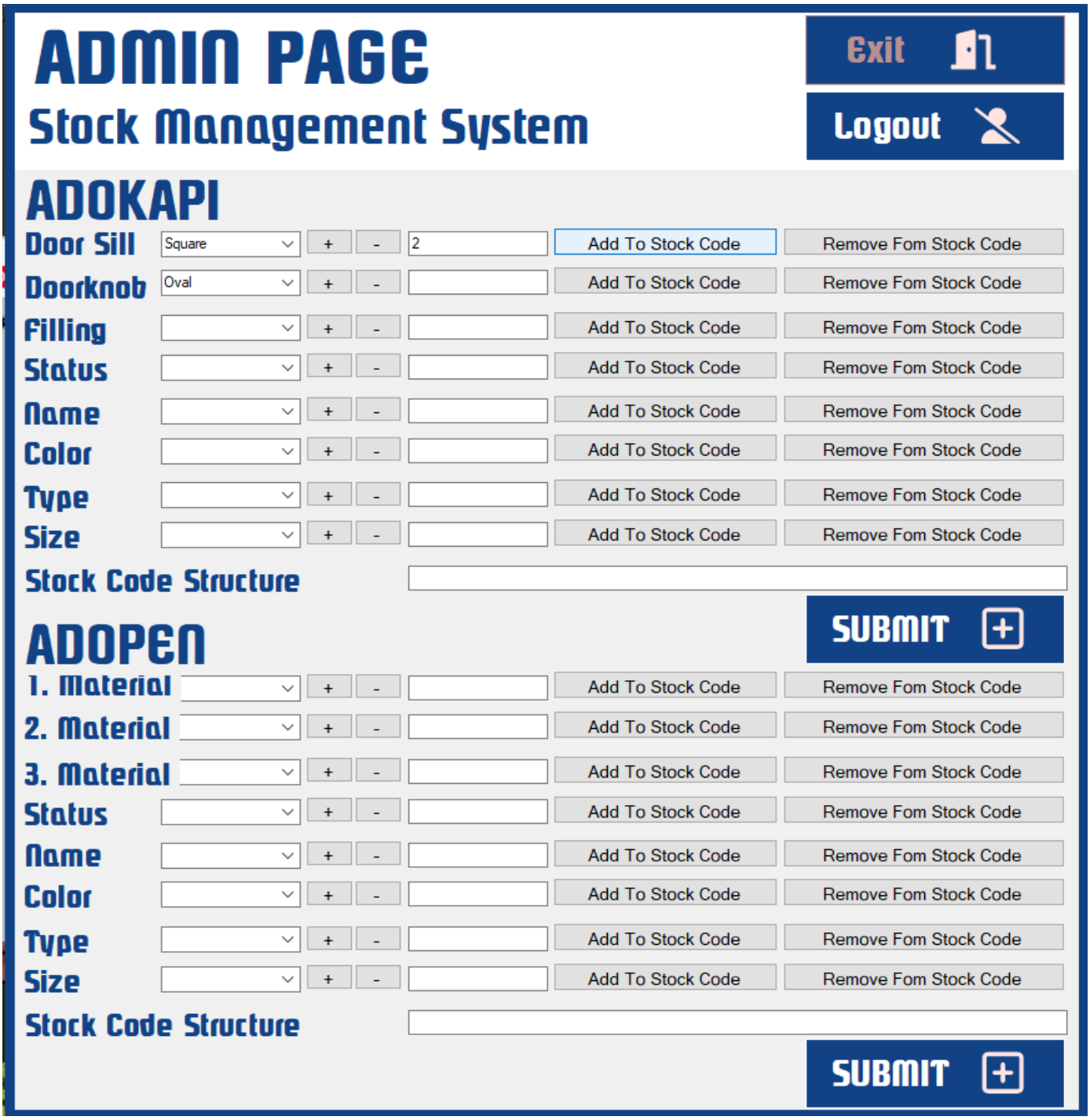


Figure 18:Admin Page Successful Stock Addition

5. SOFTWARE TESTING

5.1. USE CASE TESTS

In this part of the report testing steps are provided for the application and databse integration. Since structure is very similar for door and floor interfaces , the steps followed are very similar to each other. Therefore, I tried to avoid redundant showcases in this part of the report.

First, I installed and set up a PostgreSQL database on my local machine. Then, I created a database and a table to store the door or floor objects using the SQL statement provided, as mentioned earlier in the “Database Design” part of the report. Next, I compiled and ran the C# code, and the DoorForm form and FloorForm form was displayed on my screen according to department I am sellecting from the login page dropdown box. It was the test that shows that my login page is working well. It is already illusturated in “Figure 7: Login Page Design” and in the “Figure 8: ADOKAPI Form”.

After entering DoorForm form I tested the iconButtonInsert button by entering some values in the combo boxes and text boxes, and clicking the button. The new door object was added to the database and displayed in the dataGridView as expected. As you can see in the figures below.

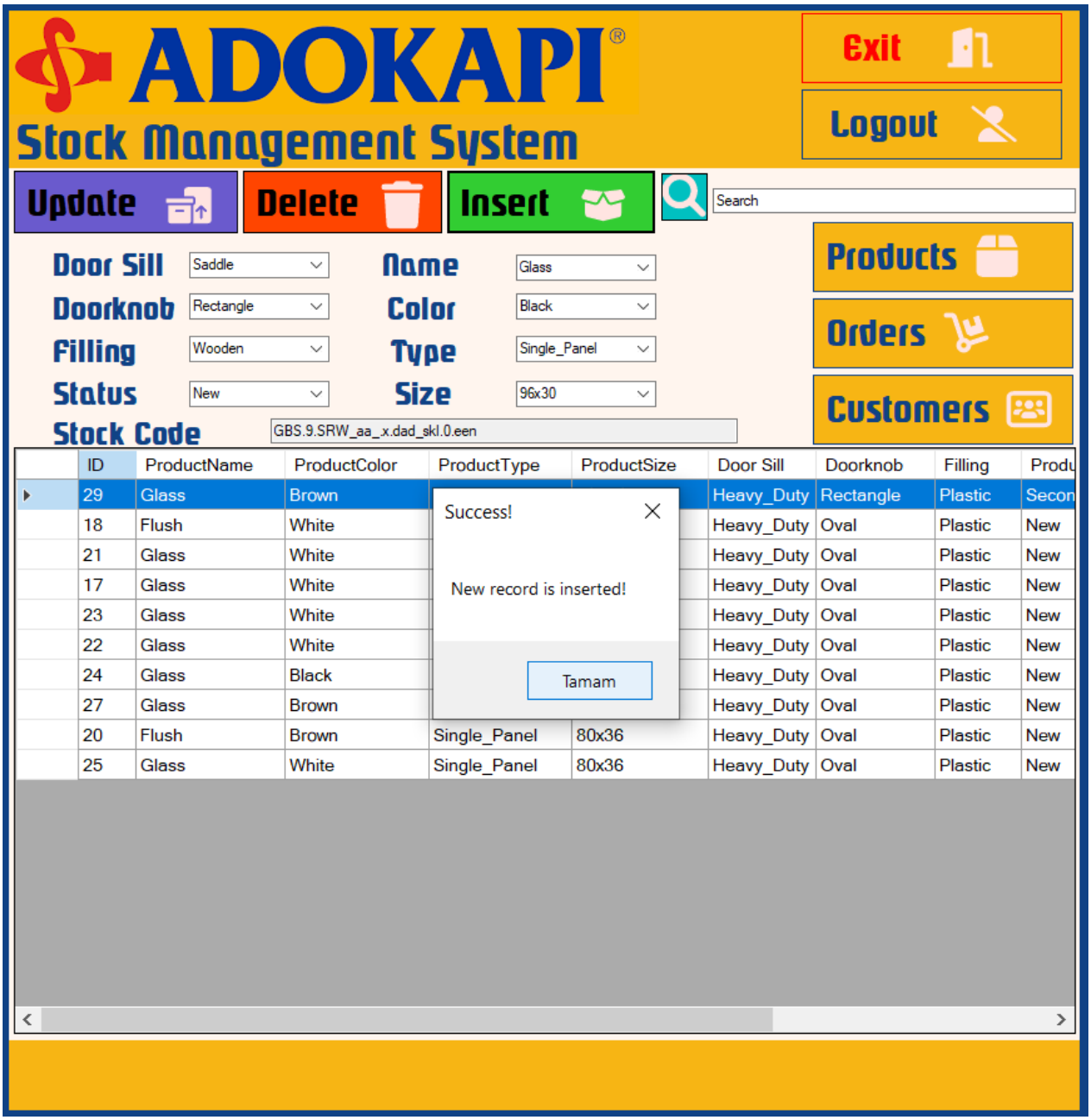


Figure 19: Clicking the Insert Button

Figure 20: After Clicking the Insert Button

public.p_door/Adopen/postgres@PostgreSQL 14

Query Editor | Query History | Data Output | Explain | Notifications

	id [PK] integer	door_sill character varying (100)	doorknob character varying (100)	filling character varying (100)	productstatus character varying (100)	productname character varying (100)	productcolor character varying (100)	producttype character varying (10
1	17	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
2	18	Heavy_Duty	Oval	Plastic	New	Flush	White	Two_Panel
3	20	Heavy_Duty	Oval	Plastic	New	Flush	Brown	Single_Panel
4	21	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
5	22	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
6	23	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
7	24	Heavy_Duty	Oval	Plastic	New	Glass	Black	Two_Panel
8	25	Heavy_Duty	Oval	Plastic	New	Glass	White	Single_Panel
9	27	Heavy_Duty	Oval	Plastic	New	Glass	Brown	Single_Panel
10	29	Heavy_Duty	Rectangle	Plastic	Second_hand	Glass	Brown	Single_Panel
11	35	Saddle	Rectangle	Wooden	New	Glass	Black	Single_Panel

Figure 21:Database After Clicking the Insert Button

I also tested the iconButtonUpdate button by selecting a door object in the dataGridView, modifying the values in the combo boxes and text boxes, and clicking the button. The selected door object was updated in the database and the dataGridView was refreshed to display the updated values. As shown in the figure below.

ID	ProductName	ProductColor	ProductType	ProductSize	Door Sill	Doorknob	Filling	Produ
29	Glass	Brown			_Duty	Rectangle	Plastic	Secon
35	Glass	Black			e	Rectangle	Wooden	New
18	Flush	White			_Duty	Oval	Plastic	New
21	Glass	White			_Duty	Oval	Plastic	New
17	Glass	White			_Duty	Oval	Plastic	New
23	Glass	White			_Duty	Oval	Plastic	New
22	Glass	White			_Duty	Oval	Plastic	New
24	Glass	Black			_Duty	Oval	Plastic	New
27	Glass	Brown	Single_Panel	80x36	Heavy_Duty	Oval	Plastic	New
20	Flush	Brown	Single_Panel	80x36	Heavy_Duty	Oval	Plastic	New
25	Glass	White	Single_Panel	80x36	Heavy_Duty	Oval	Plastic	New

Figure 22: Clicking the Update Button

Figure 23: After Clicking the Update Button

public.p_door/Adopen/postgres@PostgreSQL 14

Query Editor | Query History | Data Output | Explain | Notifications

	id [PK] integer	door_sill character varying (100)	doorknob character varying (100)	filling character varying (100)	productstatus character varying (100)	productname character varying (100)	productcolor character varying (100)	producttype character varying (100)	productsize character varying (100)	stockcode character varying (100)
1	17	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel	80x36	GWT.8.HOP_aIP.3._as_sel. .ylc
2	18	Heavy_Duty	Oval	Plastic	New	Flush	White	Two_Panel	80x36	FWT.8.HOP_uIP.3._as_hel. .ylc
3	20	Heavy_Duty	Oval	Plastic	New	Flush	Brown	Single_Panel	80x36	FBS.8.HOP_uo_.3._as_hnl. .ylc
4	21	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel	80x36	GWT.8.HOP_aIP.3._as_sel. .ylc
5	22	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel	80x36	GWT.8.HOP_aIP.3._as_sel. .ylc
6	23	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel	80x36	GWT.8.HOP_aIP.3._as_sel. .ylc
7	24	Heavy_Duty	Oval	Plastic	New	Glass	Black	Two_Panel	80x36	GBT.8.HOP_aaP.3._as_skl. .ylc
8	25	Heavy_Duty	Oval	Plastic	New	Glass	White	Single_Panel	80x36	GWS.8.HOP_ai_.3._as_sel. .ylc
9	27	Heavy_Duty	Oval	Plastic	New	Glass	Brown	Single_Panel	80x36	GBS.8.HOP_ao_.3._as_snl. .ylc
10	29	Saddle	Rectangle	Wooden	New	Steel	White	Three_Panel	96x30	SWT.9.SRW_ei_.x.dad_lel.0.een
11	35	Saddle	Rectangle	Wooden	New	Glass	Black	Three_Panel	96x30	GBT.9.SRW_aa_.x.dad_skl.0.ee

Figure 24: Database After Clicking the Update Button

I then tested the ıconButtonDelete button by selecting a door object in the dataGridView and clicking the button. The selected door object was deleted from the database and the dataGridView was refreshed to remove the deleted door object. As shown in the figures below.

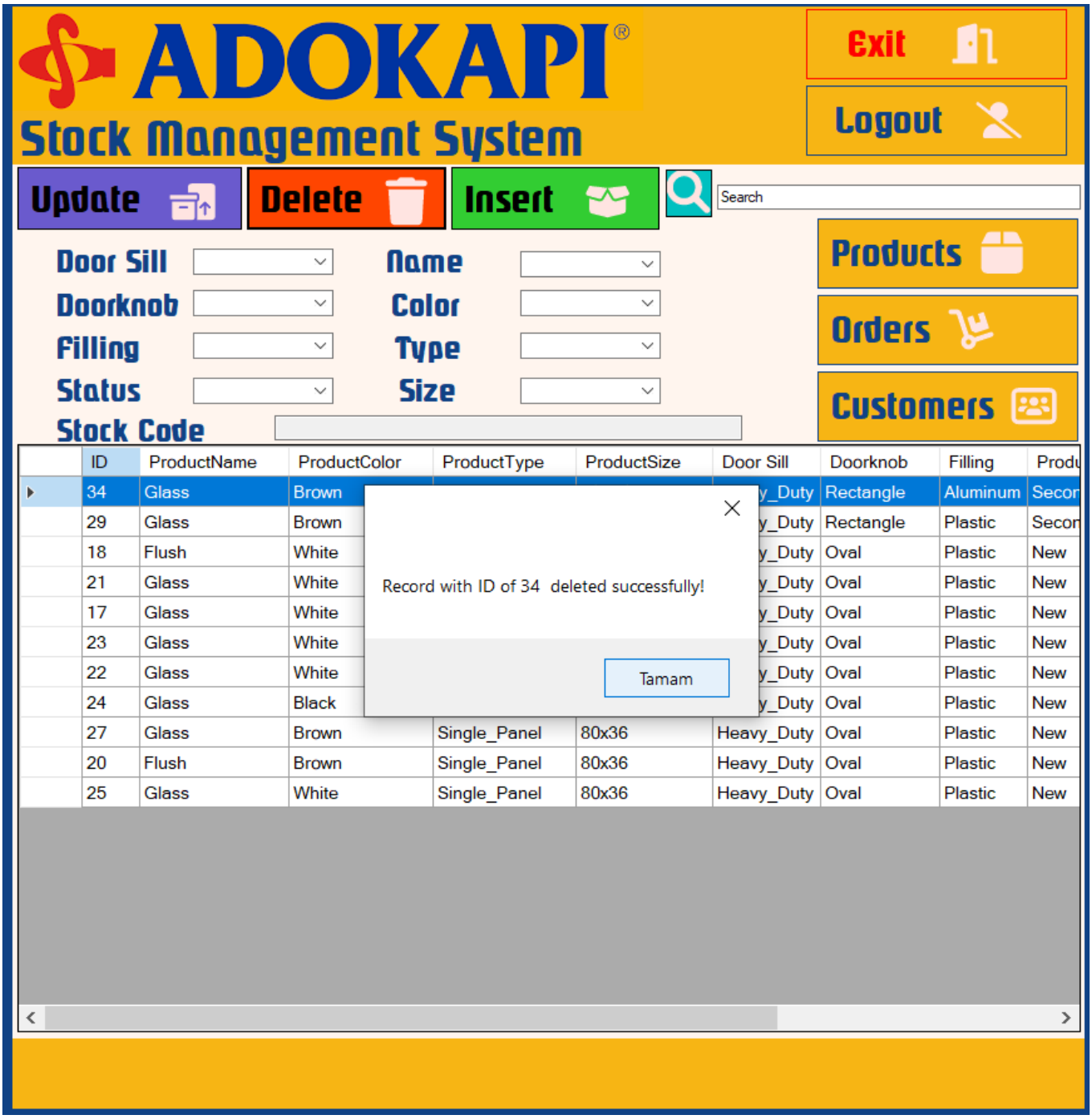


Figure 25: Clicking the Delete Button

public.p_door/Adopen/postgres@PostgreSQL 14

Query Editor | Query History | Data Output | Explain | Notifications

	id [PK] integer	door_sill character varying (100)	doorknob character varying (100)	filling character varying (100)	productstatus character varying (100)	productname character varying (100)	productcolor character varying (100)	producttype character varying
1	17	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
2	18	Heavy_Duty	Oval	Plastic	New	Flush	White	Two_Panel
3	20	Heavy_Duty	Oval	Plastic	New	Flush	Brown	Single_Panel
4	21	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
5	22	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
6	23	Heavy_Duty	Oval	Plastic	New	Glass	White	Two_Panel
7	24	Heavy_Duty	Oval	Plastic	New	Glass	Black	Two_Panel
8	25	Heavy_Duty	Oval	Plastic	New	Glass	White	Single_Panel
9	27	Heavy_Duty	Oval	Plastic	New	Glass	Brown	Single_Panel
10	29	Heavy_Duty	Rectangle	Plastic	Second_hand	Glass	Brown	Single_Panel

Figure 26: Database After Clicking the Delete Button

Finally, I tested the ıconButtonExit and ıconButtonLogout buttons by clicking them, and the application closed and the login form was displayed as expected.

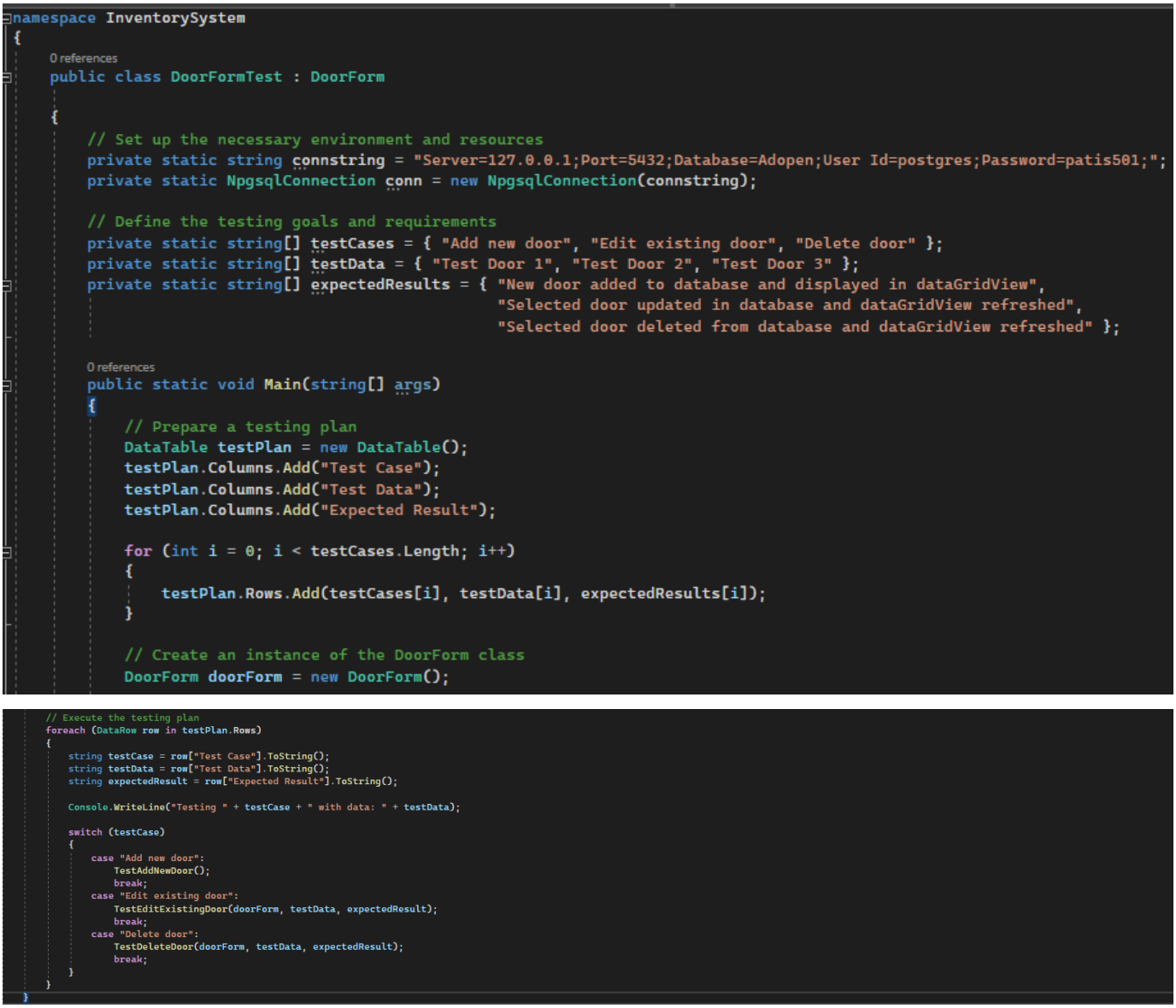
5.2 TESTING WITH THE METHODS

As the developer of these methods, I was responsible for ensuring that they were functional and properly implemented. However, another software testing engineer was responsible for conducting tests on these methods to verify and identify any potential issues of my project. We worked together to plan and execute a series of tests on these methods, using a variety of test cases and data to thoroughly evaluate their behavior. During the testing process, we encountered some issues, and I made necessary corrections to the methods to resolve these problems.

Unfortunately, I was unable to include the test results in my report because they were generated and collected by the software testing engineer, who is responsible for managing and documenting the testing process. As the developer, my role was limited to implementing and debugging the methods for testing, and I did not have access to the complete test results. However, I made sure to thoroughly communicate the testing process and any issues or corrections made to the testing engineer, and worked with them to ensure that the methods were fully functional and ready for deployment.

As you can see below, I have provided a detailed explanation of the methods and their functionalities. I have described the purpose and behavior of these methods in detail,

including how they are used and what they are designed to accomplish. This information gives a clear understanding of the methods and their role in the system, and how they are used for testing purposes.



```
namespace InventorySystem
{
    0 references
    public class DoorFormTest : DoorForm

    {
        // Set up the necessary environment and resources
        private static string connstring = "Server=127.0.0.1;Port=5432;Database=Adopen;User Id=postgres;Password=patis501;";
        private static NpgsqlConnection conn = new NpgsqlConnection(connstring);

        // Define the testing goals and requirements
        private static string[] testCases = { "Add new door", "Edit existing door", "Delete door" };
        private static string[] testData = { "Test Door 1", "Test Door 2", "Test Door 3" };
        private static string[] expectedResults = { "New door added to database and displayed in dataGridView",
                                                    "Selected door updated in database and dataGridView refreshed",
                                                    "Selected door deleted from database and dataGridView refreshed" };

        0 references
        public static void Main(string[] args)
        {
            // Prepare a testing plan
            DataTable testPlan = new DataTable();
            testPlan.Columns.Add("Test Case");
            testPlan.Columns.Add("Test Data");
            testPlan.Columns.Add("Expected Result");

            for (int i = 0; i < testCases.Length; i++)
            {
                testPlan.Rows.Add(testCases[i], testData[i], expectedResults[i]);
            }

            // Create an instance of the DoorForm class
            DoorForm doorForm = new DoorForm();
```

```
        // Execute the testing plan
        foreach (DataRow row in testPlan.Rows)
        {
            string testCase = row["Test Case"].ToString();
            string testData = row["Test Data"].ToString();
            string expectedResult = row["Expected Result"].ToString();

            Console.WriteLine("Testing " + testCase + " with data: " + testData);

            switch (testCase)
            {
                case "Add new door":
                    TestAddNewDoor();
                    break;
                case "Edit existing door":
                    TestEditExistingDoor(doorForm, testData, expectedResult);
                    break;
                case "Delete door":
                    TestDeleteDoor(doorForm, testData, expectedResult);
                    break;
            }
        }
    }
```

Figure 27: DoorFormTest Class

The Main method is the entry point of the DoorFormTest class, which is a unit testing class that is used to verify the functionality of the DoorForm class. The Main method sets up the necessary environment and resources for the tests by creating a connection to the database

using the NpgsqlConnection class and a connection string that specifies the server, port, database, user ID, and password for the database.

The Main method also defines the testing goals and requirements by defining three test cases, test data, and expected results. The test cases are represented as strings and include "Add new door", "Edit existing door", and "Delete door". The test data and expected results are also represented as strings and are associated with each test case. The Main method then prepares a testing plan by creating a DataTable and adding three columns for the test case, test data, and expected result. It then populates the DataTable with the test cases, test data, and expected results that were defined earlier. Next, the Main method creates an instance of the DoorForm class and stores it in a variable called doorForm. This instance of the DoorForm class will be used in the TestEditExistingDoor and TestDeleteDoor methods to perform the necessary operations on the doors. Finally, the Main method executes the testing plan by iterating over the rows in the DataTable and executing the appropriate test case for each row. It does this by using a switch statement to determine which test case to execute based on the value of the Test Case column. If the test case is "Add new door", it calls the TestAddNewDoor method. If the test case is "Edit existing door", it calls the TestEditExistingDoor method and passes in the doorForm instance, the Test Data value, and the Expected Result value as arguments. If the test case is "Delete door", it calls the TestDeleteDoor method and passes in the doorForm instance, the Test Data value, and the Expected Result value as arguments.

As shown in the figure below, the TestAddNewDoor method is a unit test that is designed to verify the correctness of the functionality for adding a new door to the database. To do this, the method constructs and executes an INSERT statement using the NpgsqlCommand class. This statement inserts a new door into the p_door table in the database, with the specified parameters such as the name, color, type, size, door sill, doorknob, filling, price, and stock code of the door.

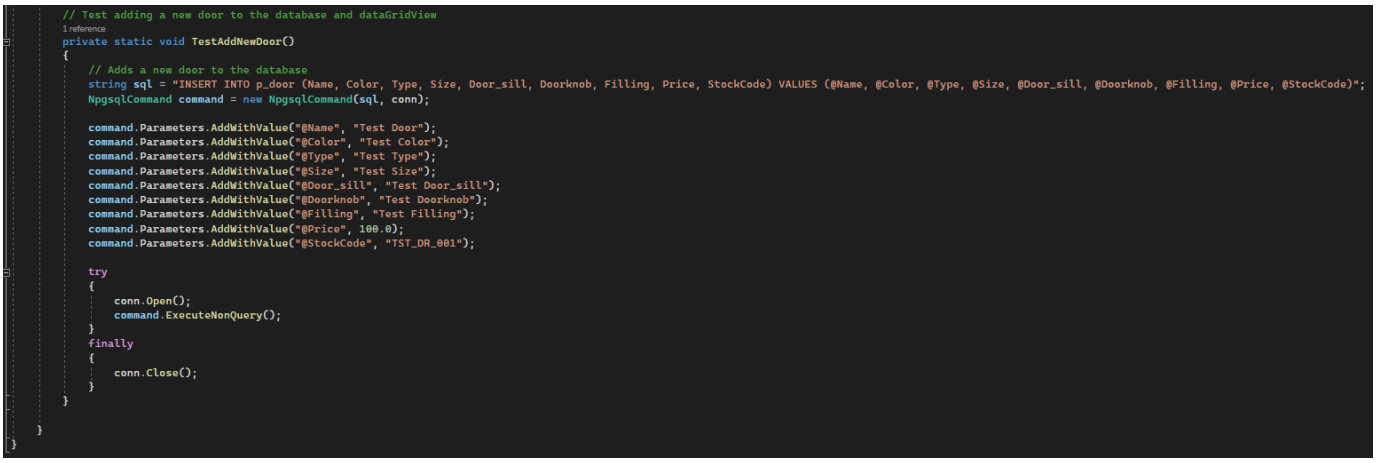
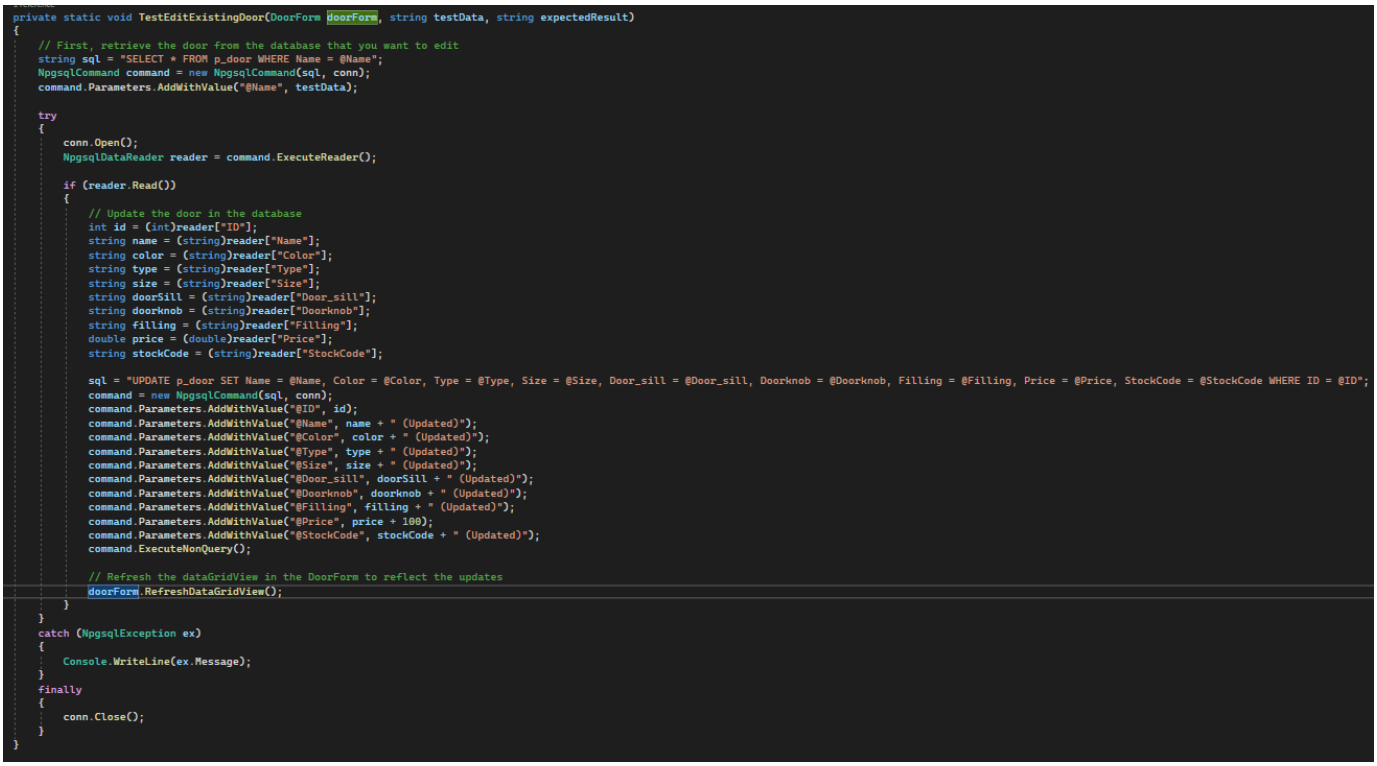


Figure 28: TestAddNewDoor Method

As shown in the figure below, the TestEditExistingDoor method is a unit test that verifies the functionality for editing an existing door in the database. It begins by using an SELECT statement and the NpgsqlCommand class to retrieve the door from the p_door table based on the name of the door, which is provided as a parameter to the method. Once the door has been retrieved, the method updates the door in the database by constructing and executing an UPDATE statement using the NpgsqlCommand class. This statement updates the specified fields of the door in the p_door table, such as the name, color, type, size, door sill, doorknob, filling, price, and stock code. Finally, the method calls the RefreshDataGridView method of the DoorForm class to refresh the dataGridView and reflect the updates. This allows the user to see that the door has been successfully updated in the user interface, and ensures that the data displayed in the dataGridView is consistent with the data in the database.



```
private static void TestEditExistingDoor(DoorForm doorForm, string testData, string expectedResult)
{
    // First, retrieve the door from the database that you want to edit
    string sql = "SELECT * FROM p_door WHERE Name = @Name";
    NpgsqlCommand command = new NpgsqlCommand(sql, conn);
    command.Parameters.AddWithValue("@Name", testData);

    try
    {
        conn.Open();
        NpgsqlDataReader reader = command.ExecuteReader();

        if (reader.Read())
        {
            // Update the door in the database
            int id = (int)reader["ID"];
            string name = (string)reader["Name"];
            string color = (string)reader["Color"];
            string type = (string)reader["Type"];
            string size = (string)reader["Size"];
            string doorSill = (string)reader["Door_sill"];
            string doorknob = (string)reader["Doorknob"];
            string filling = (string)reader["Filling"];
            double price = (double)reader["Price"];
            string stockCode = (string)reader["StockCode"];

            sql = "UPDATE p_door SET Name = @Name, Color = @Color, Type = @Type, Size = @Size, Door_sill = @Door_sill, Doorknob = @Doorknob, Filling = @Filling, Price = @Price, StockCode = @StockCode WHERE ID = @ID";
            command = new NpgsqlCommand(sql, conn);
            command.Parameters.AddWithValue("@ID", id);
            command.Parameters.AddWithValue("@Name", name + " (Updated)");
            command.Parameters.AddWithValue("@Color", color + " (Updated)");
            command.Parameters.AddWithValue("@Type", type + " (Updated)");
            command.Parameters.AddWithValue("@Size", size + " (Updated)");
            command.Parameters.AddWithValue("@Door_sill", doorSill + " (Updated)");
            command.Parameters.AddWithValue("@Doorknob", doorknob + " (Updated)");
            command.Parameters.AddWithValue("@Filling", filling + " (Updated)");
            command.Parameters.AddWithValue("@Price", price + 100);
            command.Parameters.AddWithValue("@StockCode", stockCode + " (Updated)");
            command.ExecuteNonQuery();

            // Refresh the dataGridView in the DoorForm to reflect the updates
            doorForm.RefreshDataGridView();
        }
    }
    catch (NpgsqlException ex)
    {
        Console.WriteLine(ex.Message);
    }
    finally
    {
        conn.Close();
    }
}
```

Figure 29: TestEditExistingDoor Method

As shown in the figure below, the TestDeleteDoor method is a unit test that verifies the functionality for deleting a door from the database. It retrieves the door from the database using an SELECT statement and the NpgsqlCommand class, based on the name of the door provided as a parameter. Then, it deletes the door from the database by executing a DELETE statement using the NpgsqlCommand class. Finally, it calls the RefreshDataGridView method of the DoorForm class to refresh the dataGridView and reflect the deletion, allowing the user to see that the door is no longer displayed in the user interface.

```
private static void TestDeleteDoor(DoorForm doorForm, string testData, string expectedResult)
{
    // First, retrieve the door from the database that you want to delete
    string sql = "SELECT * FROM p_door WHERE Name = @Name";
    NpgsqlCommand command = new NpgsqlCommand(sql, conn);
    command.Parameters.AddWithValue("@Name", testData);

    try
    {
        conn.Open();
        NpgsqlDataReader reader = command.ExecuteReader();

        if (reader.Read())
        {
            // Delete the door from the database
            int id = (int)reader["ID"];
            sql = "DELETE FROM p_door WHERE ID = @ID";
            command = new NpgsqlCommand(sql, conn);
            command.Parameters.AddWithValue("@ID", id);
            command.ExecuteNonQuery();

            // Refresh the dataGridView in the DoorForm to reflect the deletion
            doorForm.RefreshDataGridView();
        }
    }
    catch (NpgsqlException ex)
    {
        Console.WriteLine(ex.Message);
    }
    finally
    {
        conn.Close();
    }
}
```

Figure 30: TestDeleteDoor Method

6. CONCLUSIONS

I spent twenty business days working on software design and testing topics throughout my internship. I developed a stock management desktop application with database integration since the company's old application was not updatable. Also, I have designed a login page for the same desktop application for different departments of my company, even for the admin. Additionally, for the admin, I have created an admin page that allows them to change the whole departmental page dropdowns. I completed all the project pages using C# in the Visual Studio development environment. I have tested my application with some users in the company. I took their feedback and made some updates. Since my application allows users to manage their departmental product stocks modularly, it is also adjustable for new departments. The application also has many error controls and warning mechanisms. It checks the database for duplicate stock codes and avoids redundant and inconsistent data.

Furthermore, I have followed agile development rules and restrictions in this project. My supervisors and coworkers have regularly given me feedback and guided me with several tutorials. It helped me to shorten the delivery time. I have understood the importance of a collaborative working environment with innovative coworkers.

Finally, I thank İbrahim Kuş and other Information Technologies workers in ADOPEN since they provided me with a safe and disciplined professional working environment.

REFERENCES

About us | ADOPEN. (n.d.). Pvc Window and Door Systems | ADOPEN. https://www.adopen.com/about-us

ADO introduction. (n.d.). W3Schools Online Web Tutorials. https://www.w3schools.com/asp/ado_intro.asp

Anandmeg. (n.d.). Create a Windows forms app with C# - Visual studio (Windows). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/enus/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022

C# tutorial (C sharp). (n.d.). W3Schools Online Web Tutorials. https://www.w3schools.com/cs/index.php

Erkec, E. (2021, September 24). SQL connection strings tips. SQL Shack - articles about database auditing, server performance, data recovery, and more. https://www.sqlshack.com/sql-connection-strings-tips/

(2022, November 3). PostgreSQL. https://www.postgresql.org