**ДИПЛОМЕН ПРОЕКТ**

**ТЕМА:**

**РАЗРАБОТВАНЕ НА НАСТОЛНО ПРИЛОЖЕНИЕ ЗА ФИРМА ЗА ЕЛЕКТРОНИКА**

**ДИПЛОМАНТ: Константин Любенов Станоев**

**СПЕЦИАЛНОСТ: *код 4810201 „Системно програмиране“***

**ПРОФЕСИЯ:** ***код 481020 „Системен програмист“***

**РЪКОВОДИТЕЛ-КОНСУЛТАНТ: инж. Иван Киров**

**СОФИЯ**

**2024**

ЗАДАНИЕ

# Съдържание

Contents

[Съдържание 4](#_Toc195475162)

[Раздел I – Теоретична част 6](#_Toc195475163)

[Увод: 6](#_Toc195475164)

[Анализ на съществуващи решения: 7](#_Toc195475165)

[Изследване на популярни платформи 7](#_Toc195475166)

[Технологичен анализ на наличните решения 8](#_Toc195475167)

[Идентифицирани проблеми и възможности 8](#_Toc195475168)

[Заключение от анализа 9](#_Toc195475169)

[Основи на релационните бази данни 9](#_Toc195475170)

[Историческо развитие и концепция 9](#_Toc195475171)

[Основни компоненти на релационния 10](#_Toc195475172)

[Видове връзки между таблици 10](#_Toc195475173)

[Нормализация на бази от данни 10](#_Toc195475174)

[SQL – език за работа с релационни бази 11](#_Toc195475175)

[Използване на MySQL и C# в проекта 11](#_Toc195475176)

[Технологии за разработка 12](#_Toc195475177)

[C# като език за програмиране 12](#_Toc195475178)

[Visual Studio 12](#_Toc195475179)

[ADO.NET комуникация с база от данни 13](#_Toc195475180)

[MySQL като релационна база данни 13](#_Toc195475181)

[MySQL Workbench 14](#_Toc195475182)

[Взаимодействие между компонентите 14](#_Toc195475183)

[Алтернативни технологии (сравнение и обосновка на избора) 14](#_Toc195475184)

[Език за програмиране 15](#_Toc195475185)

[Система за управление на бази данни 15](#_Toc195475186)

[Тип платформа: настолна, уеб или мобилна 16](#_Toc195475187)

[Среда за разработка 17](#_Toc195475188)

[Обобщение 17](#_Toc195475189)

[Работа с бази от данни в C# 17](#_Toc195475190)

[ADO.NET и MySql.Data 18](#_Toc195475191)

[Отваряне и затваряне на връзка 18](#_Toc195475192)

[Изпълнение на SQL команди 18](#_Toc195475193)

[Параметризирани заявки 19](#_Toc195475194)

[Зареждане на данни в DataGridView 19](#_Toc195475195)

[Обработка на грешки 19](#_Toc195475196)

[Структура и организация на кода 20](#_Toc195475197)

[Заключение 20](#_Toc195475198)

[Архитектура на системата 20](#_Toc195475199)

[Основни функционалности 21](#_Toc195475200)

[Административен панел 21](#_Toc195475201)

[Потребителски интерфейс 22](#_Toc195475202)

[Бъдещи подобрения и възможности за разширяване 22](#_Toc195475203)

[Функционални характеристики на приложението 23](#_Toc195475204)

[Презентационен слой (Presentation Layer) 23](#_Toc195475205)

[Логически слой (Business Logic Layer) 24](#_Toc195475206)

[Данни и достъп до база данни (Data Access Layer) 24](#_Toc195475207)

[База данни (Database Layer) 25](#_Toc195475208)

[Възможности за разширяване 25](#_Toc195475209)

[Заключение 26](#_Toc195475210)

[Раздел II – Практическа част 27](#_Toc195475211)

[Обща структура на системата 27](#_Toc195475212)

[База от данни – структура и свързаност 27](#_Toc195475213)

[Работа с формуляри и действия в интерфейса 28](#_Toc195475214)

[Управление на формата за вход (LoginForm) 31](#_Toc195475215)

[Основни функционалности на LoginForm: 31](#_Toc195475216)

[Автоматично поведение при повторно показване: 32](#_Toc195475217)

[Примерен сценарий: добавяне на нов клиент 32](#_Toc195475218)

[Примерен сценарий: редактиране на вече създадена поръчка 34](#_Toc195475219)

[Примерен сценарий: изтриване на продукт 36](#_Toc195475220)

[Взаимодействие с базата от данни 38](#_Toc195475221)

[Заключение 39](#_Toc195475222)

[Раздел III - Източници 39](#_Toc195475223)

[Раздел IV - Кодът 39](#_Toc195475224)

[LoginForm 39](#_Toc195475225)

[ProcessingForm 41](#_Toc195475226)

# Раздел I – Теоретична част

## Увод:

В съвременната дигитална епоха ефективното управление на информацията е от съществено значение за успешното функциониране на бизнеса. Компаниите, занимаващи се с търговия и дистрибуция на компютърна техника, ежедневно обработват големи обеми от данни, свързани с продукти, клиенти, поръчки и доставчици. Тази информация трябва да бъде надеждно съхранявана, лесно достъпна и подлежаща на ефективна обработка, за да се гарантира бързо и безпроблемно обслужване. В този контекст необходимостта от специализирани софтуерни решения, които оптимизират тези процеси, става все по-голяма.

Настоящият дипломен проект е посветен на разработването на настолно приложение за фирма, специализирана в продажбата на компютърна техника, включително **настолни компютри, лаптопи и сървъри**. Основната цел на приложението е да автоматизира ключови бизнес процеси, като осигури интуитивен интерфейс за управление на **продуктовия каталог, клиентските профили, поръчките и доставките**. Чрез използването на **релационна база данни** ще се гарантира структурирано и ефективно съхранение на информацията, а интегрираният потребителски интерфейс ще позволи бърз достъп до всички основни функционалности.

Дигитализацията и автоматизацията на бизнес процесите носят значителни предимства за търговските компании, като подобряват вътрешната организация и повишават конкурентоспособността. Въвеждането на софтуерна система, базирана на релационна база данни, ще даде възможност за **оптимизация на работните процеси, минимизиране на грешките и повишаване на ефективността**. Чрез детайлно анализиране на съществуващите решения и проектиране на функционалностите ще бъде създадена платформа, съобразена със спецификите на фирмата, която ще подобри управлението на ресурсите и ще повиши качеството на предлаганите услуги.

В следващите глави ще бъде направен анализ на съществуващите софтуерни решения, ще се разгледат принципите на релационните бази данни и ще се обоснове изборът на използваните технологии. Освен това ще бъдат представени основните функционалности на разработваното приложение и неговите предимства спрямо алтернативните методи за управление на информацията.

## Анализ на съществуващи решения:

Преди да се пристъпи към проектиране и разработка на собствена система за управление на онлайн магазин за електроника, е от съществено значение да бъдат анализирани съществуващите решения в този домейн. Това включва както проучване на конкурентни платформи и тяхната функционалност, така и използваните от тях технологии и архитектурни подходи. Анализът служи като основа за определяне на силните и слабите страни на наличните решения, за да се избегнат техните недостатъци и да се надградят добрите практики.

Изследване на популярни платформи Онлайн търговията с електроника е добре развита област, в която доминират големи играчи като Amazon, eMAG и Technopolis. Тези платформи предлагат широк набор от функции:

* Интуитивен потребителски интерфейс
* Филтриране и търсене на продукти
* Потребителски профили и история на поръчките
* Сигурно плащане и доставка
* Управление на наличности и категории продукти

Въпреки че тези платформи са мащабни и комплексни, те често са твърде тежки за нуждите на малки и средни бизнеси. Това налага разработката на по-леки и адаптивни решения, съобразени с конкретни изисквания.

Технологичен анализ на наличните решения Много от съществуващите платформи използват популярни среди като PHP и JavaScript за уеб-базирани решения, в комбинация с бази от данни като MySQL, PostgreSQL или MongoDB. В настоящата дипломна работа обаче се прави избор за използване на:

* **C#** за бекенд логиката, поради стабилността и богатата екосистема на .NET.
* **MySQL** за базата от данни, тъй като тя е съвместима, надеждна и лесна за управление с **MySQL Workbench**.
* **Visual Studio** като интегрирана среда за разработка, която предлага богати възможности за дебъгване, тестване и визуален дизайн.

Това решение осигурява по-добра интеграция и гъвкавост при разработка на десктоп базирана система, която не изисква хостинг или специфични сървърни конфигурации.

Идентифицирани проблеми и възможности **за подобрение** При анализа на съществуващи решения се наблюдават някои често срещани слабости:

* Сложен потребителски интерфейс, който затруднява навигацията
* Ограничена възможност за персонализиране от страна на малкия бизнес
* Слаба интеграция между модулите за поръчки, инвентар и клиенти

Системата, разработена в рамките на дипломната работа, цели да адресира тези проблеми чрез:

* Изчистен и опростен дизайн
* Централизирано управление на данни чрез нормализирана релационна база
* Интуитивна връзка между отделните функционалности (продукти, клиенти, поръчки)

Заключение от анализа Проучването на съществуващи решения е от решаващо значение за дефиниране на обхвата и функционалността на разработваната система. Изборът на конкретни технологии и архитектурни подходи – C#, MySQL, Visual Studio – предоставя стабилна основа за разработка на адаптивно и функционално десктоп приложение, което отговаря на нуждите на електронна търговия в среден и малък мащаб.

## Основи на релационните бази данни

Релационните бази данни са гръбнакът на съвременните информационни системи, осигуряващи надеждно и последователно управление на структурирани данни. Те позволяват ефективно съхранение, извличане и манипулация на информация, като осигуряват висока степен на логическа яснота и поддръжка на сложни зависимости между различни данни. За дипломния проект, фокусиран върху разработката на система за онлайн търговия с електроника, релационният модел предлага стабилна основа за създаване на надеждна и добре организирана база от данни, която е съвместима с инструментите MySQL и езика за програмиране C#.

Историческо развитие и концепция Релационният модел е въведен от Едгар Ф. Код през 1970 г. като теоретичен подход към организацията на данни. Основният му принцип е, че всички данни трябва да бъдат представени във вид на релации (таблици), състоящи се от редове (записи) и колони (атрибути). За разлика от по-старите йерархични и мрежови модели, релационният подход предлага по-голяма гъвкавост и логическа прозрачност, като позволява дефиниране на връзки между обекти от реалния свят по ясен и стандартизиран начин.

Основни компоненти на релационния **модел**

* **Таблици** – основни структури, които представят конкретни обекти, като продукти, клиенти, поръчки и т.н.
* **Редове (записи)** – всяка таблица съдържа множество редове, като всеки ред представя уникален запис.
* **Колони (атрибути)** – всяка колона представя конкретно свойство на обекта, напр. име, цена, дата.
* **Ключове:**
  + **Първичен ключ (Primary Key)** – уникален идентификатор за всеки ред, който гарантира целостта на данните.
  + **Външен ключ (Foreign Key)** – използва се за дефиниране на връзки между таблици, чрез препращане към първичен ключ от друга таблица.

### Видове връзки между таблици

* **Един към един (1:1)** – рядко използвана връзка, напр. един клиент има една потребителска настройка.
* **Един към много (1:N)** – най-често срещаната връзка, напр. един клиент може да направи много поръчки.
* **Много към много (M:N)** – реализира се чрез допълнителна (свързваща) таблица, напр. поръчки и продукти.

Нормализация на бази от данни Нормализацията е процес на разделяне на големи таблици на по-малки и логически свързани таблици, с цел избягване на дублирани данни и аномалии при актуализации. Този процес преминава през няколко нормални форми:

* **1NF (Първа нормална форма)** – елиминиране на многозначни колони.
* **2NF (Втора нормална форма)** – премахване на частични зависимости.
* **3NF (Трета нормална форма)** – елиминиране на транзитивни зависимости. Нормализираната база е по-ефективна, по-надеждна и по-лесна за поддръжка.

SQL – език за работа с релационни бази SQL (Structured Query Language) е основен език за комуникация с релационни бази данни. Неговите основни команди включват:

* **SELECT** – извличане на данни
* **INSERT** – добавяне на записи
* **UPDATE** – промяна на съществуващи записи
* **DELETE** – изтриване на данни
* **CREATE, ALTER, DROP** – управление на структурата на базата Тези команди се използват активно чрез C# кода и ADO.NET компонентите в настоящия проект.

Използване на MySQL и C# в проекта Настоящата дипломна работа използва MySQL като система за управление на бази от данни, поради нейната надеждност, безплатен лиценз и добра поддръжка на SQL стандарта. C# и ADO.NET са използвани за осъществяване на връзката между приложението и базата, позволявайки динамично извличане, редактиране и визуализация на данни.

* **MySQL Workbench** се използва за визуално моделиране на базата, както и за ръчно тестване на заявки.
* **Visual Studio** и **C#** осигуряват потребителския интерфейс и логиката на приложението.

Използването на тези технологии осигурява стабилна и гъвкава архитектура, която позволява лесно разширяване и адаптиране към бъдещи бизнес нужди.

## Технологии за разработка

Изборът на технологии за реализирането на софтуерен проект е от ключово значение за неговата стабилност, ефективност и поддръжка. При настоящото дипломно приложение, чиято цел е изграждането на система за управление на онлайн магазин за електроника, са използвани утвърдени и широко разпространени инструменти и програмни езици – **C#**, **MySQL**, **Visual Studio** и **MySQL Workbench**. Тези технологии предлагат богата екосистема от библиотеки, документация и възможности за интеграция, което ги прави идеални за създаването на пълноценна информационна система.

C# като език за програмиране **-** C# (C Sharp) е обектно-ориентиран език, разработен от Microsoft като част от .NET платформата. Той съчетава мощта и производителността на C++ със сигурността и управлението на паметта на Java. Сред основните предимства на C# за разработка на настолни приложения са:

* Поддръжка на богати графични интерфейси чрез Windows Forms или WPF
* Лесна интеграция с бази от данни чрез ADO.NET
* Добра типова безопасност и автоматично управление на паметта
* Развита среда за дебъгване и писане на код във Visual Studio

Visual Studio Visual Studio - е интегрирана среда за разработка (IDE), която предоставя широк набор от инструменти за писане, тестване и дебъгване на приложения, разработени на C#. Нейните предимства включват:

* Поддръжка на шаблони за проекти и форми
* Интегриран дизайнер за графичен интерфейс
* Инструменти за контрол на версиите (Git)
* Възможност за разширяване с допълнителни разширения и плъгини

ADO.NET комуникация с база от данни **-** ADO.NET е компонент на .NET Framework, който предоставя набор от библиотеки за работа с бази от данни. С негова помощ могат да се изпълняват SQL заявки, да се четат и модифицират данни, както и да се обработват резултати чрез DataTable и DataSet. Основни характеристики:

* Работа със свързан и несвързан модел
* Поддръжка на параметризирани заявки за сигурност срещу SQL инжекции
* Лесна интеграция с MySQL чрез MySQL Connector/NET

MySQL като релационна база данни **-** MySQL е един от най-популярните системи за управление на релационни бази от данни с отворен код. Тя се използва широко за уеб и десктоп приложения поради своята:

* Висока производителност при големи обеми от данни
* Поддръжка на транзакции, индекси, тригери и процедури
* Съвместимост с .NET приложения чрез съответни драйвери
* Добре документиран синтаксис и поддръжка на стандартен SQL

MySQL Workbench **-** MySQL Workbench е визуален инструмент за работа с MySQL бази данни. Използва се за:

* Създаване и моделиране на таблици, релации и ограничения
* Изпълнение на SQL заявки
* Преглед и модификация на данни
* Създаване на ER диаграми за визуализиране на структурата

Взаимодействие между компонентите **-** В системата, реализирана по време на проекта, взаимодействието между компонентите е реализирано по следния начин:

* Потребителят работи с графичен интерфейс, създаден с Windows Forms в средата на Visual Studio
* Чрез C# и ADO.NET се изпращат заявки към базата данни
* Базата от данни, създадена и управлявана в MySQL Workbench, отговаря на заявките и връща резултатите
* Резултатите се визуализират обратно в потребителския интерфейс

Тази архитектура осигурява ясно разделение между представянето, логиката и данните, като същевременно гарантира мащабируемост и възможност за бъдещо надграждане. Използваните технологии са стабилни, добре документирани и с широка поддръжка в професионалната общност, което допринася за успеха на проекта.

## Алтернативни технологии (сравнение и обосновка на избора)

При проектирането на информационна система е важно не само да се изгради правилната архитектура, но и да се изберат подходящи технологии, които да отговарят на конкретните цели, мащаб и технически възможности на проекта. В тази точка се разглеждат възможни алтернативи на използваните технологии – език за програмиране, база данни, среда за разработка и тип платформа – и се обосновава защо настоящият избор е най-подходящ.

### Език за програмиране

В настоящия проект е използван езикът **C#**, който предоставя високо ниво на структурираност, яснота и обектно-ориентирани възможности. Като алтернативи биха могли да се използват **Java** или **Python**.

Java е също силно типизиран език и традиционно се използва за големи уеб и enterprise приложения. Той е особено подходящ за кросплатформени решения, но за desktop GUI разработка изисква повече усилия при оформяне на потребителския интерфейс (например чрез Swing или JavaFX).

Python е лесен за научаване и използване, особено за бързо прототипиране. Въпреки това, той не предлага същото ниво на производителност и контрол върху графичен интерфейс в сравнение с C#. Също така, Python разчита на допълнителни библиотеки (напр. Tkinter, PyQt), които не са така добре интегрирани с инструменти като Visual Studio.

Изборът на C# е обоснован с факта, че той работи отлично в среда като **Windows Forms**, предлага директна поддръжка за изграждане на графични приложения и осигурява висока производителност и стабилност.

### Система за управление на бази данни

В проекта се използва **MySQL** – релационна база от данни с отворен код, която е доказана в малки и средни бизнес системи. Сред възможните алтернативи са **PostgreSQL**, **SQLite** и **Microsoft SQL Server**.

PostgreSQL е известна със своята мощ и разширени възможности, включително по-строги правила за транзакции, поддръжка на JSON и сложни заявки. Въпреки това, за мащаба на настоящото приложение тя би представлявала по-сложен избор без конкретни предимства.

SQLite е изключително лека и подходяща за мобилни или вградени системи, но не е предназначена за многопотребителска работа и не поддържа паралелни връзки на същото ниво като MySQL.

Microsoft SQL Server е мощна и богата на функции платформа, но е по-тежка за внедряване и изисква лицензиране при по-големи приложения. Също така интеграцията ѝ с Visual Studio е отлична, но за отворен проект като този MySQL остава предпочитан.

В обобщение, MySQL осигурява лекота на използване, добър контрол над структурата на данните и лесна интеграция с C# чрез ADO.NET и библиотеката MySql.Data.

### Тип платформа: настолна, уеб или мобилна

Системата е реализирана като **настолно приложение**, тъй като тя е предназначена за вътрешна употреба от служители – например в търговски офис, склад или магазин. Desktop приложението предлага висока производителност, не зависи от интернет свързаност и осигурява пълен контрол върху потребителския интерфейс.

Ако системата беше реализирана като **уеб приложение**, това би увеличило достъпността ѝ – тя би могла да се използва от различни устройства и локации, но би изисквало значително по-сложна архитектура, поддръжка на сървър, сигурност при работа в интернет и допълнителни знания (HTML, CSS, JavaScript, backend framework).

Мобилната платформа (Android/iOS) също е възможна опция в бъдеще, но тя изисква отделна реализация, обикновено чрез допълнителни технологии като Xamarin или MAUI (за C#), Flutter (Dart) или React Native.

Като първа версия, desktop реализацията е най-подходяща: тя е бърза, стабилна, позволява локално съхранение и максимален контрол над интерфейса.

### Среда за разработка

В проекта е използвана **Visual Studio** – водеща интегрирана среда за разработка, особено подходяща за C# и .NET приложения. Тя предлага мощен GUI дизайнер, отстраняване на грешки (debugger), интеграция с база данни, управление на зависимости (NuGet), както и удобна работа с формуляри (Windows Forms).

Алтернативите като **IntelliJ IDEA** (за Java) или **VS Code** (многоезикова лека среда) имат своите предимства, но не осигуряват същата пълноценна и интегрирана поддръжка за графично програмиране с C#.

### Обобщение

Изборът на C#, MySQL, Visual Studio и Windows Forms е резултат от внимателен баланс между производителност, достъпност, лекота на реализация и стабилност. Тези технологии са достатъчно мощни, за да изградят завършена и надеждна система, като същевременно са гъвкави за бъдещо разширение. Възможностите за надграждане към уеб или мобилни платформи остават отворени, ако бизнес нуждите се разширят.

## Работа с бази от данни в C#

Една от основните цели на проекта е да осигури ефективно и стабилно взаимодействие между приложението и базата от данни. За тази цел е използван езикът **C#** в средата **Visual Studio**, в комбинация с **MySQL** като система за управление на релационни бази от данни. Свързването между тях е реализирано чрез технологията **ADO.NET**, която предоставя пълен контрол върху SQL операциите, като същевременно гарантира сигурност и гъвкавост.

### ADO.NET и MySql.Data

За свързване с MySQL в проекта е използвана библиотеката **MySql.Data**, която се импортира чрез **NuGet пакета** MySql.Data и се използва за създаване на обекти като:

* MySqlConnection – представлява връзката със сървъра;
* MySqlCommand – изпълнява SQL команди;
* MySqlDataReader – чете резултати от SELECT заявки;
* MySqlDataAdapter и DataTable – използвани за зареждане на данни в DataGridView.

Чрез този подход се реализират директни SQL заявки и се осигурява пълен контрол върху базовите операции.

### Отваряне и затваряне на връзка

Всяка SQL операция изисква отваряне на връзка към базата чрез обект MySqlConnection. Добра практика е тя да се отваря непосредствено преди операцията и да се затваря веднага след това:

string connStr = "server=localhost;user=root;password=1234;database=onlinestore;";

MySqlConnection con = new MySqlConnection(connStr);

con.Open();

// изпълнение на заявка

con.Close();

### Изпълнение на SQL команди

В зависимост от типа на операцията, се използват различни методи:

**➤ ExecuteNonQuery()**

Използва се при заявки, които не връщат резултат – INSERT, UPDATE, DELETE:

MySqlCommand cmd = new MySqlCommand("INSERT INTO customers (FirstName, LastName) VALUES (@FirstName, @LastName)", con);

cmd.Parameters.AddWithValue("@FirstName", "Иван");

cmd.Parameters.AddWithValue("@LastName", "Иванов");

**➤ ExecuteReader()**

Използва се при заявки SELECT, които връщат множество редове:

cmd.ExecuteNonQuery();

MySqlCommand cmd = new MySqlCommand("SELECT \* FROM products", con);

MySqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

string name = reader["Name"].ToString();

}

reader.Close();

**➤ ExecuteScalar()**

Използва се за заявки, които връщат една стойност – напр. брой записи или агрегатна стойност:

MySqlCommand cmd = new MySqlCommand("SELECT COUNT(\*) FROM orders", con);

int count = Convert.ToInt32(cmd.ExecuteScalar());

### Параметризирани заявки

За защита от **SQL инжекции** се използват параметри, вместо директно вграждане на стойности в заявките. Това е изключително важна част от сигурността на приложението:

cmd.Parameters.AddWithValue("@Email", tbEmail.Text);

### Зареждане на данни в DataGridView

Информацията от базата се визуализира в таблица чрез обект DataTable:

MySqlDataAdapter da = new MySqlDataAdapter("SELECT \* FROM customers", con);

DataTable dt = new DataTable();

da.Fill(dt);

dataGridView1.DataSource = dt;

Този подход позволява автоматично обновяване на данните след всяка операция.

### Обработка на грешки

Всяка операция е обвита в try-catch блок, който гарантира, че при възникване на грешка – като проблем с връзката, невалидни данни или синтактична грешка в SQL – потребителят получава обратна връзка чрез MessageBox:

try

{

cmd.ExecuteNonQuery();

MessageBox.Show("Успешно добавяне!");

}

catch (Exception ex)

{

MessageBox.Show("Грешка: " + ex.Message);

}

### Структура и организация на кода

Всеки CRUD метод е свързан с бутон в интерфейса, като например:

* btRegistrationProducts\_Click – добавяне на продукт;
* btEditOrder\_Click – редактиране на поръчка;
* btRemoveOrder\_Click – изтриване с възстановяване на наличност;
* btFindCustomers\_Click – търсене на клиент по ID.

Това осигурява ясен и модулен подход, при който всеки компонент отговаря за една конкретна операция.

### Заключение

Работата с базата от данни в C# чрез **ADO.NET** и **MySQL** е ключова част от проекта. Тя осигурява стабилна комуникация между интерфейса и сървъра, позволява ефективна манипулация на данните и осигурява контрол върху сигурността. Използваният подход е едновременно гъвкав и професионален, позволяващ надграждане и поддръжка на системата в дългосрочен план.

## Архитектура на системата

Функционалните характеристики представляват основата на всяко софтуерно приложение. Те дефинират какво ще може да прави системата, как ще комуникира с потребителя и какви операции ще се извършват върху данните. Разработеното в рамките на тази дипломна работа приложение за управление на онлайн магазин за електроника е ориентирано към улесняване на процесите по продажба, инвентаризация и обслужване на клиенти. Системата е разработена с помощта на **C#** в **Visual Studio** и използва **MySQL** база от данни, управлявана чрез **MySQL Workbench**. Основната ѝ цел е да предостави цялостно решение за управление на продукти, поръчки, клиенти и административни дейности, като в същото време поддържа високо ниво на сигурност и ефективност.

### Основни функционалности

* **Управление на продукти**: Приложението позволява добавяне, редактиране и изтриване на продукти. Всеки продукт съдържа информация за име, описание, цена, наличност, категория и тип (настолен компютър, лаптоп или сървър). Категоризацията помага на потребителите да навигират по-лесно из продуктовия каталог, а интегрираният механизъм за филтриране и търсене осигурява бърз достъп до желаните артикули.
* **Управление на клиенти**: В системата се поддържа детайлна база данни с клиентска информация, включително пълно име, телефонен номер, имейл адрес и физически адрес. Това позволява персонализирано обслужване и по-добро планиране на маркетингови стратегии.
* **Създаване и управление на поръчки**: Приложението предоставя възможност за създаване на поръчки чрез лесен и интуитивен интерфейс. Системата генерира уникален номер на поръчката, отбелязва дата и час на извършване, и съхранява пълна информация за транзакцията. Статусите на поръчките (на изчакване, обработка, завършена, отказана) се управляват динамично.
* **Филтриране и търсене**: Реализиран е механизъм за бързо и ефективно намиране на продукти и клиенти по различни критерии: име, ID, категория, цена, наличност. Това значително улеснява потребителя при работа с големи обеми от данни и подобрява цялостната потребителска ефективност.

Административен панел Системата разполага със защитен административен панел, предназначен за използване от персонал с високо ниво на достъп. Функционалностите включват:

* Достъп до пълната база данни и всички таблици
* Възможност за изпълнение на CRUD операции (Create, Read, Update, Delete) върху всички обекти
* Управление на категории и наличности

Потребителски интерфейс Потребителският интерфейс е реализиран с помощта на **Windows Forms** в средата на Visual Studio. Спазени са основни принципи на UX/UI дизайн:

* **Интуитивност**: Интерфейсът е логически структуриран, така че дори потребители без задълбочени технически познания да могат лесно да навигират в системата.
* **Удобство**: Формите са минималистични и функционални, с добре подредени контроли, осигуряващи бърз достъп до всички основни действия.
* **Скорост и производителност**: Използваните елементи са леки и оптимизирани за бързо зареждане и реакция, което прави приложението подходящо дори за по-стари машини.
* **Адаптивност**: Интерфейсът е разработен така, че да може лесно да бъде модифициран в бъдещи версии, в случай че се премине към уеб-базирана реализация.

Бъдещи подобрения и възможности за разширяване Предвид динамичното развитие на електронната търговия и нуждите на бизнеса, системата е проектирана така, че да може да бъде разширявана. Някои от възможните бъдещи подобрения включват:

* **Интеграция с онлайн разплащателни системи** – за осигуряване на автоматизирани разплащания чрез PayPal, кредитни/дебитни карти и др.
* **Журнал на действията** – история на административни дейности за целите на проследимост и сигурност
* **Преглед на системна статистика в реално време** (общ брой поръчки, среден приход, най-продавани артикули)
* **Изпращане на известия и напомняния** – чрез имейл или SMS при нова поръчка, промоции или нужда от доставка
* **Система за лоялни клиенти** – начисляване на точки при всяка покупка, които могат да се обменят за отстъпки или подаръци
* **Миграция към уеб базирана платформа** – използване на ASP.NET и Blazor за създаване на уеб версия с достъп от всяко устройство
* **Интеграция с мобилно приложение** – за мобилен достъп до най-важните функции на системата от страна на администратори и клиенти
* **Аналитични модули с графики и визуализации** – за по-ефективен бизнес анализ и прогнозиране
* **Справки и отчети** – генерирането на отчети, която подпомага мениджърския процес

Разработената функционалност поставя стабилна основа за създаване на пълноценна и професионална система за управление на онлайн магазин за електроника, която е в състояние да се развива и адаптира спрямо бъдещите нужди и технологични тенденции.

## Функционални характеристики на приложението

Архитектурата на разработената система играе ключова роля в осигуряването на стабилна, разширяема и сигурна платформа за електронна търговия. Избраната архитектурна структура следва принципите на слоест дизайн (multi-layered architecture), при който логиката на приложението е разделена на отделни слоеве, всеки със свои отговорности. Това позволява лесна поддръжка, подобрения и повторна употреба на кода.

Системата се състои от следните основни слоеве:

Презентационен слой (Presentation Layer) Този слой отговаря за взаимодействието с крайния потребител. Реализиран е чрез **Windows Forms** в **Visual Studio** с помощта на **C#**. Интерфейсът е съобразен с основните принципи на удобство за потребителя (UX) и включва:

* Форма за вход с валидация на потребителски имена и пароли
* Форми за CRUD операции (създаване, редактиране, изтриване и преглед)
* Динамично зареждане на данни от базата чрез DataGridView компоненти
* Потвърждения за действия и съобщения за грешки

Логически слой (Business Logic Layer) Този слой реализира бизнес логиката на приложението, като обработва данните, получени от потребителския интерфейс, и ги подготвя за съхранение в базата. Отговорностите му включват:

* Проверка на валидност на въведени данни
* Изчисление на обща стойност на поръчките
* Генериране на уникални идентификатори
* Управление на статуса на поръчки
* Комуникация с логиката за отчетност и филтриране

Данни и достъп до база данни (Data Access Layer) Този слой осъществява връзката с **MySQL** базата данни чрез използване на **MySql.Data** библиотеката. Всички SQL заявки и операции с базата преминават през този слой. Основните функции включват:

* Изпълнение на заявки за извличане и обновяване на информация (SELECT, INSERT, UPDATE, DELETE)
* Свързване към базата чрез стрингове за връзка (connection strings)
* Работа с параметризирани заявки с цел предотвратяване на SQL инжекции
* Прехвърляне на данни към логическия слой чрез обекти и структури

База данни (Database Layer) Базата данни е създадена и поддържана с помощта на **MySQL Workbench** и се състои от няколко взаимносвързани таблици:

* **Customers** – съдържа информация за клиентите (име, имейл, адрес, телефон)
* **Products** – информация за продуктите (име, цена, количество, категория, производител)
* **Orders** – информация за поръчките (дата, клиент, количество, състояние, продукт)
* **Suppliers** – информация за доставчиците (име, имейл, телефон)
* Всяка таблица е свързана чрез външни ключове (foreign keys), което осигурява цялостта на данните и логическите зависимости между обектите

Възможности за разширяване Архитектурата е гъвкава и предразполага към бъдещи надграждания, като:

* Миграция към уеб-базирано приложение чрез **ASP.NET Core MVC**
* Добавяне на REST API за свързване с външни системи
* Включване на допълнителни модули – плащания, известия, мобилни приложения
* Използване на нива на достъп – администраторите имат пълен контрол, а потребителите – ограничени възможности
* Защита на вход чрез хеширане на пароли (при евентуална бъдеща реализация)

Тази архитектура гарантира стабилност, мащабируемост и ясна организация на кода. Тя позволява лесно откриване и отстраняване на грешки, както и бързо адаптиране към нови изисквания или промени в бизнес логиката на приложението.

## Заключение

Разработката на системата за управление на електронен магазин за техника представлява успешно реализиране на съвременен софтуерен проект, който интегрира ключови принципи на обектно-ориентираното програмиране, структурирано моделиране на бази данни и съвременни технологии за разработка. Чрез използването на **Visual Studio** и езика **C#** в комбинация с **MySQL Workbench** и **MySQL** базата данни, бе създадено надеждно и функционално приложение, отговарящо на реални бизнес нужди.

Системата обхваща всички основни аспекти на управление на онлайн магазин – от регистриране и управление на клиенти, през организиране и поддръжка на продуктова гама, до извършване на поръчки и проследяване на техния статус. Изграденият потребителски интерфейс е ясен и достъпен, като осигурява високо ниво на използваемост, а архитектурата на системата гарантира възможности за лесна поддръжка, развитие и адаптация към бъдещи изисквания.

По време на разработката бяха приложени редица добри практики като разделяне на логическите отговорности чрез слоест модел, използване на параметризирани заявки за сигурност срещу SQL инжекции, и модулно структуриране на функционалностите. Това не само повиши качеството на крайния продукт, но и допринесе за професионално развитие в работата с реални софтуерни инструменти и технологии.

Системата разполага с потенциал за надграждане чрез добавяне на модули за електронни плащания, интеграция с външни услуги чрез API, мобилна версия на приложението и дори преминаване към облачна архитектура. По този начин настоящата дипломна работа поставя здрава основа, върху която могат да се изграждат по-сложни решения, отговарящи на нарастващите нужди на съвременния дигитален бизнес.

Разработката демонстрира как с помощта на достъпни, но мощни технологии като **C#** и **MySQL**, може да бъде създадено професионално софтуерно решение, което ефективно автоматизира и оптимизира процесите в един реален бизнес контекст.

# Раздел II – Практическа част

В този раздел се представя реализацията на системата за управление на онлайн магазин за електроника на практика. Тук се разглежда как работи приложението чрез неговия графичен интерфейс, логика на взаимодействие с базата от данни и реални примери за изпълнение на операции като създаване, редактиране и изтриване на записи. Разработката е извършена с помощта на **Visual Studio**, език за програмиране **C#** и базата данни **MySQL**, проектирана с **MySQL Workbench**.

## Обща структура на системата

Системата се състои от две основни части:

* **Клиентска част (Windows Forms приложение)** – отговаря за потребителския интерфейс и взаимодействие с потребителя;
* **Сървърна част (MySQL база от данни)** – съхранява данните и обслужва заявки, изпращани от приложението.

Клиентската част се грижи за визуализиране на съдържание, приемане на вход от потребителя и логическа обработка на действия като добавяне на нови записи, редактиране и търсене. Сървърната част се състои от ясно структуриран модел от таблици, свързани с външни ключове и логически зависимости, поддържащи цялостта на информацията и улесняващи нейната обработка.

## База от данни – структура и свързаност

Базата от данни е проектирана с помощта на **ER диаграма**, разработена в **MySQL Workbench**. Тя включва следните основни таблици:

* **products** – съдържа информация за наличните продукти: име, категория, цена, наличност и доставчик;
* **customers** – съдържа данни за клиентите: име, фамилия, имейл, телефон и адрес;
* **orders** – съхранява информация за направените поръчки: клиент, продукт, количество, дата и статус;
* **suppliers** – съдържа данни за доставчиците на електроника: име, имейл, телефон.

Таблиците са свързани помежду си чрез **външни ключове**. Например, всяка поръчка в таблицата orders е свързана с клиент от customers и продукт от products. Това осигурява **референтна цялост** на информацията и улеснява проследяването на транзакции и наличности. Таблицата products съдържа връзка към suppliers, позволявайки лесно проследяване откъде е закупен всеки артикул. Всички полета са нормализирани, така че да се избягва дублиране на данни и да се улесни бъдещата поддръжка.

## Работа с формуляри и действия в интерфейса

Приложението разполага с няколко форми за работа с различни данни:

* **LoginForm** – вход в системата чрез потребителско име и парола. След успешно логване, потребителят бива пренасочен към главния интерфейс.

A screenshot of a log in

AI-generated content may be incorrect.

* **ProcessingForm** – главният прозорец за работа с базата данни. Съдържа избор на таблица, визуализация на записи в DataGridView и функционалност за CRUD операции чрез бутони и входни полета.

A screenshot of a computer

AI-generated content may be incorrect.

Потребителят избира таблица от падащо меню и според това се зарежда съответният панел:

* При избор на **products** – се активират полетата свързани с атрибутите на **products**: ProductID, Name, Category, Price, QuantityInStock, SupplierID ;
* При избор на **customers** – се активират полетата свързани с атрибутите на **customers**: CustomerID, FirstName, LastName, Email, Phone, Address ;
* При избор на **orders** – се активират полетата свързани с атрибутите на **orders**: OrderID, CustomerID, ProductID, Quantity, OrderDate, Status ;
* При избор на **suppliers** – се активират полетата свързани с атрибутите на **suppliers**: SupplierID, Name, ContactEmail, Phone ;

CRUD операциите са реализирани чрез SQL команди (INSERT, UPDATE, DELETE, SELECT), които се изпълняват чрез MySqlCommand и MySqlDataAdapter в C#. Всяко действие е обвито в try-catch блок, който показва съобщение за успех или грешка чрез MessageBox. За валидация се използват регулярни изрази, които гарантират правилен формат на телефонни номера и имейл адреси.

## Управление на формата за вход (LoginForm)

Формата за вход (LoginForm) е ключов компонент в сигурността и управлението на достъпа до приложението. Тя служи като първичен филтър, който контролира дали потребителят има право да използва системата. Интерфейсът включва полета за въвеждане на потребителско име и парола, както и бутони за вход, изчистване на полетата и изход от програмата.

### Основни функционалности на LoginForm:

* **Проверка на въведените данни:** При натискане на бутона „Вход“, се извършва валидация на въведеното потребителско име и парола. Ако данните съвпадат с валиден запис от таблица с администратори или служители (в зависимост от имплементацията), потребителят се допуска до приложението.
* **Изчистване на полетата:** При натискане на бутон „Изчисти“, съдържанието на текстовите полета автоматично се премахва, което позволява бързо въвеждане на нови данни.
* **Показване и скриване на парола:** С помощта на чекбокс или бутон, потребителят може да визуализира или скрие паролата си, което е полезно при въвеждане на по-дълги или по-сложни комбинации.
* **Изход от приложението:** Бутон „Изход“ затваря приложението изцяло, като се използва методът Application.Exit() или this.Close().

**Преход към главната форма (ProcessingForm):**

При успешно влизане, LoginForm извиква нова инстанция на ProcessingForm и скрива себе си. За да се гарантира правилното поведение при връщане назад (например при натискане на „Назад“ в главната форма), LoginForm се подава като параметър на конструктора на ProcessingForm. Така при връщане, тя се показва отново и полетата за вход се изчистват автоматично чрез метода Clear()

### Автоматично поведение при повторно показване:

За да бъде потребителското изживяване още по-интуитивно, се използва събитие Activated, което се задейства всеки път когато LoginForm се върне на преден план. То автоматично изчиства полетата и подготвя формата за нов вход:

private void LogInForm\_Activated(object sender, EventArgs e)

{

tbUsername.Clear();

tbPassword.Clear();

tbUsername.Focus();

}

Тази логика гарантира, че при всяко връщане потребителят ще започне отначало, без да се притеснява, че предишните данни са останали въведени.

## Примерен сценарий: добавяне на нов клиент

1. Потребителя се свързва с базата данни чрез бутона „Свързване“



1. Избира таблицата „customers“ от падащото меню;

A blue and white stripe

AI-generated content may be incorrect.

1. Въвежда следните данни:
   * CustomerID: (автоматично се генерира, но можете и ръчно да се напише. Ако искате автоматично да ви го създаде просто оставете полето празно)
   * FirstName: Георги
   * LastName: Иванов
   * Email: [g\_ivanov@abv.bg](mailto:g_ivanov@abv.bg)
   * Phone: 0888123456
   * Address: гр. София, ул. Лиляна 5

A screenshot of a phone

AI-generated content may be incorrect.

1. Натиска бутона „Записване“;



1. В системата се изпълнява SQL заявка и се появява съобщение „Добавихте успешно клиент!“;

A screenshot of a computer

AI-generated content may be incorrect.

1. Данните автоматично се обновяват в таблицата отляво.

A screenshot of a computer

AI-generated content may be incorrect.

## Примерен сценарий: редактиране на вече създадена поръчка

1. Потребителят избира таблицата „orders“ от падащото меню;

A blue and white striped background

AI-generated content may be incorrect.

1. Въвежда:
   * ID на вече създадена поръчкаA screenshot of a computer

     AI-generated content may be incorrect.
2. Натиска бутона „Намери“ (продуктът се намира чрез неговото ID);  
   A screenshot of a phone

   AI-generated content may be incorrect.
3. Редактира някое от вече запълнените полета:   
   Status: In Progress -> Delivered

A screenshot of a computer screen

AI-generated content may be incorrect.

1. Натиска бутона „Редактиране“;



1. В системата се изпълнява SQL заявка и се появява съобщение „Поръчката беше успешно редактиран!“;

A screenshot of a computer

AI-generated content may be incorrect.

1. Данните автоматично се обновяват в таблицата отляво.

A screenshot of a table

AI-generated content may be incorrect.

## Примерен сценарий: изтриване на продукт

1. Потребителят избира таблицата „products“ от падащото меню;

A blue and white stripe

AI-generated content may be incorrect.

1. Въвежда:
   * ID на вече създаден продукт

A screenshot of a computer

AI-generated content may be incorrect.

1. Натиска бутона „Намери“ (продуктът се намира чрез неговото ID);

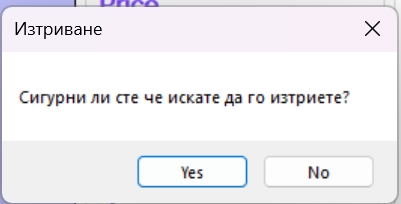
A screenshot of a computer

AI-generated content may be incorrect.

1. Натиска бутона „ Изтриване“;



1. Появява се съобщение "Сигурни ли сте че искате да го изтриете?", натискате бутона "Yes"



1. В системата се изпълнява SQL заявка и се появява съобщение „Успешно изтрихте продукта!“;

A screenshot of a computer

AI-generated content may be incorrect.

1. Данните автоматично се обновяват в таблицата отляво.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Взаимодействие с базата от данни

Приложението използва директна връзка с MySQL сървъра чрез MySqlConnection. Всеки бутон изпълнява SQL заявка:

* **SELECT** – зарежда данни в таблицата;
* **INSERT** – добавя нов ред към таблицата;
* **UPDATE** – редактира избран запис;
* **DELETE** – премахва съществуващ запис.

При зареждане на таблиците, информацията се визуализира в **DataGridView**, а потребителят може да извършва действия чрез контролите в десния панел. Данните се зареждат в реално време, без нужда от презареждане на цялото приложение. Това позволява бърза и ефективна работа дори при големи обеми информация.

## Заключение

Практическата реализация на системата демонстрира успешно взаимодействие между потребителски интерфейс и база данни. Всички основни функции – логване, добавяне на продукти, регистриране на клиенти, обработка на поръчки – са реализирани чрез интуитивни форми и ясно структурирана логика. Системата показва стабилност и бърза реакция при работа с реални данни, което я прави подходяща за внедряване в реална бизнес среда. Проектираната архитектура позволява лесна поддръжка и разширяемост, като същевременно предоставя солидна основа за бъдещи технологични надграждания и оптимизации.

# Раздел III - Източници

 Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). **Database System Concepts** (6th ed.). McGraw-Hill Education.  
*(За: Основи на релационните бази данни, нормализация, видове връзки и SQL)*

 Microsoft Docs. (n.d.). **ADO.NET overview**.   
<https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/>  
*(За: ADO.NET и връзка с бази от данни в C#)*

 Oracle (MySQL). (n.d.). **MySQL 8.0 Reference Manual**. Retrieved from:  
<https://dev.mysql.com/doc/>  
*(За: SQL заявки, структура на база от данни, FOREIGN KEY връзки)*

 Microsoft Docs. (n.d.). **Windows Forms controls**.   
<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/>  
*(За: TextBox, ComboBox, DateTimePicker, DataGridView и други контроли)*

 Freeman, A., & MacDonald, M. (2021). **Pro C# 9 with .NET 5**. Apress.  
*(За: C# като език, обектно-ориентирано програмиране, архитектура)*

 Visual Studio. (n.d.). **Product documentation and tutorials**.   
<https://learn.microsoft.com/en-us/visualstudio/>  
*(За: Visual Studio, създаване на desktop приложение, UI дизайнер)*

 MySQL Workbench Team. (n.d.). **MySQL Workbench Manual**.   
<https://dev.mysql.com/doc/workbench/en/>  
*(За: Създаване на ER диаграми, моделиране на бази данни)*

 Microsoft Learn. (n.d.). **Desktop application development with Windows Forms**.   
<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/>  
*(За: UI/UX подходи и управление на събития в WinForms)*

 Stack Overflow (2023). Различни въпроси и примери, свързани с C#, SQL и ADO.NET.  
*(Използвано за справки при обработка на грешки, параметризирани заявки и оптимизация)*

 Geeks for Geeks. (n.d.). **Difference between MySQL, PostgreSQL, SQLite and SQL Server**.  
*(За: Сравнение между СУБД – MySQL, PostgreSQL, SQLite и SQL Server)*

# Раздел IV - Кодът

## LoginForm

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DataBaseApp

{

public partial class LogInForm : Form

{

// Constructor for the LogInForm

public LogInForm()

{

InitializeComponent();

}

// Event which checks if the inputted username and password are correct and then if so,

// opens the ProcessingForm

private void button1\_Click(object sender, EventArgs e)

{

if (tbUsername.Text == "admin" && tbPassword.Text == "1234")

{

ProcessingForm pf = new ProcessingForm(this);

Hide();

pf.Show();

}

else

{

MessageBox.Show("Invalid username or password!");

tbUsername.Clear();

tbPassword.Clear();

tbUsername.Focus();

}

}

// Event which clears the fields and sets the focus to the username field

private void label2\_Click(object sender, EventArgs e)

{

tbUsername.Clear();

tbPassword.Clear();

tbUsername.Focus();

}

// Event which closes the application

private void label3\_Click(object sender, EventArgs e)

{

Application.Exit();

}

// Flag to check if the password is visible or not

private bool privacyFlag = true;

// Event which changes the visibility of the password

private void pictureBox4\_Click(object sender, EventArgs e)

{

if(privacyFlag == true)

{

pictureBox4.Image = Image.FromFile(@"../Debug/images/password\_privateOFF.png");

privacyFlag = false;

}

else

{

pictureBox4.Image = Image.FromFile(@"../Debug/images/password\_privateON.png");

privacyFlag = true;

}

tbPassword.UseSystemPasswordChar = !tbPassword.UseSystemPasswordChar;

}

// Event that clears the fields and sets the focus to the username field when the form is activated

private void LogInForm\_Activated(object sender, EventArgs e)

{

tbUsername.Clear();

tbPassword.Clear();

tbUsername.Focus();

}

}

}

## ProcessingForm

using MySql.Data.MySqlClient;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics.Eventing.Reader;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DataBaseApp

{

public partial class ProcessingForm : Form

{

// Declaration of variables

private string MyDataBase { get; set; }

private LogInForm loginForm;

MySqlConnection conDataBase1 = new MySqlConnection();

// Flag for checking if the form is closed manually

private bool IsManuallyClosed = true;

// Constructor for the ProcessingForm

public ProcessingForm(LogInForm lf)

{

InitializeComponent();

// Initialization of the connection to the database

MyDataBase = "server=localhost;uid=root;pwd=1234;Database=electronics\_company";

conDataBase1.ConnectionString = MyDataBase;

loginForm = lf;

}

// Event that shows the selected table in DataGridView and enables the corresponding group

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

dataGridView1.DataSource = null;

gbProducts.Enabled = false;

gbCustomers.Enabled = false;

gbOrders.Enabled = false;

gbSuppliers.Enabled = false;

btEditProduct.Enabled = false;

btEditCustomer.Enabled = false;

btEditOrder.Enabled = false;

btEditSupplier.Enabled = false;

btRemoveProduct.Enabled = false;

btRemoveCustomer.Enabled = false;

btRemoveOrder.Enabled = false;

btRemoveSupplier.Enabled = false;

ClearProductsGB();

ClearCustomersGB();

ClearOrdersGB();

ClearSuppliersGB();

if (radioButton2.Checked == true){}

else if (comboBox1.SelectedIndex == 0)

{

gbProducts.Enabled = true;

ShowTable(MyDataBase, dataGridView1, "products");

}

else if (comboBox1.SelectedIndex == 1)

{

gbCustomers.Enabled = true;

ShowTable(MyDataBase, dataGridView1, "customers");

}

else if (comboBox1.SelectedIndex == 2)

{

gbOrders.Enabled = true;

ShowTable(MyDataBase, dataGridView1, "orders");

}

else if(comboBox1.SelectedIndex == 3)

{

gbSuppliers.Enabled = true;

ShowTable(MyDataBase, dataGridView1, "suppliers");

}

}

// Event that connects to the database and shows the selected table

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

comboBox1.Enabled = true;

if (radioButton1.Checked == true)

{

conDataBase1.Open();

ShowTable(MyDataBase, dataGridView1, "products");

comboBox1.Items.Add("products");

comboBox1.Items.Add("customers");

comboBox1.Items.Add("orders");

comboBox1.Items.Add("suppliers");

comboBox1.SelectedItem = "products";

radioButton1.Enabled = false;

radioButton2.Enabled = true;

}

}

// Event that disconnects from the database and brings back the default state of the form

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

comboBox1.Enabled = false;

if (radioButton2.Checked == true)

{

dataGridView1.DataSource = null;

this.dataGridView1.Rows.Clear();

comboBox1.SelectedIndex = -1;

comboBox1.Items.Clear();

conDataBase1.Close();

radioButton1.Enabled = true;

radioButton2.Enabled = false;

}

}

// <-------------------------->

// Methods for adding records

private void btRegistrationProducts\_Click(object sender, EventArgs e)

{

if (Decimal.Parse(tb\_products\_Price.Text) < 0 || int.Parse(tb\_products\_QuantityInStock.Text) < 0)

{

MessageBox.Show("Цената и количеството не могат да бъдат отрицателни!");

return;

}

try

{

MySqlCommand cmdDataBase1 = new MySqlCommand("insert into products (Name,Category,Price,QuantityInStock,SupplierID) " +

"values(@Name,@Category,@Price,@QuantityInStock,@SupplierID)", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@Name", (tb\_products\_Name.Text));

cmdDataBase1.Parameters.AddWithValue("@Category", (cb\_products\_Category.SelectedIndex + 1));

cmdDataBase1.Parameters.AddWithValue("@Price", (tb\_products\_Price.Text));

cmdDataBase1.Parameters.AddWithValue("@QuantityInStock", (tb\_products\_QuantityInStock.Text));

cmdDataBase1.Parameters.AddWithValue("@SupplierID", (tb\_products\_SupplierID.Text));

cmdDataBase1.ExecuteNonQuery();

MessageBox.Show("Добавихте Успешно!");

ShowTable(MyDataBase, dataGridView1, "products");

}

catch (Exception ex)

{

MessageBox.Show("Грешка при добавяне на продукта: " + ex.Message);

}

}

private void btRegistrationCustomers\_Click(object sender, EventArgs e)

{

try

{

MySqlCommand cmdDataBase1 = new MySqlCommand

("insert into customers (FirstName,LastName,Phone,Email,Address) " +

"values(@FirstName,@LastName,@Phone,@Email,@Address)", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@FirstName", (tb\_customers\_FirstName.Text));

cmdDataBase1.Parameters.AddWithValue("@LastName", (tb\_customers\_LastName.Text));

cmdDataBase1.Parameters.AddWithValue("@Phone", (tb\_customers\_Phone.Text));

cmdDataBase1.Parameters.AddWithValue("@Email", (tb\_customers\_Email.Text));

cmdDataBase1.Parameters.AddWithValue("@Address", (tb\_customers\_Address.Text));

cmdDataBase1.ExecuteNonQuery();

MessageBox.Show("Добавихте Успешно!");

ShowTable(MyDataBase, dataGridView1, "customers");

}

catch(Exception ex)

{

MessageBox.Show("Грешка при добавяне на клиента: " + ex.Message);

}

}

private void btRegistrationOrders\_Click(object sender, EventArgs e)

{

try

{

int productId = int.Parse(tb\_orders\_ProductID.Text);

int quantityOrdered = int.Parse(tb\_orders\_Quantity.Text);

MySqlCommand checkStock = new MySqlCommand(

"SELECT QuantityInStock FROM products WHERE ProductID = @ProductID", conDataBase1);

checkStock.Parameters.AddWithValue("@ProductID", productId);

int currentStock = Convert.ToInt32(checkStock.ExecuteScalar());

if (currentStock < quantityOrdered)

{

MessageBox.Show("Няма достатъчно наличност от този продукт!");

return;

}

MySqlCommand cmdInsert = new MySqlCommand(

"INSERT INTO orders (CustomerID, ProductID, Quantity, OrderDate, Status) " +

"VALUES (@CustomerID, @ProductID, @Quantity, @OrderDate, @Status)", conDataBase1);

cmdInsert.Parameters.AddWithValue("@CustomerID", tb\_orders\_CustomerID.Text);

cmdInsert.Parameters.AddWithValue("@ProductID", productId);

cmdInsert.Parameters.AddWithValue("@Quantity", quantityOrdered);

cmdInsert.Parameters.AddWithValue("@OrderDate", tb\_orders\_OrderDate.Text);

cmdInsert.Parameters.AddWithValue("@Status", cb\_orders\_Status.SelectedIndex + 1);

cmdInsert.ExecuteNonQuery();

MySqlCommand updateStock = new MySqlCommand(

"UPDATE products SET QuantityInStock = QuantityInStock - @Qty WHERE ProductID = @ProductID", conDataBase1);

updateStock.Parameters.AddWithValue("@Qty", quantityOrdered);

updateStock.Parameters.AddWithValue("@ProductID", productId);

updateStock.ExecuteNonQuery();

MessageBox.Show("Поръчката беше добавена успешно и количеството е обновено!");

ShowTable(MyDataBase, dataGridView1, "orders");

}

catch (Exception ex)

{

MessageBox.Show("Грешка при добавяне на поръчката: " + ex.Message);

}

}

private void btRegistrationSuppliers\_Click(object sender, EventArgs e)

{

try

{

MySqlCommand cmdDataBase1 = new MySqlCommand("insert into suppliers (Name,ContactEmail,Phone) " +

"values(@Name,@ContactEmail,@Phone)", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@Name", (tb\_suppliers\_Name.Text));

cmdDataBase1.Parameters.AddWithValue("@ContactEmail", (tb\_suppliers\_ContactEmail.Text));

cmdDataBase1.Parameters.AddWithValue("@Phone", (tb\_suppliers\_Phone.Text));

cmdDataBase1.ExecuteNonQuery();

MessageBox.Show("Добавихте Успешно!");

ShowTable(MyDataBase, dataGridView1, "suppliers");

}

catch(Exception ex)

{

MessageBox.Show("Грешка при добавяне на доставчика: " + ex.Message);

}

}

// <-------------------------->

// <-------------------------->

// Methods for finding records

private void btFindProducts\_Click(object sender, EventArgs e)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("SELECT Name, Category, Price, QuantityInStock, SupplierID FROM products WHERE ProductID = @ProductID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@ProductID", tb\_products\_ProductID.Text);

MySqlDataReader reader = cmdDataBase1.ExecuteReader();

if (reader.Read())

{

tb\_products\_Name.Text = reader["Name"].ToString();

cb\_products\_Category.Text = reader["Category"].ToString();

tb\_products\_Price.Text = reader["Price"].ToString();

tb\_products\_QuantityInStock.Text = reader["QuantityInStock"].ToString();

tb\_products\_SupplierID.Text = reader["SupplierID"].ToString();

MessageBox.Show("Поръчката е намерена!");

btRemoveProduct.Enabled = true;

btEditProduct.Enabled = true;

}

else

{

MessageBox.Show("Поръчката не е намерена!");

btRemoveProduct.Enabled = false;

btEditProduct.Enabled = false;

}

reader.Close();

}

private void btFindCustomers\_Click(object sender, EventArgs e)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("SELECT FirstName, LastName, Email, Phone, Address FROM customers WHERE CustomerID = @CustomerID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@CustomerID", tb\_customers\_CustomerID.Text);

MySqlDataReader reader = cmdDataBase1.ExecuteReader();

if (reader.Read())

{

tb\_customers\_FirstName.Text = reader["FirstName"].ToString();

tb\_customers\_LastName.Text = reader["LastName"].ToString();

tb\_customers\_Email.Text = reader["Email"].ToString();

tb\_customers\_Phone.Text = reader["Phone"].ToString();

tb\_customers\_Address.Text = reader["Address"].ToString();

MessageBox.Show("Клиента е намерен!");

btRemoveCustomer.Enabled = true;

btEditCustomer.Enabled = true;

}

else

{

MessageBox.Show("Клиента не е намерен!");

btRemoveCustomer.Enabled = false;

btEditCustomer.Enabled = false;

}

reader.Close();

}

private void btFindOrders\_Click(object sender, EventArgs e)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("SELECT CustomerID, ProductID, Quantity, OrderDate, Status FROM orders WHERE OrderID = @OrderID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@OrderID", tb\_orders\_OrderID.Text);

MySqlDataReader reader = cmdDataBase1.ExecuteReader();

if (reader.Read())

{

tb\_orders\_CustomerID.Text = reader["CustomerID"].ToString();

tb\_orders\_ProductID.Text = reader["ProductID"].ToString();

tb\_orders\_Quantity.Text = reader["Quantity"].ToString();

DateTime orderDate = Convert.ToDateTime(reader["OrderDate"]);

tb\_orders\_OrderDate.Text = orderDate.ToString("yyyy-MM-dd HH:mm:ss");

cb\_orders\_Status.Text = reader["Status"].ToString();

MessageBox.Show("Поръчката е намерена!");

btRemoveOrder.Enabled = true;

btEditOrder.Enabled = true;

}

else

{

MessageBox.Show("Поръчката не е намерена!");

btRemoveOrder.Enabled = false;

btEditOrder.Enabled = false;

}

reader.Close();

}

private void btFindSuppliers\_Click(object sender, EventArgs e)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("SELECT Name, ContactEmail, Phone FROM suppliers WHERE SupplierID = @SupplierID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@SupplierID", tb\_suppliers\_SupplierID.Text);

MySqlDataReader reader = cmdDataBase1.ExecuteReader();

if (reader.Read())

{

tb\_suppliers\_Name.Text = reader["Name"].ToString();

tb\_suppliers\_ContactEmail.Text = reader["ContactEmail"].ToString();

tb\_suppliers\_Phone.Text = reader["Phone"].ToString();

MessageBox.Show("Доставчика е намерен!");

btRemoveSupplier.Enabled = true;

btEditSupplier.Enabled = true;

}

else

{

MessageBox.Show("Доставчика не е намерен!");

btRemoveSupplier.Enabled = false;

btEditSupplier.Enabled = false;

}

reader.Close();

}

// <-------------------------->

// <-------------------------->

// Methods for editing records

private void btEditProduct\_Click(object sender, EventArgs e)

{

if(Decimal.Parse(tb\_products\_Price.Text) < 0 || int.Parse(tb\_products\_QuantityInStock.Text) < 0)

{

MessageBox.Show("Цената и количеството не могат да бъдат отрицателни!");

return;

}

MySqlCommand cmdDataBase1 = new MySqlCommand("UPDATE products SET Name = @Name, Category = @Category, Price = @Price, QuantityInStock = @QuantityInStock, SupplierID = @SupplierID WHERE ProductID = @ProductID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@ProductID", tb\_products\_ProductID.Text);

cmdDataBase1.Parameters.AddWithValue("@Name", tb\_products\_Name.Text);

cmdDataBase1.Parameters.AddWithValue("@Category", cb\_products\_Category.SelectedIndex + 1);

cmdDataBase1.Parameters.AddWithValue("@Price", tb\_products\_Price.Text);

cmdDataBase1.Parameters.AddWithValue("@QuantityInStock", tb\_products\_QuantityInStock.Text);

cmdDataBase1.Parameters.AddWithValue("@SupplierID", tb\_products\_SupplierID.Text);

int rowsAffected = cmdDataBase1.ExecuteNonQuery();

if (rowsAffected > 0)

{

MessageBox.Show("Продуктът беше успешно редактиран!");

}

else

{

MessageBox.Show("Грешка при редактирането на продукта!");

}

btRemoveProduct.Enabled = false;

btEditProduct.Enabled = false;

ShowTable(MyDataBase, dataGridView1, "products");

}

private void btEditCustomer\_Click(object sender, EventArgs e)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("UPDATE customers SET FirstName = @FirstName, LastName = @LastName, Email = @Email, Phone = @Phone, Address = @Address WHERE CustomerID = @CustomerID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@CustomerID", tb\_customers\_CustomerID.Text);

cmdDataBase1.Parameters.AddWithValue("@FirstName", tb\_customers\_FirstName.Text);

cmdDataBase1.Parameters.AddWithValue("@LastName", tb\_customers\_LastName.Text);

cmdDataBase1.Parameters.AddWithValue("@Email", tb\_customers\_Email.Text);

cmdDataBase1.Parameters.AddWithValue("@Phone", tb\_customers\_Phone.Text);

cmdDataBase1.Parameters.AddWithValue("@Address", tb\_customers\_Address.Text);

int rowsAffected = cmdDataBase1.ExecuteNonQuery();

if (rowsAffected > 0)

{

MessageBox.Show("Клиентът беше успешно редактиран!");

}

else

{

MessageBox.Show("Грешка при редактирането на клиента!");

}

btRemoveCustomer.Enabled = false;

btEditCustomer.Enabled = false;

ShowTable(MyDataBase, dataGridView1, "customers");

}

private void btEditOrder\_Click(object sender, EventArgs e)

{

try

{

int orderId = int.Parse(tb\_orders\_OrderID.Text);

int newQty = int.Parse(tb\_orders\_Quantity.Text);

int productId = int.Parse(tb\_orders\_ProductID.Text);

MySqlCommand getOldQtyCmd = new MySqlCommand(

"SELECT Quantity FROM orders WHERE OrderID = @OrderID", conDataBase1);

getOldQtyCmd.Parameters.AddWithValue("@OrderID", orderId);

int oldQty = Convert.ToInt32(getOldQtyCmd.ExecuteScalar());

int qtyDifference = newQty - oldQty;

MySqlCommand getStockCmd = new MySqlCommand(

"SELECT QuantityInStock FROM products WHERE ProductID = @ProductID", conDataBase1);

getStockCmd.Parameters.AddWithValue("@ProductID", productId);

int stock = Convert.ToInt32(getStockCmd.ExecuteScalar());

if (qtyDifference > stock)

{

MessageBox.Show("Недостатъчна наличност за редактиране на поръчката!");

return;

}

MySqlCommand updateOrder = new MySqlCommand(

"UPDATE orders SET CustomerID = @CustomerID, ProductID = @ProductID, Quantity = @Quantity, " +

"OrderDate = @OrderDate, Status = @Status WHERE OrderID = @OrderID", conDataBase1);

updateOrder.Parameters.AddWithValue("@OrderID", orderId);

updateOrder.Parameters.AddWithValue("@CustomerID", tb\_orders\_CustomerID.Text);

updateOrder.Parameters.AddWithValue("@ProductID", productId);

updateOrder.Parameters.AddWithValue("@Quantity", newQty);

updateOrder.Parameters.AddWithValue("@OrderDate", tb\_orders\_OrderDate.Text);

updateOrder.Parameters.AddWithValue("@Status", cb\_orders\_Status.SelectedIndex + 1);

updateOrder.ExecuteNonQuery();

MySqlCommand updateStock = new MySqlCommand(

"UPDATE products SET QuantityInStock = QuantityInStock - @Diff WHERE ProductID = @ProductID", conDataBase1);

updateStock.Parameters.AddWithValue("@Diff", qtyDifference);

updateStock.Parameters.AddWithValue("@ProductID", productId);

updateStock.ExecuteNonQuery();

MessageBox.Show("Поръчката беше успешно редактирана и количеството е актуализирано!");

ShowTable(MyDataBase, dataGridView1, "orders");

}

catch (Exception ex)

{

MessageBox.Show("Грешка при редактиране на поръчката: " + ex.Message);

}

btRemoveOrder.Enabled = false;

btEditOrder.Enabled = false;

}

private void btEditSupplier\_Click(object sender, EventArgs e)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("UPDATE suppliers SET Name = @Name, ContactEmail = @ContactEmail, Phone = @Phone WHERE SupplierID = @SupplierID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@SupplierID", tb\_suppliers\_SupplierID.Text);

cmdDataBase1.Parameters.AddWithValue("@Name", tb\_suppliers\_Name.Text);

cmdDataBase1.Parameters.AddWithValue("@ContactEmail", tb\_suppliers\_ContactEmail.Text);

cmdDataBase1.Parameters.AddWithValue("@Phone", tb\_suppliers\_Phone.Text);

int rowsAffected = cmdDataBase1.ExecuteNonQuery();

if (rowsAffected > 0)

{

MessageBox.Show("Доставчикът беше успешно редактиран!");

}

else

{

MessageBox.Show("Грешка при редактирането на доставчика!");

}

btRemoveSupplier.Enabled = false;

btEditSupplier.Enabled = false;

ShowTable(MyDataBase, dataGridView1, "suppliers");

}

// <-------------------------->

// <-------------------------->

// Methods for removing records

private void btRemoveProduct\_Click(object sender, EventArgs e)

{

DialogResult dialogResult = MessageBox.Show("Сигурни ли сте че искате да го изтриете?", "Изтриване", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("DELETE FROM products WHERE ProductID = @ProductID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@ProductID", tb\_products\_ProductID.Text);

int rowsAffected = cmdDataBase1.ExecuteNonQuery();

if (rowsAffected > 0)

{

MessageBox.Show("Успешно изтрихте продукта!");

}

else

{

MessageBox.Show("Продуктът не беше намерен за изтриване!");

}

ClearProductsGB();

btRemoveProduct.Enabled = false;

btEditProduct.Enabled = false;

ShowTable(MyDataBase, dataGridView1, "products");

}

else if (dialogResult == DialogResult.No) { }

}

private void btRemoveCustomer\_Click(object sender, EventArgs e)

{

DialogResult dialogResult = MessageBox.Show("Сигурни ли сте че искате да го изтриете?", "Изтриване", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("DELETE FROM customers WHERE CustomerID = @CustomerID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@CustomerID", tb\_customers\_CustomerID.Text);

int rowsAffected = cmdDataBase1.ExecuteNonQuery();

if (rowsAffected > 0)

{

MessageBox.Show("Успешно изтрихте клиента!");

}

else

{

MessageBox.Show("Клиентът не беше намерен за изтриване!");

}

ClearCustomersGB();

btRemoveCustomer.Enabled = false;

btEditCustomer.Enabled = false;

ShowTable(MyDataBase, dataGridView1, "customers");

}

else if (dialogResult == DialogResult.No) { }

}

private void btRemoveOrder\_Click(object sender, EventArgs e)

{

DialogResult dialogResult = MessageBox.Show("Сигурни ли сте че искате да го изтриете?", "Изтриване", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

try

{

int orderId = int.Parse(tb\_orders\_OrderID.Text);

MySqlCommand getOrderDetails = new MySqlCommand(

"SELECT ProductID, Quantity FROM orders WHERE OrderID = @OrderID", conDataBase1);

getOrderDetails.Parameters.AddWithValue("@OrderID", orderId);

MySqlDataReader reader = getOrderDetails.ExecuteReader();

int productId = -1;

int quantityToRestore = 0;

if (reader.Read())

{

productId = Convert.ToInt32(reader["ProductID"]);

quantityToRestore = Convert.ToInt32(reader["Quantity"]);

}

else

{

reader.Close();

MessageBox.Show("Поръчката не беше намерена за изтриване!");

return;

}

reader.Close();

MySqlCommand restoreQuantity = new MySqlCommand(

"UPDATE products SET QuantityInStock = QuantityInStock + @Qty WHERE ProductID = @ProductID", conDataBase1);

restoreQuantity.Parameters.AddWithValue("@Qty", quantityToRestore);

restoreQuantity.Parameters.AddWithValue("@ProductID", productId);

restoreQuantity.ExecuteNonQuery();

MySqlCommand deleteOrder = new MySqlCommand("DELETE FROM orders WHERE OrderID = @OrderID", conDataBase1);

deleteOrder.Parameters.AddWithValue("@OrderID", orderId);

int rowsAffected = deleteOrder.ExecuteNonQuery();

if (rowsAffected > 0)

{

MessageBox.Show("Успешно изтрихте поръчката и възстановихте количеството на продукта!");

}

else

{

MessageBox.Show("Поръчката не беше намерена за изтриване!");

}

ClearOrdersGB();

btRemoveOrder.Enabled = false;

btEditOrder.Enabled = false;

ShowTable(MyDataBase, dataGridView1, "orders");

}

catch (Exception ex)

{

MessageBox.Show("Грешка при изтриване на поръчката: " + ex.Message);

}

}

}

private void btRemoveSupplier\_Click(object sender, EventArgs e)

{

DialogResult dialogResult = MessageBox.Show("Сигурни ли сте че искате да го изтриете?", "Изтриване", MessageBoxButtons.YesNo);

if (dialogResult == DialogResult.Yes)

{

MySqlCommand cmdDataBase1 = new MySqlCommand("DELETE FROM suppliers WHERE SupplierID = @SupplierID", conDataBase1);

cmdDataBase1.Parameters.AddWithValue("@SupplierID", tb\_suppliers\_SupplierID.Text);

int rowsAffected = cmdDataBase1.ExecuteNonQuery();

if (rowsAffected > 0)

{

MessageBox.Show("Успешно изтрихте доставчика!");

}

else

{

MessageBox.Show("Доставчикът не беше намерен за изтриване!");

}

ClearSuppliersGB();

btRemoveSupplier.Enabled = false;

btEditSupplier.Enabled = false;

ShowTable(MyDataBase, dataGridView1, "suppliers");

}

else if (dialogResult == DialogResult.No) { }

}

// <-------------------------->

// Method for showing the selected table in DataGridView

public void ShowTable(string dataBase, DataGridView dataGridView, string table)

{

MySqlConnection conDataBase = new MySqlConnection(dataBase);

MySqlCommand cmdDataBase = new MySqlCommand($"Select \*from {table} ", conDataBase);

MySqlDataAdapter sda = new MySqlDataAdapter();

sda.SelectCommand = cmdDataBase;

DataTable dbTable = new DataTable();

sda.Fill(dbTable);

BindingSource bSource = new BindingSource();

bSource.DataSource = dbTable;

dataGridView.AutoGenerateColumns = true;

dataGridView.DataSource = bSource;

conDataBase.Close();

}

// <-------------------------->

// Methods for clearing the group boxes

private void ClearProductsGB()

{

tb\_products\_ProductID.Clear();

tb\_products\_Name.Clear();

cb\_products\_Category.SelectedIndex = -1;

tb\_products\_Price.Clear();

tb\_products\_QuantityInStock.Clear();

tb\_products\_SupplierID.Clear();

}

private void ClearCustomersGB()

{

tb\_customers\_CustomerID.Clear();

tb\_customers\_FirstName.Clear();

tb\_customers\_LastName.Clear();

tb\_customers\_Email.Clear();

tb\_customers\_Phone.Clear();

tb\_customers\_Address.Clear();

}

private void ClearOrdersGB()

{

tb\_orders\_OrderID.Clear();

tb\_orders\_CustomerID.Clear();

tb\_orders\_ProductID.Clear();

tb\_orders\_Quantity.Clear();

tb\_orders\_OrderDate.Clear();

cb\_orders\_Status.SelectedIndex = -1;

}

private void ClearSuppliersGB()

{

tb\_suppliers\_SupplierID.Clear();

tb\_suppliers\_Name.Clear();

tb\_suppliers\_ContactEmail.Clear();

tb\_suppliers\_Phone.Clear();

}

// <-------------------------->

// Method for getting the current date and time

private void btGetCurrentTime\_Click(object sender, EventArgs e)

{

DateTime currentDateTime = DateTime.Now;

tb\_orders\_OrderDate.Text = currentDateTime.ToString("yyyy-MM-dd HH:mm:ss");

}

// Event which checks if the form is closed manually

private void ManualAditionForm\_FormClosed(object sender, FormClosedEventArgs e)

{

if (IsManuallyClosed == true)

{

conDataBase1.Close();

loginForm.Close();

}

}

// Event for closing the form and returning to the login form

private void btBack\_Click(object sender, EventArgs e)

{

IsManuallyClosed = false;

conDataBase1.Close();

this.Close();

loginForm.Show();

IsManuallyClosed = true;

}

}

}