**1. Полнота обработки ошибочных данных**

**Описание:** В проекте предусмотрена проверка допустимых данных на этапе авторизации, при создании заявок и при сохранении отчетов. Применяются такие подходы, как ограничение количества попыток ввода неверного пароля, введение CAPTCHA при нескольких ошибках входа и блокировка после определенного количества неудачных попыток. Также код содержит обработчики исключений в случае ошибок при взаимодействии с базой данных (Рисунок 1 – 2).



Рисунок 1 – Блок try



Рисунок 2 – Блок catch

**2. Наличие тестов для проверки допустимых значений входных данных**

**Описание:** Для предотвращения ввода недопустимых значений в системе реализованы неактивные кнопки (Рисунок 3), которые становятся доступными только после выполнения условий (например, обязательное заполнение полей), выпадающие списки (Рисунок 4), позволяющие выбирать лишь допустимые значения, и проверка условий для доступа к действиям, как в модуле отчетов. Эти механизмы обеспечивают корректность и безопасность данных, вводимых пользователем.

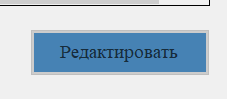


Рисунок 3 – Неактивная кнопка

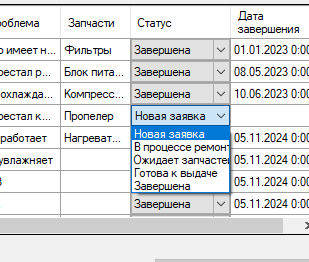


Рисунок 4 – Выпадающий список

**3. Наличие средств контроля корректности входных данных**

**Описание:** Проект включает проверку корректности введенных данных с помощью проверок try-catch при подключении к базе данных и в других операциях с базой данных (Рисунок 5), что снижает вероятность аварийного завершения работы программы при некорректных данных. Также в некоторых формах предусмотрены проверки на наличие пустых строк и ограничение на ввод специального формата данных (Рисунок 6).

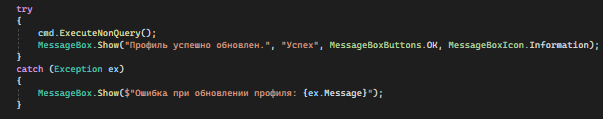


Рисунок 5 – Блок попытки выполнения запроса



Рисунок 6 – Проверка на пустое значение

**4. Наличие средств восстановления при сбоях оборудования**

**Описание:** Система имеет резервную копию базы данных, что позволяет минимизировать потерю информации при сбоях оборудования.

**5. Наличие комментариев**

**Описание: Большинство методов в проекте содержат комментарии, описывающие их назначение и основные действия. Также задокументированы ключевые блоки кода, такие как обработчики ошибок, этапы валидации и бизнес-логика обработки заявок и отчетов.**

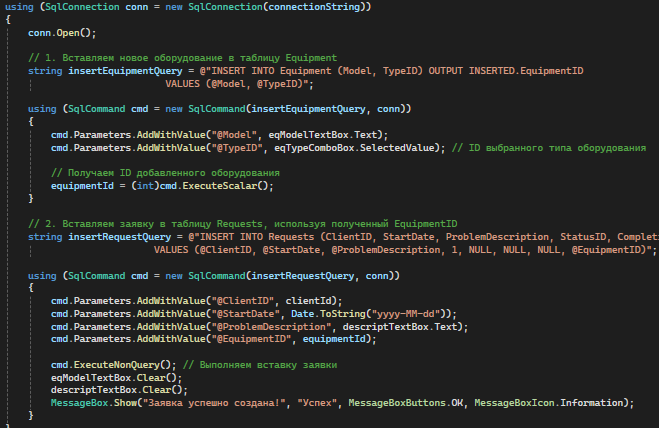
****

Рисунок 7 – Закомментированные блоки кода

**6. Наличие проверки корректности передаваемых данных**

**Описание:** Проверка корректности передаваемых данных реализована для авторизационных данных пользователя, а также при создании отчета и подачи заявки. Например, пользователь не может передать в базу данных пустое значение (Рисунок 8). В коде используется ограничение на ввод неверных данных, есть проверка на пустые строки и подтверждение значений при вводе пароля.

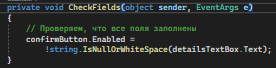


Рисунок 8 – Проверка перед вставкой в БД

**7. Описание основных функций**

**Метод авторизации** enterButton\_Click(object sender, EventArgs e)

Код:

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

{

string userLogin = loginTextBox.Text;

bool isSuccess = false;

if (failedAttempts >= 1 && captchaTextBox.Text != generatedCaptcha)

{

captchaTextBox.Clear();

MessageBox.Show("CAPTCHA введена неправильно.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

LogLoginAttempt(userLogin, false);

return;

}

try

{

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

string selectUserQuery = "SELECT UserTypeID, Password FROM Users WHERE Login = @Login";

using (SqlCommand cmd = new SqlCommand(selectUserQuery, conn))

{

cmd.Parameters.AddWithValue("@Login", userLogin);

var reader = cmd.ExecuteReader();

if (reader.Read())

{

userRole = Convert.ToInt32(reader["UserTypeID"]);

string storedPassword = reader["Password"].ToString();

if (storedPassword == passTextBox.Text)

{

isSuccess = true;

failedAttempts = 0;

MessageBox.Show("Вход выполнен успешно!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

MainMenu mainMenu = new MainMenu(userRole, userLogin);

mainMenu.Show();

Hide();

failedAttempts = 0;

ResetCaptcha();

}

else

{

HandleFailedAttempt();

}

}

else

{

HandleFailedAttempt();

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при попытке подключения к базе данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

LogLoginAttempt(userLogin, isSuccess);

}

Описание:

Метод enterButton\_Click отвечает за аутентификацию пользователя при нажатии на кнопку «Войти». Он принимает два параметра: object sender, представляющий объект, вызвавший событие (в данном случае кнопку enterButton), и EventArgs e, который содержит информацию о событии (обычно используется для привязки к событию кнопки). Метод не возвращает значения (void) и выполняет проверку логина и пароля пользователя.

Работа метода начинается с получения логина пользователя из поля loginTextBox и установки переменной isSuccess в false. Если количество неудачных попыток больше или равно 1, и введенная CAPTCHA не совпадает с сгенерированной, метод очищает текст CAPTCHA, выводит сообщение об ошибке и регистрирует неудачную попытку. Затем происходит подключение к базе данных, и выполняется SQL-запрос для получения UserTypeID и Password, соответствующих введенному логину.

Если логин найден в базе данных, происходит сравнение введенного пароля с хранимым. В случае совпадения пароля метод обновляет isSuccess на true, сбрасывает счетчик failedAttempts и CAPTCHA, и переходит к следующему окну (MainMenu), передавая туда роль пользователя (userRole) и логин (userLogin). При несоответствии пароля или отсутствии логина вызывается метод HandleFailedAttempt для обработки неудачной попытки.

После выполнения проверки логина и пароля метод вызывает LogLoginAttempt, который записывает информацию о результате попытки входа (успешной или неудачной) в таблицу LoginHistory. Этот метод тесно связан с другими методами, такими как HandleFailedAttempt для обработки неудачных попыток, и ResetCaptcha для очистки CAPTCHA после успешного входа.

**Метод добавления записи** confirmButton\_Click(object sender, EventArgs e)

Код:

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

{

DateTime Date = DateTime.Now;

string connectionString = @"Data Source=(localdb)\MSSQLLocalDB;Initial catalog=OleksenkoPract;Integrated Security=True";

int equipmentId = 0;

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// 1. Вставляем новое оборудование в таблицу Equipment

string insertEquipmentQuery = @"INSERT INTO Equipment (Model, TypeID) OUTPUT INSERTED.EquipmentID

VALUES (@Model, @TypeID)";

using (SqlCommand cmd = new SqlCommand(insertEquipmentQuery, conn))

{

cmd.Parameters.AddWithValue("@Model", eqModelTextBox.Text);

cmd.Parameters.AddWithValue("@TypeID", eqTypeComboBox.SelectedValue); // ID выбранного типа оборудования

// Получаем ID добавленного оборудования

equipmentId = (int)cmd.ExecuteScalar();

}

// 2. Вставляем заявку в таблицу Requests, используя полученный EquipmentID

string insertRequestQuery = @"INSERT INTO Requests (ClientID, StartDate, ProblemDescription, StatusID, CompletionDate, RepairParts, MasterID, EquipmentID)

VALUES (@ClientID, @StartDate, @ProblemDescription, 1, NULL, NULL, NULL, @EquipmentID)";

using (SqlCommand cmd = new SqlCommand(insertRequestQuery, conn))

{

cmd.Parameters.AddWithValue("@ClientID", clientId);

cmd.Parameters.AddWithValue("@StartDate", Date.ToString("yyyy-MM-dd"));

cmd.Parameters.AddWithValue("@ProblemDescription", descriptTextBox.Text);

cmd.Parameters.AddWithValue("@EquipmentID", equipmentId);

cmd.ExecuteNonQuery(); // Выполняем вставку заявки

eqModelTextBox.Clear();

descriptTextBox.Clear();

MessageBox.Show("Заявка успешно создана!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

Описание:

Метод confirmButton\_Click выполняет обработку создания новой заявки и нового оборудования в базе данных при нажатии на кнопку "Подтвердить". Он принимает стандартные для обработчиков событий параметры: object sender, который указывает на элемент, вызвавший событие, и EventArgs e, представляющий информацию о событии. Метод не возвращает значения (void), а выполняет ряд действий по добавлению данных в базу.

Метод начинается с получения текущей даты (DateTime.Now) и установления строки подключения connectionString для работы с базой данных OleksenkoPract. Также создается переменная equipmentId, чтобы хранить идентификатор нового оборудования, добавленного в базу.

Первый этап включает добавление нового оборудования в таблицу Equipment. Для этого создается SQL-запрос insertEquipmentQuery, который вставляет модель и тип оборудования в таблицу Equipment и возвращает EquipmentID (идентификатор добавленного оборудования) благодаря OUTPUT INSERTED.EquipmentID. Команда SqlCommand выполняет запрос с параметрами, взятыми из текстового поля eqModelTextBox и eqTypeComboBox. Результат сохраняется в переменной equipmentId для использования в следующем этапе.

На втором этапе метод добавляет новую заявку в таблицу Requests, используя идентификатор оборудования (EquipmentID), полученный на предыдущем шаге. Запрос insertRequestQuery вставляет в таблицу Requests данные, включая ClientID, дату начала заявки, описание проблемы, статус по умолчанию (например, StatusID = 1) и EquipmentID. После выполнения запроса поля eqModelTextBox и descriptTextBox очищаются, и отображается сообщение об успешном создании заявки.

Весь метод защищен использованием using, что автоматически закрывает соединение с базой данных после выполнения, обеспечивая оптимизацию ресурсов и безопасность.

**Метод изменения записи** UpdateRequest(int requestId, int masterId, int statusId, DateTime? startDate, DateTime? completionDate)

Код:

private void UpdateRequest(int requestId, int masterId, int statusId, DateTime? startDate, DateTime? completionDate)

{

string query = @"UPDATE Requests

SET MasterID = @MasterID, StatusID = @StatusID, StartDate = @StartDate, CompletionDate = @CompletionDate

WHERE RequestID = @RequestID";

using (SqlConnection conn = new SqlConnection(connectionString))

{

SqlCommand cmd = new SqlCommand(query, conn);

cmd.Parameters.AddWithValue("@RequestID", requestId);

cmd.Parameters.AddWithValue("@MasterID", masterId);

cmd.Parameters.AddWithValue("@StatusID", statusId);

// Устанавливаем значения для StartDate и CompletionDate с проверкой на null

cmd.Parameters.AddWithValue("@StartDate", startDate.HasValue ? (object)startDate.Value : DBNull.Value);

cmd.Parameters.AddWithValue("@CompletionDate", completionDate.HasValue ? (object)completionDate.Value : DBNull.Value);

conn.Open();

cmd.ExecuteNonQuery();

}

}

Описание:

Метод UpdateRequest обновляет информацию о заявке в базе данных по заданному requestId. Этот метод принимает пять параметров:

requestId — идентификатор обновляемой заявки (целое число),

masterId — идентификатор мастера, связанного с заявкой (целое число),

statusId — идентификатор статуса заявки (целое число),

startDate и completionDate — необязательные параметры DateTime? для даты начала и даты завершения заявки, которые могут быть null (если данные о дате не указаны).

Метод строит SQL-запрос UPDATE, который обновляет в таблице Requests данные по мастеру (MasterID), статусу (StatusID), дате начала (StartDate) и дате завершения (CompletionDate). Далее открывается соединение с базой данных через строку подключения connectionString. Команда SqlCommand создает запрос, в который последовательно добавляются параметры:

@RequestID, @MasterID и @StatusID со значениями requestId, masterId и statusId соответственно,

@StartDate и @CompletionDate — значения дат, если они заданы, или DBNull.Value для пустых значений.

Метод выполняет команду ExecuteNonQuery, завершая процесс обновления информации о заявке, и закрывает соединение по завершении работы с базой данных.