**Качественные характеристики кода**

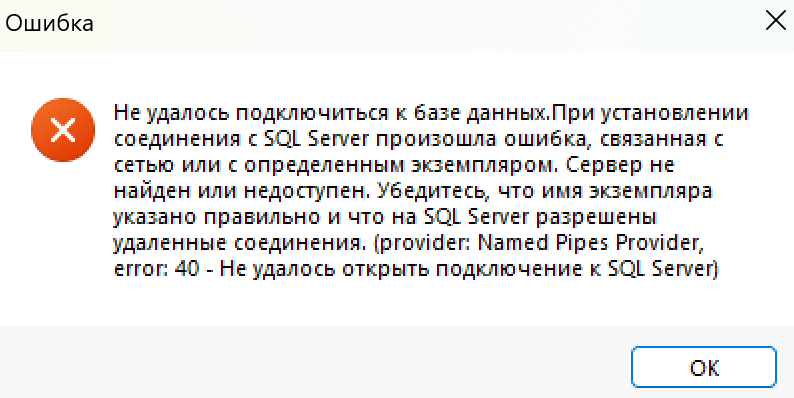
**Полнота обработки ошибочных данных.**

* В коде реализована базовая обработка ошибок для нескольких случаев. Например, при вводе неверных данных авторизации (логина или пароля) и неверной капчи выводится соответствующее сообщение об ошибке.

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

* В некоторых методах обрабатываются исключения SqlException для случаев ошибок доступа к базе данных, что минимизирует риск внезапного завершения программы при сбоях в подключении к базе. Обработка исключений при подключении к базе данных с выводом сообщения об ошибке, что позволяет пользователю понять причину сбоя.



* В случае нескольких неудачных попыток входа реализована блокировка с ограничением повторных попыток и оповещением пользователя.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

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

* Реализация капчи предотвращает попытки автоматизированного взлома.

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

**Наличие средств контроля корректности входных данных**

* В коде реализована базовая проверка корректности входных данных. Например, введенная капча проверяется перед допуском к авторизации, а также осуществляется верификация введенных логина и пароля через базу данных.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

* Использование передаваемых в comboBox данных из бд, что обеспечивает корректность входных данных.

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

**Наличие средств восстановления при сбоях оборудования**

Код не включает явных средств восстановления при сбоях оборудования, таких как перезапуск при сбое подключения к базе данных или повторные попытки при сетевых сбоях. При возникновении сбоя соединения с базой данных (например, из-за временного отключения сервера) выполнение программы просто завершается.

Для повышения устойчивости программы к аппаратным сбоям следовало бы добавить механизмы, позволяющие выполнять **восстановление при сбоях оборудования**. Например, повторные попытки подключения к базе данных при ее недоступности или уведомление пользователей о сбоях с рекомендацией повторить операцию позже.

**Наличие комментариев**

В коде есть незначительные комментарии, поясняющие некоторые ключевые моменты, например пояснение мс в с и дополнительная строка подключения к БД

Изображение выглядит как текст, программное обеспечение, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

**Наличие проверки корректности передаваемых данных**

Проверка корректности передаваемых данных выполняется перед их записью в базу данных:

* Используются параметризированные SQL-запросы, что гарантирует корректную передачу данных и защищает от SQL-инъекций.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Однако не проверяется корректность ввода формата или длины данных в поля.

**Наличие описаний основных функций**

**Метод:** ValidateUser  
**Тип:** int  
**Параметры:**

* **username** (тип: string): Логин пользователя, который используется для проверки в базе данных.
* **password** (тип: string): Пароль пользователя, который используется для проверки в базе данных.

**Возвращаемое значение:**

* Возвращает int — идентификатор пользователя (UserId), если введенные логин и пароль корректны.
* В случае ошибок или несоответствия данных возвращается -1.

**Описание:**  
Метод **ValidateUser** проверяет наличие пользователя в базе данных, используя введенные логин и пароль. Если пользователь с таким логином и паролем существует, возвращается его уникальный идентификатор (UserId). Если пользователь не найден или данные некорректны, возвращается значение **-1**.

**Работа метода:**

1. Формируется SQL-запрос, который выбирает идентификатор пользователя из таблицы Users на основе введенных логина и пароля:

* Запрос выглядит как "SELECT UserId FROM Users WHERE Login = @Username AND Password = @Password", где @Username и @Password являются параметрами.

1. Создается объект **SqlCommand**, который используется для выполнения SQL-запроса.

* Параметры запроса (@Username и @Password) добавляются через метод **AddWithValue** для предотвращения SQL-инъекций.

1. Метод **connection.Open()** устанавливает соединение с базой данных.
2. Команда выполняется с помощью **ExecuteScalar()**, которая возвращает первый столбец первой строки из результата запроса, то есть **UserId** пользователя.

* Если запись найдена, результат преобразуется в **int** и возвращается.

1. Если данные не найдены (т.е. результат выполнения запроса равен null), возвращается значение **-1**, указывая на то, что пользователя с таким логином и паролем не существует.
2. После выполнения запроса соединение с базой данных закрывается методом **connection.Close()**.

**Исключения:**  
Метод перехватывает любые исключения, которые могут возникнуть при подключении к базе данных или выполнении SQL-команды. В случае ошибки выводится сообщение с описанием проблемы.

**Метод:** Addfailure\_plus  
**Тип:** void  
**Параметры:**

* **userID** (тип: int): Идентификатор пользователя, для которого необходимо зарегистрировать неудачную попытку входа.

**Возвращаемое значение:** тип void (метод не возвращает значения).

**Описание:**  
Метод **Addfailure\_plus** используется для регистрации удачной попытки входа пользователя в систему. Он записывает информацию о удачной попытке входа в таблицу LoginAttempts, фиксируя идентификатор пользователя, дату и время попытки, а также указывая, что попытка удачна (параметр failure равен 1).

**Работа метода:**

1. Метод проверяет, является ли переданный идентификатор пользователя userID значением -1. Если это так, выполнение метода прекращается (это может означать, что пользователь не существует, и нет смысла регистрировать попытку входа).
2. Создается SQL-запрос для вставки данных о удачной попытке входа в таблицу LoginAttempts:

INSERT INTO LoginAttempts (UserId, loginDate, failure)

VALUES (@UserId, @Date, 1);

* В запросе используется параметр @UserId для записи идентификатора пользователя.
* Параметр @Date записывает текущее время (поле loginDate).
* Значение failure равно 1, что указывает на удачную попытку.

1. Создается объект SqlCommand с запросом, и параметры добавляются с помощью AddWithValue:

* Параметр @UserId принимает значение идентификатора пользователя userID.
* Параметр @Date принимает текущее время, полученное с помощью DateTime.Now.

1. Метод открывает соединение с базой данных с помощью connection.Open().
2. Команда выполняется через cmd.ExecuteNonQuery(), которая записывает данные о удачной попытке в таблицу LoginAttempts.
3. После выполнения запроса соединение с базой данных закрывается с помощью connection.Close().

**Исключения:**  
1) Метод не содержит явной обработки исключений. Если при выполнении запроса возникнут ошибки (например, проблемы с подключением к базе данных или нарушением целостности данных), будет выброшено исключение.

2) Для повышения устойчивости работы кода, можно было бы добавить обработку ошибок с использованием блока **try-catch**. Это позволило бы отловить и корректно обработать возможные исключения (например, ошибки соединения с базой данных).