Тема 18. MongoDB Интеграция с приложением.

Учебные вопросы:

1. Подготовка среды и драйвер
2. Мапинг данных (POCO-классы)
3. Архитектура подключения (Singleton/Client)
4. Реализация CRUD на C#
5. Синхронизация с интерфейсом (Binding)

1. Подготовка среды и драйвер

Что такое MongoDB Driver?

C# не умеет работать с MongoDB «из коробки» (в отличие от работы с файлами или простыми типами).

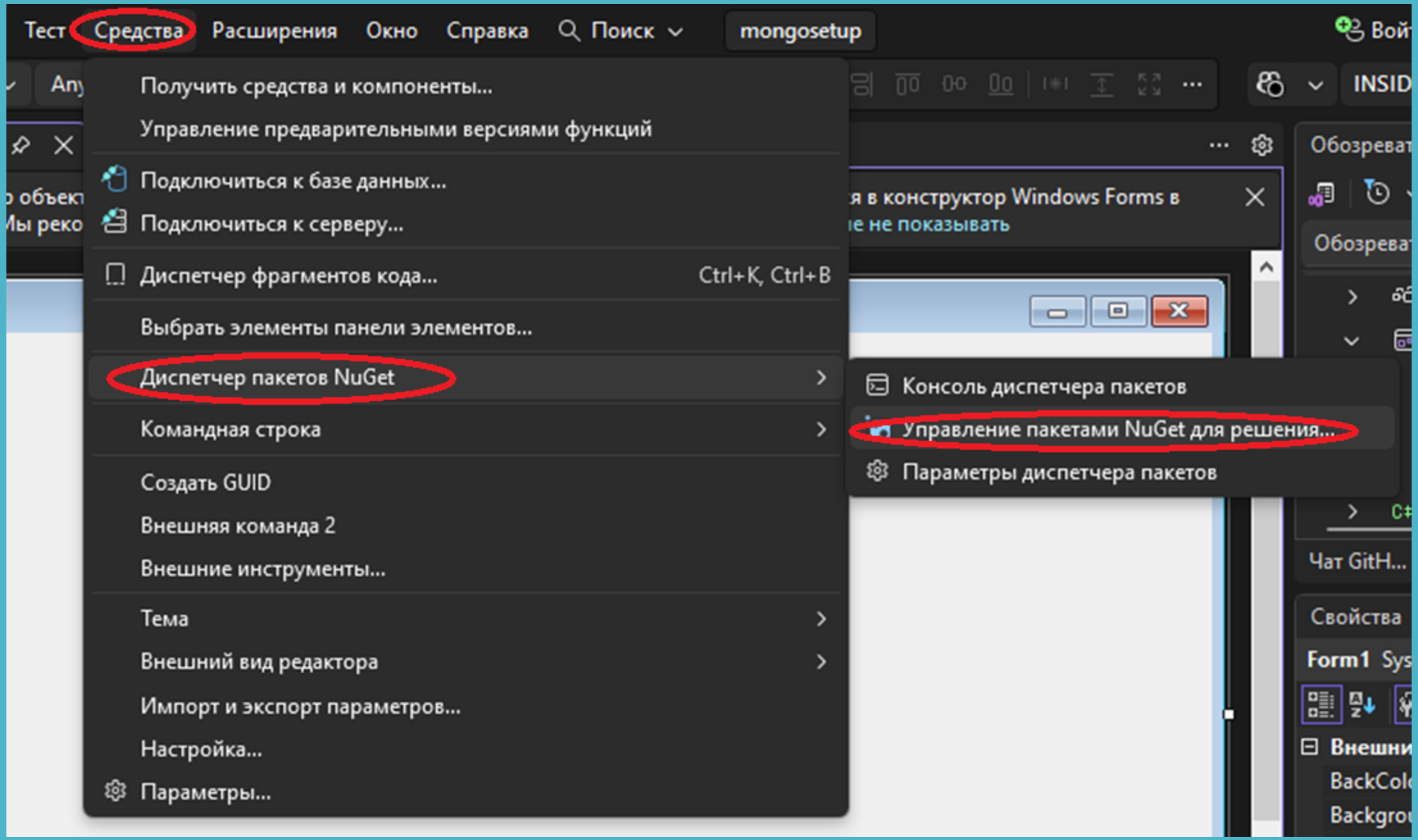
- Драйвер — это библиотека (набор DLL-файлов), которая берет ваши C#-объекты, превращает их в бинарный формат BSON и отправляет серверу по сети.
- Официальный драйвер поддерживает современные фишки: асинхронность (async/await), LINQ-запросы и автоматическое управление подключениями.

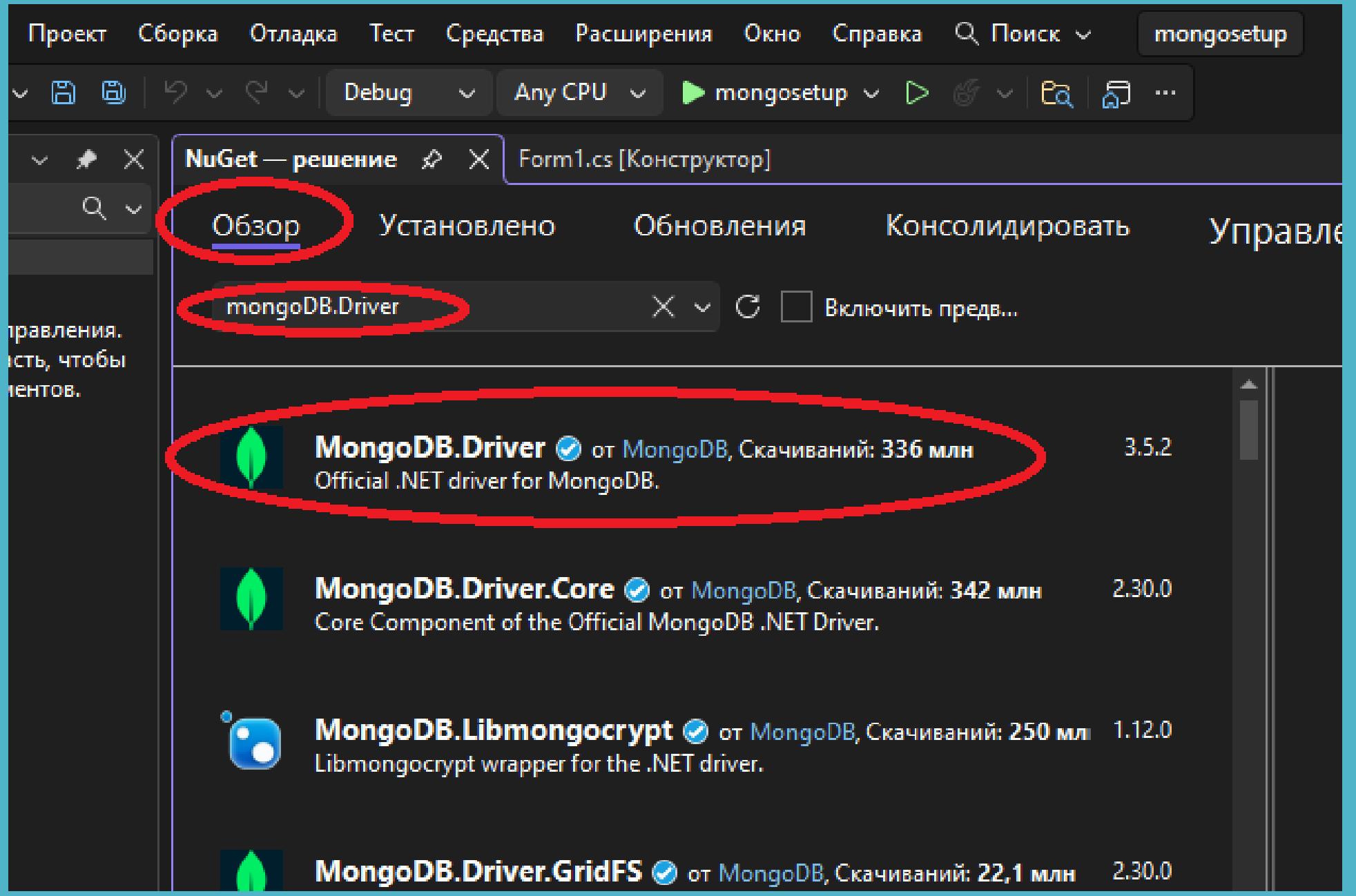
Установка через NuGet

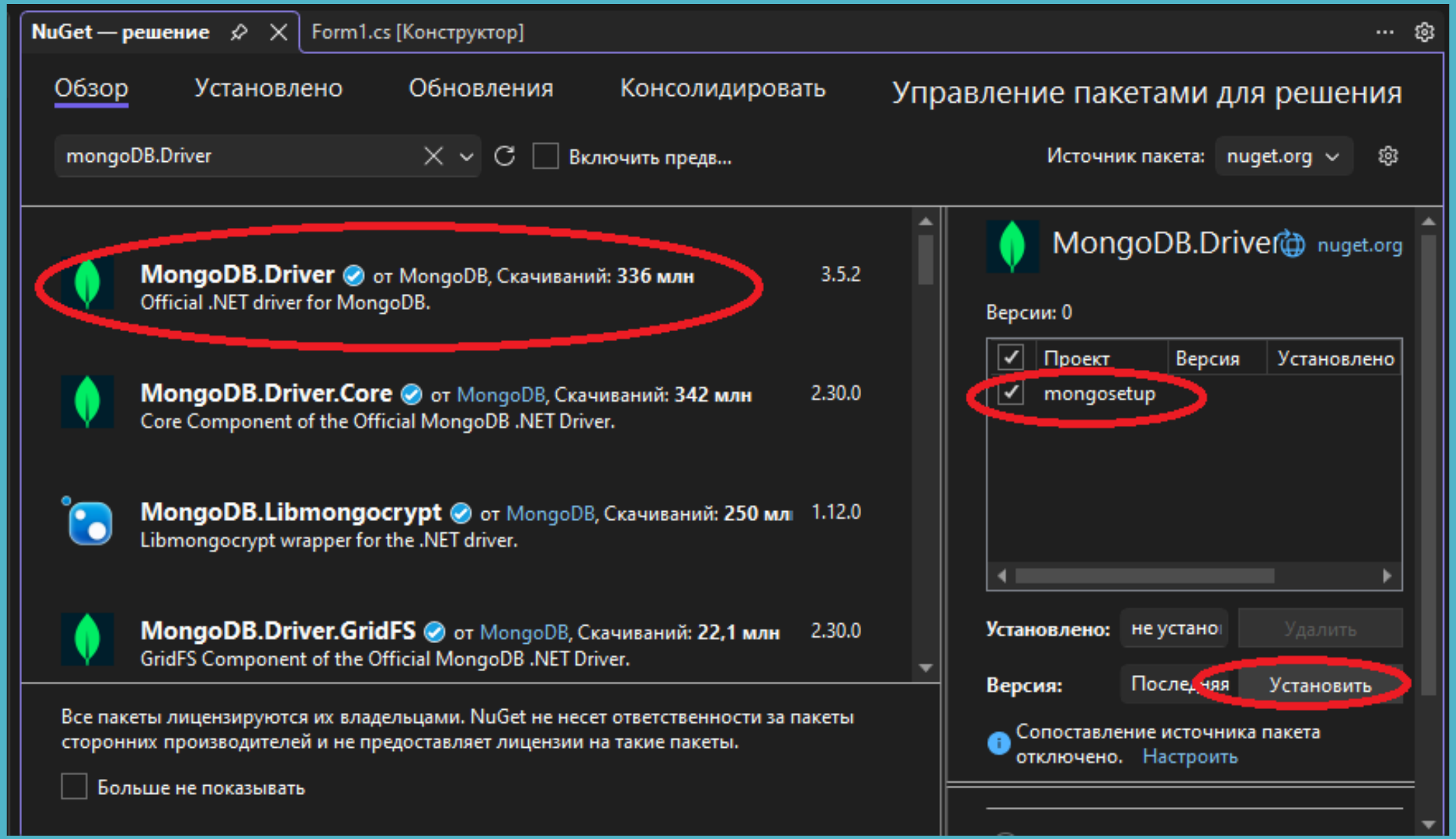
Самый простой и правильный способ добавить поддержку MongoDB в ваш проект WinForms:

- Откройте ваш проект в Visual Studio.
- Перейдите в меню Средства (Tools) -> Диспетчер пакетов NuGet -> Управление пакетами для решения.
- Вкладка Обзор (Browse) -> введите в поиске: MongoDB.Driver.
- Выберите проект и нажмите Установить (Install).

Важно: Вместе с основным пакетом подтянутся зависимости (MongoDB.Bson, MongoDB.Driver.Core и др.) — это нормально, они необходимы для работы.


Тест
Средства
Расширения
Окно
Справка
Поиск
mongosetup
Получить средства и компоненты...
Управление предварительными версиями функций
Подключиться к базе данных...
Подключиться к серверу...
Диспетчер фрагментов кода... Ctrl+K, Ctrl+B
Выбрать элементы панели элементов...
Диспетчер пакетов NuGet
Командная строка
Создать GUID
Внешняя команда 2
Внешние инструменты...
Тема
Внешний вид редактора
Импорт и экспорт параметров...
Настройка...
Параметры...
Консоль диспетчера пакетов
Управление пакетами NuGet для решения...
Параметры диспетчера пакетов
Обозреват
Чат GitH...
Свойства
Form1


Проект
Сборка
Отладка
Тест
Средства
Расширения
Окно
Справка
Поиск
mongosetup
Debug
Any CPU
mongosetup
NuGet — решение
Form1.cs [Конструктор]
Обзор
Установлено
Обновления
Консолидировать
Управле
mongoDB.Driver
Включить предв...
MongoDB.Driver от MongoDB, Скачиваний: 336 млн
Official .NET driver for MongoDB.
3.5.2
MongoDB.Driver.Core от MongoDB, Скачиваний: 342 млн
Core Component of the Official MongoDB .NET Driver.
2.30.0
MongoDB.Libmongocrypt от MongoDB, Скачиваний: 250 мл
Libmongocrypt wrapper for the .NET driver.
1.12.0
MongoDB.Driver.GridFS от MongoDB, Скачиваний: 22,1 млн
2.30.0


NuGet — решение
Form1.cs [Конструктор]
Обзор
Установлено
Обновления
Консолидировать
Управление пакетами для решения
mongoDB.Driver
Включить предв...
Источник пакета:
nuget.org
MongoDB.Driver от MongoDB, Скачиваний: 336 млн
Official .NET driver for MongoDB.
3.5.2
MongoDB.Driver.Core от MongoDB, Скачиваний: 342 млн
Core Component of the Official MongoDB .NET Driver.
2.30.0
MongoDB.Libmongocrypt от MongoDB, Скачиваний: 250 мл
Libmongocrypt wrapper for the .NET driver.
1.12.0
MongoDB.Driver.GridFS от MongoDB, Скачиваний: 22,1 млн
GridFS Component of the Official MongoDB .NET Driver.
2.30.0
Все пакеты лицензируются их владельцами. NuGet не несет ответственности за пакеты сторонних производителей и не предоставляет лицензии на такие пакеты.
Больше не показывать
MongoDB.Driver
nuget.org
Версии: 0
Проект
Версия
Установлено
mongosetup
Установлено:
не устано
Удалить
Версия:
Последняя
Установить
Сопоставление источника пакета отключено.
Настроить

Строка подключения (Connection String)

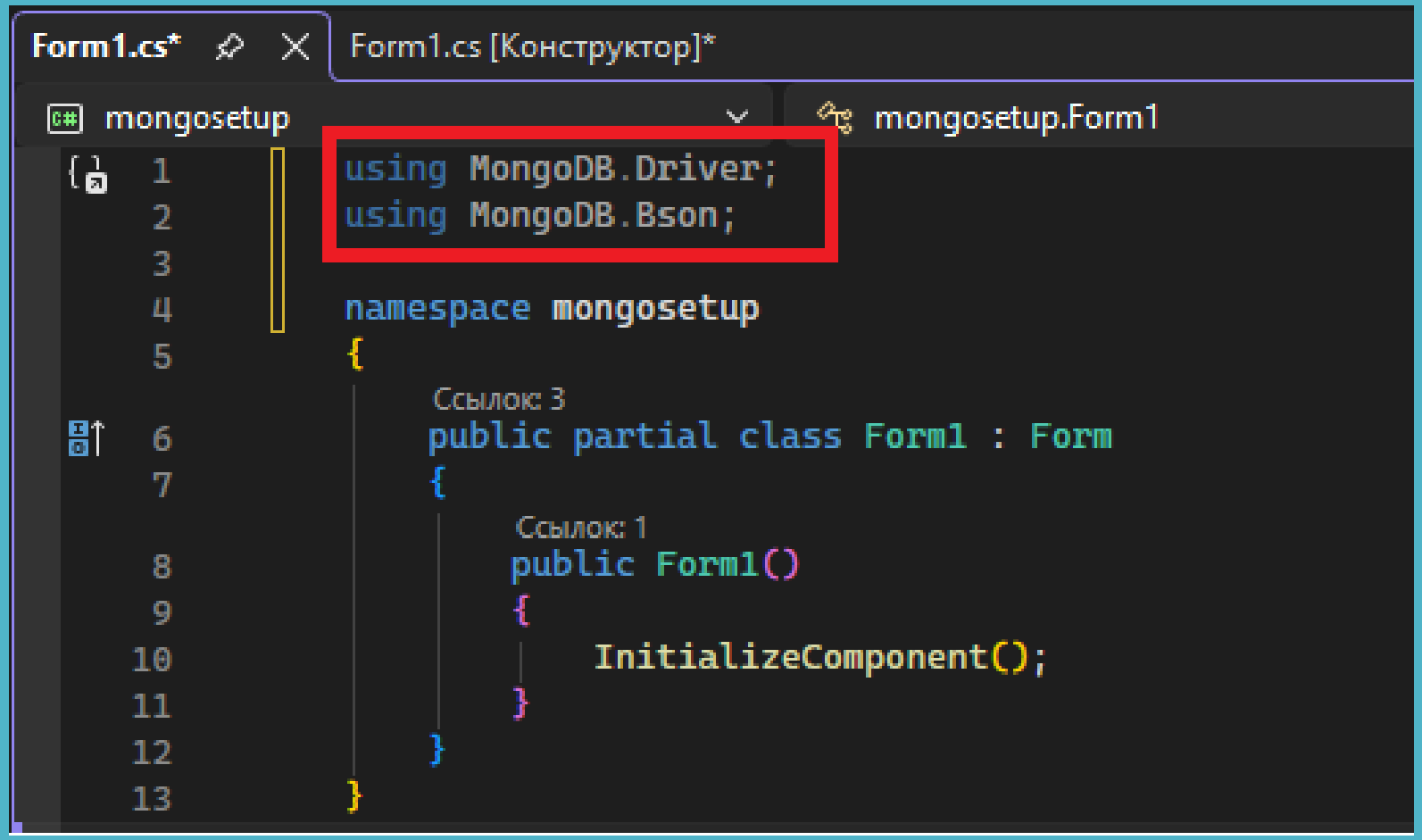
Чтобы приложение знало, куда «стучаться», используется специальная строка адреса.

Стандартный формат:
mongodb://[username:password@]host[:port]

- Для локальной базы: **mongodb://localhost:27017**
- Для облака (Atlas): Строка будет сложнее, вида mongodb+srv://user:password@cluster.mongodb.net/.

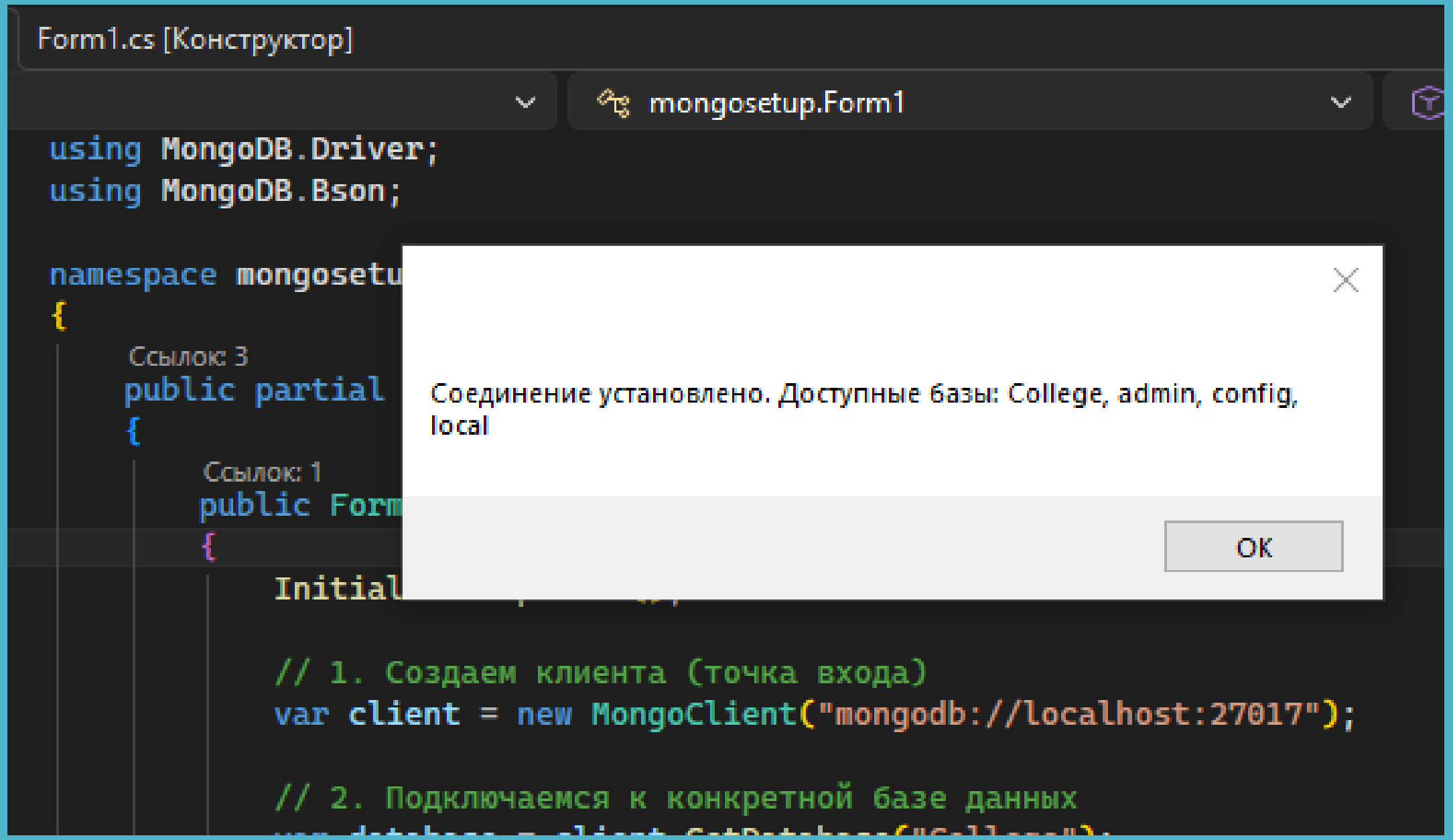
Проверка подключения в коде

После установки пакетов, в коде формы (Form1.cs) нужно добавить пространство имен:



Минимальный код для «рукопожатия» с базой:

```
public Form1()
{
    InitializeComponent();

    // 1. Создаем клиента (точка входа)
    var client = new MongoClient("mongodb://localhost:27017");

    // 2. Подключаемся к конкретной базе данных
    var database = client.GetDatabase("College");

    // 3. Получаем доступ к коллекции (таблице)
    // Пока используем BsonDocument (универсальный тип без привязки к классу)
    var collection = database.GetCollection<BsonDocument>("Students");

    // 4. Проверяем, что связь действительно есть
    // Получаем список имен баз данных
    using (var cursor = client.ListDatabaseNames())
    {
        var databaseNames = cursor.ToList();
        MessageBox.Show("Соединение установлено. Доступные базы: " + string.Join(", ", databaseNames));
    }

}
```


Form1.cs [Конструктор]
mongosetup.Form1
using MongoDB.Driver;
using MongoDB.Bson;
namespace mongosetu
{
Ссылок: 3
public partial
{
Ссылок: 1
public Form
{
Initial
// 1. Создаем клиента (точка входа)
var client = new MongoClient("mongodb://localhost:27017");
// 2. Подключаемся к конкретной базе данных
Соединение установлено. Доступные базы: College, admin, config, local
OK

Пояснение к коду:

Что такое cursor?

Cursor (Курсор) — это временный «пропуск» или «указатель» на результаты твоего запроса. Он не скачивает все данные в память компьютера мгновенно, а стоит в начале списка и ждет команды «Давай следующую запись».

Разбор кода:

client.ListDatabaseNames(): Запрашиваем список имен баз. MongoDB не кидает готовый список, она открывает курсор (канал связи) к этому списку.

using (var cursor = ...): Конструкция using важна — она гарантирует, что как только мы получим данные, «канал связи» (курсор) закроется и не будет тратить ресурсы сети и сервера.

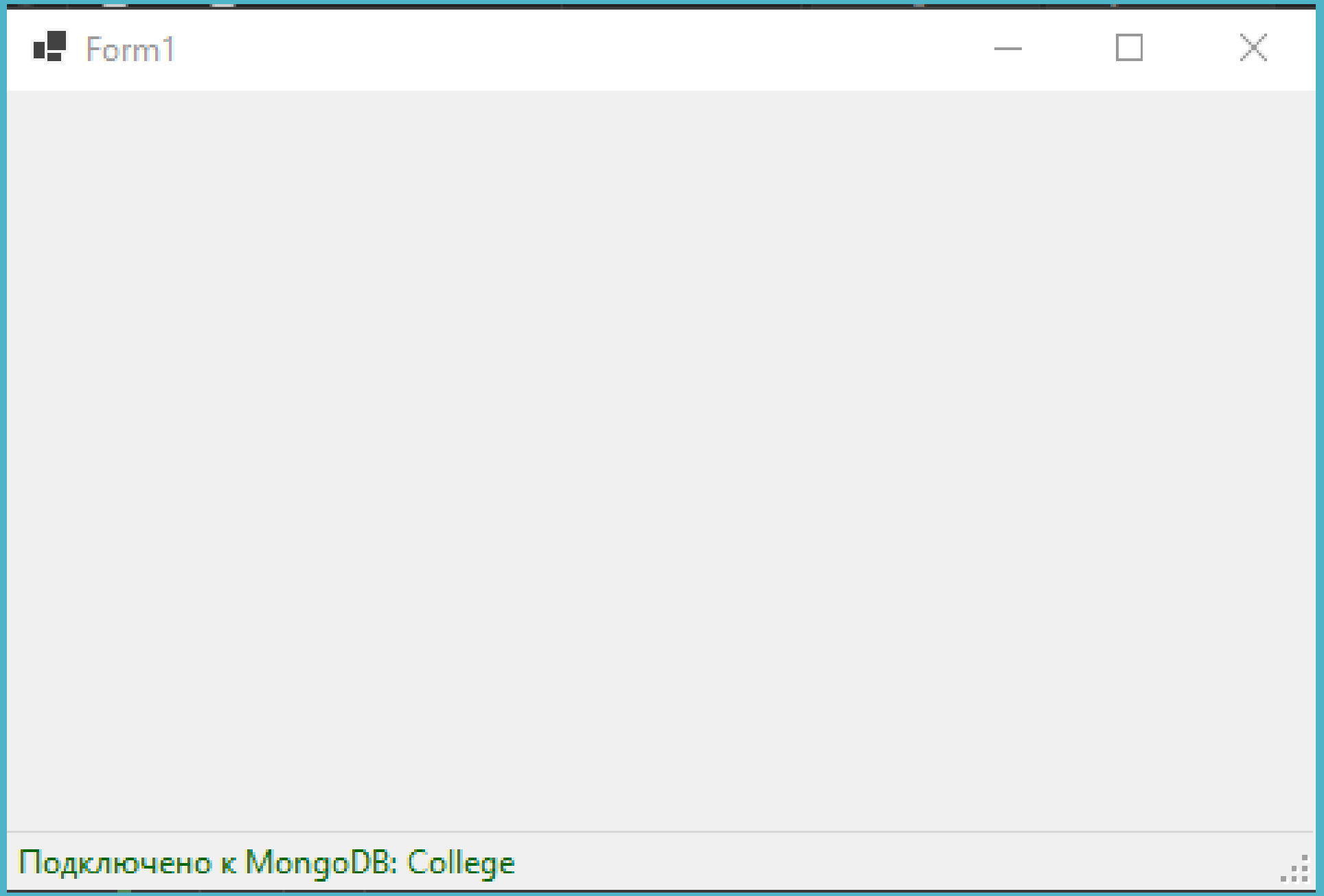
cursor.ToList(): Это команда курсору: «Пробегись по всем записям прямо сейчас, собери их и преврати в обычный список List<string>, с которым удобно работать в C#».

Вынесем код подключения в отдельную функцию. Для отображения статуса используем StatusStrip

```
public partial class Form1 : Form
{
    // Объявляем переменные на уровне класса (поля), чтобы они были доступны везде
    private IMongoCollection<BsonDocument> _collection;
    private IMongoDatabase _database;
    private MongoClient _client;

    Ссылок: 1
    public Form1()
    {
        InitializeComponent();

        // Сразу после инициализации компонентов пробуем подключиться
        ConnectToMongo();
    }
```

```
private void ConnectToMongo()
{
    // Устанавливаем начальное состояние
    toolStripStatusLabel1.Text = "Попытка подключения...";
    toolStripStatusLabel1.ForeColor = Color.Black;

    try
    {
        // Настройки клиента (добавляем таймаут, чтобы приложение не зависло надолго)
        var settings = MongoClientSettings.FromConnectionString("mongodb://localhost:27017");
        settings.ServerSelectionTimeout = TimeSpan.FromSeconds(3); // Ждем максимум 3 сек.

        _client = new MongoClient(settings);
        _database = _client.GetDatabase("College");
        _collection = _database.GetCollection<BsonDocument>("Students");

        // Выполняем реальную проверку (Ping)
        _database.RunCommand((Command<BsonDocument>)"{ping:1}");

        // Если Ping прошел успешно:
        toolStripStatusLabel1.Text = "Подключено к MongoDB: College";
        toolStripStatusLabel1.ForeColor = Color.DarkGreen;
    }
```

```
catch (Exception ex)
{
    // Если произошла ошибка:
    toolStripStatusLabel1.Text = "Ошибка соединения!";
    toolStripStatusLabel1.ForeColor = Color.Red;

    // Подробности ошибки можно оставить в ToolTip или Output
    System.Diagnostics.Debug.WriteLine(ex.Message);
}
```


Form1
Подключено к MongoDB: College

Пояснение к коду:

```
_database.RunCommand((Command<BsonDocument>)"{ping:1}");
```

RunCommand — это прямая «смс-ка» серверу. {ping:1} — это текст сообщения: «Ты тут?».

Если база на связи, она мгновенно ответит «Да», и код пойдет дальше. Если база выключена или адрес неверный, код «споткнется» и выдаст ошибку (исключение).

Мы используем это как мгновенную проверку связи.

2. Мапинг данных (POCO-классы)

Мапинг (Mapping) — это «создание карты соответствия» или правила перевода данных из одного формата в другой.

Представьте, что у вас есть два берега реки:

- На одном берегу — MongoDB: там данные лежат в виде JSON-текста (BSON).
- На другом берегу — C# (WinForms): здесь данные должны быть объектами с четкими свойствами.

Мапинг — это мост между ними. Вы объясняете программе: «Поле, которое в базе называется name, в моем коде на C# должно попадать в свойство Name класса Student».

Что такое POCO?

POCO (Plain Old CLR Object) — это простой C#-класс, который не содержит никакой логики, а только свойства (поля).

Драйвер MongoDB умеет автоматически превращать BSON-документ из базы в объект такого класса и наоборот.

Это называется десериализация и сериализация.

Создание базовой модели

Допустим, в базе у нас студенты. Создадим для них класс. Чтобы драйвер понимал, как сопоставить поля, мы используем специальные атрибуты.

```
[BsonIgnoreExtraElements] // Важнейший атрибут: игнорирует поля в базе, которых нет в этом классе
Ссылок: 3
public class Student
{
    [BsonId] // Помечает это поле как первичный ключ (_id)
    [BsonRepresentation(BsonType.ObjectId)] // Позволяет C# работать с ID как со строкой
    Ссылок: 0
    public string Id { get; set; }

    [BsonElement("name")] // Явно указываем имя поля в базе (если оно отличается от имени свойства)
    Ссылок: 1
    public string Name { get; set; }

    Ссылок: 0
    public int age { get; set; } // Если имена совпадают, атрибут [BsonElement] не нужен
}
```

Зачем нужны эти атрибуты?

- [BsonId]: В MongoDB каждый документ обязан иметь поле _id. В C# мы привыкли называть это просто Id. Этот атрибут говорит драйверу: «Свяжи мой Id с их _id».
- [BsonRepresentation(BsonType.ObjectId)]: В базе _id — это 12 байт бинарных данных. Чтобы вам не мучиться с ними в коде, этот атрибут позволяет C# видеть этот ID как обычную строку string.
- [BsonIgnoreExtraElements]: Представьте, что ваш коллега добавил студенту в базу поле HomeAddress, а в вашем классе его нет. Без этого атрибута программа выдаст ошибку, так как «не будет знать, куда положить адрес». С ним она просто проигнорирует лишнее.

Использование в коде (изменяем подключение)

Теперь, когда класс готов, мы меняем инициализацию коллекции. Вместо универсального BsonDocument указываем наш класс Student:

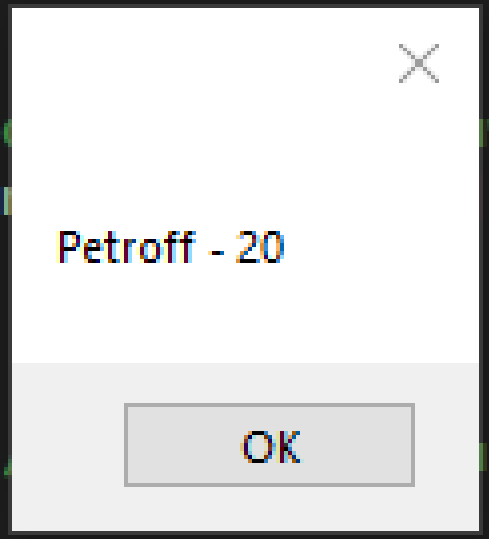
```
// Было:
private IMongoCollection<BsonDocument> _collection;

// Стало:
private IMongoCollection<Student> _students;
```

Исправим метод соединения с базой:

```
// Настройки клиента (добавляем таймаут, чтобы приложение не зависло надолго)
var settings = MongoClientSettings.FromConnectionString("mongodb://localhost:27017");
settings.ServerSelectionTimeout = TimeSpan.FromSeconds(3); // Ждем максимум 3 сек.

_client = new MongoClient(settings);
_database = _client.GetDatabase("College");
//_collection = _database.GetCollection<BsonDocument>("Students");
_students = _database.GetCollection<Student>("Students");

// Выполняем реальную проверку (Ping)
_database.RunCommand((Command<BsonDocument>)"{ping:1}");

// Если Ping прошел успешно:
toolStripStatusLabel1.Text = "Подключено к MongoDB: College";
toolStripStatusLabel1.ForeColor = Color.DarkGreen;

// Получаем всех студентов
List<Student> list = _students.Find(_ => true).ToList();

// Обращаемся к данным через точку
foreach (var s in list)
{
    MessageBox.Show(s.Name + " - " + s.age);
}
```

Проверяем:

```
[BsonIgnoreExtraElements] // Важнейший атрибут: игнорирует поля
Ссылок: 3
public class Student
{
    [BsonId] // Помечает эт                    чный ключ (_id)
    [BsonRepresentation(Bso                    // Позволяет C# рабо
    Ссылок: 0
    public string Id { get;

    [BsonElement("name")] /                    имя поля в базе (ес
    Ссылок: 1
    public string Name { get; set; }

    Ссылок: 1
    public int age { get; set; } // Если имена совпадают, атриб
}
```



3. Архитектура подключения (Singleton/Client)

Проблема «Множественных подключений»

В WinForms начинающие разработчики часто создают new MongoClient везде, где нужно нажать кнопку.

Результат: Каждое нажатие открывает новый "канал" связи.

Последствия: Трата оперативной памяти, исчерпание лимита подключений на сервере и медленная работа интерфейса.

Решение: Паттерн Singleton (Одиночка)

Singleton — это объект, который существует в программе в единственном экземпляре. Для работы с MongoDB мы создадим класс-посредник (обычно его называют DbContext или MongoService), который один раз подключается к базе и раздает это подключение всем формам.

Реализация класса MongoService

Создадим в проекте новый класс MongoService.cs. Этот код станет "сердцем" приложения для работы с данными:

```
internal class MongoService
{
    // Статическая переменная для хранения единственного экземпляра сервиса
    private static MongoService _instance;

    // Поля для работы с MongoDB
    private readonly IMongoDatabase _database;

    // Приватный конструктор (никто не сможет написать new MongoService() снаружи)
    Ссылок: 1
    private MongoService()
    {
        var client = new MongoClient("mongodb://localhost:27017");
        _database = client.GetDatabase("College");
    }

    // Свойство для получения экземпляра класса (точка доступа)
    Ссылок: 0
    public static MongoService Instance => _instance ??= new MongoService();

    // Универсальный метод для получения любой коллекции
    Ссылок: 0
    public IMongoCollection<T> GetCollection<T>(string name)
    {
        return _database.GetCollection<T>(name);
    }
}
```

Как это использовать в любой форме?

Теперь вам не нужно настраивать MongoClient в каждой форме. Достаточно вызвать MongoService.Instance.

Пример в Form1.cs:

```
private IMongoCollection<Student> _students;

Ссылок: 1
public Form1()
{
    InitializeComponent();

    // Получаем коллекцию через наш сервис-синглтон
    _students = MongoService.Instance.GetCollection<Student>("Students");
}
```

Преимущества такого подхода:

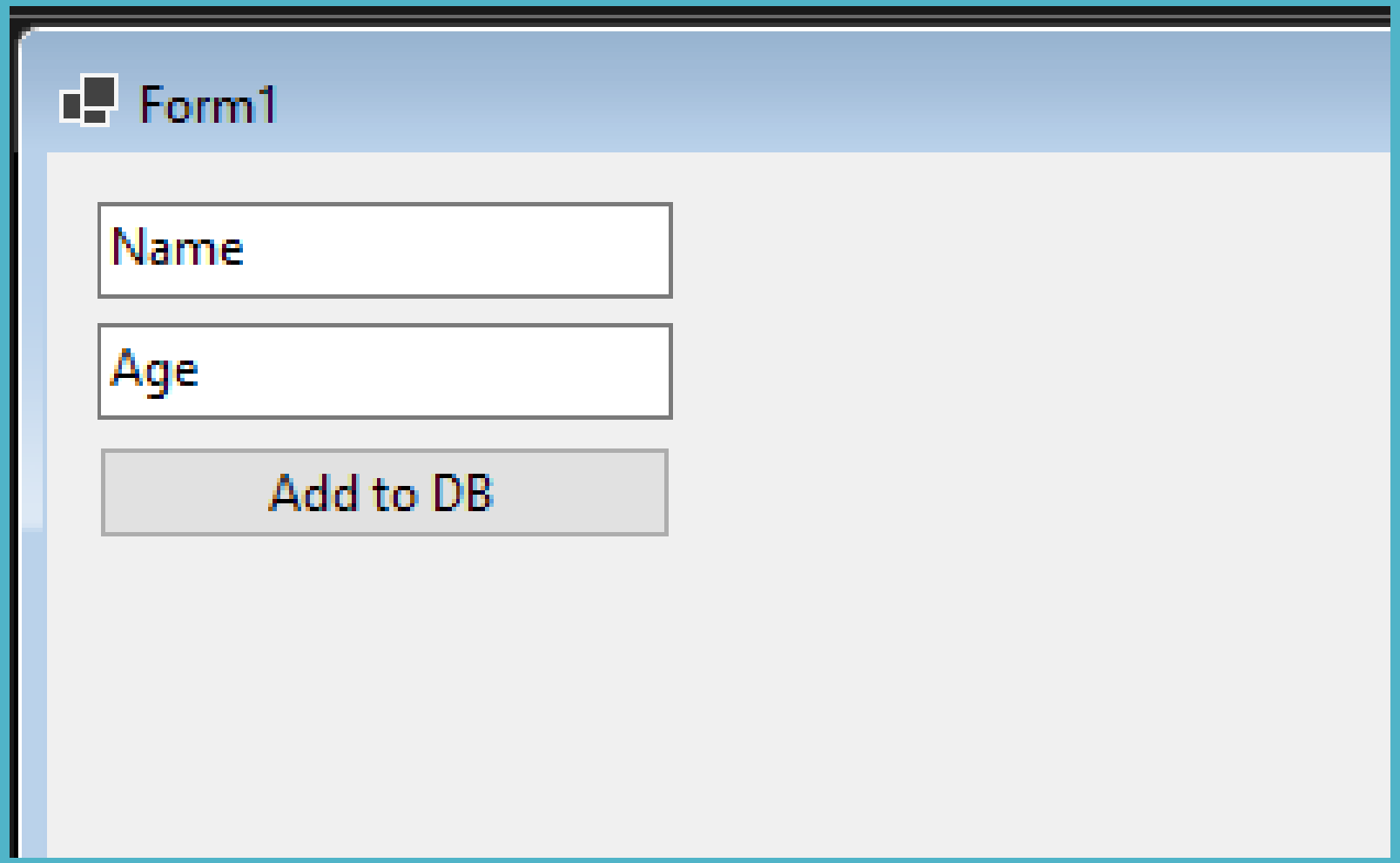
- Централизованная настройка: Если адрес базы изменится, вы поменяете его только в одном месте (в классе MongoService).
- Экономия ресурсов: Одно приложение — одно соединение.
- Удобство: Вам не нужно передавать переменные client или database между формами через конструкторы.

4. Реализация CRUD на C#

Мы разберем, как выполнять четыре основные операции (Create, Read, Update, Delete), используя наш класс Student и асинхронный подход.

В современном C# операции с базами данных принято делать асинхронными (async/await). Это гарантирует, что форма WinForms не будет «зависать», пока программа ждет ответа от сервера MongoDB.

Добавим на форму два текстовых поля для данных и кнопку:

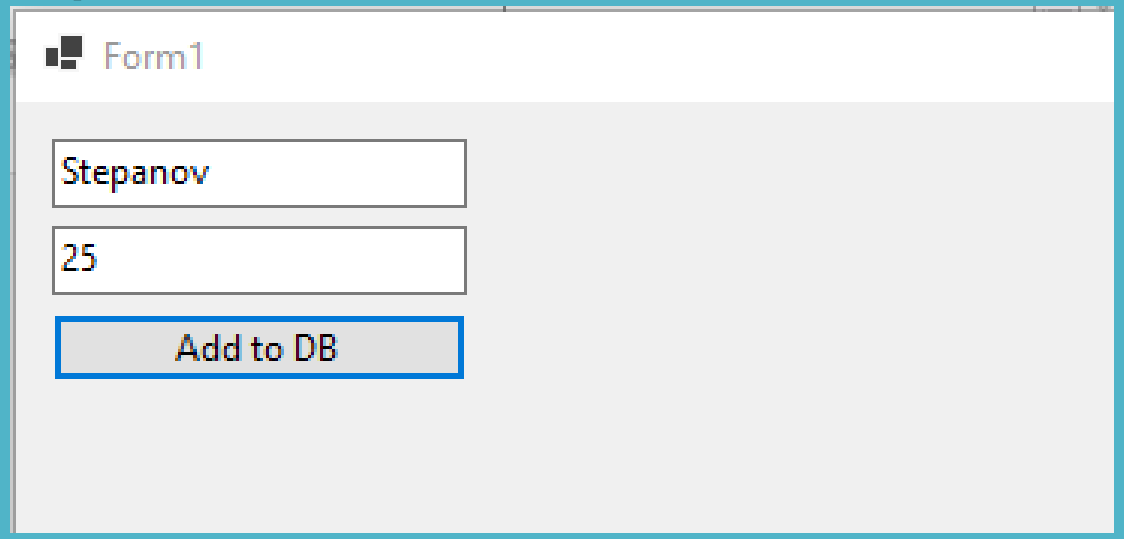


Create (Создание записи)

Чтобы добавить новый объект в базу, мы создаем экземпляр нашего класса и вызываем метод **InsertOneAsync**.

```
private async void btnAdd_Click(object sender, EventArgs e)
{
    var newStudent = new Student
    {
        Name = txtName.Text,
        age = int.Parse(txtAge.Text),
    };

    await _students.InsertOneAsync(newStudent);
    toolStripStatusLabel1.Text = "Студент успешно добавлен!";
}
```

Проверяем, работает:

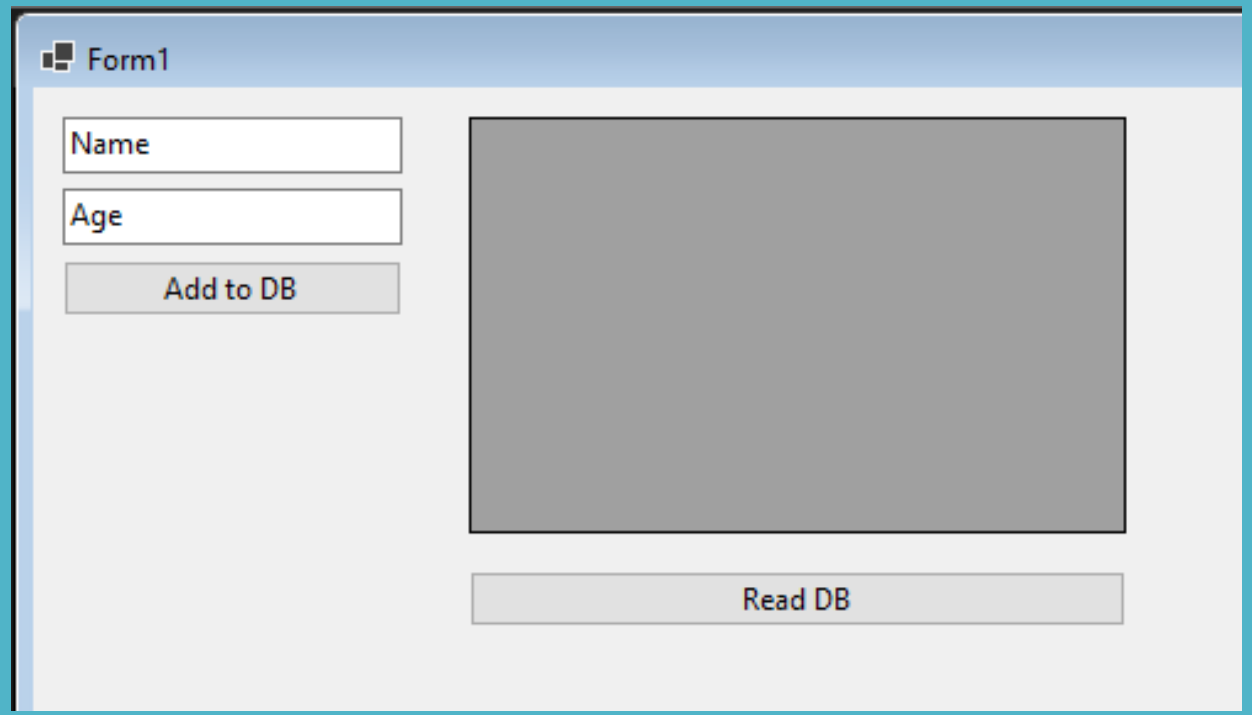



```
_id: ObjectId('695bccb4124e03d7131a0e1a')
name : "Petroff"
age : 20

_id: ObjectId('695bcdc0124e03d7131a0e1c')
name : "Ivanoff"
age : 22

_id: ObjectId('695cf35ee06580e802af14e3')
name : "Stepanov"
age : 25
```

Read (Чтение / Получение данных)

Для получения всех данных мы используем метод Find. Чтобы отобразить их в DataGridView, удобнее всего преобразовать результат в List.

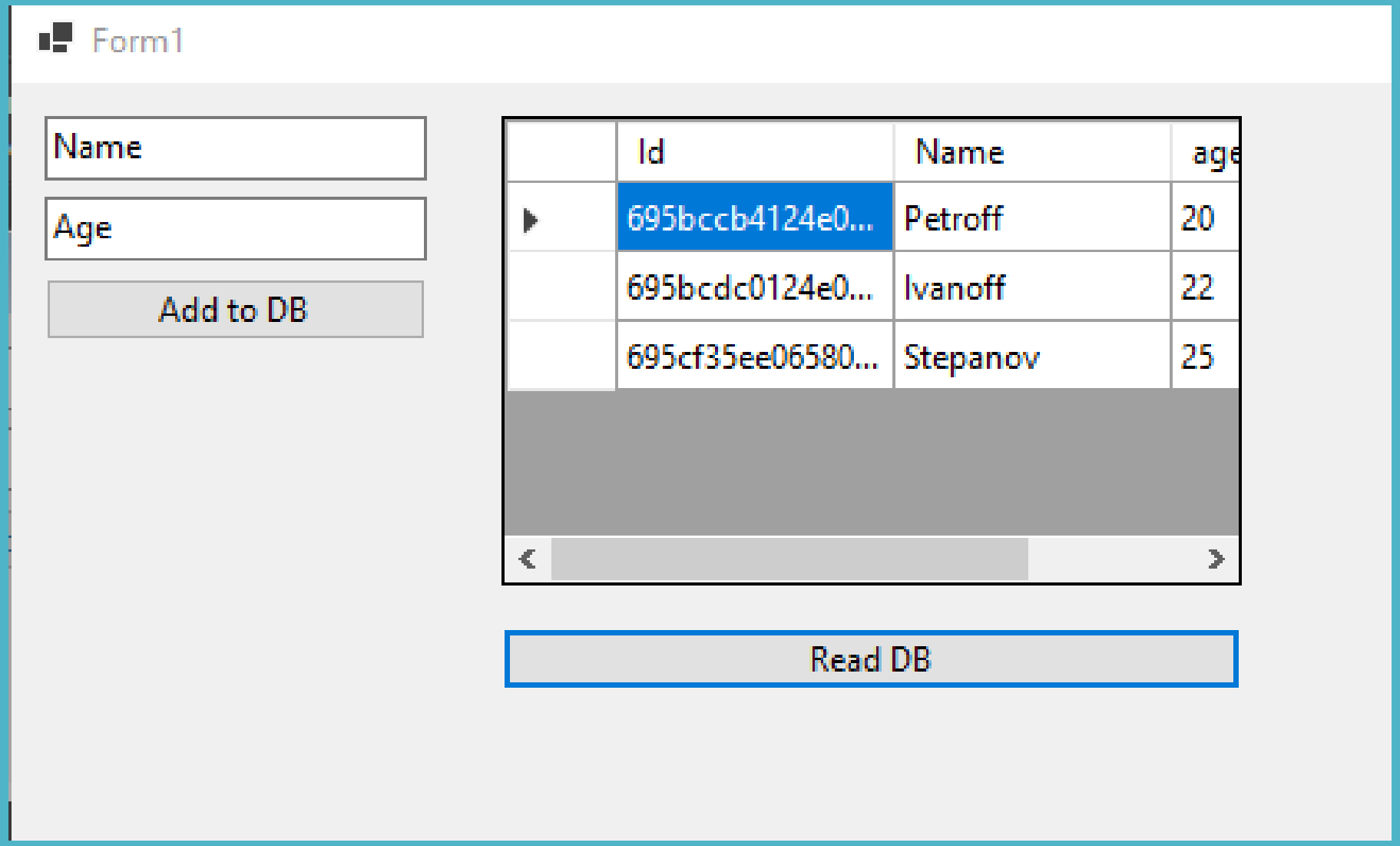
Добавим DataGridView и кнопку чтения на форму:



Добавим обработчик для кнопки:

```
private async void btnLoad_Click(object sender, EventArgs e)
{
    // Фильтр-пустышка (найти всё)
    var filter = Builders<Student>.Filter.Empty;

    // Получаем данные асинхронно
    var list = await _students.Find(filter).ToListAsync();

    // Привязываем список к таблице на форме
    dataGridView1.DataSource = list;
}
```

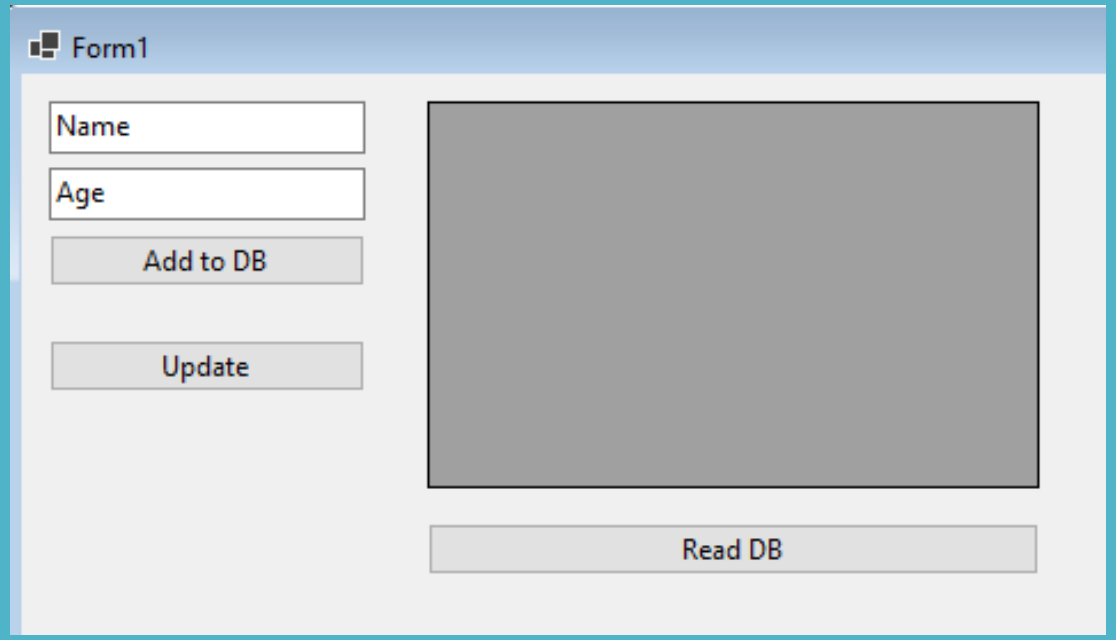
Проверяем, работает:



Update (Обновление записи)

Обычно обновление делается по идентификатору Id. Мы используем ReplaceOneAsync (полная замена документа) или UpdateOneAsync (изменение конкретного поля).

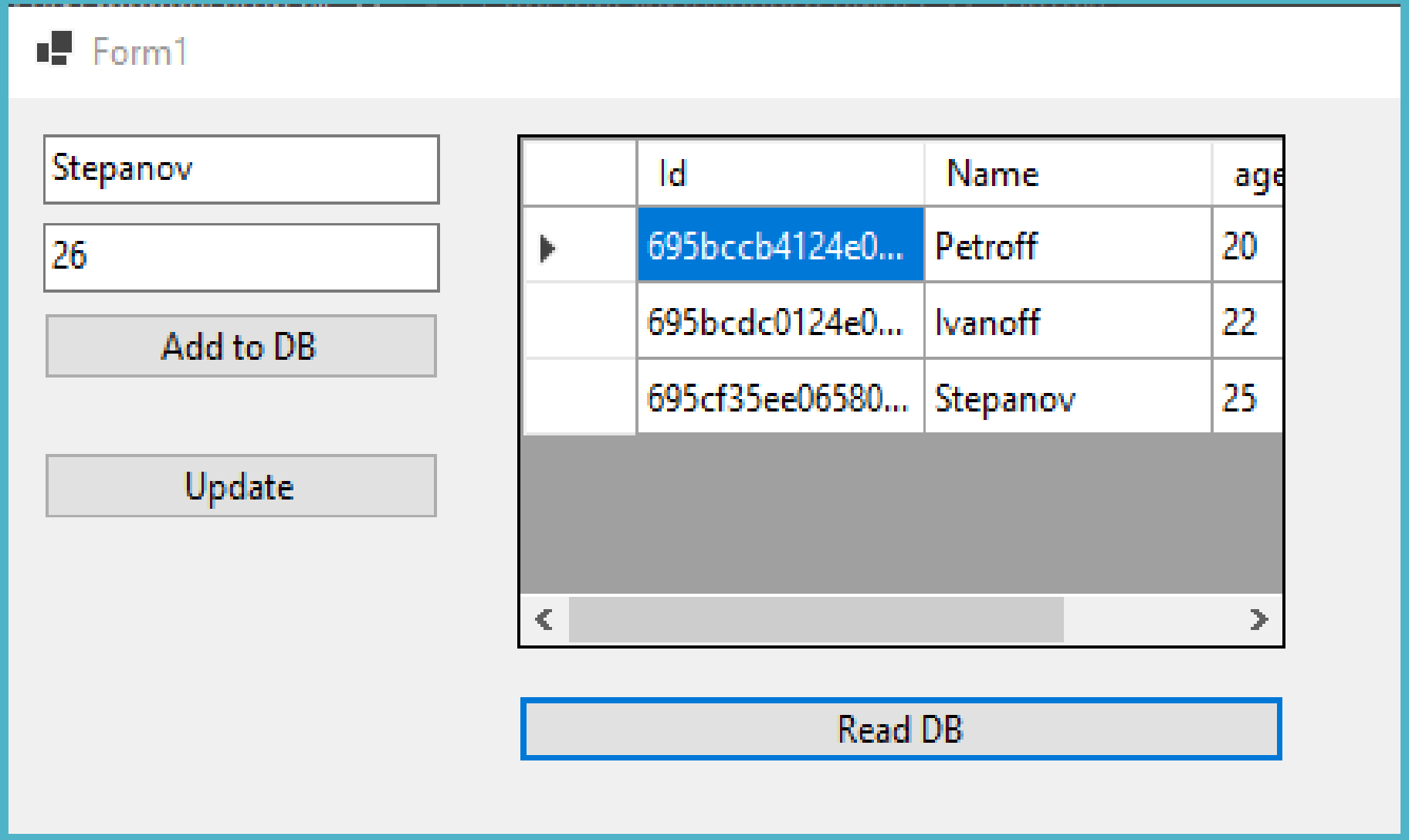
Добавим еще одну кнопку:

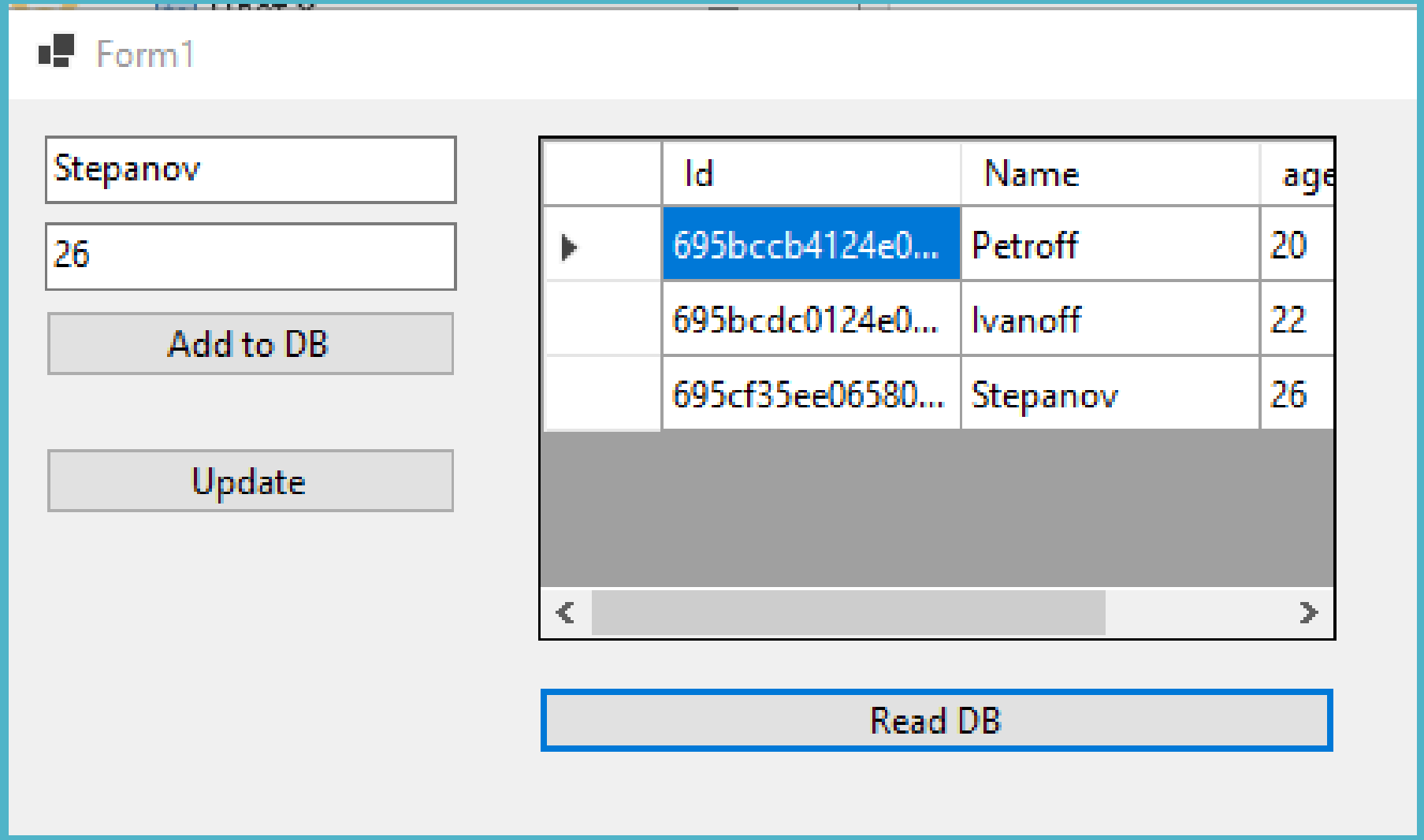


Обработчик:

```
Ссылок: 0
private async void btnUpdate_Click(object sender, EventArgs e)
{
    // Допустим, мы меняем возраст студента по его имени
    var filter = Builders<Student>.Filter.Eq(s => s.Name, txtName.Text);
    var update = Builders<Student>.Update.Set(s => s.age, int.Parse(txtAge.Text));

    await _students.UpdateOneAsync(filter, update);
}
```

Проверяем:


Form1
Stepanov
26
Add to DB
Update
Id
Name
age
695bccb4124e0...
Petroff
20
695bcdc0124e0...
Ivanoff
22
695cf35ee06580...
Stepanov
25
Read DB

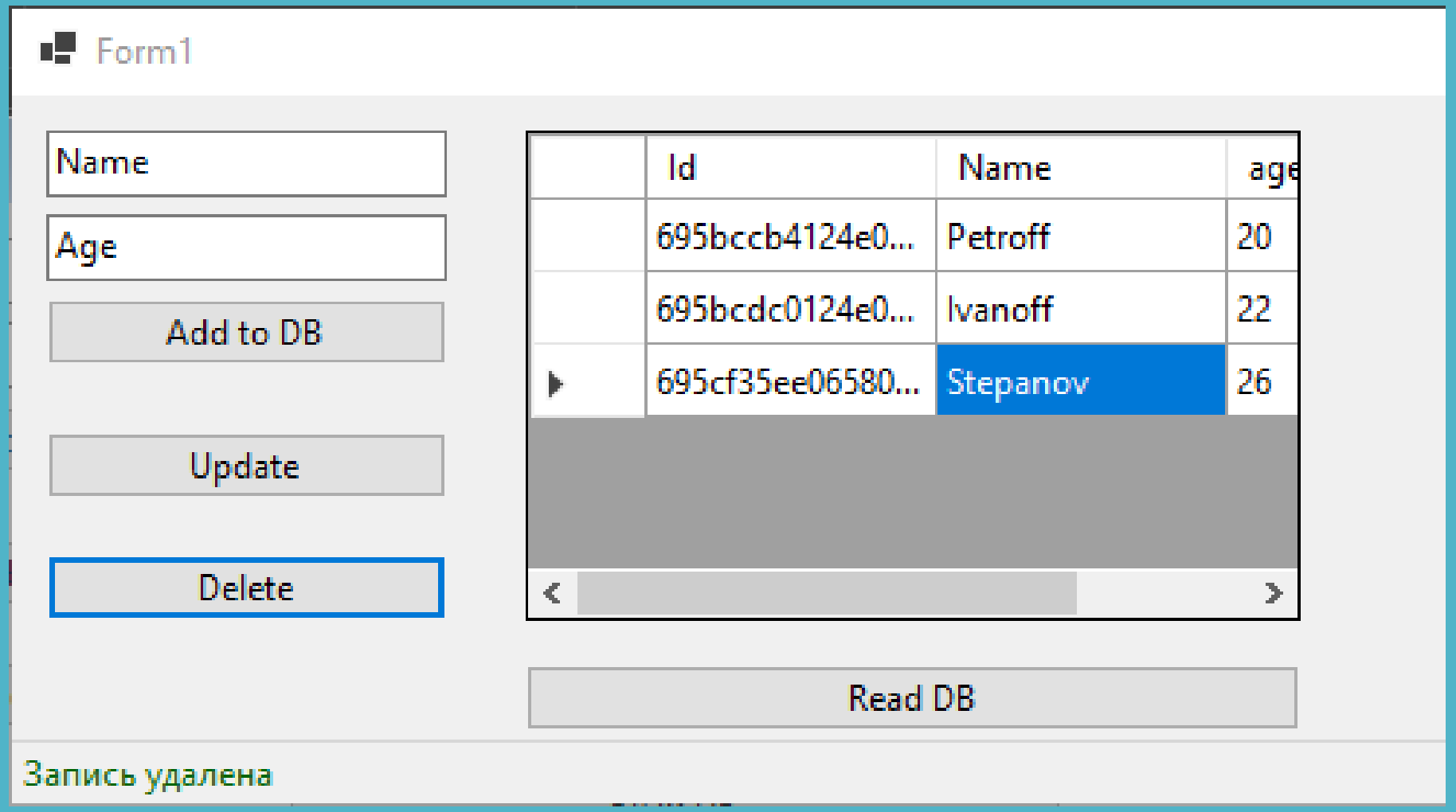

Form1
Stepanov
26
Add to DB
Update
Id
Name
age
695bccb4124e0...
Petroff
20
695bcdc0124e0...
Ivanoff
22
695cf35ee06580...
Stepanov
26
Read DB

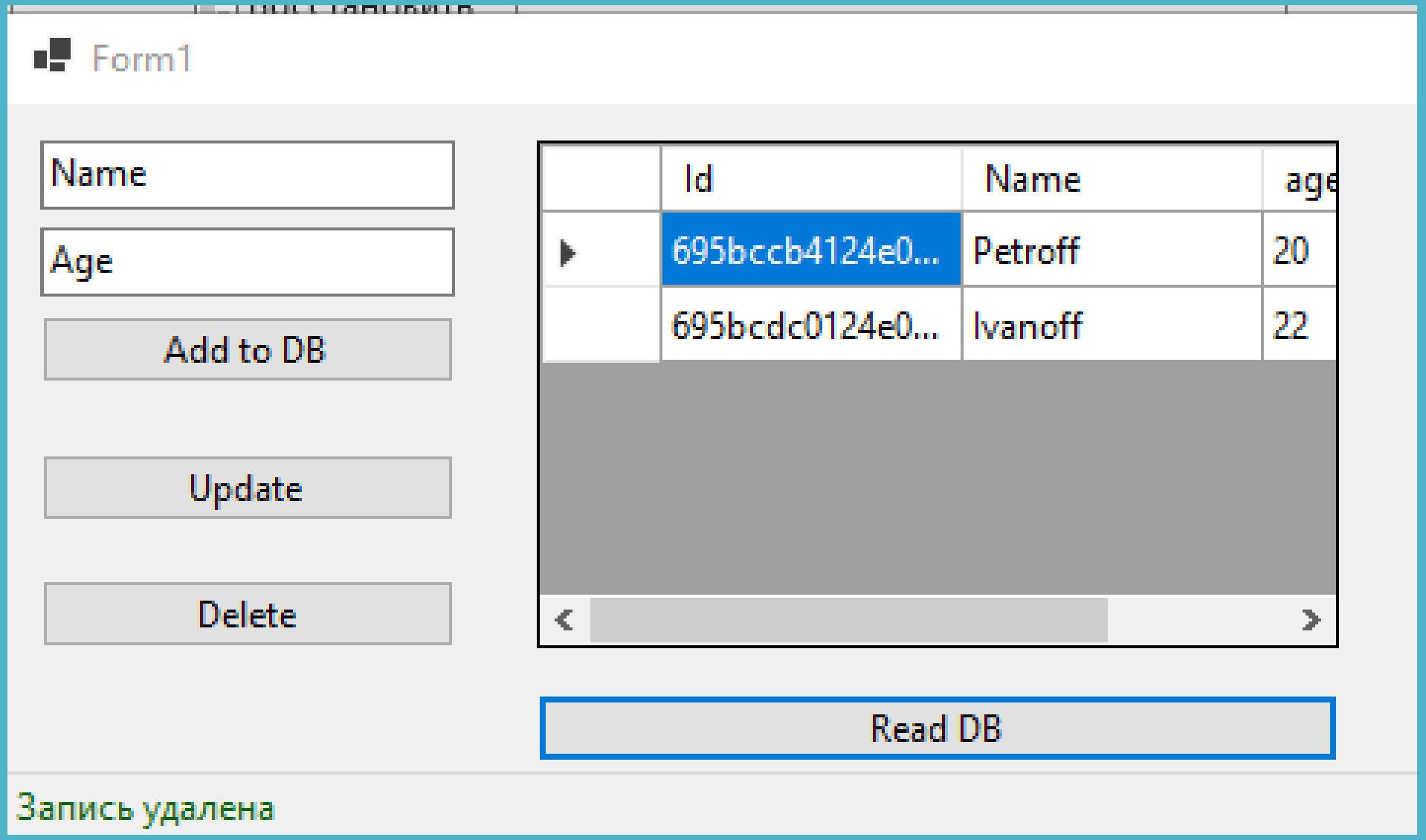
Delete (Удаление записи)

Для удаления используется метод DeleteOneAsync с фильтром.

```
private async void btnDelete_Click(object sender, EventArgs e)
{
    // Удаляем студента, у которого Id совпадает с выбранным в таблице
    var id = dataGridView1.CurrentRow.Cells["Id"].Value.ToString();
    var filter = Builders<Student>.Filter.Eq(s => s.Id, id);

    await _students.DeleteOneAsync(filter);
    toolStripStatusLabel1.Text = "Запись удалена";
}
```

Проверяем:


Form1
Name
Age
Add to DB
Update
Delete
Id
Name
age
695bccb4124e0...
Petroff
20
695bcdc0124e0...
Ivanoff
22
695cf35ee06580...
Stepanov
26
Read DB
Запись удалена


Form1
Name
Age
Add to DB
Update
Delete
Id
Name
age
695bccb4124e0...
Petroff
20
695bcdc0124e0...
Ivanoff
22
Read DB
Запись удалена

Важные инструменты драйвера: Builders<T>

В коде выше вы заметили Builders<Student>. Это специальный «конструктор» запросов в MongoDB драйвере:

- Builders<T>.Filter: используется для создания условий поиска (равно, больше, содержит и т.д.).
- Builders<T>.Update: используется для описания того, какие именно поля нужно изменить.

5. Синхронизация с интерфейсом (Binding)

В WinForms данные не "текут" сами по себе.

Чтобы изменения в базе мгновенно отображались в таблице, а клик по строке заполнял текстовые поля, используется связка BindingSource + DataBindings.

Правильная инициализация BindingSource

Чтобы избежать ошибки ArgumentException (dataMember), нужно инициализировать источник данных с указанием типа.

Это позволяет системе заранее узнать структуру класса Student.

```
public partial class Form1 : Form
{
    private IMongoCollection<Student> _students;
    // Указываем тип заранее, чтобы DataBindings видел свойства Name и Age
    private BindingSource _studentBindingSource = new BindingSource { DataSource = typeof(Student) };
```

Однократная привязка (Data Binding)

В конструкторе формы мы "склеиваем" свойства объекта со свойствами элементов управления.

Теперь нам не нужно событие CellClick — данные будут подставляться сами при навигации по таблице.

```
public Form1()
{
    InitializeComponent();

    // Настраиваем таблицу один раз при запуске
    dataGridView1.DataSource = _studentBindingSource;

    // Привязываем текстовые поля к источнику данных
    // Теперь это не вызовет ошибку, так как BindingSource знает про класс Student
    txtName.DataBindings.Add("Text", _studentBindingSource, "Name", true, DataSourceUpdateMode.OnPropertyChanged);
    txtAge.DataBindings.Add("Text", _studentBindingSource, "Age", true, DataSourceUpdateMode.OnPropertyChanged);
```

Автоматическая загрузка при старте

Чтобы пользователь не видел пустую таблицу при запуске, вызываем метод загрузки сразу после того, как установили соединение с базой.

```
try
{
    // 1. Получаем коллекцию
    _students = MongoService.Instance.GetCollection<Student>("Students");

    // чтобы данные загружались сразу:
    btnLoad_Click(null, null);
```

Принудительная синхронизация после CRUD

После любой операции (добавление, удаление, правка) данные в MongoDB изменились, но список List<Student> в оперативной памяти — еще нет.

Чтобы обновить картинку, нужно заново выкачать данные.

```
private async void btnAdd_Click(object sender, EventArgs e)
{
    var newStudent = new Student
    {
        Name = txtName.Text,
        Age = int.Parse(txtAge.Text),
    };

    await _students.InsertOneAsync(newStudent);

    // Мгновенно обновляем таблицу на экране
    btnLoad_Click(null, null);
    toolStripStatusLabel1.Text = "Студент успешно добавлен!";
}
```

Главные преимущества такой архитектуры:

- Отсутствие ручного копирования: Вам больше не нужно писать txtName.Text = student.Name. Привязка делает это за вас.
- Безопасность ID: Мы можем скрыть колонку с ID в DataGridView, но она все равно будет доступна через _studentBindingSource.Current. Это позволяет удалять и обновлять записи без риска задеть однофамильцев.
- Единый источник истины: BindingSource всегда знает, какой объект сейчас выбран, и предоставляет его через свойство .Current.

Материалы лекций:

https://github.com/ShViktor72/Education2025