Introduction

This lesson will discuss BSON objects and how to map them to a C# object. It will also introduce a database and its C# equivalent which we will use for learning purposes.

WE'LL COVER THE FOLLOWING

Recap

In previous chapters, Mongo DB basics part 1 and part 2, we covered some of the foundations of this database. We learned how to create databases, collections, documents, and we learned how to use them. We also learned information about distributing MongoDB, shards, replica sets, and how to manipulate each of these entities

Nevertheless, there is no single line of code, and everything is explained in a general manner (more or less). MongoDB Shell Client was used to demonstrate all of the crucial operations, so readers were not exposed to any coding lessons.

That is why, here, we'll try to show you how MongoDB can be used, in the code, to automatically do some of the operations shown in the previous lessons. So, let's take a look at how MongoDB can be used in .NET, by implementing a simple MongoDB CRUD Repository.

Mapping BSON to Strongly Typed C# Object

If you have read the previous chapters, you will have learned that MongoDB stores documents in **BSON** format, which are basically *binary* JSON files (ok, it is a little bit more complicated than that).

When we use .NET driver, we consume documents through BsonDocument. What we want to do in this exercise is to map those BsonDocuments to strongly typed C# objects. But before that, let's see what our database, and its entities, look like.

Example Database

For the purpose of this exercise, we will use a database called – `blog`.

This database has a collection, `users`, which contains a document for each *user*.

The JSON document for an individual user would look like something like this:

```
{
  "_id" : ObjectId("59ce6b34f48f171624840b05"),
  "name" : "Nikola",
  "blog" : "rubikscode.net",
  "age" : 30,
  "location" : "Beograd"
}
```

C# Equivalent

An equivalent of this in C# looks like this:

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace MongoDb
{
    /// <summary>
    /// Class used in business logic to represent user.
    /// </summary>
    public class User
    {
        [BsonId]
        public ObjectId Id { get; set; }
        [BsonElement("name")]
        public string Name { get; set; }
        [BsonElement("blog")]
        public string Blog { get; set; }
        [BsonElement("age")]
        public int Age { get; set; }
        [BsonElement("location")]
        public string Location { get; set; }
    }
}
```

Explanation

You will notice that there are attributes for each property of this class. These attributes allow us to automatically map data from BsonDocument to our class.

`BsonId` **(line 11)** is used to map a unique document identifier. Every document in MongoDB has the element:

```
ObjectId.BsonElement(field_name)
```

Its type is used to map other fields from the object on the object properties.

Since we know that document databases don't have a strict schema, it is possible to have more fields in the JSON document than we really want to use in our application.

Also, it is possible that we may not want to pick up all the fields from the database. For this, we can use

```
BsonIgnoreExtraElements
```

attribute on the class itself.

In this lesson, we just touched upon the database, and its structure, that we will be using in this chapter. In the next lesson, we will further discuss the operations that can be performed.