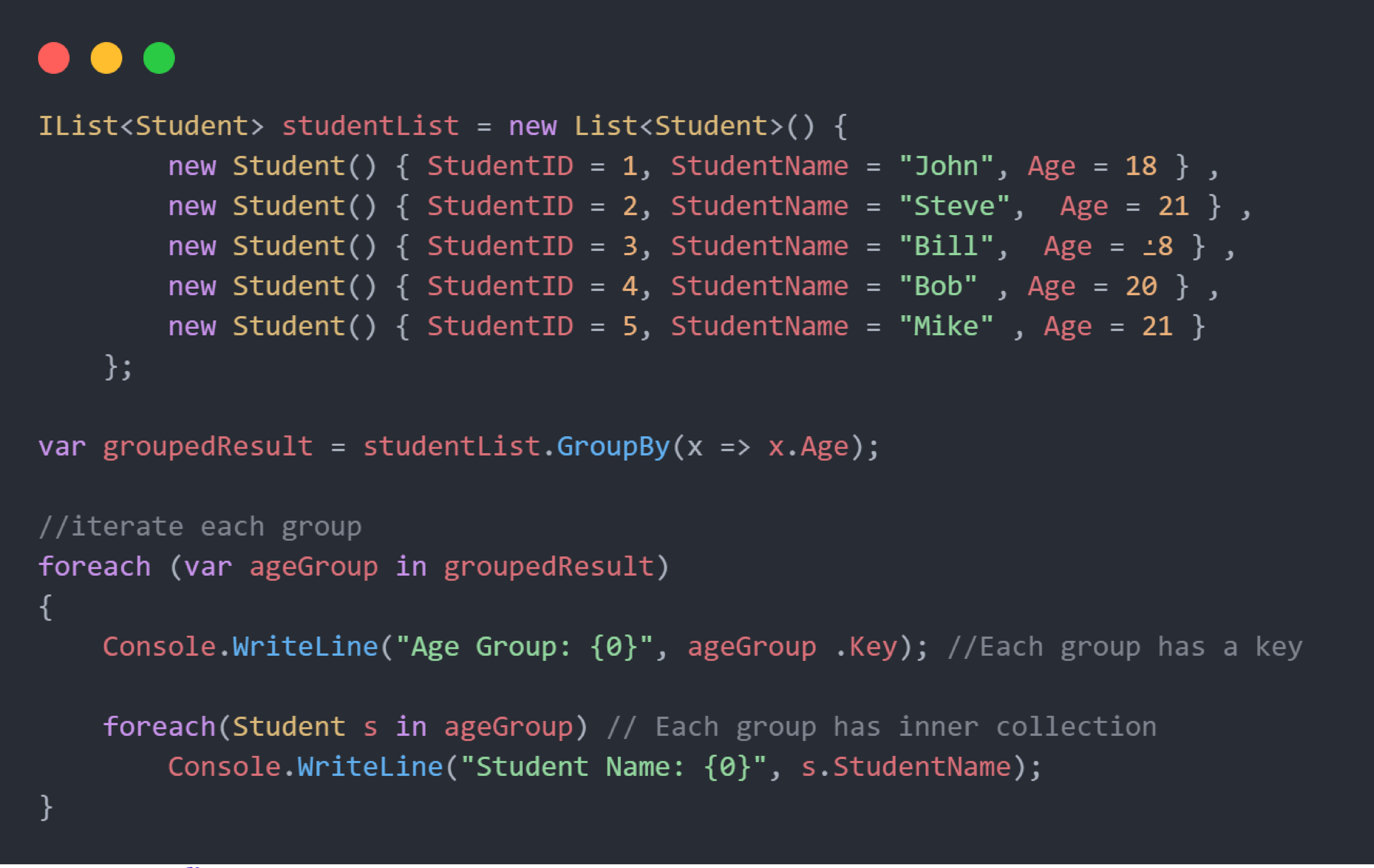
.NET Core

Linq GroupBy

GROUPING YOUR DATA EASILY

@romain-od in
www.code-review.tech

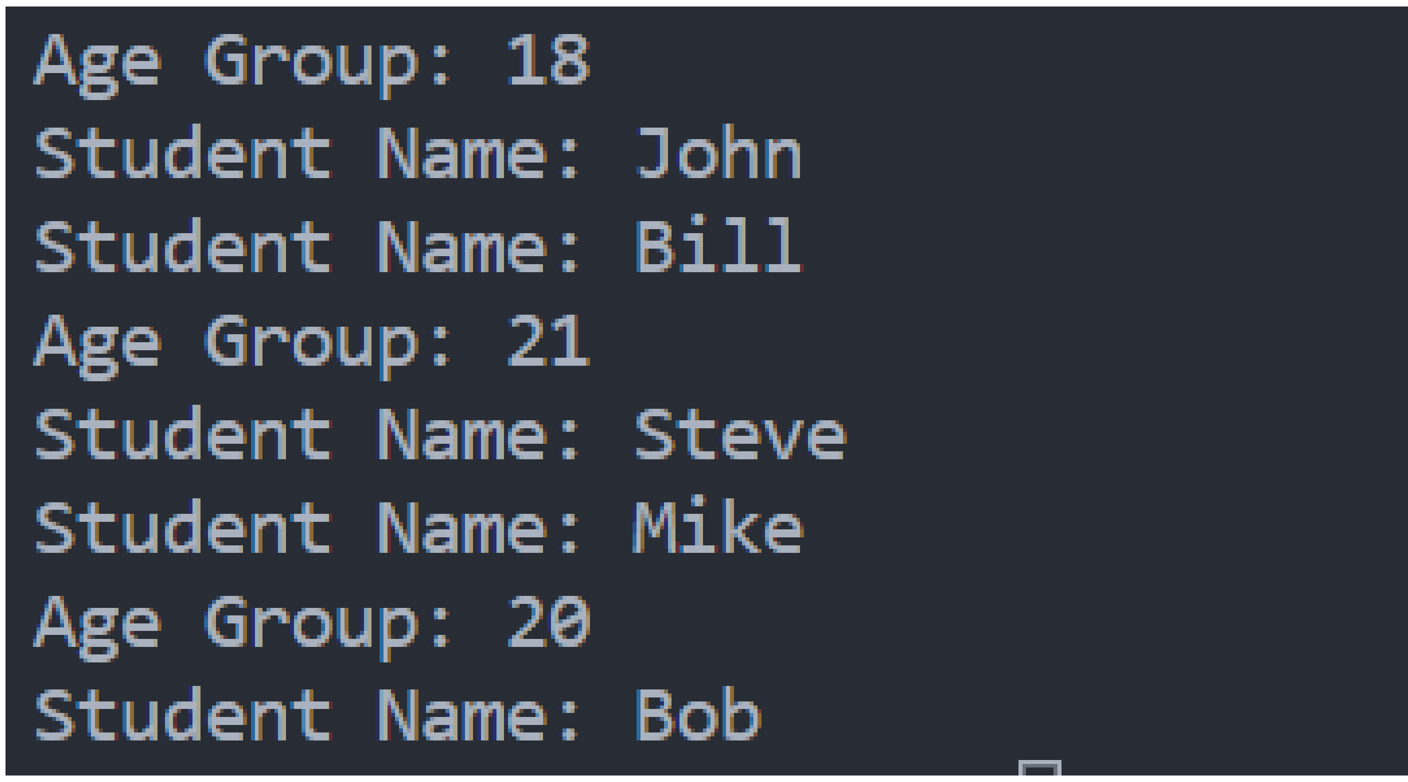


```
IList<Student> studentList = new List<Student>() {
        new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,
        new Student() { StudentID = 2, StudentName = "Steve",  Age = 21 } ,
        new Student() { StudentID = 3, StudentName = "Bill",  Age = 18 } ,
        new Student() { StudentID = 4, StudentName = "Bob" , Age = 20 } ,
        new Student() { StudentID = 5, StudentName = "Mike" , Age = 21 }
    };

var groupedResult = studentList.GroupBy(x => x.Age);

//iterate each group
foreach (var ageGroup in groupedResult)
{
    Console.WriteLine("Age Group: {0}", ageGroup .Key); //Each group has a key

    foreach(Student s in ageGroup) // Each group has inner collection
        Console.WriteLine("Student Name: {0}", s.StudentName);
}
```



```
Age Group: 18
Student Name: John
Student Name: Bill
Age Group: 21
Student Name: Steve
Student Name: Mike
Age Group: 20
Student Name: Bob
```

@romain-od

www.code-review.tech

Students of the same age will be in the same collection and each grouped collection will have a key and inner collection, where the key will be the age and the inner collection will include students whose age is matched with a key.

Let's see what happens when we serialize it as a result of a minimal API endpoint, for example

```
app.MapGet("", () => {
    return studentList.GroupBy(x => x.Age);
});
```

```
[
    [
      {
        "studentID": 1,
        "studentName": "John",
        "age": 18
      },
      {
        "studentID": 3,
        "studentName": "Bill",
        "age": 18
      }
    ],
    [
      {
        "studentID": 2,
        "studentName": "Steve",
        "age": 21
      },
      {
        "studentID": 5,
        "studentName": "Mike",
```

@romain-od

www.code-review.tech

Our student list is split into multiple sub-collections, and each one of those has a Key property of whatever type we grouped on. In our example, it is the age.

Imagine if we don't want the entire body of our student item in our grouped sub-collection. In that case, we can use the overload method that takes a *selector*.

```csharp
app.MapGet("", () => {
    return studentList.GroupBy(x => x.Age,
                               x => x.StudentName);
});
```

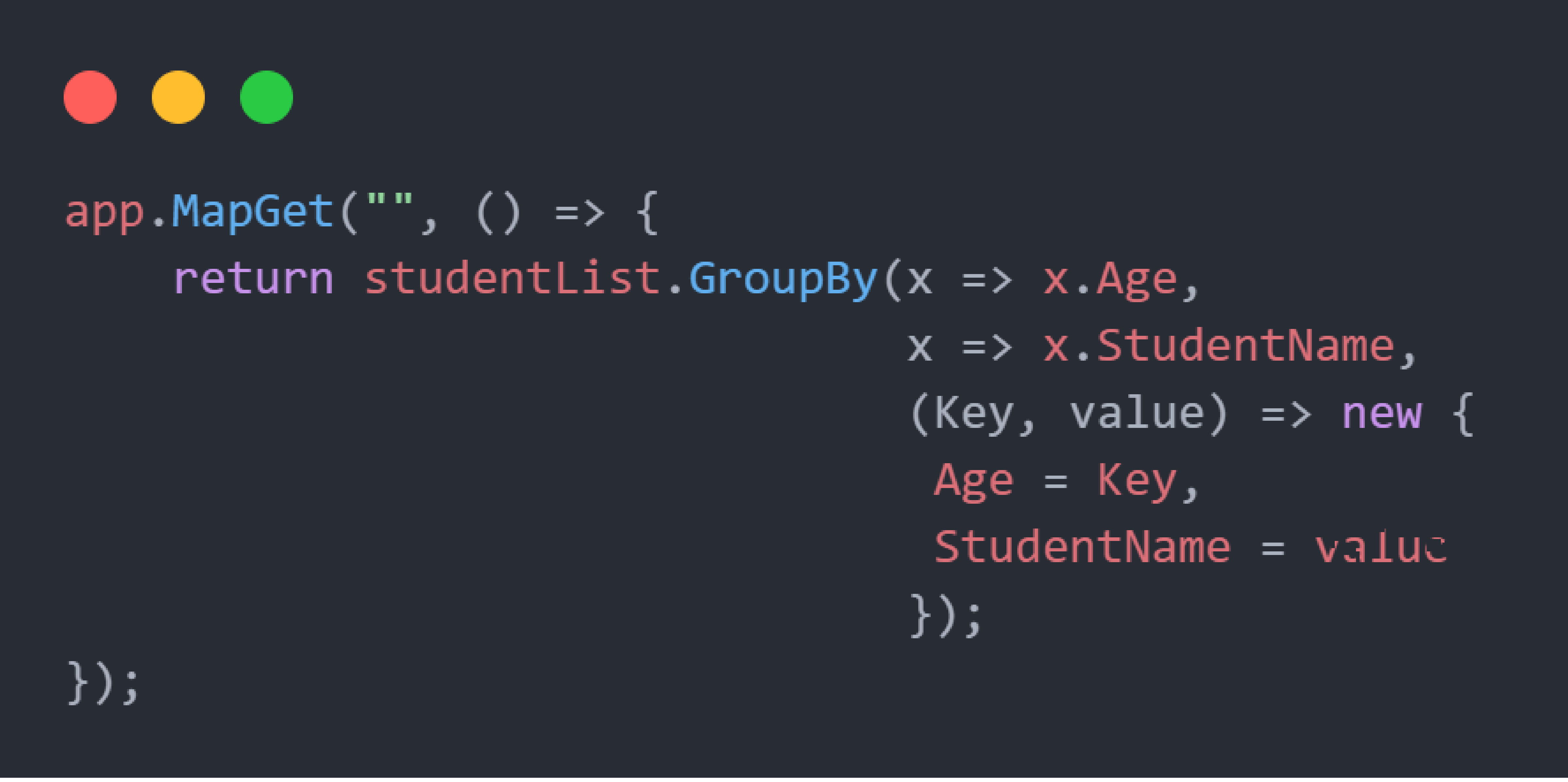
```
[
    [
      "John",
      "Bill"
    ],
    [
      "Steve",
      "Mike"
    ],
    [
      "Bob"
    ]
  ]
```

@romain-od

www.code-review.tech

You can notice that the group key is not included in the serialized results and so we don’t see the age listed.

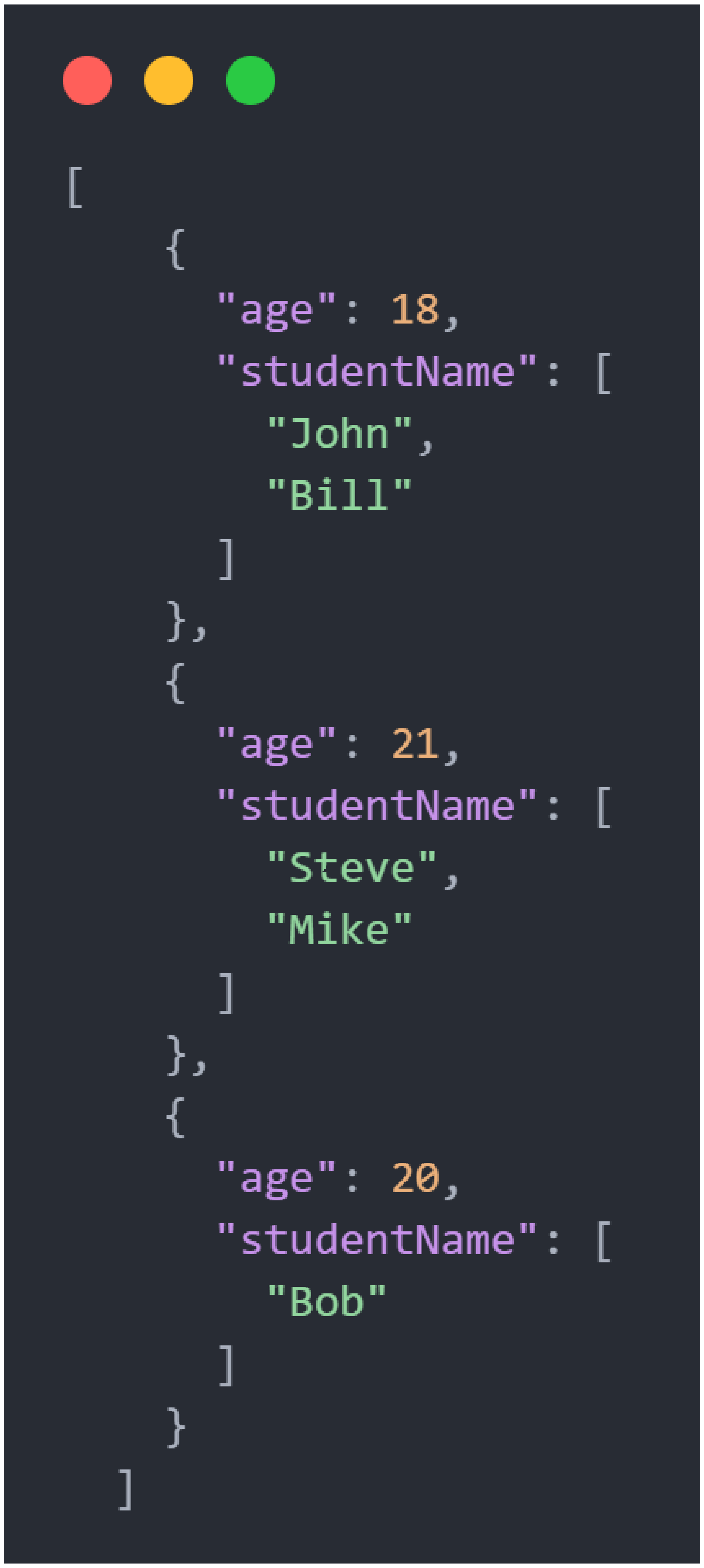
Let's resolve this using result selector



```
app.MapGet("", () => {
    return studentList.GroupBy(x => x.Age,
                               x => x.StudentName,
                               (Key, value) => new {
                                Age = Key,
                                StudentName = value
                               });
});
```

@romain-od

www.code-review.tech



```
[
    {
      "age": 18,
      "studentName": [
        "John",
        "Bill"
      ]
    },
    {
      "age": 21,
      "studentName": [
        "Steve",
        "Mike"
      ]
    },
    {
      "age": 20,
      "studentName": [
        "Bob"
      ]
    }
  ]
```

@romain-od
www.code-review.tech

THANKS FOR READING

SHARE YOUR THOUGHTS

FOLLOW ME FOR MORE SHARING

@romain-od

www.code-review.tech