Improving Data Access with Abstractions

Steve Smith

@ardalis

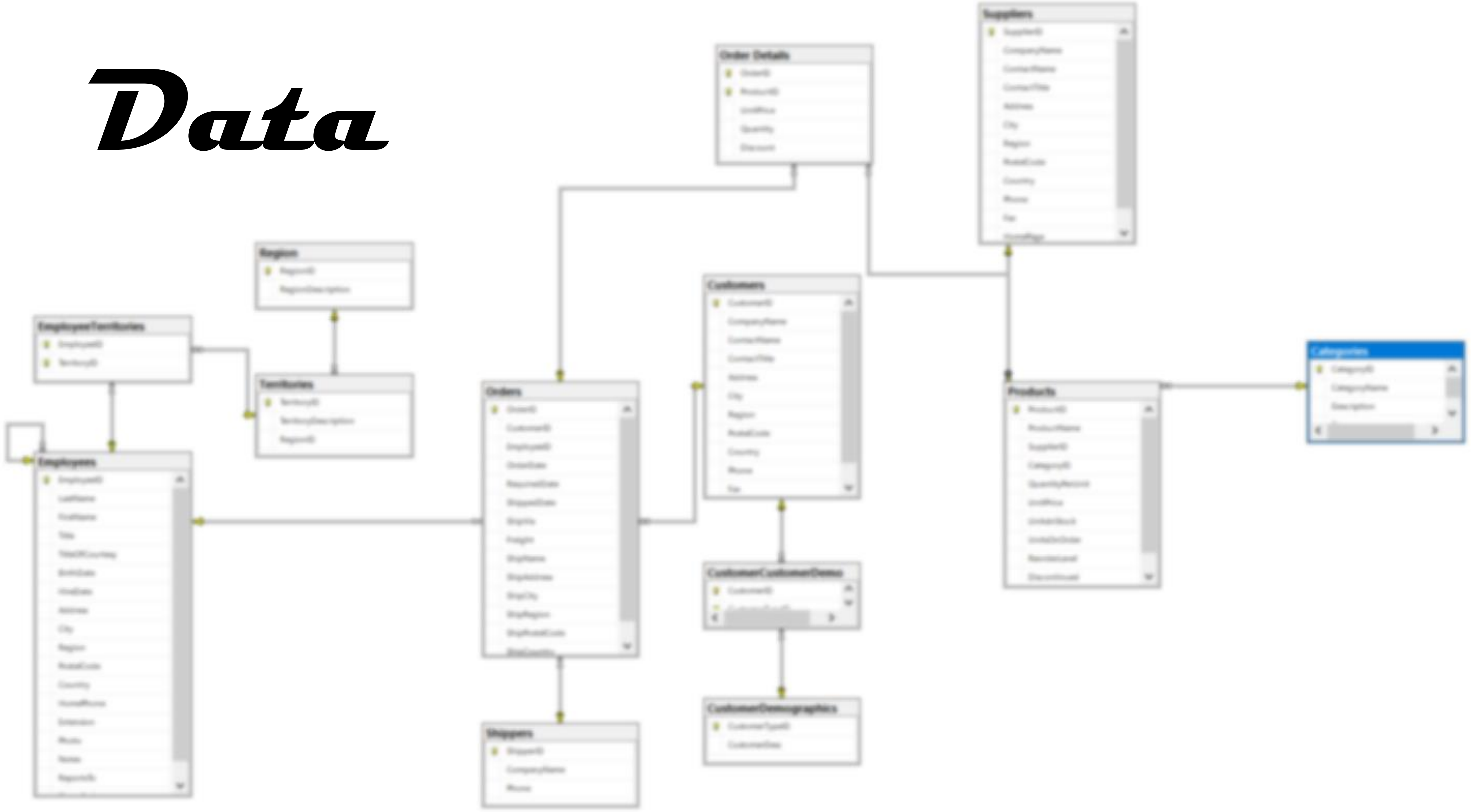
steve@nimblepros.com | NimblePros.com

NimblePros

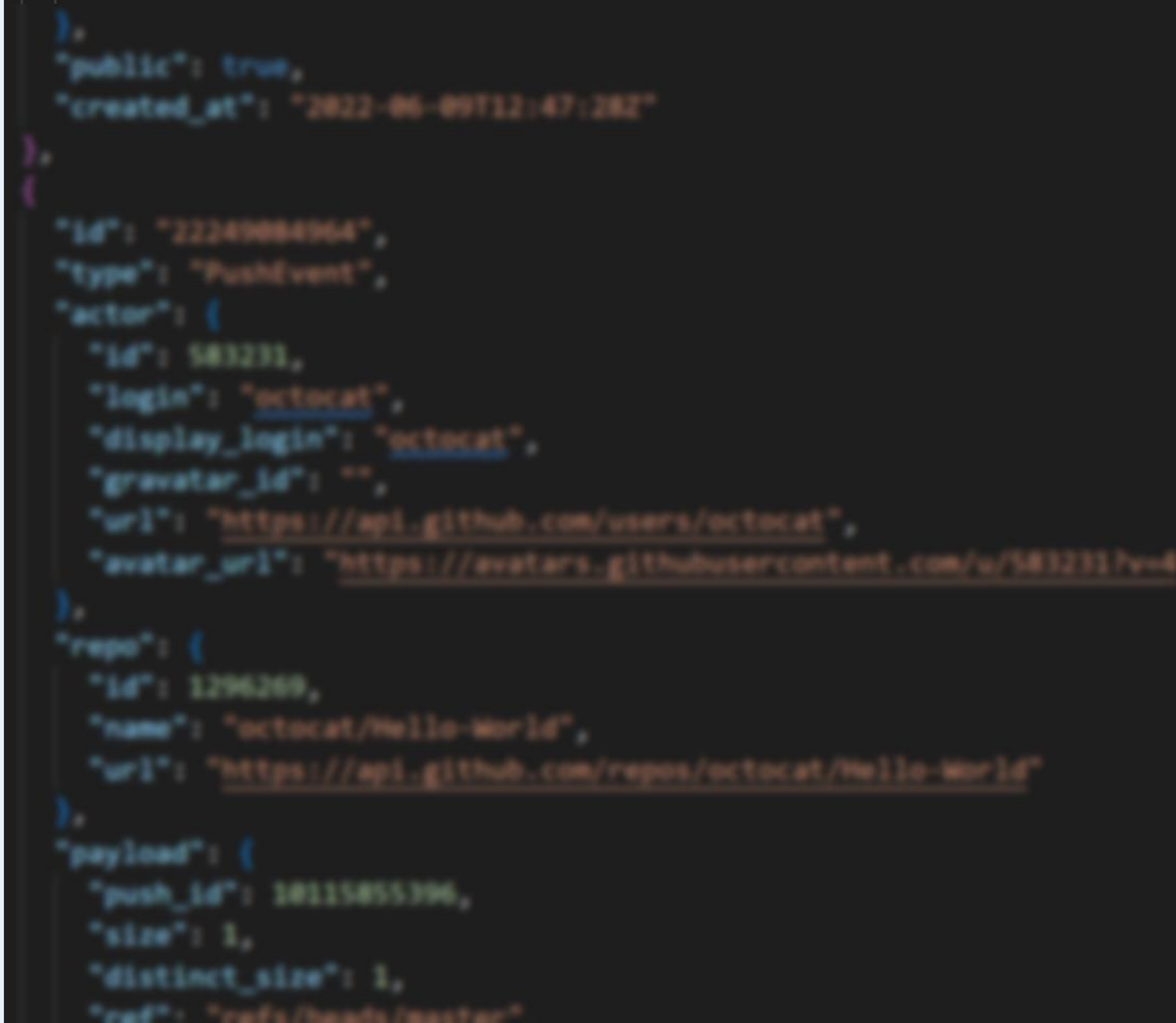
Better software development. Delivered.

@ardalis@fosstodon.org

Data


Order Details
Suppliers
Region
EmployeeTerritories
Territories
Customers
Categories
Orders
Products
Employees
CustomerCustomerDemo
CustomerDemographics
Shippers

Data



```
    },
    "public": true,
    "created_at": "2022-06-09T12:47:28Z"
  },
  {
    "id": "22249084964",
    "type": "PushEvent",
    "actor": {
      "id": 583231,
      "login": "octocat",
      "display_login": "octocat",
      "gravatar_id": "",
      "url": "https://api.github.com/users/octocat",
      "avatar_url": "https://avatars.githubusercontent.com/u/583231?v=
    },
    "repo": {
      "id": 1296269,
      "name": "octocat/Hello-World",
      "url": "https://api.github.com/repos/octocat/Hello-World"
    },
    "payload": {
      "push_id": 10115855396,
      "size": 1,
      "distinct_size": 1,
      "ref": "refs/heads/master",
```

Bring Me The Data!

Fetching data the old-fashioned way

Using ADO.NET

```
var authors = new List<AuthorDTO>();
using var conn = new SqlConnection(_connString);
var sql = "SELECT * FROM Authors";
var cmd = new SqlCommand(sql, conn);
conn.Open();
using var reader = cmd.ExecuteReader();
if(reader.HasRows)
{
    while (reader.Read())
    {
        var author = new AuthorDTO();
        author.Id = reader.GetInt32(0);
        author.Name = reader.GetString(1);
        authors.Add(author);
    }
}
```

Using ADO.NET

```csharp
    var author = new AuthorWithCoursesDTO();
    using var conn = new SqlConnection(_connString);
    var sql = @"SELECT a.Id, a.Name, ca.RoyaltyPercentage, ca.CourseId, ca.AuthorId, c.Title
FROM Authors a
LEFT JOIN CourseAuthor ca ON a.Id = ca.AuthorId
LEFT JOIN Courses c ON c.Id = ca.CourseId
WHERE a.Id = @AuthorId";
    using var cmd = new SqlCommand(sql, conn);
    cmd.Parameters.AddWithValue("@AuthorId", id);
    conn.Open();
    _logger.LogInformation("Executing query: {sql}, {parameters}", sql, cmd.Parameters);
    using var reader = cmd.ExecuteReader();
    if (reader.HasRows)
    {
        while (reader.Read())
        {
            author.Id = reader.GetInt32(0);
            author.Name = reader.GetString(1);
            if (!reader.IsDBNull(3))
            {
                author.Courses.Add(new CourseDTO
                {
                    Id = reader.GetInt32(3),
                    AuthorId = reader.GetInt32(4),
                    RoyaltyPercentage = reader.GetInt32(2),
                    Title = reader.GetString(5)
                });
            }
        }
    }
```

Code Characteristics

What are some characteristics we can use to measure a given approach in our code?

Code Characteristics (ilities)

Readability

Security

Performance

Testability

Maintainability

Pure ADO.NET Report Card

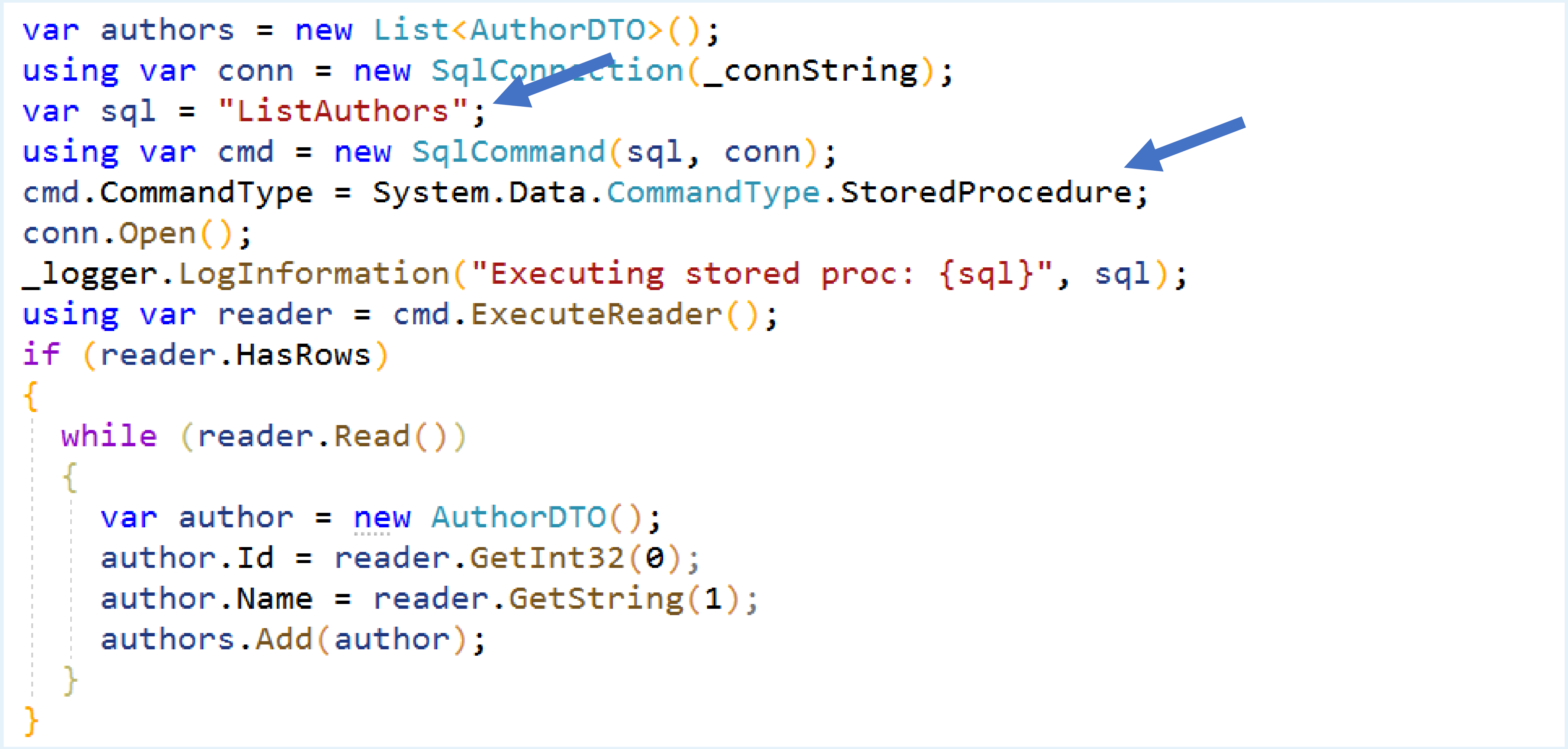
Area	Grade
Readability	☆☆☆
Security	☆☆
Performance	☆☆☆☆☆
Testability	☆
Maintainability	☆☆

Pain Driven Development

Bring Me The Data!

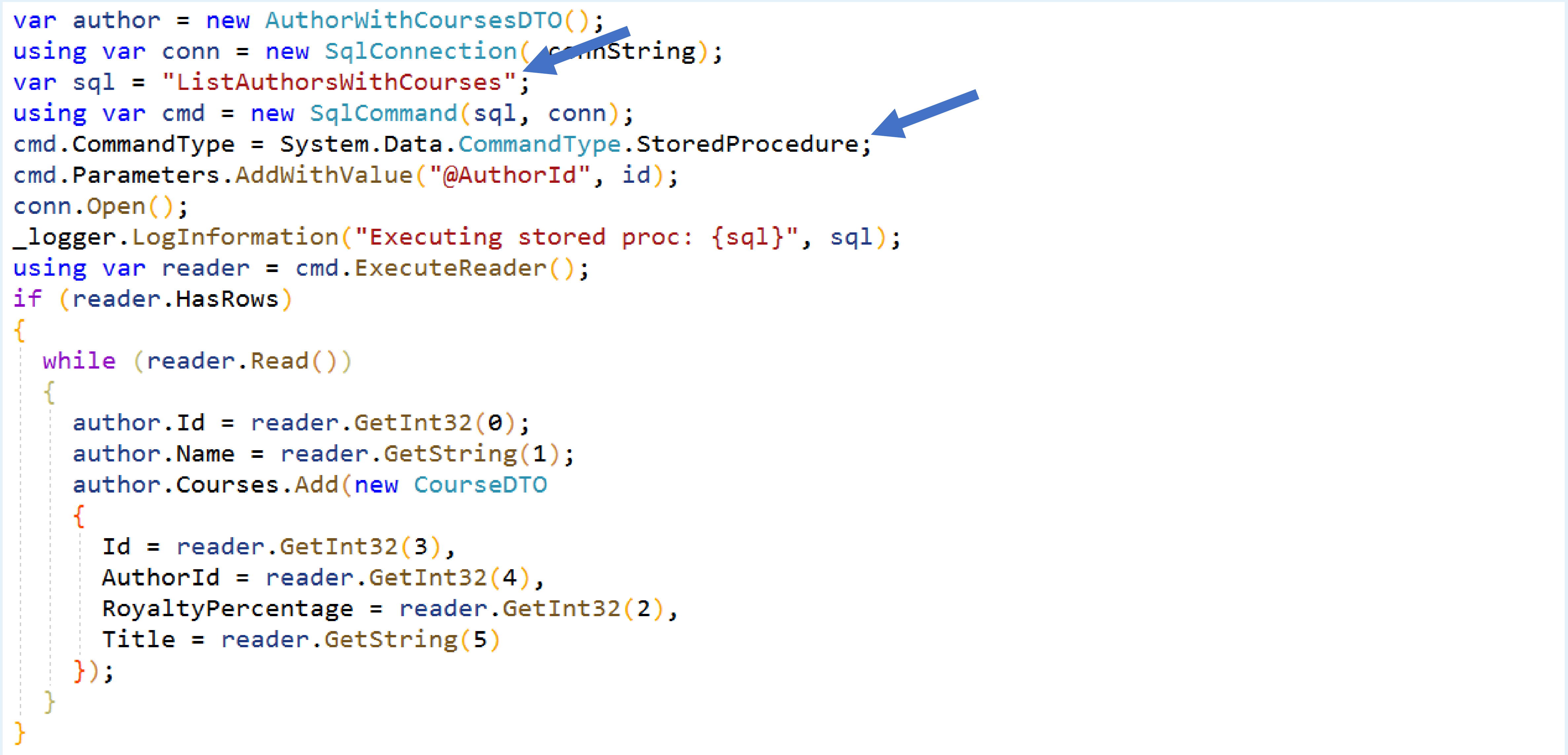
But with stored procedures

Using ADO.NET… with SPROCS!



```csharp
var authors = new List<AuthorDTO>();
using var conn = new SqlConnection(_connString);
var sql = "ListAuthors";
using var cmd = new SqlCommand(sql, conn);
cmd.CommandType = System.Data.CommandType.StoredProcedure;
conn.Open();
_logger.LogInformation("Executing stored proc: {sql}", sql);
using var reader = cmd.ExecuteReader();
if (reader.HasRows)
{
  while (reader.Read())
  {
    var author = new AuthorDTO();
    author.Id = reader.GetInt32(0);
    author.Name = reader.GetString(1);
    authors.Add(author);
  }
}
```

Using ADO.NET… with SPROCS!



```
var author = new AuthorWithCoursesDTO();
using var conn = new SqlConnection(connString);
var sql = "ListAuthorsWithCourses";
using var cmd = new SqlCommand(sql, conn);
cmd.CommandType = System.Data.CommandType.StoredProcedure;
cmd.Parameters.AddWithValue("@AuthorId", id);
conn.Open();
_logger.LogInformation("Executing stored proc: {sql}", sql);
using var reader = cmd.ExecuteReader();
if (reader.HasRows)
{
  while (reader.Read())
  {
    author.Id = reader.GetInt32(0);
    author.Name = reader.GetString(1);
    author.Courses.Add(new CourseDTO
    {
      Id = reader.GetInt32(3),
      AuthorId = reader.GetInt32(4),
      RoyaltyPercentage = reader.GetInt32(2),
      Title = reader.GetString(5)
    });
  }
}
```

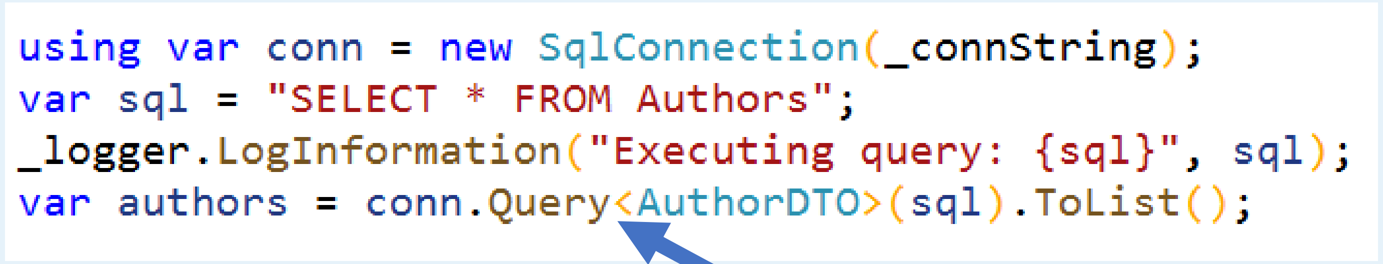
Pure ADO.NET (SPROCS) Report Card

Area	Grade
Readability	☆☆☆
Security	☆☆☆☆
Performance	☆☆☆☆☆
Testability	☆
Maintainability	☆☆☆

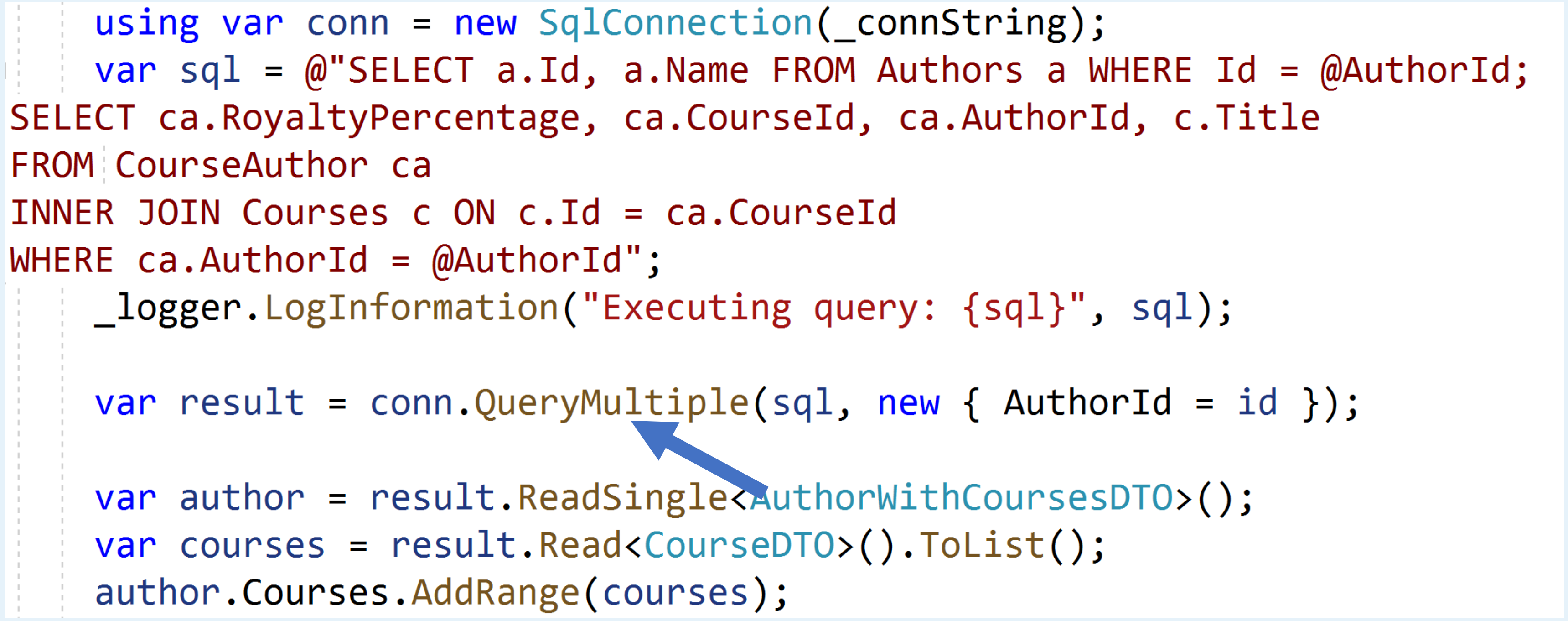
Shrink the Code!

What if we apply Dapper (not dapr)?

Using Dapper



```csharp
using var conn = new SqlConnection(_connString);
var sql = "SELECT * FROM Authors";
_logger.LogInformation("Executing query: {sql}", sql);
var authors = conn.Query<AuthorDTO>(sql).ToList();
```



```csharp
    using var conn = new SqlConnection(_connString);
    var sql = @"SELECT a.Id, a.Name FROM Authors a WHERE Id = @AuthorId;
SELECT ca.RoyaltyPercentage, ca.CourseId, ca.AuthorId, c.Title
FROM CourseAuthor ca
INNER JOIN Courses c ON c.Id = ca.CourseId
WHERE ca.AuthorId = @AuthorId";
    _logger.LogInformation("Executing query: {sql}", sql);

    var result = conn.QueryMultiple(sql, new { AuthorId = id });

    var author = result.ReadSingle<AuthorWithCoursesDTO>();
    var courses = result.Read<CourseDTO>().ToList();
    author.Courses.AddRange(courses);
```

Using Dapper with SPROCS

```
using var conn = new SqlConnection(_connString);
var sql = "ListAuthors";
var authors = conn.Query<AuthorDTO>(sql,
  commandType: CommandType.StoredProcedure)
    .ToList();
```

```
using var conn = new SqlConnection(_connString);
var sql = "ListAuthorsWithCoursesMulti";

_logger.LogInformation("Executing stored proc: {sql}", sql);

var result = conn.QueryMultiple(sql, new { AuthorId = id },
  commandType: CommandType.StoredProcedure);

var author = result.ReadSingle<AuthorWithCoursesDTO>();
var courses = result.Read<CourseDTO>().ToList();
author.Courses.AddRange(courses);
```

Dapper Report Card

Area	Grade
Readability	☆☆☆☆
Security	☆☆☆☆
Performance	☆☆☆☆☆
Testability	☆
Maintainability	☆☆☆

What about updates?

```
using var conn = new SqlConnection(_connString);
var sql = "UPDATE Authors SET name = @Name WHERE Id = @Id";

conn.Execute(sql, new { Name = value, Id = id });
```

Add Change Tracking!

Instead of a micro-ORM, how about a full ORM, like Entity Framework Core (EF Core)?

Using EF Core - Queries

```
var authors = _dbContext.Authors
    .Select(a => new AuthorDTO { Id = a.Id, Name = a.Name })
    .ToList();
```

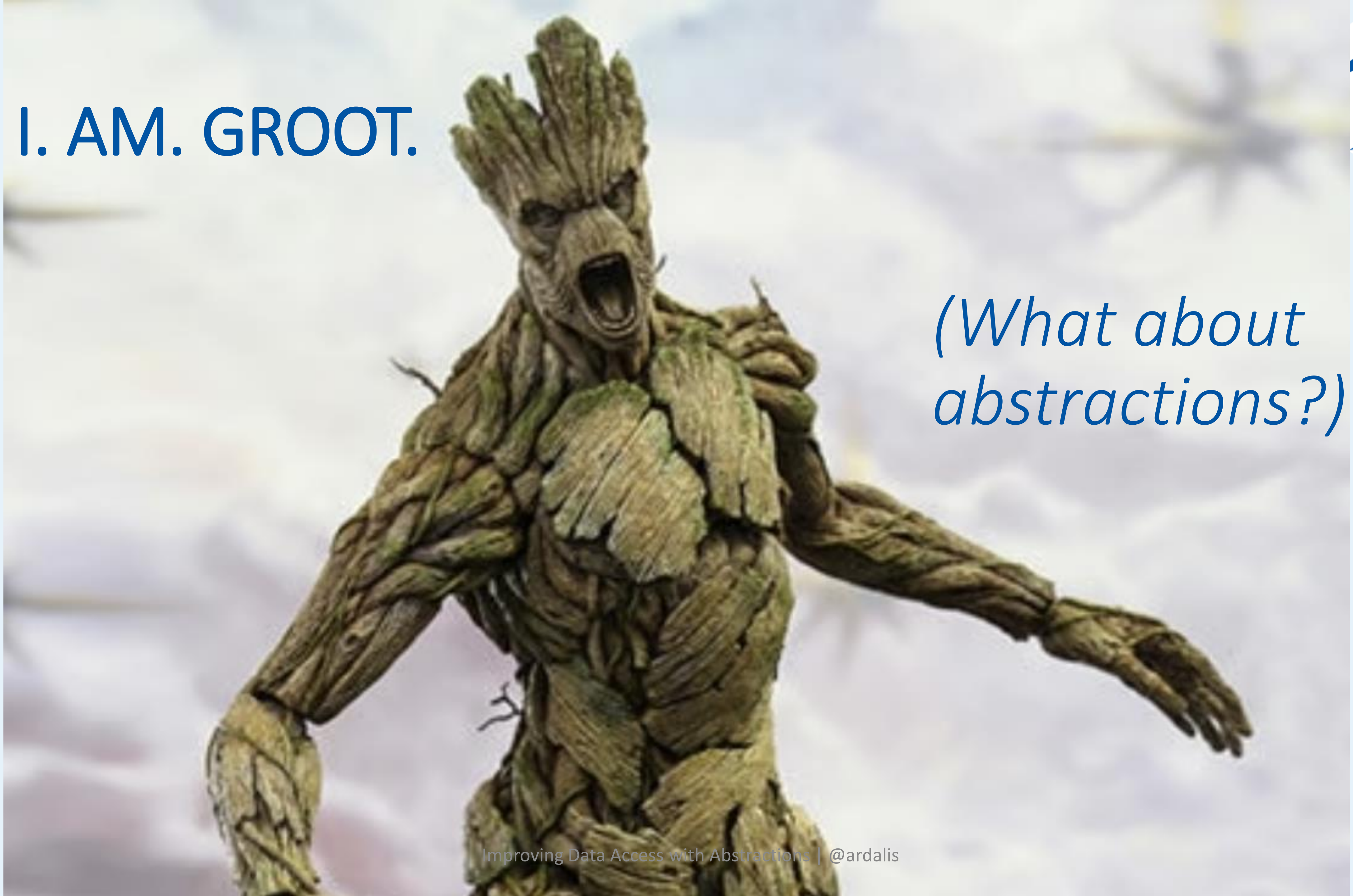
```
var author = _dbContext.Authors
    .Include(author => author.Courses)
    .ThenInclude(ca => ca.Course)
    .Select(a => new AuthorWithCoursesDTO
    {
        Id = a.Id,
        Name = a.Name,
        Courses = a.Courses.Select(c => new CourseDTO
        {
            Id = c.Id,
            Title = c.Course.Title,
            AuthorId = a.Id,
            RoyaltyPercentage = c.RoyaltyPercentage
        }).ToList()
    })
    .SingleOrDefault(a => a.Id == id);
```

Using EF Core - Updates

```
var authorToUpdate = _dbContext.Authors.Find(id);
if (authorToUpdate is null) return NotFound();

authorToUpdate.Name = value;

_dbContext.Update(authorToUpdate);
_dbContext.SaveChanges();
```

Pure EF Core Report Card

Area	Grade
Readability	☆☆☆☆
Security	☆☆☆☆
Performance	☆☆☆
Testability	☆☆
Maintainability	☆☆☆



I. AM. GROOT.

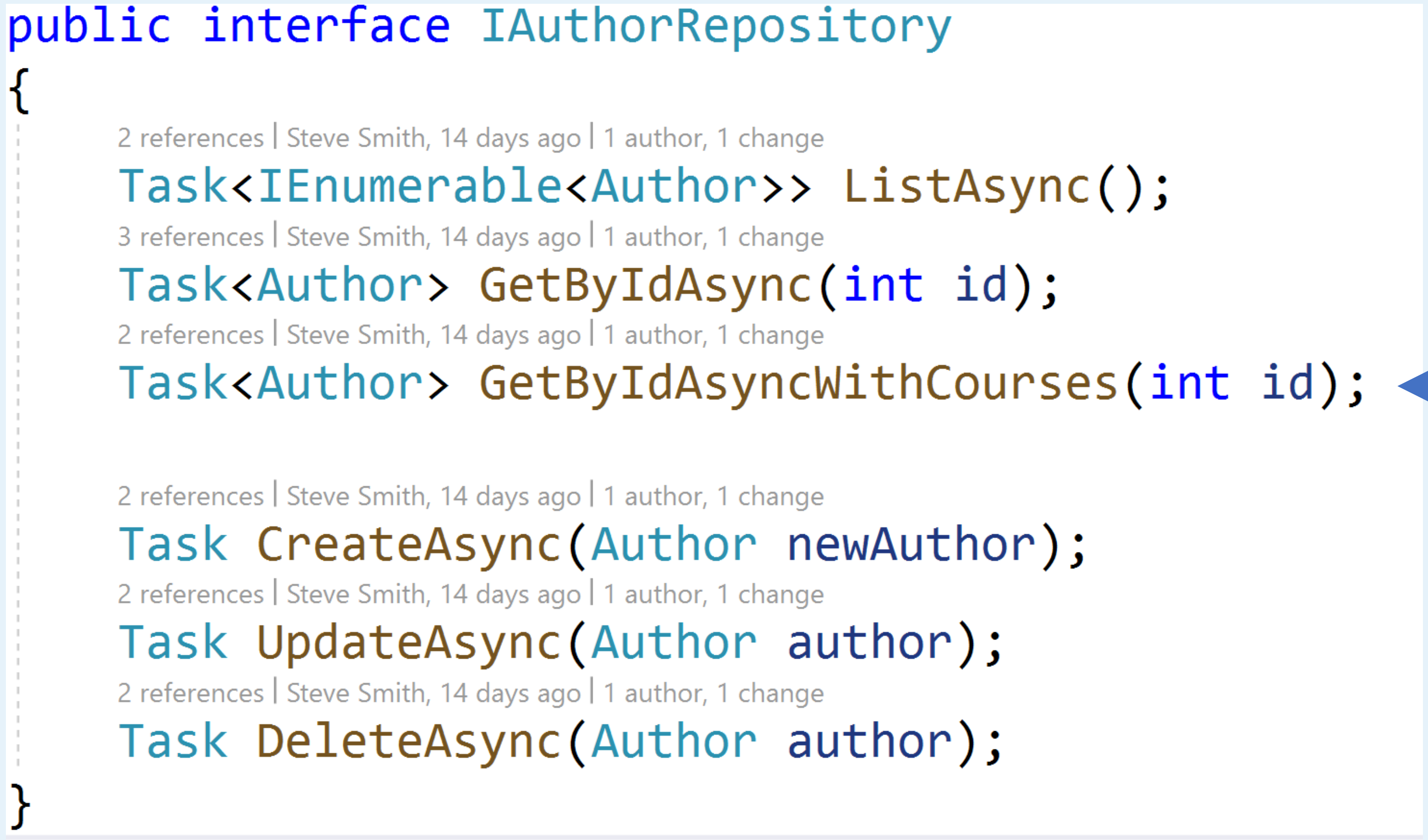
(What about abstractions?)

Let’s add an abstraction!

We’ll call it a Repository, because that’s the established pattern name for such abstractions.

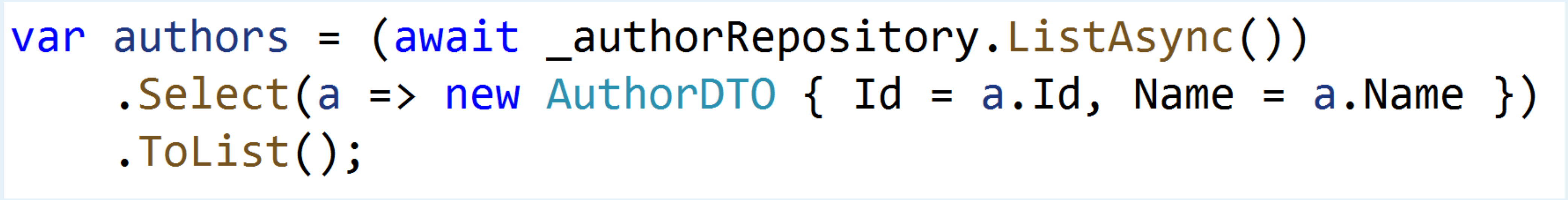
No, an EF DbContext is not an abstraction – it’s an implementation

An abstraction is just an interface

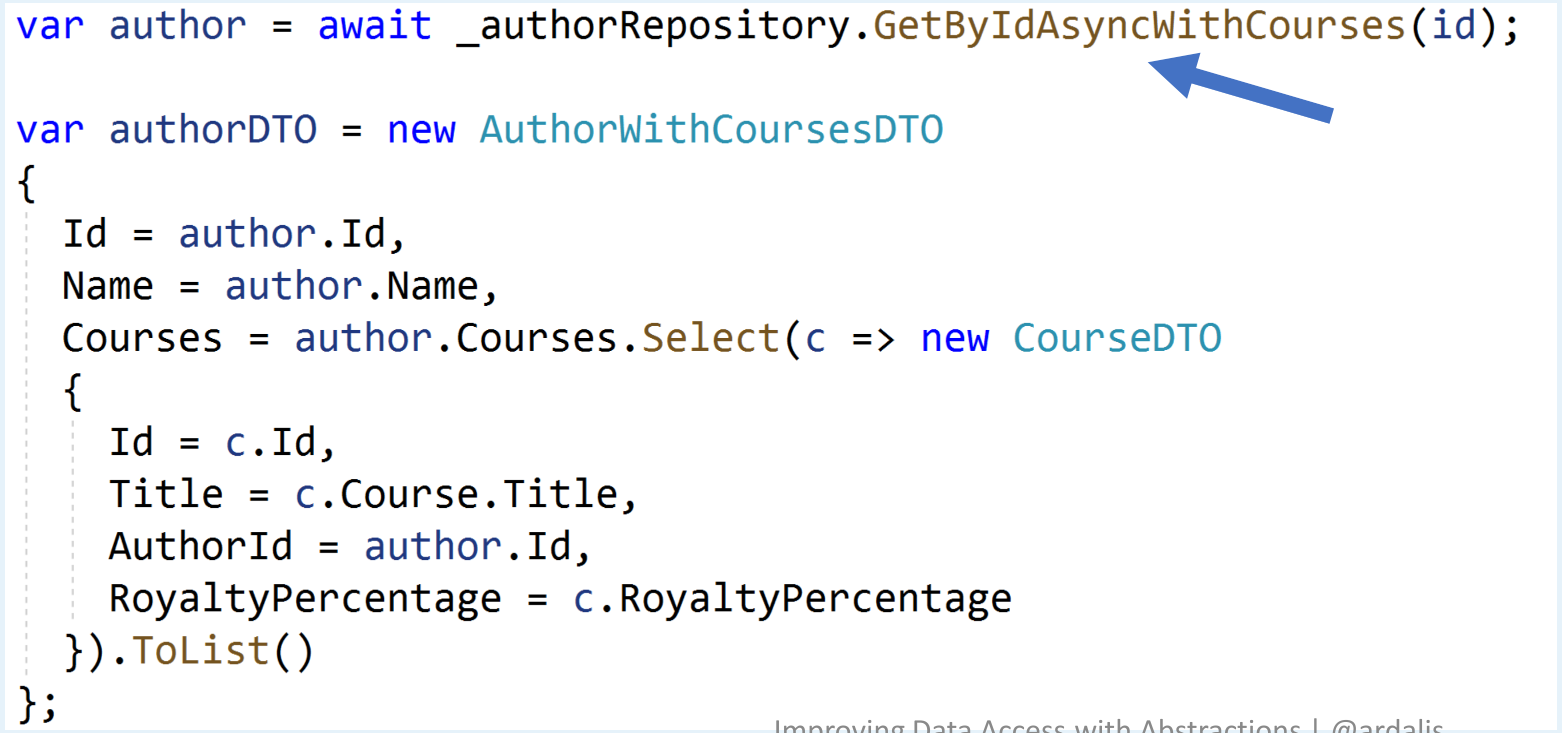


```
public interface IAuthorRepository
{
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task<IEnumerable<Author>> ListAsync();
        3 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task<Author> GetByIdAsync(int id);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task<Author> GetByIdAsyncWithCourses(int id);

        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task CreateAsync(Author newAuthor);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task UpdateAsync(Author author);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task DeleteAsync(Author author);
}
```

Working with the Repository

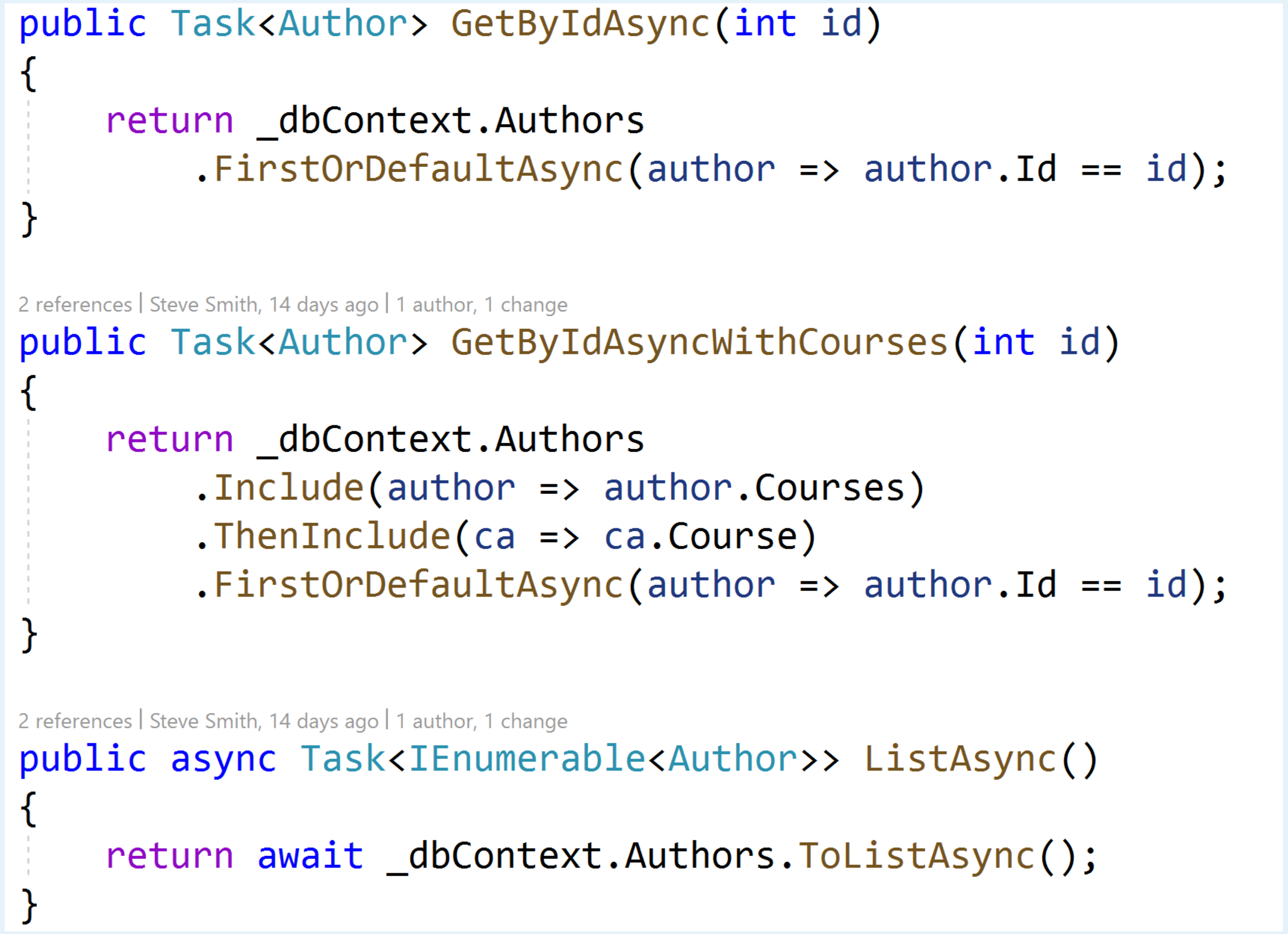


```
var authors = (await _authorRepository.ListAsync())
    .Select(a => new AuthorDTO { Id = a.Id, Name = a.Name })
    .ToList();
```



```
var author = await _authorRepository.GetByIdAsyncWithCourses(id);

var authorDTO = new AuthorWithCoursesDTO
{
  Id = author.Id,
  Name = author.Name,
  Courses = author.Courses.Select(c => new CourseDTO
  {
    Id = c.Id,
    Title = c.Course.Title,
    AuthorId = author.Id,
    RoyaltyPercentage = c.RoyaltyPercentage
  }).ToList()
};
```

Repo Implementation



```
public Task<Author> GetByIdAsync(int id)
{
    return _dbContext.Authors
        .FirstOrDefaultAsync(author => author.Id == id);
}

2 references | Steve Smith, 14 days ago | 1 author, 1 change
public Task<Author> GetByIdAsyncWithCourses(int id)
{
    return _dbContext.Authors
        .Include(author => author.Courses)
        .ThenInclude(ca => ca.Course)
        .FirstOrDefaultAsync(author => author.Id == id);
}

2 references | Steve Smith, 14 days ago | 1 author, 1 change
public async Task<IEnumerable<Author>> ListAsync()
{
    return await _dbContext.Authors.ToListAsync();
}
```

Two Big Benefits!

Modularity

Testability

What still hurts? Scaling the codebase.

- Need an interface per entity
- Need an implementation per entity
- Need a method per interface and implementation per custom query

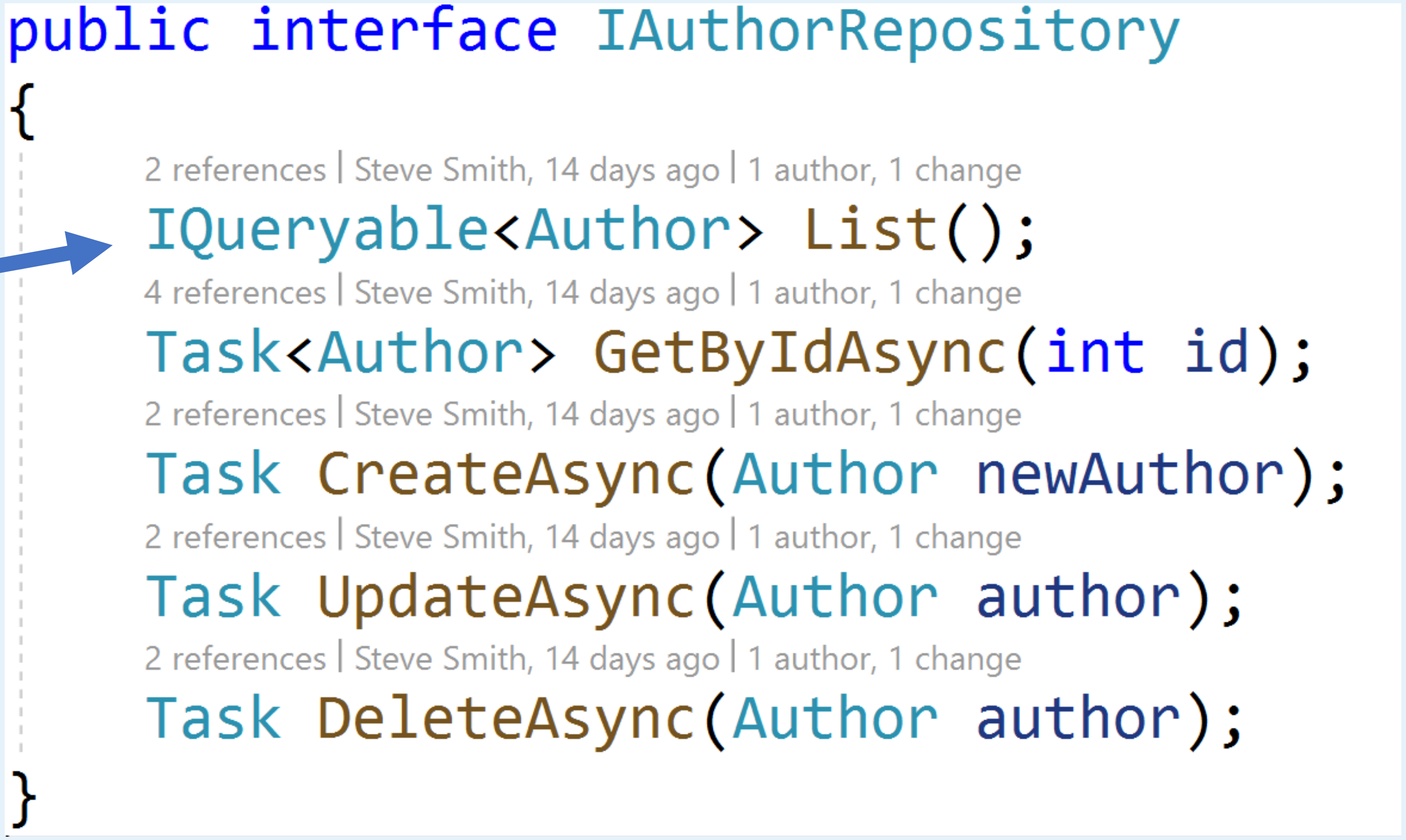
Simple Author Repo Report Card

Area	Grade
Readability	☆☆☆☆
Security	☆☆☆☆
Performance	☆☆☆
Testability	☆☆☆☆☆
Maintainability	☆☆☆

IQueryable!

We can evolve this design into a more perfect version by using IQueryable! This way, we will no longer need to create custom methods for custom queries!

A Simpler Interface



```
public interface IAuthorRepository
{
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
        IQueryable<Author> List();
        4 references | Steve Smith, 14 days ago | 1 author, 1 change
        Task<Author> GetByIdAsync(int id);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
        Task CreateAsync(Author newAuthor);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
        Task UpdateAsync(Author author);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
        Task DeleteAsync(Author author);
}
```

Query logic moves to calling code

```
var authorsWithCourses = _authorRepository.List()
  .Include(a => a.Courses)
  .Select(author => new AuthorWithCoursesDTO
  {
    Id = author.Id,
    Name = author.Name,
    Courses = author.Courses.Select(c => new CourseDTO
    {
      Id = c.Id,
      Title = c.Course.Title,
      AuthorId = author.Id,
      RoyaltyPercentage = c.RoyaltyPercentage
    }).ToList()
  })
  .FirstOrDefault(a => a.Id == id);
```

Query Logic – ANYWHERE… EVERYWHERE

- You can add query logic anywhere
- **Just because you can doesn’t mean you should**
- All data access encapsulation is eliminated

IQueryable is like the dark side…

"Once you start down the dark path [of exposing IQueryable], forever will it dominate your destiny. Consume you it will."

Master Yoda

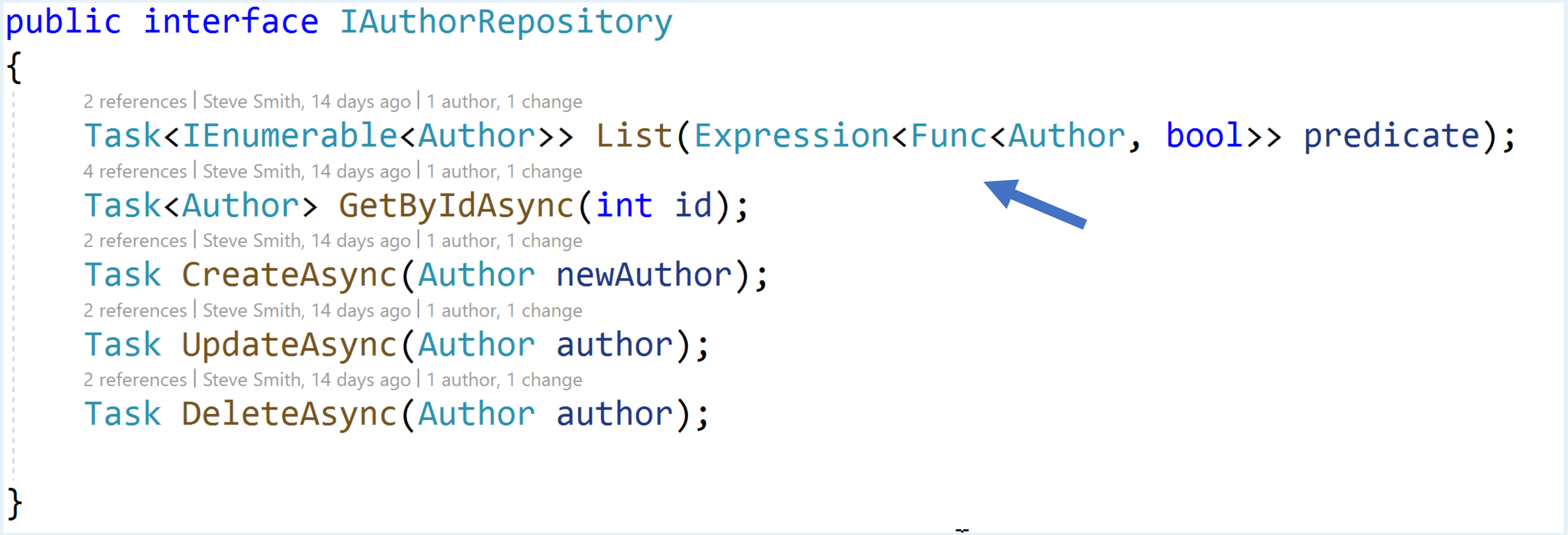
IQueryable Repo Report Card

Area	Grade
Readability	☆☆☆
Security	☆☆☆☆
Performance	☆☆☆
Testability	☆☆☆☆
Maintainability	☆☆

Pass in an Expression

Less pollution of logic throughout our app

Our Revised Interface



```
public interface IAuthorRepository
{
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task<IEnumerable<Author>> List(Expression<Func<Author, bool>> predicate);
        4 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task<Author> GetByIdAsync(int id);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task CreateAsync(Author newAuthor);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task UpdateAsync(Author author);
        2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task DeleteAsync(Author author);

}
```

Calling the Revised Interface

```
// example passing an expression
var steve = (await _authorRepository
                .ListAsync(a => a.Name == "Steve Smith"))
                .FirstOrDefault();
```

Encapsulation (re)achieved!

What still hurts?

- Still too many interfaces and implementations
- Still a lot of LINQ logic required in the calling code

Expression-Based Repo Report Card

Area	Grade
Readability	☆☆☆☆
Security	☆☆☆☆
Performance	☆☆☆
Testability	☆☆☆☆☆
Maintainability	☆☆☆

Introduce Specifications

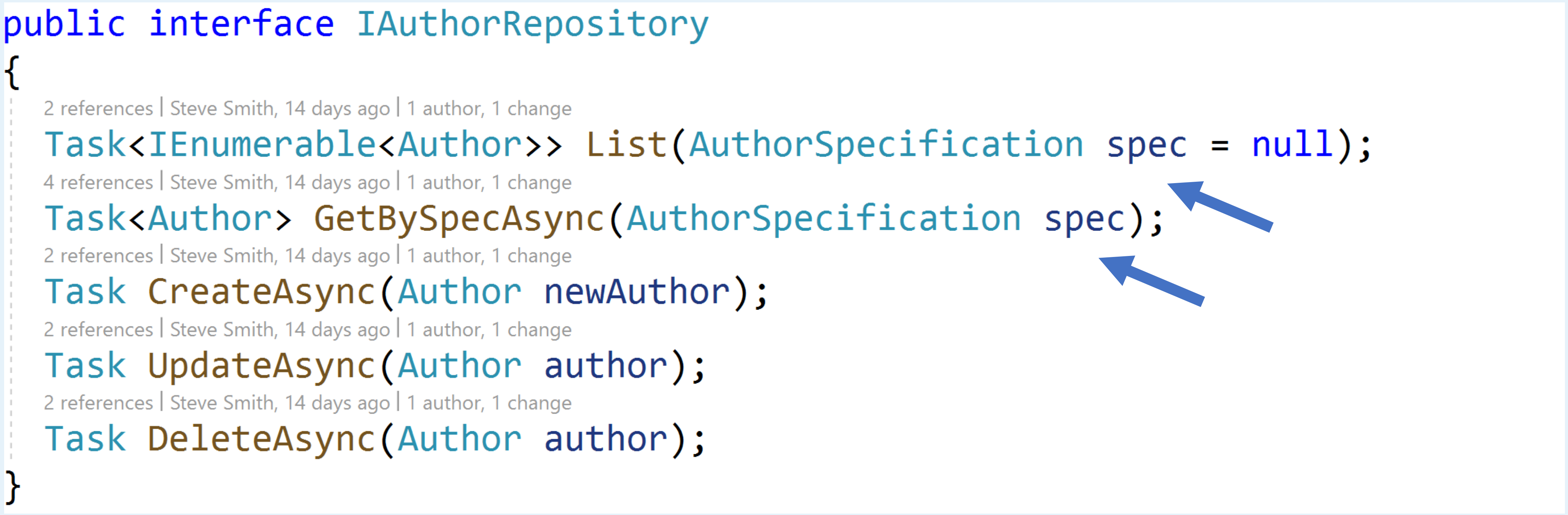
Use an actual class per query type

What’s a Specification?

- A well-named class

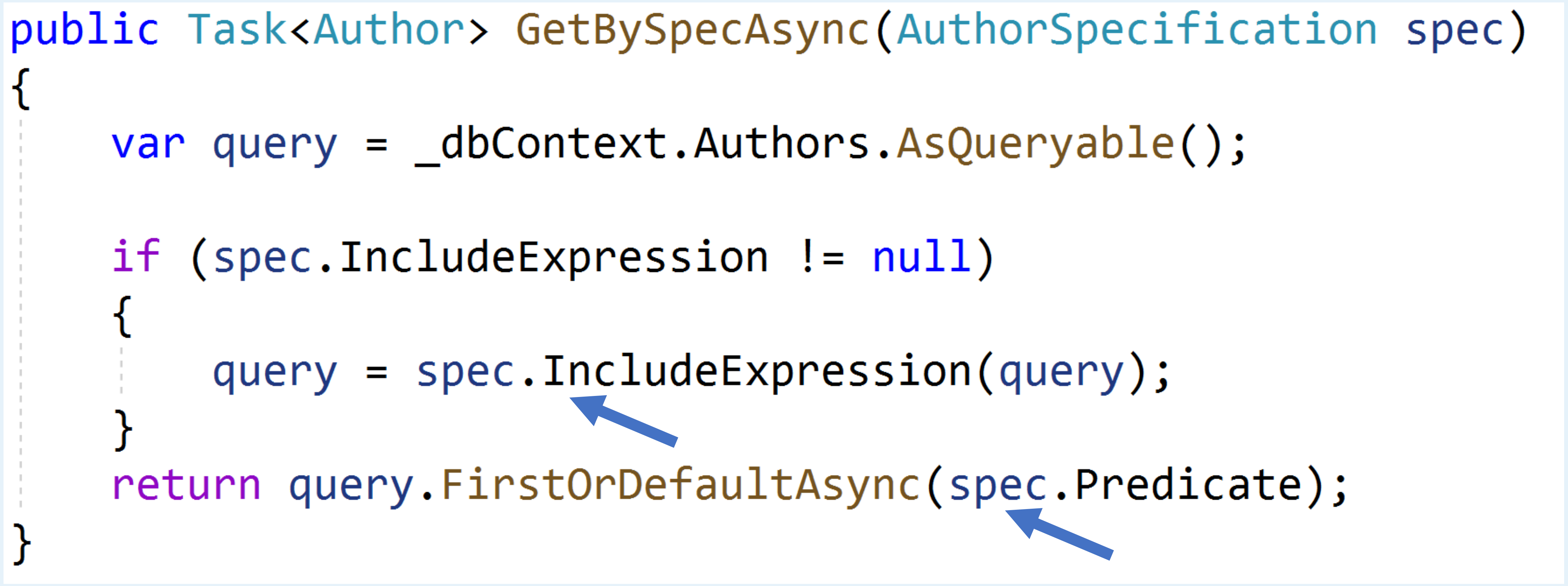
- Properties for:
  - Filtering
  - Includes
  - Projection
  - Paging
  - Caching
  - Etc.

Revised Interface



```
public interface IAuthorRepository
{
    2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task<IEnumerable<Author>> List(AuthorSpecification spec = null);
    4 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task<Author> GetBySpecAsync(AuthorSpecification spec);
    2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task CreateAsync(Author newAuthor);
    2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task UpdateAsync(Author author);
    2 references | Steve Smith, 14 days ago | 1 author, 1 change
    Task DeleteAsync(Author author);
}
```

Revised Implementation



```
public Task<Author> GetBySpecAsync(AuthorSpecification spec)
{
    var query = _dbContext.Authors.AsQueryable();

    if (spec.IncludeExpression != null)
    {
        query = spec.IncludeExpression(query);
    }
    return query.FirstOrDefaultAsync(spec.Predicate);
}
```

Usage

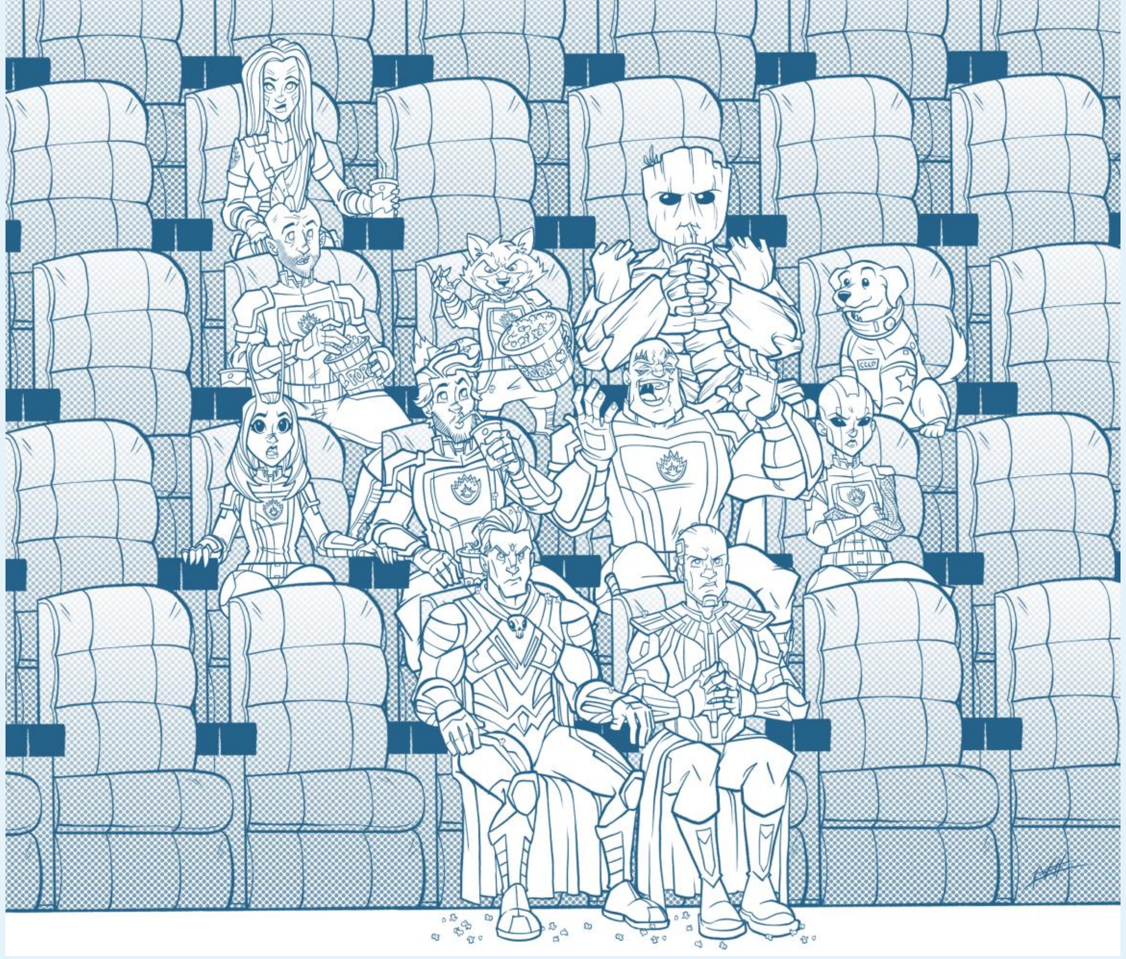
```
var spec = new AuthorByIdSpecification(id);
var author = await _authorRepository.GetBySpecAsync(spec);

var authorDTO = new AuthorWithCoursesDTO
{
  Id = author.Id,
  Name = author.Name,
  Courses = author.Courses.Select(c => new CourseDTO
  {
    Id = c.Id,
    Title = c.Course.Title,
    AuthorId = author.Id,
    RoyaltyPercentage = c.RoyaltyPercentage
  }).ToList()
};
```

What still hurts?

- Still too many interfaces and implementations
- ~~Still a lot of LINQ logic required in the calling code~~

Specification-Based Repo Report Card

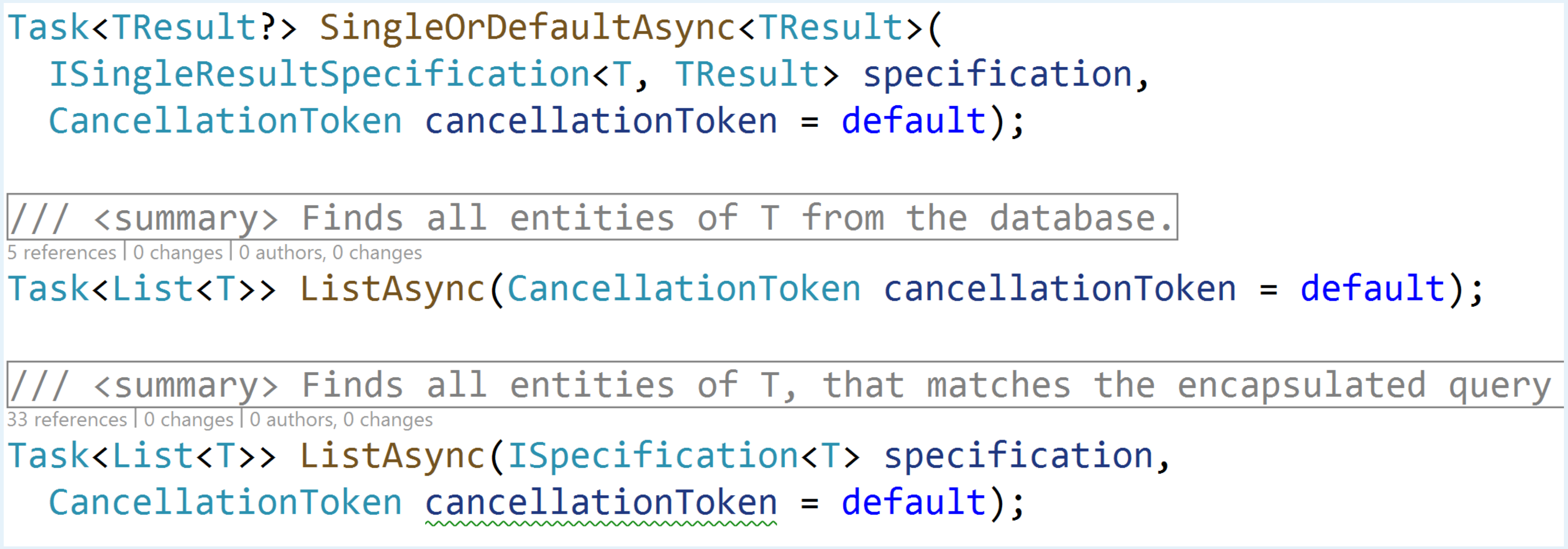
Area	Grade
Readability	☆☆☆☆
Security	☆☆☆☆
Performance	☆☆☆
Testability	☆☆☆☆☆
Maintainability	☆☆☆☆



Generics

Let’s (finally) cut down the number of classes needed for all of this.

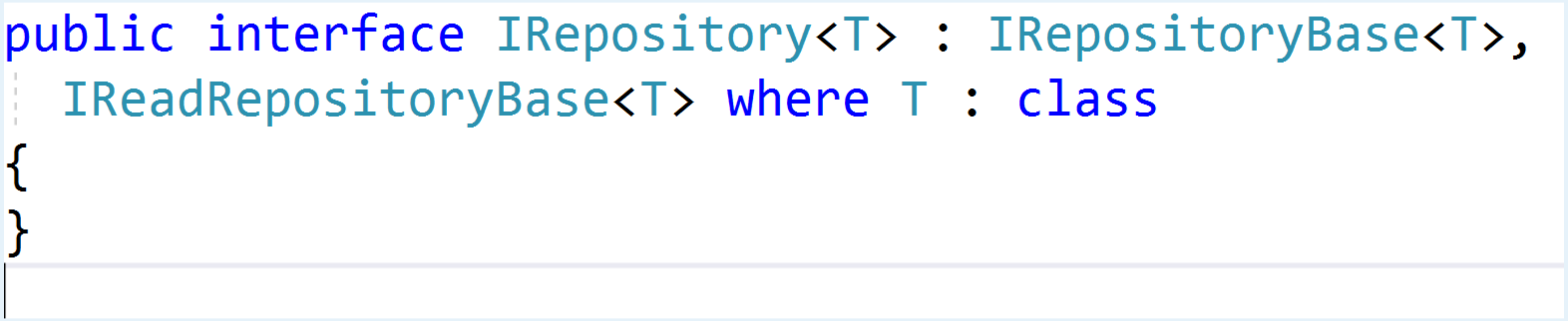
One Interface (to rule them all)



```
Task<TResult?> SingleOrDefaultAsync<TResult>(
  ISingleResultSpecification<T, TResult> specification,
  CancellationToken cancellationToken = default);

/// <summary> Finds all entities of T from the database.
5 references | 0 changes | 0 authors, 0 changes
Task<List<T>> ListAsync(CancellationToken cancellationToken = default);

/// <summary> Finds all entities of T, that matches the encapsulated query
33 references | 0 changes | 0 authors, 0 changes
Task<List<T>> ListAsync(ISpecification<T> specification,
  CancellationToken cancellationToken = default);
```

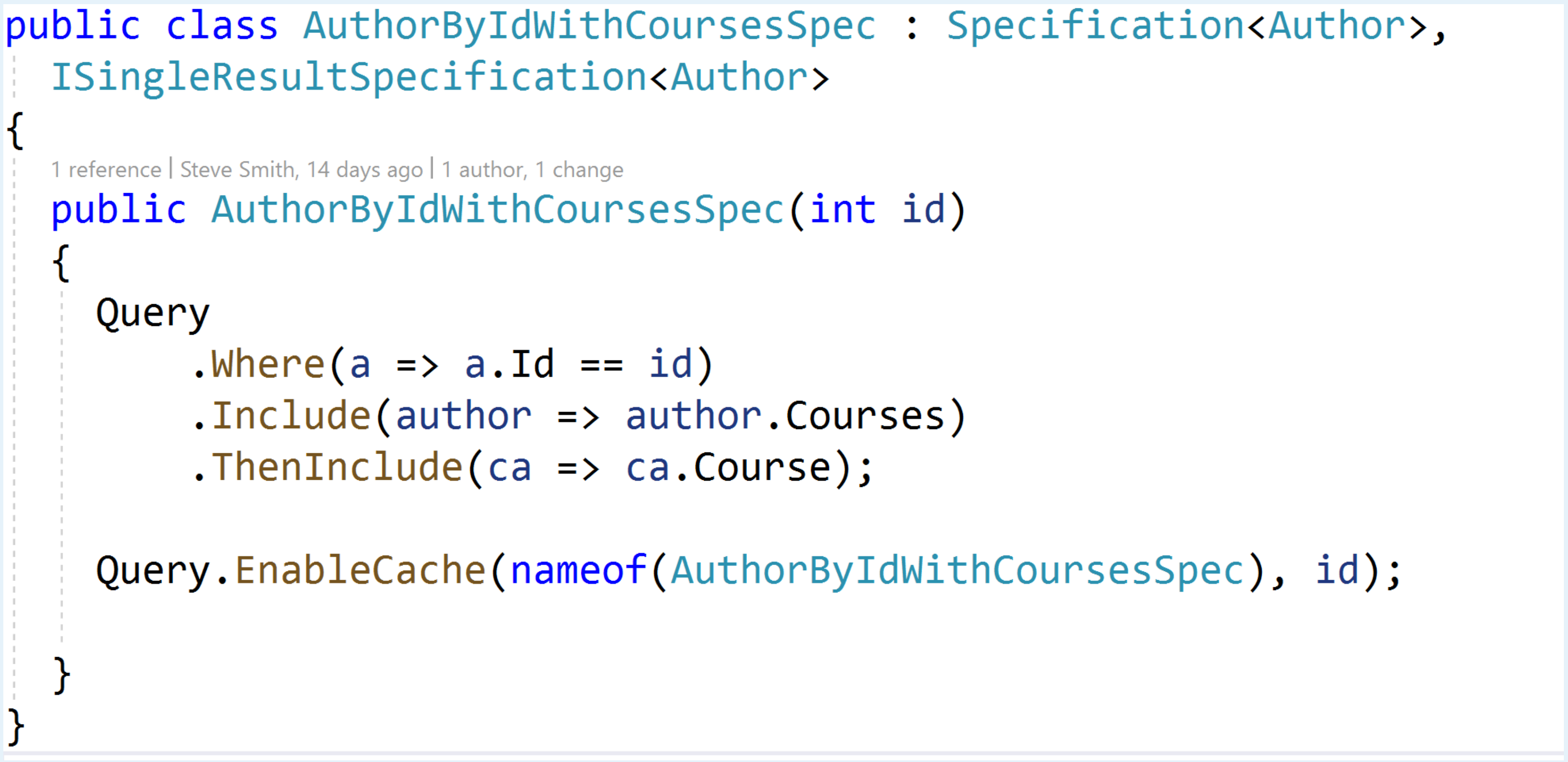
From Ardalis.Specification NuGet Package

One Interface (to rule them all)



```csharp
public interface IRepository<T> : IRepositoryBase<T>,
  IReadRepositoryBase<T> where T : class
{
}
```

Sample Specification

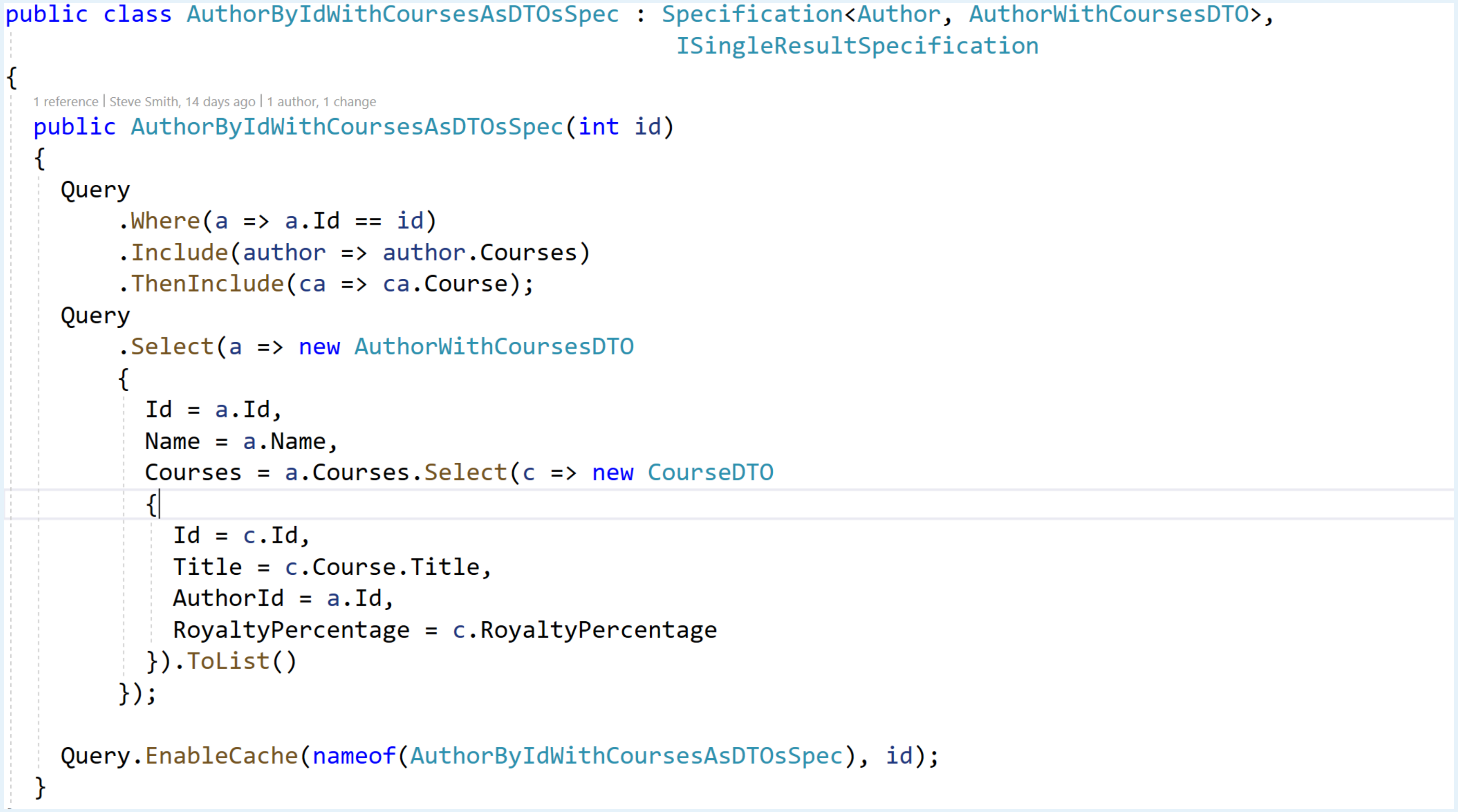


```csharp
public class AuthorByIdWithCoursesSpec : Specification<Author>,
  ISingleResultSpecification<Author>
{
  1 reference | Steve Smith, 14 days ago | 1 author, 1 change
  public AuthorByIdWithCoursesSpec(int id)
  {
    Query
        .Where(a => a.Id == id)
        .Include(author => author.Courses)
        .ThenInclude(ca => ca.Course);

    Query.EnableCache(nameof(AuthorByIdWithCoursesSpec), id);

  }
}
```

Usage

```
var spec = new AuthorByIdWithCoursesSpec(id);
var author = await _authorRepository.SingleOrDefaultAsync(spec);

var authorDTO = new AuthorWithCoursesDTO
{
  Name = author.Name,
  Id = author.Id,
  Courses = new List<CourseDTO>()
};
authorDTO.Courses = author.Courses
    .Select(ca => new CourseDTO
    {
      Id = ca.Course.Id,
      AuthorId = ca.AuthorId,
      RoyaltyPercentage = ca.RoyaltyPercentage,
      Title = ca.Course.Title
    }).ToList();
```

Add Mapping



```
public class AuthorByIdWithCoursesAsDTOsSpec : Specification<Author, AuthorWithCoursesDTO>,
                                               ISingleResultSpecification
{
  1 reference | Steve Smith, 14 days ago | 1 author, 1 change
  public AuthorByIdWithCoursesAsDTOsSpec(int id)
  {
    Query
        .Where(a => a.Id == id)
        .Include(author => author.Courses)
        .ThenInclude(ca => ca.Course);
    Query
        .Select(a => new AuthorWithCoursesDTO
        {
          Id = a.Id,
          Name = a.Name,
          Courses = a.Courses.Select(c => new CourseDTO
          {
            Id = c.Id,
            Title = c.Course.Title,
            AuthorId = a.Id,
            RoyaltyPercentage = c.RoyaltyPercentage
          }).ToList()
        });

    Query.EnableCache(nameof(AuthorByIdWithCoursesAsDTOsSpec), id);
  }
```

Usage with Mapping in Spec

```
var spec = new AuthorByIdWithCoursesAsDTOsSpec(id);
var authorDTO = (await _authorRepository.ListAsync(spec))
                .SingleOrDefault();
```

Generic Repo Report Card

Area	Grade
Readability	☆☆☆☆☆
Security	☆☆☆☆
Performance	☆☆☆
Testability	☆☆☆☆☆
Maintainability	☆☆☆☆☆

Generic CachedRepo Report Card

Area	Grade
Readability	☆☆☆☆☆
Security	☆☆☆☆
Performance	☆☆☆☆☆
Testability	☆☆☆☆☆
Maintainability	☆☆☆☆☆

Overall

Pattern	Read	Security	Perf	Test	Maint
Pure ADO.NET	3	2	5	1	2
Pure ADO.NET w/Stored Procedures	3	4	5	1	3
Dapper	4	4	5	1	3
Dapper w/Stored Procedures	4	4	5	1	3
Pure EF Core	4	4	3	2	3
Simple Repository	4	4	3	5	3
IQueryable Repository	3	4	3	4	2
Expression-Parameter Repository	4	4	3	5	3
Specification + Repository	4	4	3	5	4
Generic Specification + Repository	5	4	3	5	5
Cached Generic Specification + Repository	5	4	5	5	5

Summary

- Different approaches have different trade-offs
- Low level code should be encapsulated
- Complex query logic should be encapsulated
- Calling code should be simple and readable
- Leverage the right patterns
- Combine patterns to unlock huge gains
- Check out NuGet package **Ardalis.Specification**

Thank you!

- If you enjoyed this talk, let me know on Twitter – mention **@ardalis**
- If your team needs mentoring, contact me via **NimblePros.com**
- **Demos:**
  - **https://github.com/ardalis/DotNetDataAccessTour**
- Enjoy the rest of Stir Trek!