Implementing Dynamic Metadata in C#

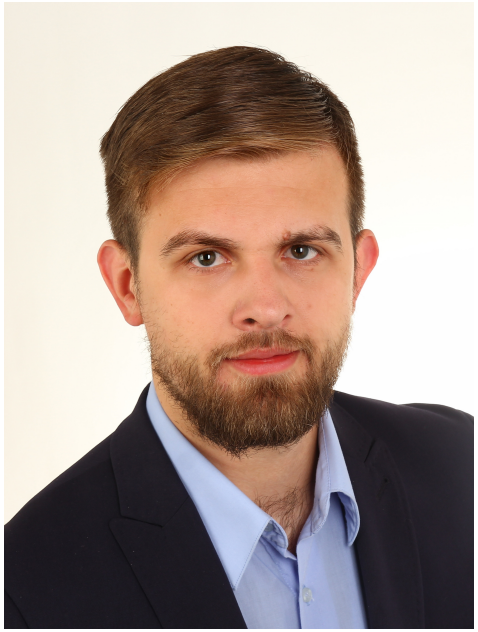
Paweł Łabno

Seville More Helory

SFIonline webdev 2021

Table of Contents

About speaker



- Senior Software Developer @ Seville More Helory
- Graduated @ Jagiellonian University & AGH
- Former Microsoft Student Partner
- pawellabno1992@gmail.com

Table of Contents

What is "metadata"

data that provides information about other data

`https://en.wikipedia.org/wiki/Metadata`

Problem description

Business scenario:
You are developing Movie database with complete information about that movie including:

Problem description

Business scenario:
You are developing Movie database with complete information about that movie including: actors, production team, postproductions, other people involved, recording locations & dates.

Problem description

Business scenario:
You are developing Movie database with complete information about that movie including: actors, production team, postproductions, other people involved, recording locations & dates. That data would be later processed by Your partners Data Analysts to prepare movies people will love most.

Business requirement - records preview

No.	Movie	Year	Locations	BO (bn $)	Director
1	Avengers: Endgame	2019	USA, Ireland	2.798	Russo Brothers
2	Avatar	2009	USA, New Zeland	2.79	James Cameron
3	Titanic	1997	Canada, Mexico	2.195	James Cameron

Technical requirements

Technical requirements for our solution:

Technical requirements

Technical requirements for our solution:

1. Our application should be written in C#

Technical requirements

Technical requirements for our solution:

1. Our application should be written in C#
2. Application should be portable

Technical requirements

Technical requirements for our solution:

1. Our application should be written in C#
2. Application should be portable
3. Some updates could be done after few seconds

Technical requirements

Technical requirements for our solution:

1. Our application should be written in C#
2. Application should be portable
3. Some updates could be done after few seconds
4. Our system should be easy to extend

Technical specification

1. We will use .NET 5 to build application

Technical specification

1. We will use .NET 5 to build application
2. We will use not-relational database approach to make easier extension in future

Technical specification

1. We will use .NET 5 to build application
2. We will use not-relational database approach to make easier extension in future
3. For purposes of this session we will use InMemoryDatabase

Standard approach

```
class Movie {
        List<Recording> Recordings {get;}
        Person Director {get;}
        List<Role> Actors {get;}
        List<Person> Producers {get;}
}
```

Standard approach description

In standard solution we are developing well-defined classes (e.g. Actor or Director) with specified all properties. There is a little risk that variable wouldn't be understood by developer. However adding new metadata type would be implementing completely new class (maybe with little reusage of other components). Adding new property would always require software developer work.

Pros & cons of standard approach

pros	cons
Standard format of data, easily to learn	Very hard to extend, each new feature require implementing new class and properties
"Harder" to break system with invalid data/configuration	Unable to extend by business team / admin
Integration require knowledge about components and relations	
Exploration of data "column by column"	

Pros & cons of standard approach

pros	cons
Standard format of data, easily to learn	Very hard to extend, each new feature require implementing new class and properties
"Harder" to break system with invalid data/configuration	Unable to extend by business team / admin
Integration require knowledge about components and relations	
Exploration of data "column by column"	

Extending requires devteam work

Table of Contents

Dynamic metadata concept

In dynamic metadata approach we treat metadata and its configuration as a metadata with standard. E.g. in our movie database scenario metadata could be "Director" or "Actor" both storing some information about that person and or her/his role in movie production. Adding new metadata kind would require just add new definition to database.

In overall in that case development team will provide solution with definitions, values and property types.

Metadata definition

What that object should contains?

- Identifier or key
- Names to easily select by user
- Collection of properties
  - Identifier or key
  - Name
  - Type
  - Default value
- Validators
  - Cross-property validation configuration
  - Context validation configuration

Metadata definition - code example

```
public class DynamicMetadata
{
    public IList<DynamicMetadataProperty> Properties { get

    public string DefinitionId { get; set; }
    public string Id { get; set; }
    public string Name { get; set; }
}
```

Metadata definition - key points

1. Type specifications
2. Validation pipelines, middlewares, complex scenarios
3. Cross-property & cross-item validations
4. Default values & configuration

Metadata definition - possible bad solutions

1. Not specified type or trying to generalize some data
2. Passing complete configuration to target object (e.g. Movie)
3. Creating on database layer "tree-kind" structures
4. Validators on property level (on database, it could be fine for front-end validation)

Metadata usage

How should we extend default Movie class to better match our solution?

Metadata usage

How should we extend default Movie class to better match our solution?

In our project there should be new list of properties storing information about metadata. Each of this object should contains value and metadata definition and ...

Metadata usage

How should we extend default Movie class to better match our solution?

In our project there should be new list of properties storing information about metadata. Each of this object should contains value and metadata definition and ... that's all :)

Sample request

```
"metadataDefinitionId": "actor",
"dynamicProperties": [
  {
    "id": "person",
    "type": "person",
    "value": {
        "id": "someone"
    }
  },
  {
    "id": "character",
    "type": "string",
    "value": "Anonymous"
  },
  {
      "id": "main-character",
      "type": "boolean",
      "value": True
  }
]
```

Sample request - 2nd Part

On previous slide You could saw that "Value" had three different data types assigned to field. With little work we could allow to this flexibility, however there is a hidden cost - we have to validate data passed in requests.

Layers separation

Standard ASP.NET Core project is divided into three layers

Layers separation

Standard ASP.NET Core project is divided into three layers

1. Presentation

Layers separation

Standard ASP.NET Core project is divided into three layers

1. Presentation
2. Business logic

Layers separation

Standard ASP.NET Core project is divided into three layers

1. Presentation
2. Business logic
3. Data access

Layers separation

Standard ASP.NET Core project is divided into three layers

1. Presentation
2. Business logic
3. Data access

Proper storage of data and communication with outer world is crucial for our problem.

Parsing input data

```
case "person":
{
    if (value == null)
        return new PersonProperty { Id = id, WrappedValue = null };
    var item = (JsonElement) value;
    var inputData = JsonSerializer.Deserialize<PersonDto>( json: item.GetRawText(),
    {
        PropertyNameCaseInsensitive = true
    });
    return new PersonProperty
    {
        Id = id,
        WrappedValue = new Person
        {
            Id = inputData.Id,
            FirstName = inputData.FirstName,
            LastName = inputData.LastName
        }
    };
}
```

Table of Contents

Data consistency

For data analysis it is required to data be consistent in context of whole ecosystem. Since our data would be dynamic there are few additional issues that system should handle and in few following slides we will discuss data aspects.

System should be sure that if data is described as *Person* that it contains person information, not numeric one.

Constant up-do-date aspect

To help processing data in system, developers should prepare solution that would keeps data up-to-data and having all actual information. To achieve that point they should prepare proper event handler, and when definition would change then all *"Movies described with that description"* should be updated too.

This topic we will continue in next section.

Validation pipeline

During creation of new metadata entry or updating existing one we should run validation process to item. Since all configuration is stored in MetadataDefinition object we will generate validator based on that part of code. Validation process by default is invoked in context including *Movie* and *MetadataProperty*.

Validation pipeline - algorithm flow

1. Create validator for MetadataDefinition
    1. Find all validator configurations for definition
    2. For each validator
        1. Get new instance of specified validator
        2. Set validation parameters
        3. Add validator to composite validator
    3. Return composite validator
2. Validate item using validator
    1. For each validator
        1. Run validation on (metadata, movie) context
        2. Add errors to result collection
    2. Return list of all errors
3. If validator returns any error - throw validation exception

IValidation interface

```
using System.Collections.Generic;
using MovieDatabase.Model;

namespace MovieDatabase.Services.Validation
{
    public interface IValidator
    {
        string Key { get; }
        List<Error> Validate(DynamicMetadata dynamicMetadata, I
        void ConfigureValidator(Dictionary<string, string> vali
    }
}
```

Property validators

This is the simplies kind of validator - we process just property data - e.g. we could check if year is in range or if connected user exists in proper database collection.

Cross property validators

This is a little more complex validator - in that case it is crucial that properties are properly named (we should use unique keys). In that case system will process a bit bigger chunk of data.

Cross-items validators

For example we want to have just one director for movie or ensure that actor really exist.

Missing data

This is first serious problem system should handle. First part of that issue could be handled by adding/removing/updating all properties updated in definition but there is another case.

Missing data

This is first serious problem system should handle. First part of that issue could be handled by adding/removing/updating all properties updated in definition but there is another case.
What if property doesn't exist in request?

Missing data

This is first serious problem system should handle. First part of that issue could be handled by adding/removing/updating all properties updated in definition but there is another case.
What if property doesn't exist in request? E.g. for actor we specify person but we don't specify character?

Missing data

This is first serious problem system should handle. First part of that issue could be handled by adding/removing/updating all properties updated in definition but there is another case.
What if property doesn't exist in request? E.g. for actor we specify person but we don't specify character? System should create missing properties and then validates content or returns missing property exception.

Duplicated property

Another serious problem -

Duplicated property

Another serious problem - What if we are victims of somebody bad trying to break our system with invalid data?

Duplicated property

Another serious problem - What if we are victims of somebody bad trying to break our system with invalid data?
There are three available solutions - You could select best matching Your requirements

Duplicated property

Another serious problem - What if we are victims of somebody bad trying to break our system with invalid data?
There are three available solutions - You could select best matching Your requirements

1. Handle just first occurance of property

Duplicated property

Another serious problem - What if we are victims of somebody bad trying to break our system with invalid data?
There are three available solutions - You could select best matching Your requirements

1. Handle just first occurance of property
2. Update property with each occurance

Duplicated property

Another serious problem - What if we are victims of somebody bad trying to break our system with invalid data?
There are three available solutions - You could select best matching Your requirements

1. Handle just first occurance of property
2. Update property with each occurance
3. Raise an error

Additional property

Next serious problem -

Additional property

Next serious problem - What if breaker from previous slide would add additional properties?

Additional property

Next serious problem - What if breaker from previous slide would add additional properties?
There are two available solutions - You could select best matching Your requirements

Additional property

Next serious problem - What if breaker from previous slide would add additional properties?
There are two available solutions - You could select best matching Your requirements

1. Ignore that property

Additional property

Next serious problem - What if breaker from previous slide would add additional properties?
There are two available solutions - You could select best matching Your requirements

1. Ignore that property
2. Raise an error

Invalid data type

Final problem is with data stored in property and there could be two issues with that

Invalid data type

Final problem is with data stored in property and there could be two issues with that

1. Specified property type differs from type in definition -

Invalid data type

Final problem is with data stored in property and there could be two issues with that

1. Specified property type differs from type in definition - we have person when it should be string

Invalid data type

Final problem is with data stored in property and there could be two issues with that

1. Specified property type differs from type in definition - we have person when it should be string
2. Data has invalid format and couldn't be parsed

Invalid data type

Final problem is with data stored in property and there could be two issues with that

1. Specified property type differs from type in definition - we have person when it should be string
2. Data has invalid format and couldn't be parsed

Invalid data type

Final problem is with data stored in property and there could be two issues with that

1. Specified property type differs from type in definition - we have person when it should be string
2. Data has invalid format and couldn't be parsed

In that case it would be better to return error

Table of Contents

Definition versioning - Introduction

Sometimes would be required to change definition in many ways, but all of them will affect how our system works. We should prepare our system for:

Definition versioning - Introduction

Sometimes would be required to change definition in many ways, but all of them will affect how our system works. We should prepare our system for:

- Adding or removing property

Definition versioning - Introduction

Sometimes would be required to change definition in many ways, but all of them will affect how our system works. We should prepare our system for:

- Adding or removing property
- Complete deletion of metadata definition

Definition versioning - Introduction

Sometimes would be required to change definition in many ways, but all of them will affect how our system works. We should prepare our system for:

- Adding or removing property
- Complete deletion of metadata definition
- Changing type or default value

Definition versioning - Introduction

Sometimes would be required to change definition in many ways, but all of them will affect how our system works. We should prepare our system for:

- Adding or removing property
- Complete deletion of metadata definition
- Changing type or default value
- Changing validation configuration

Definition versioning - Introduction

Sometimes would be required to change definition in many ways, but all of them will affect how our system works. We should prepare our system for:

- Adding or removing property
- Complete deletion of metadata definition
- Changing type or default value
- Changing validation configuration

And we will discuss all of these cases in following slides

Handling changes

One of the biggest advantage of *NoSQL* databases (e.g. MongoDB or CosmosDB) is that we don't keep explicite relations between objects. For the future discussion we will name Metadata definition as Source and Movie metadata as Target.

We should implement asynchronous update with Service Bus (Azure Service Bus or RabbitMQ) with event dispatched each time we change of definition (or at least with published change). On Target side we should implement handler to keep data up to date.

Handling changes

One of the biggest advantage of *NoSQL* databases (e.g. MongoDB or CosmosDB) is that we don't keep explicite relations between objects. For the future discussion we will name Metadata definition as Source and Movie metadata as Target.

We should implement asynchronous update with Service Bus (Azure Service Bus or RabbitMQ) with event dispatched each time we change of definition (or at least with published change). On Target side we should implement handler to keep data up to date.
NOTE: It could take several seconds to update data in Target entities.

Adding or removing property

After adding or removing property in source object natural operation is updating target object in similar way.
It would be nice to set default value for newly created properties.
Removing property on source object should be implemented very carefully since we could break validation. We should handle also all validators connected to this property or block removal in case of any validators connected

Deletion of metadata definition

After confirmation of delete on purpose and dispatching event we have two acceptable options to implement:

- Delete items created with definition from DB
  - We will have clean database without deleted items
- Mark items as deleted
  - In that case data could be visible to users (but marked as deleted, not updatable) or invisible, but in that case we should handle deleted items each time (and users could be confused by lack of some items)

Remember that it would be better to mark items as deleted rather than deleting them permanently.

Type or default value change

Data definition could be hard to handle in many cases since we will have validators attached to these properties. Simlary to content of previous slide we could block these kind of changes or invalidate existing validator.
In case of type change system should update data with provided converter - implemented by software development team - or mark it out of date and require action by some administrator.
In case of default value change we could simply iterate over collection of items and for properties marked with default value set new value. This would require add new property **IsDefaultValue** to dynamic property.

Configuration change

The best solution in case of change of validator configuration would be process all data created basing on definition. Each instance with failed validation status we should mark as out of date.

Metadata history - definition

It could be helpful to track changes in definition object. However this would require some changes into our code:

Metadata history - definition

It could be helpful to track changes in definition object. However this would require some changes into our code:

- We should create new object "DefinitionVersion" with unique Id, but referencing by key/id to MetadataDefinition

Metadata history - definition

It could be helpful to track changes in definition object. However this would require some changes into our code:

- We should create new object "DefinitionVersion" with unique Id, but referencing by key/id to MetadataDefinition
- In MetadataDefinition we should store published version of item, with reference to it

Metadata history - definition

It could be helpful to track changes in definition object. However this would require some changes into our code:

- We should create new object "DefinitionVersion" with unique Id, but referencing by key/id to MetadataDefinition
- In MetadataDefinition we should store published version of item, with reference to it
- There should be new endpoint to "Create new Definition Version"

Metadata history - definition

It could be helpful to track changes in definition object. However this would require some changes into our code:

- We should create new object "DefinitionVersion" with unique Id, but referencing by key/id to MetadataDefinition
- In MetadataDefinition we should store published version of item, with reference to it
- There should be new endpoint to "Create new Definition Version"
- All operations regarding changes in Definitions should have additional parameter "VersionId"

Metadata history - definition

It could be helpful to track changes in definition object. However this would require some changes into our code:

- We should create new object "DefinitionVersion" with unique Id, but referencing by key/id to MetadataDefinition
- In MetadataDefinition we should store published version of item, with reference to it
- There should be new endpoint to "Create new Definition Version"
- All operations regarding changes in Definitions should have additional parameter "VersionId"
- All "DefinitionChanges" Events should be handled only after publication of new version

Metadata history - definition

It could be helpful to track changes in definition object. However this would require some changes into our code:

- We should create new object "DefinitionVersion" with unique Id, but referencing by key/id to MetadataDefinition
- In MetadataDefinition we should store published version of item, with reference to it
- There should be new endpoint to "Create new Definition Version"
- All operations regarding changes in Definitions should have additional parameter "VersionId"
- All "DefinitionChanges" Events should be handled only after publication of new version

Metadata history - movie

I imagine that sometimes it would be nice to keep history of changes of Movie object. Since we could have two sources of changes for "Movie" object it wouldn't be so easy to implement similar solution as in scenario. We could simply create new object version for each time we change object - eg. add new metadata or we update metadata definition. But after little amount of time we could learn that we have one huge object (if we store information in one object) or we have a lot of big objects (in case we store each version separately).

Metadata history - movie

I imagine that sometimes it would be nice to keep history of changes of Movie object. Since we could have two sources of changes for "Movie" object it wouldn't be so easy to implement similar solution as in scenario. We could simply create new object version for each time we change object - eg. add new metadata or we update metadata definition. But after little amount of time we could learn that we have one huge object (if we store information in one object) or we have a lot of big objects (in case we store each version separately).

Neither of these options is good for me

Change as commit

There is design that would allows us to store many of changes with only one stored final object (to greatly simplify calculations).

Change as commit

There is design that would allows us to store many of changes with only one stored final object (to greatly simplify calculations).
But we should start with ...

Change as commit

There is design that would allows us to store many of changes with only one stored final object (to greatly simplify calculations).
But we should start with ... GIT

Change as commit

There is design that would allows us to store many of changes with only one stored final object (to greatly simplify calculations).
But we should start with ... GIT
We could treat each change (regardless type of these changes) as an operation - e.g. we will have operation to add new dynamic metadata, add new property or change value - each of these could be simplyfied to atomic (in context of system) operation.
Each *Save or publish* click would invoke processing of several atomic operations - like commit includes list of changes made inside files.
In this solution (recommended but hardest) We don't store full objects but we store changes.

EventSourcing

There is design pattern called EventSourcing in which we store operations (Events). In specified case basing on stored event system rebuilds final object.

EventSourcing

There is design pattern called EventSourcing in which we store operations (Events). In specified case basing on stored event system rebuilds final object.
This rebuild operation could be invoked when new event appears (better in our case) or when user requests data.

EventSourcing

There is design pattern called EventSourcing in which we store operations (Events). In specified case basing on stored event system rebuilds final object.
This rebuild operation could be invoked when new event appears (better in our case) or when user requests data.
However there could be a lot of events stored in database and in that case rebuild would take ages

EventSourcing

There is design pattern called EventSourcing in which we store operations (Events). In specified case basing on stored event system rebuilds final object.
This rebuild operation could be invoked when new event appears (better in our case) or when user requests data.
However there could be a lot of events stored in database and in that case rebuild would take ages - To handle this problem we should prepare snapshot, materialized state of object in one point at history. For further rebuilds we could start from event no. 750, not 1.

EventSourcing - Example

- Movie
  - Test Actor (Actor)

AddActorOperation

—

- Movie
  - Test Actor (Actor)
  - New Actor (Actor)

EventSourcing + Commits

To sum up this part we could prepare solution that includes Commits that will contains atomic changes to movie metadatas. There could be more than one atomic change in one metadata. Each commit will generate new version of object and during execution of commit we will execute more all operation events.

Table of Contents

Starting data

It would be nice to have some default data - sample metadata ready to work with at day 0. Then we could prepare seeding operation. This process should generate same structures as they could be defined by API actions, and services allowing that operations should be readable for software developers.

Metadata definition

For purpose of item generation we will use *Fluent interface approach* where as result of method invocation You return invoked object. This design allow to create pipelines generating data properties one by one, during complete process using the same object as a root.

Property definition

Since properties differs from each other in metadata definition builder we should include mutliple builders, each for each type.

Table of Contents

Caching ideas

Since some operations would appears more frequently so we could optimize time required to get data. It is necessary to properly let cache service know that item is out of date.

Search & Data analysis

Our data is ready to explore by specialists. We could implement search operations suggesting best metadata definitions for movie of our content. Also we could implement AI mechanisms for suggesting content of newly created metadata entry.

Table of Contents

Thank you

You could reach me:

- pawellabno1992@gmail.com
- LinkedIn: Paweł Łabno