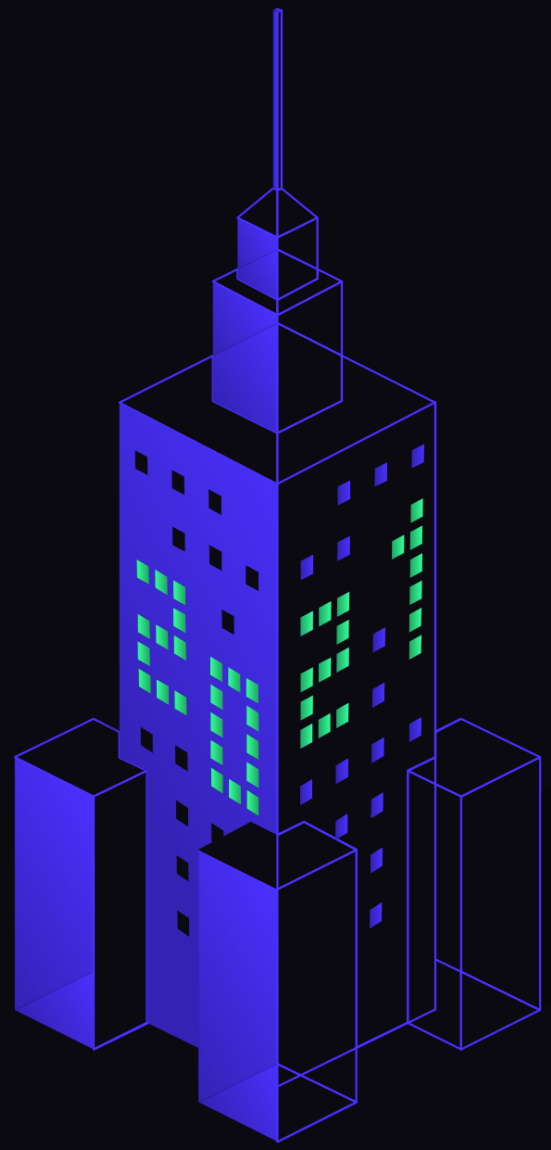
WARSAW
IT DAYS



Dynamic metadata system

Paweł Łabno
Senior software enginner, Seville More Helory

Speaker

- Senior software developer @ Seville More Helory
- Graduated @ AGH & UJ
- Runner & boardgame player

Agenda

1. Introduction
2. Dynamic project concept
3. Data validation
4. Versioning
5. Sample data population

Note

Presentation available on Github:

https://github.com/paqaos/WDI2021-DynamicMetadata

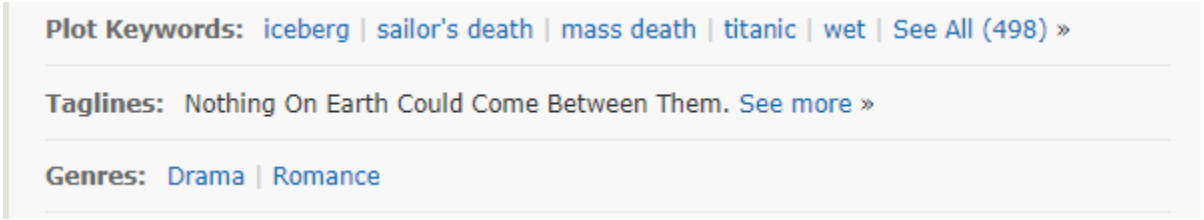
1/ Metadata concept

1/ What is „metadata”?

data that provides information about other data

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

1/ Metadata example



1/ Business idea

- Movie database
- Store standard data as actors, genres or directors
- Include other people involved in production of movie
- Store information about CGI & recording locations

1/ Business idea – possible usage

- Hiring best crew for similar movie
- Basing on Box office trends increase income from tickets

1/ Business idea – records preview

	Movie	Year	Locations	Box office	Director
1	Avengers: Endgame	2019	USA, Ireland	2.798 bln $	Russo brothers
2	Avatar	2009	USA, New Zeland	2.79 bln $	James Cameron
3	Titanic	1997	Canada, Mexico	2.195 bln $	James Cameron

1/ Business idea – technical aspects

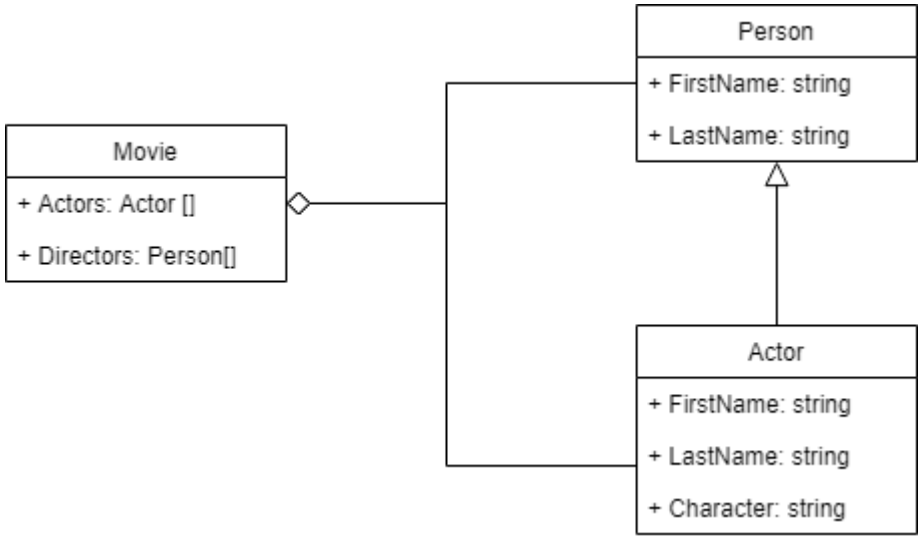
Requirements

- Our application should be written in C#
- Application should be portable
- Some updates could be done after few seconds
- Our system should be easy to extend

Framework

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

1/ Standard definition class diagram



1/ Metadata standard approach

Pros

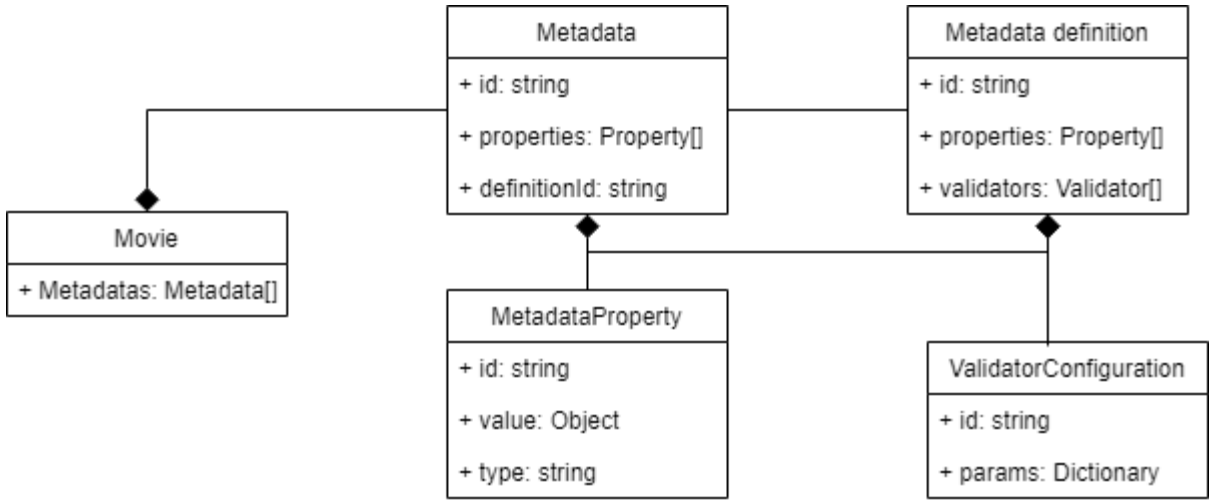
- Standard format of data, easy to learn
- Harder to break system with invalid data/configuration
- Easier integration with other components
- Intuitive exploration of data by column

Cons

- Harder to extend
- Extension couldn’t be „clicked”

2/ Dynamic metadata

2/ Dynamic metadata concept



2/ Key aspects

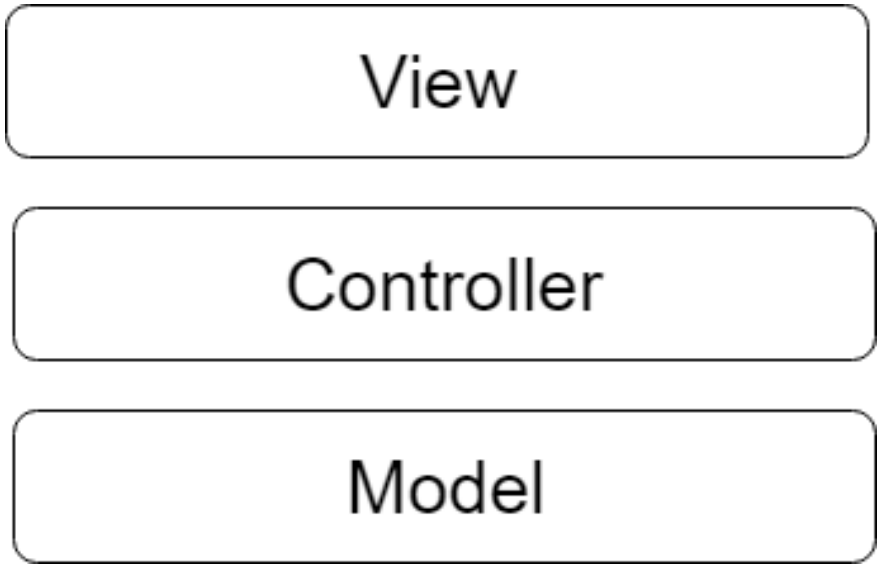
- Type specification
- Pipelines – validation, management, usage
- Validators – kinds, configurations
- Default values

2/ 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
    }
  ]
}
```

request.json

2/ Application layer’s separation



2/ Layers separation - issues

- Deserialization from json
- Serialization to json

2/ Dynamic variable type

```
public class DynamicPropertyDto
{
    public  string Id   { get; set; }
    public  string Type { get; set; }
    public  object Value { get; set; }
    public  PropertyState State { get; set; }
    public  bool HasDefaultValue { get; set; }
}
```

DynamicPropertyDto.cs

2/ Common issues

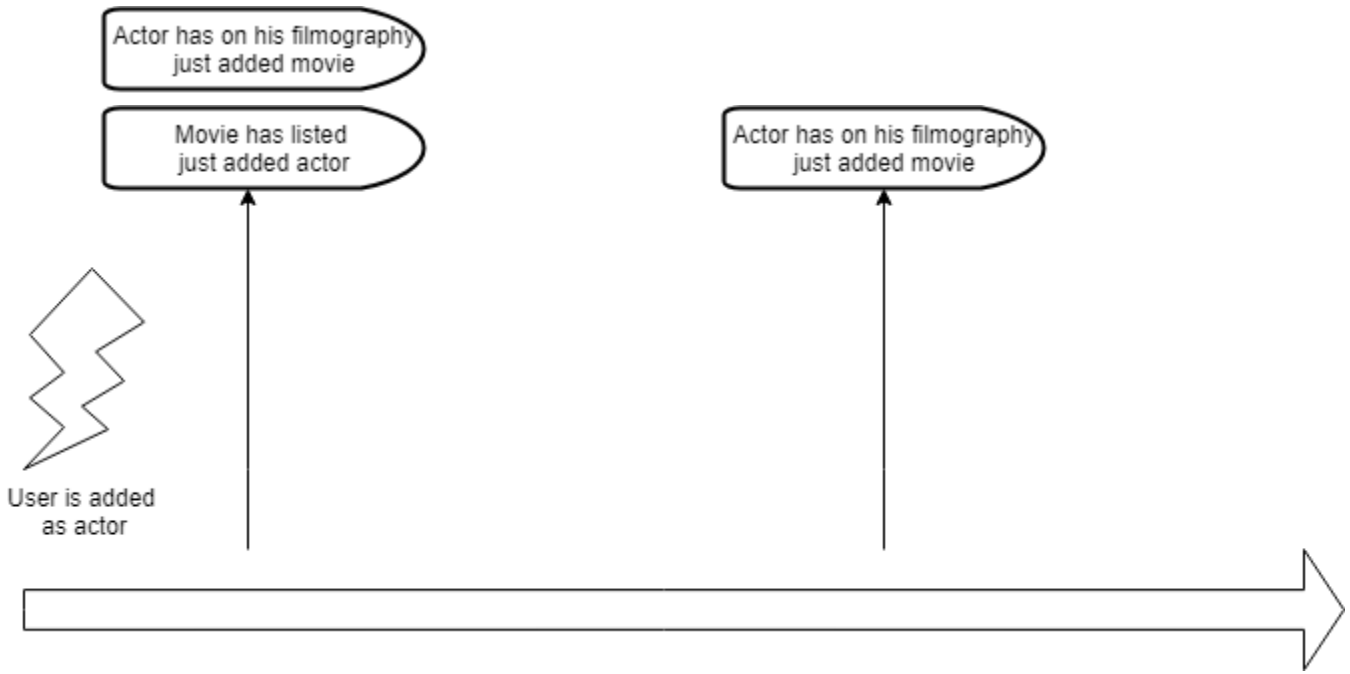
- Parsing data
- Validation of properties
- Integrations with other services
- Initial population of data
- Merging different data types
- Changes on deployed service

2/ Common mistakes

- Metadata without or with generic type
- Generalize data “in advance”
- Duplication of configuration
- Data aggregation at database layer
- Validators at property level
- Similar types with completely different implementation
- Manual configuration in database
- Not validated input data

3/ Data consistency

3/ Eventually consistency

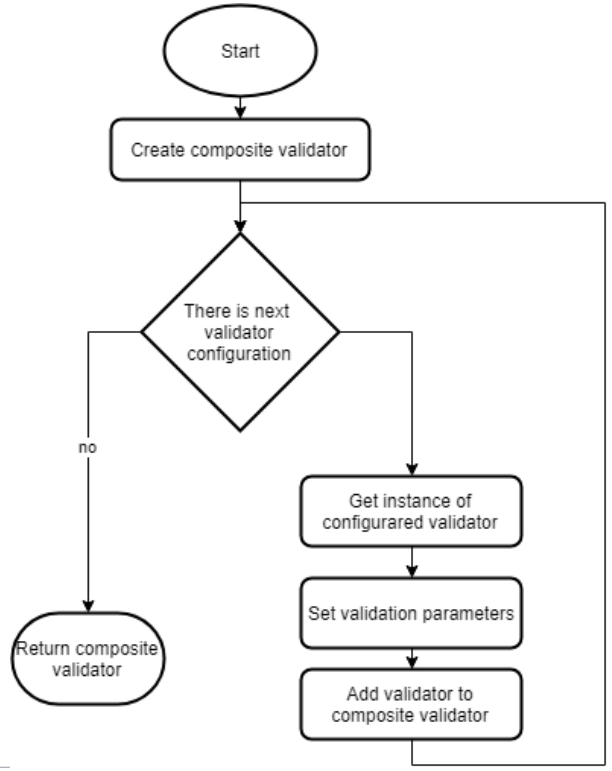


3/ Validation pipeline

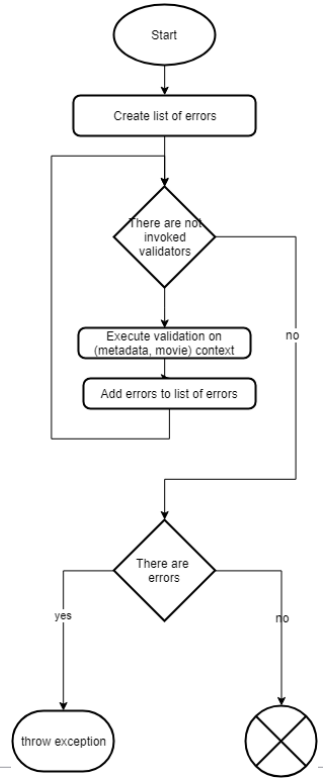
- Context of validation
- Validation abstraction
- Validation result

3/ Validation - algorithm

Create validator



Validate items



3/ Validation – algorithm usage

```
public interface IValidator
{
	string Key { get; }
	List<Error> Validate(DynamicMetadata dynamicMetadata, Movie context);
	void ConfigureValidator(Dictionary<string, string> validatorKeyParameters);
	bool WillUseProperty(string propertyKey);
	bool IsValidValidator(DynamicMetadataDefinition definition);
}
```

IValidator.cs

3/ Basic validators

- Property validators – range, is null
- Cross property validators

NumericRangePropertyValidator.cs

3/ Basic validators – not null validator

```
public List<Error> Validate(DynamicMetadata dynamicMetadata, Movie context)
        {
            var property = dynamicMetadata.Properties.First(x => x.Id == Property);

            if (property.Value != null)
            {
                return new List<Error>();
            }

            return new List<Error>
            {
                new Error(102312, "Null value")
            };
        }
```

NotNullValidator.cs

3/ Complex validators

- Cross property validation
- Database verification
- Cross item validation

3/ Complex validators

```
public List<Error> Validate(DynamicMetadata dynamicMetadata, Movie context)
     {
          var property = dynamicMetadata.Properties.First(x => x.Id == Property);
          if (property.Value != null)
          {
               Model.Person value = (Model.Person)property.Value;
               var dbItem = _personReadService.GetById(value.Id);
               if (dbItem == null)
               {
                    return new List<Error>
                    {
                         new Error(23123123, "Person doesnt exist")
                    };
               }
          }
          return new List<Error>();
     }
```

ExistingPersonValidator.cs

3/ Missing property

- Case when in request there is missing property
- Solutions
  - Default values
  - Missing property exceptions

CreateMovieMetadataCommandHandler.cs

3/ Missing property

```
foreach (var propertyInDefinition in definition.Properties)
{
    var existingProperty = metadata.Properties.FirstOrDefault(x => x.Id == propertyInDefinition.Id);
    if (existingProperty == null)
    {
        metadata.Properties.Add(propertyInDefinition);
    }
}
```

CreateMovieMetadataCommandHandler.cs

3/ Duplicated property

- Property appears more than once
- Solutions
  - Exception
  - Dismissing all properties but one*

3/ Additional property

- There is property not appearing in defeinition
- Solutions
  - Throws exception
  - Skip that item

3/ Invalid content

- Available scenarios
  - Data not matched declared type
  - Type in request differs from type in definition
- Solutions
  - Exception

3/ Invalid content

```
if (propertyDefinition.Type != metadataEntry.Type) {
    throw new AggregatedValidationException("Type mismatch")  {
                        ErrorCode = 11231,
                        ValidationErrors = new List<Error>()
                    };
}
if (metadata.Properties.SingleOrDefault(x => x.Id == metadataEntry.Id) == null) {
                    try {
                        var parsedValue = _inputDataParser.DeserializeItem(metadataEntry.Id, metadataE
ntry.Type, metadataEntry.Value);
                        metadata.Properties.Add(parsedValue);
                    }
                    catch (JsonException exception) {
                        throw new AggregatedValidationException("Invalid type mismatch") {
                            ErrorCode = 11231,
                            ValidationErrors = new List<Error>()
                        };
                    }
                }
```

4/ Changes tracking

4/ Content changes

- Adding or removing property
- Delete metadata definition
- Change type / default value
- Modify validation configuration

4/ General description

Basing on eventually consistent concept we will give the system a few seconds to clean up the state after changes. After „modification” we will invoke specific operations on existing data to match the updated state.

EventHandler.cs

EventDispatcher.cs

4/ Adding or removing property

After adding or removing property in source object natural operation is updating target object in similar way. 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.

PropertyRemovingFromDefinitionCommandHandler.cs

PropertyRemovedEventHandler.cs

PropertyAddedEventHandler.cs

4/ 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.

4/ Change of type

Solutions:

- Block changes
- mark it as out of date and require action by some administrator
- Implement converter from old type to new one

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.

MetadataPropertyInDefinitionChangedEventHandler.cs

4/ Validation configuration

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.

RefreshValidationEventHandler.cs

5/ Data population

5/ Smart data population

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.

5/ Metadata definition

For purpose of item generation, we will use Fluent interface approach whereas 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.

MetadataDefinitionBuilder.cs

5/ Property definition

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

PropertyBuilders.cs

5/ Property initialization

```
x.Add(new DefinitionBuilder("recording_location", "Location of recording")
                    .AddValidator(validatorCfg => validatorCfg
                                                  .UseKey("numeric-range")
                                                  .UseValidatorConfiguration("property-key", "year")
                                                  .UseValidatorConfiguration("max", "2021"))
                    .AddProperty<NumberPropertyBuilder>("year", propertyCfg => propertyCfg.DefaultValue(2020))
                    .AddProperty<StringPropertyBuilder>("scene name",
                        propertyCfg => propertyCfg.DefaultValue("Cracow Old City"))
                    .Build());

x.Add(new DefinitionBuilder("director", "Director")
                    .AddProperty<PersonPropertyBuilder>("person", propertyCfg => propertyCfg.DefaultValue(null))
                    .AddValidator(validatorCfg => validatorCfg.UseKey("single-type-item"))
                    .AddValidator(validatorCfg => validatorCfg.UseKey("not-
null").UseValidatorConfiguration("property-key", "person"))
                    .AddValidator(validatorCfg => validatorCfg.UseKey("existing-
person").UseValidatorConfiguration("property-key", "person"))
                    .Build());

x.Add(new DefinitionBuilder("actor", "actor")
                    .AddProperty<PersonPropertyBuilder>("person", propertyCfg => propertyCfg.DefaultValue(null))
                    .Build());
```

Sum up

New type actions

- Add type definition
- Add validators
- Serialization & deserialization methods
- Add data population
- Presentation

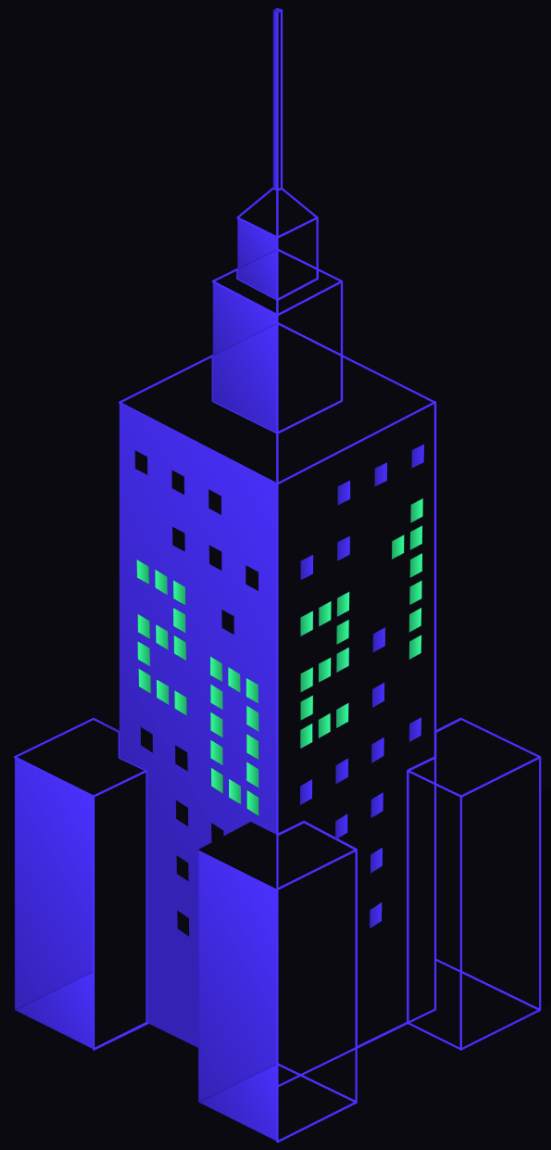
New definition actions

- Click out defintion from broad possibilities of types and validators

WARSAW
IT DAYS

Thank you for watching!

Remember to rate the presentation and
leave your questions in the section below.



www.WarszawskieDniInformatyki.pl