ibexa Summit 25

Introduction to the Discounts system in Ibexa DXP

Konrad Oboza

About myself

Konrad Oboza
Senior Software Engineer
In **Ibexa** for over 7 years

Pasionate about **rock climbing**
and **road cycling**

https://github.com/konradoboza

https://www.linkedin.com/in/konrad-oboza

konrad.oboza@ibexa.co

ibexa

Agenda

1. What ibexa/discounts is all about?
2. Architectural concepts
3. API overview
4. How to extend?
5. Example customization
6. Feature future ;)

Rules

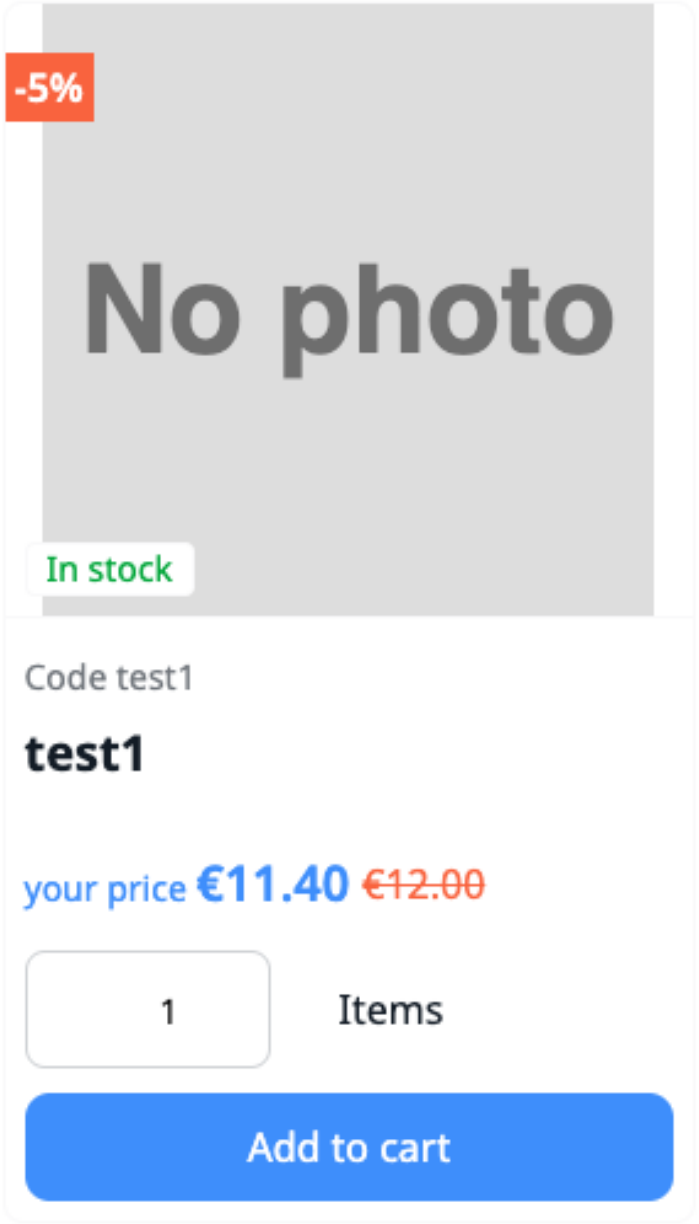
Questions will be answered at the end of the session

ibexa

1. What is `ibexa/discounts`?

4.6 LTS update:

- available for Ibexa Commerce tier
- opt-in package
- builds on top of: ibexa/cart, ibexa/checkout, ibexa/storefront, ibexa/product-catalog and ibexa/order-management



2. Architectural concepts

Symfony Expression Language

- put logic into an existing expression notation
- computing on the server side instead of painful database querying
- easier to extend
- supports caching
- fits scenario where discount object needs to meet specific conditions
- handles even very complex logic – **Expression Provider**

```php
public function getExpression(): string
{
    return 'currency.getCode() === code';
}
```

```php
public function getExpression(): string
{
    return 'summary
        .getTotalPrice()
        .getAmount() >= minimal_purchase_amount';
}
```

```php
public function getExpression(): string
{
    return 'is_user_in_customer_group(
        user,
        customer_group
    )';
}
```

ibexa

Expression Provider

```php
final class ExpressionLanguageProvider implements ExpressionFunctionProviderInterface
{
    public function getFunctions(): array
    {
        return [
            new ExpressionFunction(
                'is_user_in_customer_group’,
                static function (): string {
                    return 'is_user_in_customer_group(user, customer_group)’;
                },
                [$this, 'isUserInCustomerGroup’]
            ),
        ];
    }

    public function isUserInCustomerGroup(array $arguments): bool
    {
        $customerGroup = $arguments['customer_group’];
        $user = $arguments['user’];

        //checking the above parameters via API (…)
        return true;
    }
}
```

ibexa

Rules vs Conditions?

Differences

- the way discount is calculated is shaped by **rule**
- discount needs to meet **all the conditions** to be selected
- rule is calculated to return the final price
- condition is evaluated (true/false)
- there can be **only one rule** but **multiple conditions**

Similarities

- both are based on Expression Language
- both can be extended by adding custom implementations

```php
final class FixedAmount
extends AbstractDiscountExpressionAware
implements DiscountRuleInterface
{
    public const TYPE = 'fixed_amount';

    public function getType(): string
    {
        return self::TYPE;
    }

    public function getExpression(): string
    {
        return 'amount - discount_amount';
    }
}

final class IsInCurrency
extends AbstractDiscountExpressionAware
implements DiscountConditionInterface
{
    public function getIdentifier(): string
    {
        return 'is_in_currency';
    }

    public function getExpression(): string
    {
        return 'currency.getCode() === currency_code';
    }
}
```

ibexa

3. API overview

Managing discounts

`Ibexa\Contracts\Discounts\DiscountService`

1. creating/updating/reading (by id or identifier)/deleting
2. activating/deactivating
3. fetching using criteria and sort clauses

`Ibexa\Contracts\Discounts\DiscountPrioritizationStrategyInterface`

1. order of discount fields for the resolving
2. only active and not outdated discounts are taken into account + the default order:
   a) type (**cart vs catalog**)
   b) priority
   c) creation date

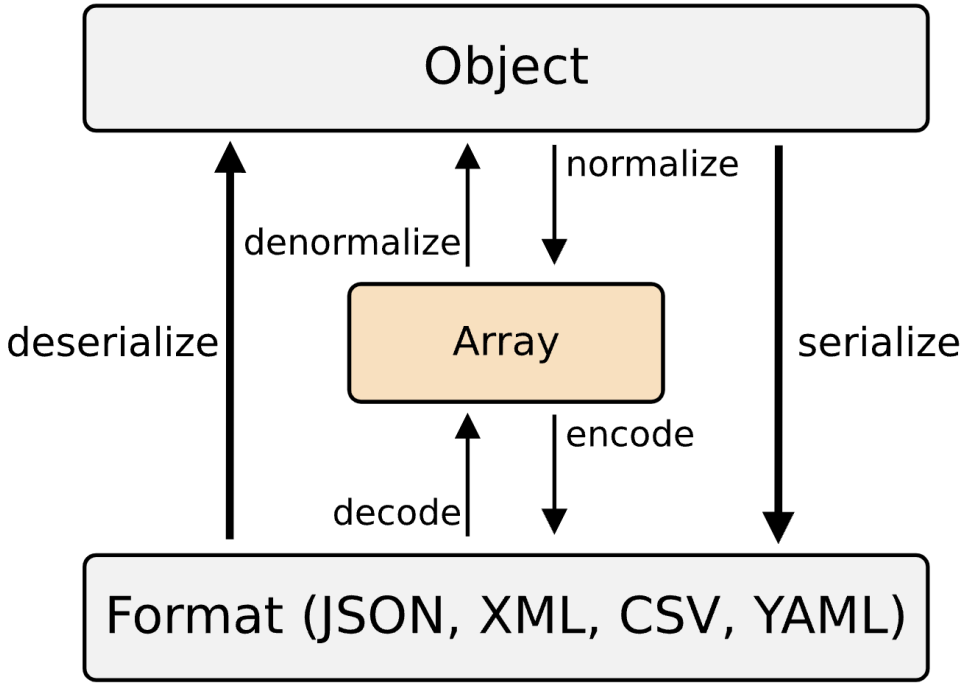
Built-in discount rules/conditions

Rules:

1. Fixed amount (e.g. -100 EUR)
2. Percentage (e.g. – 10%)
3. Buy X get Y (in progress)

Conditions:

1. IsInCategory()
2. IsInCurrency()
3. IsProductCurrentlyProcessed()
4. IsProductInArray()
5. IsProductInQuantityInCart()
6. IsUserInCustomerGroup()
7. MinimalPurchaseAmount()



Serializable context

- Continuation of the concept already settled across Ibexa Commerce packages (cart/checkout/order-management/shipping/payment)
- Serializable data (Symfony Serializer under the hood)
- Normalizer must each time correspond with your custom object that you want to be stored

```
App\Normalizer\MyCoolNormalizer:
  tags:
    - {
        name: ibexa.discounts.serializer.normalizer,
        priority: -100
      }
```

just take a picture of the QR code!

Resolving discounts and prices

```
Ibexa\Discounts\Price\DiscountPriceResolver:
    decorates: Ibexa\Contracts\ProductCatalog\PriceResolverInterface
    arguments:
        $innerResolver: '@.inner’
```

```
Ibexa\Discounts\DiscountResolver
```

```
Ibexa\Discounts\Value\Price\Stamp\DiscountStamp
```

```
interface PriceStampInterface extends StampInterface
{
    public function getNewPrice(): Money;

    public function getOriginalPrice(): Money;
}
```

Managing forms

`Ibexa\Contracts\Discounts\DiscountFormMapperInterface`

Allows manipulating steps of the creation form (wizard)

Extending new Discounts to suit your needs
by **Paweł Niedzielski**

Tomorrow, Tech Track, 10:00 – 10:30

4. How to extend?

#1 Rule/condition class

```
final class MyCustomRule extends AbstractDiscountExpressionAware implements DiscountRuleInterface
{
    // your custom logic goes here
}
```

```
final class MyCustomCondition extends AbstractDiscountExpressionAware implements DiscountConditionInterface
{
    // your custom logic goes here
}
```

#2 Rule/condition factory

```php
final class MyCustomRuleDiscountRuleFactory implements DiscountRuleFactoryInterface
{
    /**
     * @param array<string, mixed>|null $expressionValues
     */
    public function createDiscountRule(?array $expressionValues = null): DiscountRuleInterface {
        return new MyCustomRule($expressionValues);
    }
}
```

```php
final class MyCustomConditionDiscountConditionFactory implements DiscountConditionFactoryInterface
{
    /**
     * @param array<string, mixed>|null $expressionValues
     */
    public function createDiscountCondition(?array $expressionValues = null): DiscountConditionInterface {
        return new MyCustomCondition($expressionValues);
    }
}
```

#3 Registering factories

```yaml
App\DiscountRule\MyCustomRuleDiscountRuleFactory:
      tags:
        - { name: ibexa.discounts.rule.factory, discriminator: 'my_custom_rule' }
```

```yaml
App\DiscountCondition\MyCustomConditionDiscountConditionFactory:
      tags:
        - { name: ibexa.discounts.condition.factory, discriminator: 'my_custom_condition’ }
```

#3 Registering factories

```yaml
App\DiscountRule\MyCustomRuleDiscountRuleFactory:
    tags:
      - { name: ibexa.discounts.rule.factory, discriminator: 'my_custom_rule' }
```

```yaml
App\DiscountCondition\MyCustomConditionDiscountConditionFactory:
    tags:
      - { name: ibexa.discounts.condition.factory, discriminator: 'my_custom_condition’ }
```

Note: Discriminator must match rule type and condition identifier respectively!

5. Example customization

What is your goal?

- Custom, seasonal, referral-based discount
- Triggered by the URL containing a registered event code
- Applicable only for specific, predefined product code
- Discount formula is dependent on some external data

Steps:

1	2	3	4	5
Custom rule	**Custom condition**	**Discount via API**	**Handling referral**	**Calculating price**
Determine the percentage discount based on the current EUR / USD ratio	Check specific referral code and product code	Create a discount with proper contextual data about EUR / USD ratio	Intercept discount code from the provided URL	Display all the necessary data and the discount value in the cart summary

#1 Custom rule

```php
<?php

namespace App\Discounts;

use Ibexa\Contracts\Discounts\Value\DiscountRuleInterface;
use Ibexa\Discounts\Value\AbstractDiscountExpressionAware;

final class ReferralRule extends AbstractDiscountExpressionAware implements DiscountRuleInterface
{
    public function getType(): string
    {
        return 'referral_rule’;
    }

    public function getExpression(): string
    {
        return '(euro_to_dollar_ratio < 1)
            ? amount - (amount * 0.1)
            : amount - (amount * 0.09)
        ';
    }
}
```

#2 Custom condition

```php
<?php

namespace App\Discounts;

use Ibexa\Contracts\Discounts\Value\DiscountConditionInterface;
use Ibexa\Discounts\Value\AbstractDiscountExpressionAware;

final class ReferralCondition extends AbstractDiscountExpressionAware implements DiscountConditionInterface
{
    public function getIdentifier(): string
    {
        return 'referral_condition';
    }

    public function getExpression(): string
    {
        return 'product.getCode() === product_code and
                referral_code === \'IbexaSummit2025\'';
    }
}
```

#3 Discount via API

```
$struct = new DiscountCreateStruct(
    'referral_discount',
    DiscountType::CART, //type „catalog” vs „cart”
    10,     //priority
    true,   //is discount active?
    $this->getCurrentUser(),  //creator
    new ReferralRule(),
    new DateTimeImmutable('2025-01-01 08:00:00'), //start date
    [
        new IsInCurrency(['currency_code' => 'EUR']),
        new ReferralCondition(['product_code' => 'ibexa_summiter']),
    ],
    [
        new DiscountTranslationStruct(
           'eng-GB',
           'Ibexa Summit Referral',
        ),
    ],
    new DateTimeImmutable('2025-01-31 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00’),

    new MutableArrayMap([
        'euro_to_dollar_ratio' => 1.5,
    ])
);

$this->discountService->createDiscount($struct);
```

#3 Discount via API

```
$struct = new DiscountCreateStruct(
    'referral_discount',
    DiscountType::CART, //type „catalog” vs „cart”
    10,     //priority
    true,   //is discount active?
    $this->getCurrentUser(),  //creator
    new ReferralRule(),
    new DateTimeImmutable('2025-01-01 08:00:00’), //start date
    [
        new IsInCurrency(['currency_code' => 'EUR']),
        new ReferralCondition(['product_code' => 'ibexa_summiter']),
    ],
    [
        new DiscountTranslationStruct(
           'eng-GB',
           'Ibexa Summit Referral',
        ),
    ],
    new DateTimeImmutable('2025-01-31 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00’),

    new MutableArrayMap([
        'euro_to_dollar_ratio' => 1.5,
    ])
);

$this->discountService->createDiscount($struct);
```

#3 Discount via API

```
$struct = new DiscountCreateStruct(
    'referral_discount',
    DiscountType::CART, //type „catalog” vs „cart”
    10,     //priority
    true,   //is discount active?
    $this->getCurrentUser(),  //creator
    new ReferralRule(),
    new DateTimeImmutable('2025-01-01 08:00:00’), //start date
    [
        new IsInCurrency(['currency_code' => 'EUR']),
        new ReferralCondition(['product_code' => 'ibexa_summiter']),
    ],
    [
        new DiscountTranslationStruct(
           'eng-GB',
           'Ibexa Summit Referral',
        ),
    ],
    new DateTimeImmutable('2025-01-31 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00’),

    new MutableArrayMap([
        'euro_to_dollar_ratio' => 1.5,
    ])
);

$this->discountService->createDiscount($struct);
```

#3 Discount via API

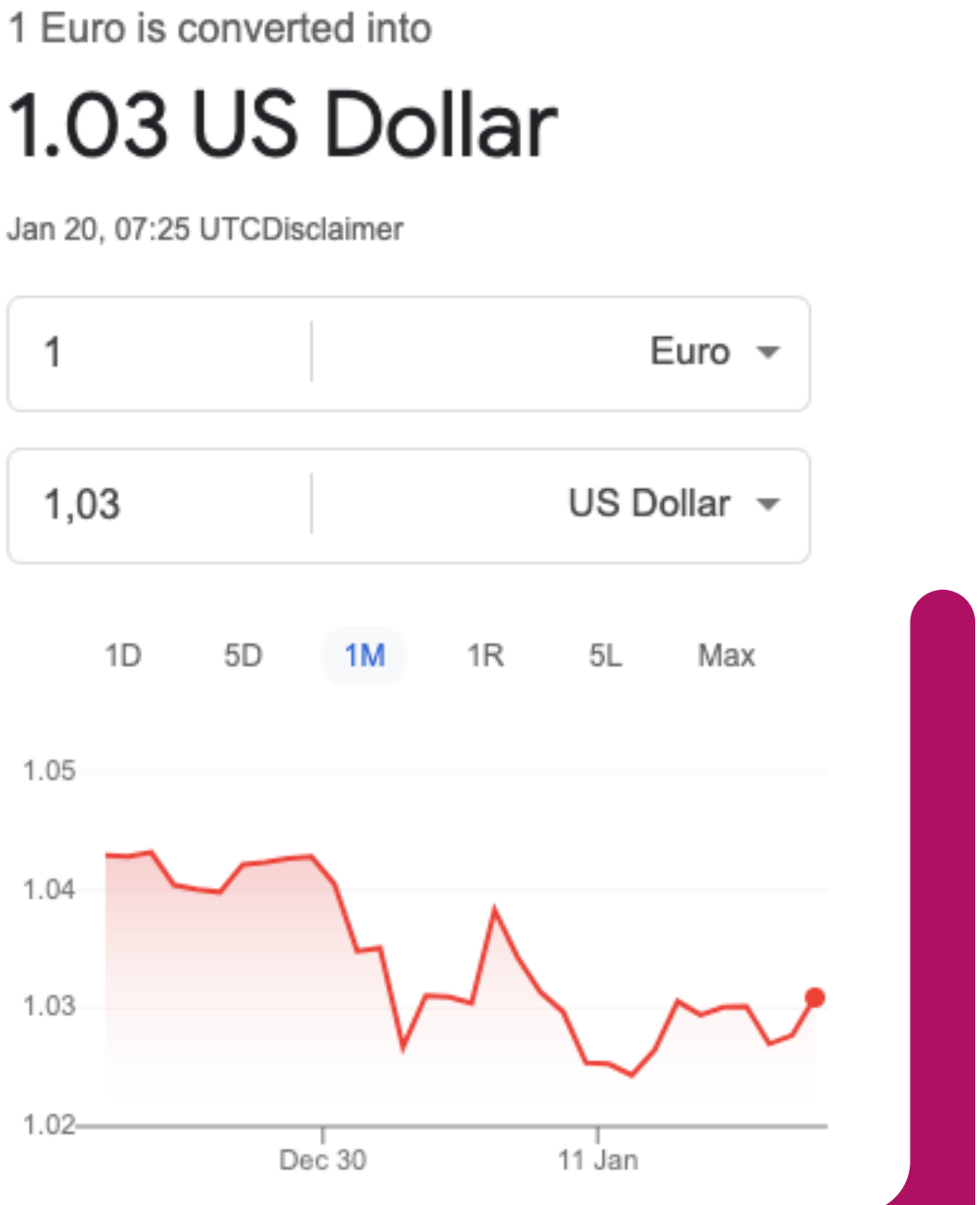
```
$struct = new DiscountCreateStruct(
    'referral_discount',
    DiscountType::CART, //type „catalog” vs „cart”
    10,     //priority
    true,   //is discount active?
    $this->getCurrentUser(),  //creator
    new ReferralRule(),
    new DateTimeImmutable('2025-01-01 08:00:00’), //start date
    [
        new IsInCurrency(['currency_code' => 'EUR']),
        new ReferralCondition(['product_code' => 'ibexa_summiter']),
    ],
    [
        new DiscountTranslationStruct(
           'eng-GB',
           'Ibexa Summit Referral',
        ),
    ],
    new DateTimeImmutable('2025-01-31 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00'),
    new DateTimeImmutable('2025-01-01 16:30:00’),

    new MutableArrayMap([
        'euro_to_dollar_ratio' => 1.5,
    ])
);

$this->discountService->createDiscount($struct);
```

#4 Handling referral

```
$cart = $this->getCart($identifier);
$cartContext = $cart->getContext();

$cartContextArray = $cartContext === null ? [] : $cartContext->toArray();
$cartContextArray['referral_code'] = $request->query->get('discount_code');

$updatedCart = $this->cartService->updateCartMetadata(
    $cart,
    new CartMetadataUpdateStruct(
        null,
        null,
        null,
        new MutableArrayMap($cartContextArray)
    )
);
```

#4 Handling referral

```
$cart = $this->getCart($identifier);
$cartContext = $cart->getContext();

$cartContextArray = $cartContext === null ? [] : $cartContext->toArray();
$cartContextArray['referral_code'] = $request->query->get('discount_code');

$updatedCart = $this->cartService->updateCartMetadata(
    $cart,
    new CartMetadataUpdateStruct(
        null,
        null,
        null,
        new MutableArrayMap($cartContextArray)
    )
);
```



#5 Calculating price

- Make sure your discount is resolved
- “Cart” discount type, high priority and creation date will increase this chance (prioritization strategy)
- `{"euro_to_dollar_ratio":1.5}` each value can be calculated by pulling some data in external service that can be e.g. put in cron
- This service can fetch our discount and update the above value via `$discount->setContext()` daily

Customization summary:

- custom rule for calculating price based on EUR/USD ratio context parameter
- custom condition applying when:
  - „IbexaSumit2025” referral exists
  - product of code „ibexa_summiter” is selected
- discount referral from the current URL
- discount object via API

Let us see all of this in practice!

The road ahead…

1. Improving UI
2. Easing up customizing forms and custom rules/conditions
3. Testing edge cases
4. Providing more discount conditions OOTB
5. Distant future: price history, discount performance block, statistics etc.

Thank you!