Journey of the Cart

Don't break the wheels

@r4pt0s

About me

👩‍💻 Name

Wolfgang Kreminger

🎨 Role

Fullstack Developer

🔨 I work for

0815 Onlinehandel GmbH

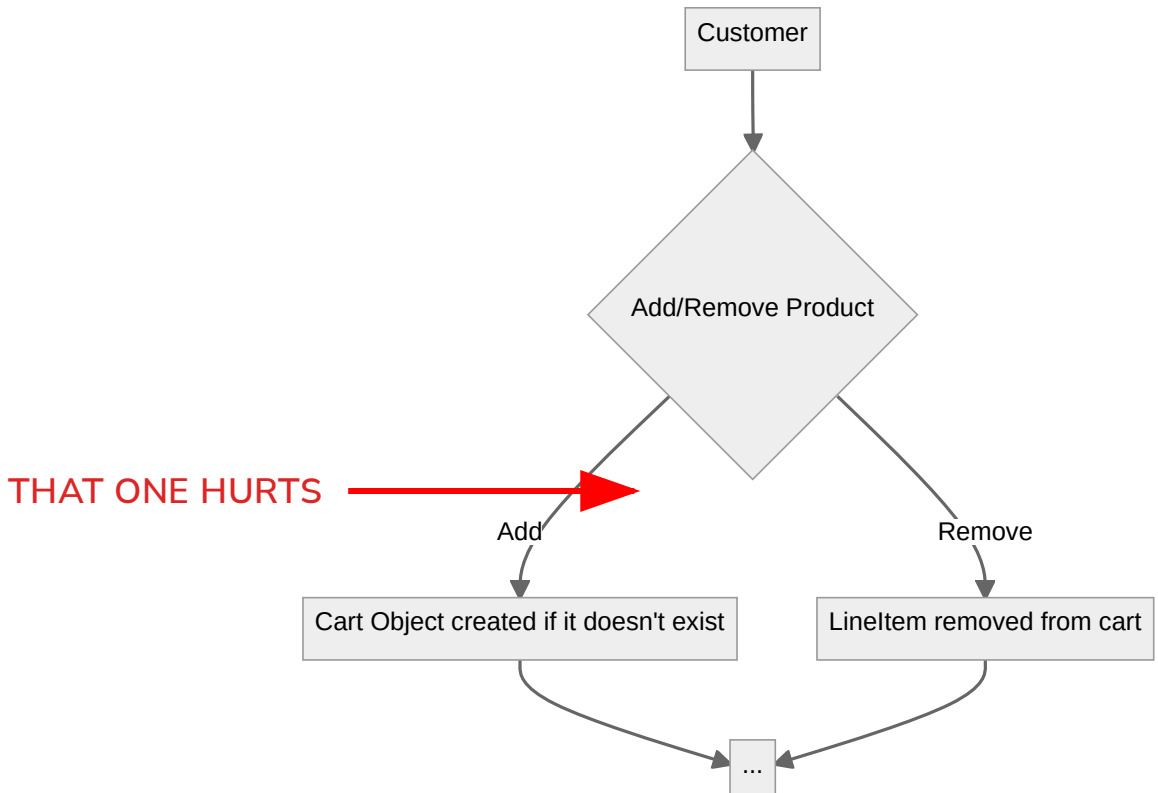
🤣 One job is not enough

Teaching assistant at zerotomastery.io

<Coding>4 LifeBalance</Coding>

LET'S JUMP RIGHT INTO IT ->

Cart in the database



Product Associations the good and evil

The Evil 😈

- Associations which you add via Product event subscribers will go into your cart
- CustomFields of products - all of them - will go into your cart
- Thumbnails are not always as helpful as you might guess

The Good 😇

- You have the control over added associations
- Usable in the Front-end for enhancing UX
- Cleanup before storing the cart is pretty straight forward

CartDataCollection - curse and saviour

Saviour part 👀

- Necessay for shopware logic
  - Product data store
  - Promotions data store
  - Shipping method data store
- Contains entire product data for the corresponding LineItem (key = "product-<Uuid>")
- Is used to keep track of product data inside the cart

CartDataCollection - curse and saviour

Curse part 🧐

- Contains data which you might not need for each single product
  - CustomFields
  - All thumbnails
  - Entire Description
  - All loaded Associations

Why you should care about cleaning up the Cart?

- Database Memory
- Performance
  - During the shopping process of a customer, add to cart takes longer for each product (linear time)
  - Database backup (final db dump file size)
  - Server (serialize, deserialize)

One Event - Cleanup before persisting

```php
// vendor/shopware/core/Checkout/Cart/CartPersister.php

/**
 * @throws InvalidUuidException
 */
public function save(Cart $cart, SalesChannelContext $context): void
{
    // .. rest of code

    $event = new CartVerifyPersistEvent($context, $cart, $shouldPersist);

    $this->eventDispatcher->dispatch($event);

    // ... rest of code
}
```

What do we get from this event?

```php
class CartVerifyPersistEvent extends Event implements ShopwareSalesChannelEvent
{
// ... rest
    public function getContext(): Context
    {
```

Cleanup in EventSubscriber

```
// custom/plugins/ExamplePlugin/src/Resources/config/services.xml
```

```
<!— rest before —>
<service id="ExamplePlugin\Subscriber\CartVerifyPersistSubscriber">
    <tag name="kernel.event_subscriber"/>
</service>
<!— rest after —>
```

```
//ExamplePlugin/src/Subscriber/CartVerifyPersistSubscriber.php

class CartVerifyPersistSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents(): array
```

Performance impact

- Reduced data leads to faster serialization/deserialization of the cart object
- Faster response time of the server
- Less database load because of less amount of data
- Less storage in Database needed

Is there something left which we can do? 🤔

🤯 SAVE A ENTIRE REQUEST FOR LINEITEM ADD 🤯

Save a entire request - The Path

- By default, shopware redirects to `@Route "/checkout/offcanvas"`

  via `$this->createActionResponse($request);`

  in `@Route "/checkout/line-item/add"`

- `@Route "/checkout/offcanvas"` uses `Shopware\Storefront\Page\Checkout\Offcanvas\OffcanvasCartPageLoader::load`

- `OffcanvasCartPageLoader::load` method uses `Shopware\Core\Checkout\Cart\SalesChannel\CartService::get`

- `CartService::get` will load the entire cart from the database again (+ deserialization) altough we already have it in our line-item/add route 🤔

Save a entire request - Decorate CartLineItemController

- Override the entire `public function addLineItems`
- Copy the entire method body of

  `\Shopware\Storefront\Controller\CartLineItemController::addLineItems`
- Instead of calling createActionResponse do

```php
$redirectTo = $requestDataBag->get('redirectTo');

if ($redirectTo === 'frontend.checkout.cart.page') {
    // we want to do redirect in this case
    return $this->createActionResponse($request);
}
// call the new custom loader which doesn't fetch the cart again
$page = $this->cartToPageLoader->prepareOffcanvasCartPage($cart->getCart(), $request, $salesChannelContext);

//render the storefront instead of doing a redirect
return $this->renderStorefront(
  '@Storefront/storefront/component/checkout/offcanvas-cart.html.twig',
  ['page' => $page]
);
```

Save a entire request - Build Custom Page Loader
The important loader

```php
// custom/plugins/ExamplePlugin/src/Storefront/Page/Checkout/Offcanvas/CartToPageLoader.php

public function prepareOffcanvasCartPage(
  Cart $cart,
  Request $request,
  SalesChannelContext $salesChannelContext
  ): OffcanvasCartPage
{
    $page = $this->genericLoader->load($request, $salesChannelContext);

    $page = OffcanvasCartPage::createFrom($page);
    $page->setCart($cart);
    $page->setShippingMethods($this->getShippingMethods($salesChannelContext));

    // make sure to dispatch event to keep shopware event flow
    $this->eventDispatcher->dispatch(
        new OffcanvasCartPageLoadedEvent($page, $salesChannelContext, $request)
    );

    return $page;
}
```

Save a entire request - Build Custom Page Loader Necessary method

```php
// custom/plugins/ExamplePlugin/src/Storefront/Page/Checkout/Offcanvas/CartToPageLoader.php

private function getShippingMethods(SalesChannelContext $context): ShippingMethodCollection
{
    $request = new Request();
    $request->query->set('onlyAvailable', '1');

    $shippingMethods = $this->shippingMethodRoute
      ->load($request, $context, new Criteria())
      ->getShippingMethods();

    if (!$shippingMethods->has($context->getShippingMethod()->getId())) {
        $shippingMethods->add($context->getShippingMethod());
    }

    return $shippingMethods;
}
```

Save a entire request - Add service

```
// custom/plugins/ExamplePlugin/src/Resources/config/services.xml
```

```
<service id="ExamplePlugin\Storefront\Controller\CartLineItemController"
         decorates="Shopware\Storefront\Controller\CartLineItemController">
    <argument type="service" id="Shopware\Core\Checkout\Cart\SalesChannel\CartService"/>
    <argument type="service" id="sales_channel.product.repository"/>
    <argument type="service" id="Shopware\Core\Framework\Util\HtmlSanitizer"/>
    <argument type="service" id="ExamplePlugin\Storefront\Page\Checkout\Offcanvas\CartToPageLoader"/>
    <call method="setContainer">
        <argument type="service" id="service_container"/>
    </call>
</service>

<service id="ExamplePlugin\Storefront\Page\Checkout\Offcanvas\CartToPageLoader">
    <argument type="service" id="Shopware\Storefront\Page\GenericPageLoader"/>
    <argument type="service" id="Shopware\Core\Checkout\Shipping\SalesChannel\ShippingMethodRoute"/>
    <argument type="service" id="event_dispatcher"/>
</service>
```

Performance improvement by Shopware - RedisCartPersister to the rescue (maybe)

Personal thoughts 🤔

- Performance might be better but what is still the same is serialization/deserialization of the cart object
- Doesn’t solve the problem of bloated cart data which might get stored
- Think about the redis setup
  - Deployments might delete all customer carts which are stored in redis
- Doesn’t solve the problem of one *"unnecessary"* request which happens by adding a product to the cart (redirect offcanvas)

✅ You can totally use the same code also if you use the RedisCartPersister

Lifetime of a cart object in the data store

Shopware default behaviour

- Customer enjoys searching around in the online shop
- Customer adds products to the cart
- Customer proceeds through the checkout
- Customer finializes the checkout with the payment process and hits the finish checkout button

Lifetime of a cart object in the data store

Shopware default behaviour

Something went wrong while fininalizing the Order 🤔

- If the payment process failed, cart gets still deleted from the data store (not available for the customer anymore)
- Customer might not be satisfied because it looks like the cart is gone and products need to get added from scratch again
- Of course, there is the account/order/edit page, but cart can’t get changed anymore (just the payment method)
- In the end it’s like a dead end of the customer journey

Expected lifetime of a cart

- In case something went wrong, cart should not get dropped
- Customer should have the ability to change the cart till the order is finially finished
- It should always be possible to jump back and forth between payment and the cart

🙂=>😀=>😄

One possible solution

- At the point where shopware drops the cart (cart to order conversion), add the cart back to the data store
- Add "Modify cart" button to the account/order/edit page or redirect to checkout/confirm directly
- Store the current order ID in the user session on finalize order button click
- Check if a order ID is stored in the user session if yes, cancel old order before creating new one
- If customer reaches the final order page, finially drop the cart from data store

Part 1 - Add cart token to session, save cart again

- Store current cart token on `CheckoutConfirmPageLoadedEvent::class`

```php
// ExamplePlugin/src/Subscriber/CheckoutConfirmPageLoadedEventSubscriber.php
public function setCartTokenToSession(CheckoutConfirmPageLoadedEvent $event): void
{
    $request = $event->getRequest();
    // set cart token to session
    $request->getSession()->set(XgxCartExtension::SESSION_CART_TOKEN, $event->getPage()->getCart()->getToken());
}
```

- Decorate and override `Shopware\Core\Checkout\Cart\SalesChannel\CartOrderRoute::order`
- Save cart after it got deleted from Shopware

```php
// ExamplePlugin/src/Core/Checkout/Cart/SalesChannel/CartOrderRoute.php
public function order(Cart $cart, SalesChannelContext $context, RequestDataBag $data): CartOrderRouteResponse
{
    $result = $this->originalCartOrderRoute->order($cart, $context, $data);

    $this->cartPersister->save($cart, $context); // ⟸ that is the key

    return $result;
}
```

Part 2 - Store current order ID in user session

- Create OrderPlacedSubscriber and subscribe to `` `CheckoutOrderPlacedEvent::class` ``
- Inject RequestStack
- If a order ID is already stored in user session, cancel previous order then store new one
- Store current order ID into user session

```php
// ExamplePlugin/src/Subscriber/OrderPlacedSubscriber.php
public function onOrderPlaced(CheckoutOrderPlacedEvent $event): void
{
    $request = $this->requestStack->getCurrentRequest();
    if (!$request || !$request->hasSession()) {
        return;
    }

    $openOrderId = $request->getSession()->get(ExamplePlugin::SESSION_KEY_ORDER_ID);
    if ($openOrderId) {
        $this->orderService->orderStateTransition(
            $openOrderId,
            'cancel',
            new ParameterBag(),
            $event->getContext()
        );
    }
```

Part 3 - Delete cart finally

- Once a customer checked out successfully, we can delete the cart
- A good entry point would be in

  `Shopware/Core/Checkout/Payment/Controller/PaymentController::finalize-transaction`
- Depends on which payment methods are used
- Latest point in order process where it makes sense is after succesful payment

```php
// ExamplePlugin/src/Core/Checkout/Payment/Controller/PaymentController.php

public function cleanupSessionStorage(
    Request $request,
    SalesChannelContext $salesChannelContext,
    ?string $finishUrl
  ): void
{
    $cartToken = $request->getSession()->get(ExamplePlugin::SESSION_CART_TOKEN);

    $request->getSession()->remove(ExamplePlugin::SESSION_KEY_ORDER_ID);

    if ($finishUrl) {
        // in case you need finalize order url somehow
        $request->getSession()->set(ExamplePlugin::SESSION_FINALIZE_ORDER_URL, $finishUrl);
```

Thanks for your attention!!

Feel free to ask Questions now

END