ItkLoyaltyPoints

Entwicklerdokumentation

André Fischer

30.03.2026

Inhaltsverzeichnis

1 Technischer Überblick **2**
1.1 Symfony / Shopware-Konzepte (knapp) 2

2 Datenmodell `itk_loyalty_transaction` **3**

3 `services.xml` und technische Verdrahtung **4**

4 Konfigurationsgruppen **5**

5 Wichtige Services **5**

6 Twig- und Storefront-API **6**

7 Warenkorb und Einlösung **6**

8 Club-Zugriff und Ausschlusslogik **7**

9 Bestell- und Statuslogik **7**
9.1 OrderPlacedSubscriber 7
9.2 OrderStateChangeSubscriber 7

10 Storefront-Controller und Routen **8**

11 Subscriber (vollständige Liste laut Codebasis) **9**

12 CMS **9**

13 Scheduled Tasks **9**

14 Console Commands **10**

15 Admin **11**

16 Migrationen **11**

17 Vollständige PHP-Klassenliste (`src/`) **11**

18 Logging und Debugging **13**

1 Technischer Überblick

Plugin-Klasse: `ItkComputerGmbh\\ItkLoyaltyPoints\\ItkLoyaltyPoints`
Admin-Label: „25now Loyalty Club“ **Namespace:** `ItkComputerGmbh\\ItkLoyaltyPoints`
DAL-Entity: `itk_loyalty_transaction` (**LoyaltyTransactionDefinition**)
Konfigurationspräfix: `ItkLoyaltyPoints.config.*` (siehe `Resources/config/config.xml`)

Schwerpunkte im Code:

- **Cart: RedeemPointsCartProcessor** (Einlösung im Warenkorb)
- **Order: OrderPlacedSubscriber**, **OrderStateChangeSubscriber** (Award, Storno, Erstattung)
- **Kunde: CustomerRegisterSubscriber** (Willkommenspunkte)
- **Club / Newsletter:** mehrere Subscriber und Controller (Redirect, DOI, AC)
- **PassCreator:** eigener Unterbaum `PassCreator/` (Service, Webhook, Subscriber, CMS)
- **Scheduled Tasks: LoyaltyExpiringPointsEmailTask**, **PassCreatorPointsSyncTask** + Handler
- **Admin API: LoyaltyCustomerController**
- **Twig:** `Storefront\\Twig\\LoyaltyPointsExtension`
- **CMS:** Resolver für Loyalty-, Club- und Pass-Elemente

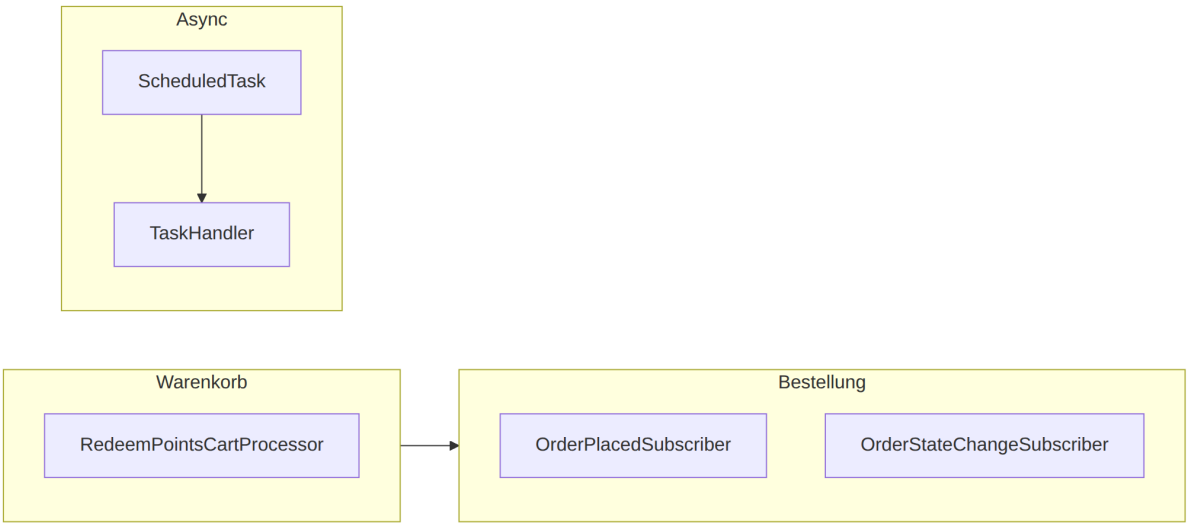
1.1 Symfony / Shopware-Konzepte (knapp)

- **CartProcessorInterface:** Shopware ruft beim Berechnen des Warenkorbs eine **Kette von Prozessoren** auf. **RedeemPointsCartProcessor** ist einer davon – er kann Line Items hinzufügen/entfernen und Preise anpassen, **bevor** die Bestellung entsteht.

- **State Machine:** Bestell- und Zahlungsstatus ändern sich über **Transitionen**. **OrderStateChangeSubscriber** lauscht auf passende Events und bucht Punkte nach oder storniert sie – du erweiterst das System selten direkt an der State Machine, sondern über **Subscriber**.

- **ScheduledTask + MessageHandler:** Ein Task wird in der DB registriert; Shopware (Worker) startet den **Handler** asynchron. Ohne laufenden **Messenger-Consumer** passiert nichts.

- **CMS Data Resolver:** Mappt Erlebniswelt-Elemente auf **Structs**, die im Twig-Template landen – analog zu „View Models“.



---

2 Datenmodell `itk_loyalty_transaction`

Felder laut **LoyaltyTransactionDefinition**:

- **id**
- **customerId**
- **customerNumber**
- **salesChannelId**
- **orderId**
- **points**
- **description**
- **transactionType**
- **lockedUntil**
- **expiresAt**
- **expiryEmailSentAt**
- **reversedTransactionId**

Assoziationen:

- **customer**
- **salesChannel**
- **order**
- **reversedTransaction**

Repository: `itk_loyalty_transaction.repository`

---

3 `services.xml` und technische Verdrahtung

Das Plugin nutzt fast alle klassischen Shopware-Service-Typen gleichzeitig:

- **Entity-Definition**
- **normale Services**
- **Cart-Processor**
- **Event Subscriber**
- **Storefront-Controller**
- **Admin-Controller**
- **CMS-Resolver**
- **Scheduled Tasks + Message Handler**

Wichtige Tags und ihre Bedeutung:

Tag	Beispiel	Zweck
shopware.entity.definition	**LoyaltyTransactionDefinition**	registriert die DAL-Entity
shopware.cart.processor	**RedeemPointsCartProcessor**	hängt Einlöse-Logik in die Cart-Pipeline
kernel.event_subscriber	**OrderPlacedSubscriber**	reagiert auf Shopware-/State-Machine-Events
controller.service_arguments	Storefront-Controller	macht Controller inkl. Injects nutzbar
shopware.cms.data_resolver	CMS-Resolver	füllt Erlebniswelt-Elemente mit Daten
shopware.scheduled.task	ScheduledTask-Klassen	registriert geplante Hintergrundjobs
messenger.message_handler	TaskHandler	verarbeitet die Tasks asynchron

Der wichtigste technische Punkt: **RedeemPointsCartProcessor** läuft mit Priorität `4500`. Wenn du Konflikte mit anderen Checkout-Plugins hast, ist genau diese Stelle meistens der Grund.

4 Konfigurationsgruppen

Die `config.xml` ist fachlich breit. Für das Verständnis im Code lohnt es sich, die Keys in Blöcken zu denken:

- **Aktivierung / globales Verhalten:** z. B. Plugin aktiv, systemglobales Verhalten, Terms-/Club-Freigaben
- **Punkteberechnung:** z. B. Punktewert, Mindestbestellwert, Sperr- und Verfallslogik
- **Einlösung im Checkout:** Limits, Darstellung, Checkout-Regeln
- **Club / Kundengruppe / Redirects:** Club-Zugang, AC-Integration, DOI-/Newsletter-Nähe
- **Willkommenspunkte:** Höhe, Bedingungen und Trigger bei Registrierung
- **E-Mail / Ablaufwarnung:** Warnfenster, Absender, Mail-Template-Logik
- **PassCreator:** API-/Sync-Einstellungen für Wallet-Funktionen

Für Code-Änderungen wichtig: Fast alle fachlichen Entscheidungen hängen an `ItkLoyaltyPoints.config.*` und sind sales-channel-spezifisch.

Für Support und Debugging solltest du dir merken:

- Storefront-Controller arbeiten meist mit **SalesChannelContext**
- Webhook-/Admin-/CLI-Pfade arbeiten eher mit **Context**
- dieselbe Konfiguration kann sich je nach Sales Channel anders verhalten

Das erklärt viele „funktioniert in Kanal A, aber nicht in Kanal B“-Fälle.

5 Wichtige Services

Klasse	Aufgabe
LoyaltyTransactionService	Buchungen, Historie, Geschäftsregeln
LoyaltyPointsService	Oberflächliche Punkte-API für Storefront/Integrationen
PointsCalculatorService	Berechnung aus Bestellpositionen
LoyaltyEmailService	Versand Ablauf-Warnung

Klasse	Aufgabe
ActiveCampaignService	Club-Liste subscribe/unsubscribe
ClubAccessModalDataService	Daten für CMS-Modal
PassCreatorService / **LoyaltyIntegrationService**	PassCreator-API & Verknüpfung

---

6 Twig- und Storefront-API

Die Twig-Extension `Storefront\\Twig\\LoyaltyPointsExtension` registriert:

- `loyalty_config(context, key, default)`: liest Konfiguration für den aktuellen Sales Channel
- `loyalty_points_to_euro(context, points)`: rechnet Punkte anhand **pointsValueInCent** in Euro um
- `loyalty_is_active(context)`: prüft `ItkLoyaltyPoints.config.activate`
- Filter **floor**: simple Abrundung für Storefront-Darstellung

Sonderfall: **loyaltyTermsValidFrom** wird bei Bedarf als ISO-Datum zurückgegeben, damit das Storefront es sauber vergleichen und anzeigen kann.

---

7 Warenkorb und Einlösung

RedeemPointsCartProcessor ist als `shopware.cart.processor` mit Priorität `4500` registriert. Das ist relevant, weil die Einlösung relativ früh in der Cart-Pipeline passiert und dadurch mit anderen Warenkorb-Prozessoren kollidieren kann.

Praktisch heißt das:

- hier wird aus Konfiguration, Kundensaldo und Checkout-Zustand entschieden, ob Punkte einlösbar sind
- Einlösung läuft nicht über „einfachen Preisabzug“, sondern über echte Cart-Logik
- wenn im Checkout falsche Werte auftauchen, ist der Cart-Processor fast immer die erste Debug-Stelle

---

8 Club-Zugriff und Ausschlusslogik

Im Code existiert mit **CustomerClubProgramExclusion** eine explizite fachliche Ausschlusslogik. Sie arbeitet u. a. mit dem Custom Field:

- `custom_customer_loyalty_club_program_excluded`

Das ist wichtig, weil viele Supportfragen nicht an der Punkteberechnung scheitern, sondern schon früher: Der Kunde darf fachlich gar nicht teilnehmen oder ist bewusst vom Club-Programm ausgeschlossen. Wenn Punkte „ohne ersichtlichen Grund“ nicht vergeben oder zurückgebucht werden, immer zuerst diese Club-/Gruppenlogik mitprüfen.

---

9 Bestell- und Statuslogik

9.1 OrderPlacedSubscriber

Verarbeitet den „positiven“ Fall: Nach einer gültigen Bestellung werden Punkte vergeben oder Buchungen vorbereitet. Zusammen mit **PointsCalculatorService** und **LoyaltyTransactionService** entsteht daraus die fachliche Gutschrift.

9.2 OrderStateChangeSubscriber

Reagiert auf zwei Event-Gruppen:

- `state_machine.order.state_changed`
- `state_machine.order_transaction.state_changed`

Wichtige Regeln aus dem Code:

- **Bestellung storniert** → Gegenbuchung für bereits vergebene Punkte
- **Zahlung refunded / partially_refunded** → Reversal-Logik über Zahlungsstatus
- **Teil-Erstattung** → Punkteabzug wird anteilig berechnet
- **Reaktivierung einer stornierten Bestellung** → frühere Storno-Gegenbuchungen werden über **expiresAt** fachlich entwertet
- **Doppelabzug vermeiden** → vor einer neuen Reversal-Buchung wird geprüft, ob schon eine aktive Gegenbuchung für die Bestellung existiert

Gerade dieser Subscriber ist der zentrale Einstiegspunkt für „Punkte stimmen nach Refund/Storno nicht mehr“.

10 Storefront-Controller und Routen

Klasse	Route / Rolle
LoyaltyAccountController	**GET /account/loyalty** → frontend.account.loyalty.page
RedeemPointsController	**POST /checkout/loyalty/redeem** → frontend.checkout.loyalty.redeem
TermsAcceptanceController	Storefront-Route zur Annahme der Teilnahmebedingungen
ClubRegisterController	Storefront-Route für Club-Registrierung
ClubNewsletterController	Storefront-Route für Newsletter-/Club-Flow
ClubMembershipController	Storefront-Route zur Mitgliedschaftsverwaltung
ClubAccessModalController	Storefront-Endpunkte für das Club-Zugangsmodal
ClubRedirectController	Storefront-Redirects im Club-Kontext
LoyaltyCardsController	Storefront-Karten-/Übersichtsseiten
PassCreatorController	Wallet-Erstellung, Download und Update
PassCreatorWebhookController	**POST /api/passcreator/webhook/pass-downloaded** → api.passcreator.webhook.pass-downloaded

API (ohne Storefront-Login), ClubJoinController:**

- **POST /api/loyalty/webhook/club-join** → `api.loyalty.webhook.club-join` (Secret per Query **secret**)
- **POST /api/loyalty/webhook/club-leave** → `api.loyalty.webhook.club-leave` (gleiche Secret-Logik)

Typische Statuscodes aus **ClubJoinController**:

- `401` bei fehlendem/falschem Secret
- `400` bei leerem Payload oder fehlender E-Mail
- `404` wenn kein passender Kunde gefunden wird
- `500` wenn Fachkonfiguration fehlt oder intern etwas schiefläuft

Wichtig für Symfony-Einsteiger: **ClubJoinController** ist zwar im Plugin-Baum „Storefront/Controller“, verhält sich technisch aber eher wie ein kleiner API-/Webhook-Controller. Er liefert JSON, läuft ohne Storefront-Login und wird nicht wie eine klassische HTML-Seite verwendet.

11 Subscriber (vollständige Liste laut Codebasis)

- **OrderPlacedSubscriber**, **OrderStateChangeSubscriber**
- **CustomerRegisterSubscriber**
- **CheckoutConfirmPageSubscriber**
- **NewsletterConfirmClubGroupSubscriber**, **NewsletterConfirmRedirectSubscriber**
- **ClubAreaCategoryRedirectSubscriber**
- `PassCreator\\Subscriber\\LoyaltyPointsChangedSubscriber`, **CustomerAccountSubscriber**

(Jeweils `getSubscribedEvents()` in der Klasse prüfen.)

---

12 CMS

- **LoyaltyPointsCmsElementResolver** + **LoyaltyPointsCmsElementStruct**
- **LoyaltyConditionsCmsElementResolver** + Struct
- **ClubAccessModalCmsElementResolver** + Struct
- `PassCreator\\Core\\Content\\Cms\\PassCreatorCmsElementResolver` + Struct

---

13 Scheduled Tasks

- **LoyaltyExpiringPointsEmailTask** + **LoyaltyExpiringPointsEmailTaskHandler**
- **PassCreatorPointsSyncTask** + **PassCreatorPointsSyncTaskHandler**

Technische Task-Namen:

- `itk_loyalty_points.expiring_points_email`
- `itk_loyalty_points.passcreator.sync_points`

Registrierung in `services.xml` / **ScheduledTask**-Definition im Plugin.

Für den Betrieb bedeutet das ganz praktisch:

- Ohne laufenden **Messenger-Consumer** oder entsprechendes Hosting-Setup werden Tasks nicht abgearbeitet.
- Relevante Befehle für die Infrastruktur sind z. B. `bin/console scheduled-task:run` und `bin/console messenger:consume`.
- Wenn Ablauf-Mails oder PassCreator-Sync „einfach nicht passieren“, ist zuerst die Worker-Infrastruktur zu prüfen, nicht die Fachlogik.

---

14 Console Commands

Wichtige Commands aus `src/Command/`:

- `itk:loyalty:ac:remove`
- `itk:loyalty:welcome:award`
- `itk:loyalty:customer:points`
- `itk:loyalty:customer:adjust`
- `itk:loyalty:bulk:create`
- `itk:loyalty:bulk:delete`
- `itk:loyalty:fraud:detect`
- `itk:loyalty:expire:mark`
- `itk:loyalty:order:recalculate`
- `itk:loyalty:process-refunded-orders`
- `itk:loyalty:transaction:create`
- `itk:loyalty:transaction:update`
- `itk:loyalty:transaction:list`
- `itk:loyalty:transaction:delete`
- `itk:loyalty:transaction:export`
- `itk:loyalty:stats:overview`
- `itk:loyalty:stats:customer`
- `itk:loyalty:generate-dummy-transactions`
- `itk:loyalty:email:expiring:run`
- `itk:loyalty:passcreator:sync:run`

---

15 Admin

`Administration\\Controller\\LoyaltyCustomerController` stellt u. a. diese Admin-API-Routen bereit:

- `/api/_action/itk-loyalty-points/customer/{customerId}/transactions`
- `/api/_action/itk-loyalty-points/customer/{customerId}/statistics`
- `/api/_action/itk-loyalty-points/customer/{customerId}/transaction`
- `/api/_action/itk-loyalty-points/transaction/{transactionId}` (update / delete)
- `/api/_action/itk-loyalty-points/customer/{customerId}/adjust-points`
- `/api/_action/itk-loyalty-points/transaction/bulk`
- `/api/_action/itk-loyalty-points/customer/{customerId}/export`
- `/api/_action/itk-loyalty-points/transaction/{transactionId}/quick-action`

Damit deckt das Plugin im Admin nicht nur Anzeige, sondern auch Korrektur, Export und Bulk-Aktionen ab.

---

16 Migrationen

Unter `src/Migration/`, u. a.:

- **Migration1764000000CustomerLoyaltyTermsAcceptedAt**
- **Migration1763000000AddNoPointsPaymentProductCustomField**
- **Migration1745000000CreateExpiringPointsMailTemplateType**

---

17 Vollständige PHP-Klassenliste (`src/`)

Root: `ItkLoyaltyPoints.php`

Commands: ActiveCampaignRemoveCommand, **AwardWelcomePointsCommand**, **BulkCreateCommand**, **BulkDeleteCommand**, **CustomerAdjustCommand**, **CustomerPointsCommand**, **ExpireMarkCommand**, **FraudDetectCommand**, **GenerateDummyTransactionsCommand**, **OrderRecalculateCommand**, **ProcessRefundedOrdersCommand**, **RunExpiringPointsEmailCommand**, **RunPassCreatorSyncCommand**, **StatsCustomerCommand**, **StatsOverviewCommand**, **TransactionCreateCommand**, **TransactionDeleteCommand**, **TransactionExportCommand**, **TransactionListCommand**, **TransactionUpdateCommand**

Core / Checkout / Cart: RedeemPointsCartProcessor

Core / Checkout / Order / Subscriber: OrderPlacedSubscriber, **OrderStateChangeSubscriber**

Core / Checkout / Customer / Subscriber: CustomerRegisterSubscriber

Core / Content / Cms: ClubAccessModalCmsElementResolver, **ClubAccessModalCmsElementStruct**, **LoyaltyConditionsCmsElementResolver**, **LoyaltyConditionsCmsElementStruct**, **LoyaltyPointsCmsElementResolver**, **LoyaltyPointsCmsElementStruct**

Entity / LoyaltyTransaction: LoyaltyTransactionCollection, **LoyaltyTransactionDefinition**, **LoyaltyTransactionEntity**

Administration / Controller: LoyaltyCustomerController

Service: ActiveCampaignService, **ClubAccessModalDataService**, **LoyaltyEmailService**, **LoyaltyPointsService**, **LoyaltyTransactionService**, **PointsCalculatorService**

Storefront / Controller: ClubAccessModalController, **ClubJoinController**, **ClubMembershipController**, **ClubNewsletterController**, **ClubRedirectController**, **ClubRegisterController**, **LoyaltyAccountController**, **LoyaltyCardsController**, **RedeemPointsController**, **TermsAcceptanceController**

Storefront / Page / Account: LoyaltyPageLoader, **LoyaltyPointsExtension** (Struct)

Storefront / Page / Checkout / Confirm: CheckoutConfirmPageSubscriber

Storefront / Subscriber: ClubAreaCategoryRedirectSubscriber, **NewsletterConfirmClubGroupSubscriber**, **NewsletterConfirmRedirectSubscriber**

Storefront / Twig: LoyaltyPointsExtension

PassCreator: `Controller\\PassCreatorController`, `Controller\\PassCreatorWebhookController`, `Service\\PassCreatorService`, `Service\\LoyaltyIntegrationService`, `Subscriber\\CustomerAccountSubscriber`, `Subscriber\\LoyaltyPointsChangedSubscriber`, `Core\\Content\\Cms\\PassCreatorCmsElementResolver`, `Struct\\PassCreatorWalletCardExtension`, **Struct** für CMS-Element

ScheduledTask: **LoyaltyExpiringPointsEmailTask**, **LoyaltyExpiringPointsEmailTaskHandler**, **PassCreatorPointsSyncTask**, **PassCreatorPointsSyncTaskHandler**

Migration: alle **Migration*.php** unter `src/Migration/`

---

18 Logging und Debugging

- Logger-Datei: `var/log/loyalty_points.log`
- typischer Einstieg bei Fehlersuche:
  - falscher Punktestand → **LoyaltyTransactionService** + betroffene Transaktionen prüfen
  - Problem im Checkout → **RedeemPointsCartProcessor**
  - Problem nach Storno/Refund → **OrderStateChangeSubscriber**
  - Problem bei Mails/Sync → **ScheduledTask** + Worker + jeweiliger Handler