Fachinformatiker für Anwendungsentwicklung

Dokumentation

Trustworthy Casino

Online-Spiele Casino

Projektzeitraum: 29.01.2025 - 12.06.2025

Projektteilnehmer:
Constantin Simonis
Phan Huy Tran
Jan-Marlon Leibl
Jan Klattenhoff
Lea Ziemke
Projektaufsicht:
Katrin Deeken
Bernd Heidemann

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CI	Continuous Integration
CI/CD	Continuous Integration/Continuous Deployment
CLI	Command Line Interface
CRM	Customer Relationship Management
CRON	Vorgangsausführung gemäß geplanten Zeitabläufen für UNIX Programme
E2E	End-to-End
eCommerce	Electronic Commerce
HTTP	Hypertext Transfer Protocol
IX	Intex Fusion Pro Omnichannel CRM
JSON	JavaScript Object Notation
M2	MAGENTO 2 eCommerce Platform
NSD	NEUSTA SOFTWARE DEVELOPMENT GmbH
NXP	NEUSTA EXPERIENCE GmbH
PHP	Hypertext Preprocessor

1 Einleitung

Die folgende Projektdokumentation behandelt den Ablauf des IHK-Abschlussprojekts, welches der Autor im Rahmen seiner Ausbildung zum Fachinformatiker mit der Fachrichtung Anwendungsentwicklung durchgeführt hat. Ausbildungsbetrieb ist die NEUSTA SOFTWARE DEVELOPMENT GmbH (NSD), ein Tochterunternehmen der team neusta Unternehmensgruppe mit Sitz in Bremen. Das Kerngeschäft der NSD umfasst die Beratung, Entwicklung und Umsetzung von komplexen Software- und Electronic Commerce (eCommerce)-Lösungen und beschäftigt zur Zeit ca. 1000 Mitarbeiter.

1.1 Projektumfeld

Auftraggeber des Projektes ist die NEUSTA EXPERIENCE GmbH (NXP), ebenfalls eine Tochtergesellschaft der team neusta Unternehmensgruppe. Zu den Leistungen der NXP gehören die Beratung und Umsetzung der Bereiche Konzeption, Design und Usability sowie die Leitung von Web- und eCommerce-Projekten.

Die technische Umsetzung von Web- und eCommerce-Projekten für Kunden der NXP wird in Zusammenarbeit mit Mitarbeitern der NSD vorgenommen.

Diese Form der Zusammenarbeit erfordert intensive, regelmäßige Kommunikation und Rücksprache zwischen beiden Parteien.

1.2 Projektziel

Ziel des Projektes ist die Entwicklung einer Schnittstelle zum Importieren und Exportieren von Produkt- und Nutzerdaten zwischen dem Customer Relationship Management (CRM) Intex Fusion Pro Omnichannel CRM (IX) und einem MAGENTO 2 eCommerce Platform (M2) Webshop.

Die NSD entwickelt für Kunden der NXP spezifische Webshops und bindet diese an das vom Kunden vorgesehene CRM an. Für IX erfordert dies eine individuelle Entwicklung pro Kunden, welche mit hohem Aufwand und Kosten verbunden ist. Dieses Projekt soll einen modularen Rahmen für Importe/Exporte zwischen M2, einer Open-Source eCommerce Platform auf PHP Basis und IX abbilden, welcher in verschiedene Kundenprojekte implementiert werden kann und so die abermalige Neuentwicklung eines Import/Export Modules ablöst.

1.3 Projektbegründung

Die NSD entwickelt derzeit für jeden Kunden, der IX als CRM nutzt, individuell ein Modul zur Erweiterung des relevanten M2. Diese Module handhaben den Kunden- sowie Produktdaten-Import von IX zu M2. Die Neuentwicklungen für jeden Kunden beinhalten Schwierigkeiten in der Wartung und Fehleranalyse, da jeder Entwickler sich in das individuell entwickelte Modul einarbeiten muss. Gleichzeitig liegt hier kein Standard vor, wie mit Fehlermeldungen, die von der IX-API empfangen werden, umgegangen wird. Dies resultiert in längerer und umständlicher Wartung und schlägt dem Kunden durch Mehraufwand zu Buche, der vermieden werden kann.

Ein modularer Rahmen mit einfachen Individualisierungsmöglichkeiten für einen IX-Importer als Modul für M2 sorgt somit für eine Minderung von Neu-Entwicklungskosten und -aufwand. Eine höhere Wartbarkeit mit geringerer Einarbeitungszeit pro Kundenprojekt wird hierdurch ebenfalls garantiert.

1.4 Projektschnittstellen

Damit die Daten, die zu M2 importiert werden sollen, vollständig ermittelt und validiert werden können, muss die Anwendung mit der IX-API interagieren können. Diese Anforderung soll mit Hilfe von cURL [1] im Umfeld der PHP Entwicklung umgesetzt werden.

Um den eigentlichen Import-Vorgang der bereits geholten Daten von IX starten zu können wird zudem eine Verbindung zur Import-API von M2 benötigt. Aus diesem Grund wird die Anwendung als M2-Modul implementiert. So wird garantiert, dass Fehler im Import-Vorgang minimiert werden, weil der Fokus darauf gelegt wird, die M2 eigenen Import-Funktionen zu nutzen.

1.5 Projektabgrenzung

Da der Projektumfang beschränkt ist, soll die Entwickung der Import- und Export-Funktionen von Bestellungen und Warenkörben nicht Bestandteil dieses Abschlussprojektes sein.

2 Projektarchitektur

2.1 Überblick

Das Casino Gaming Platform Projekt folgt einer klassischen Client-Server-Architektur mit einer klaren Trennung zwischen Frontend und Backend. Diese Architektur wurde gewählt, um

[1] cURL - Client URL `https://github.com/curl/curl`

eine saubere Separation of Concerns zu gewährleisten und die Wartbarkeit sowie Erweiterbarkeit des Systems zu fördern. Die Kommunikation zwischen den beiden Schichten erfolgt über REST-APIs, die JSON-Daten austauschen.

2.2 Technologie-Stack

2.2.1 Frontend-Technologien

Für die Entwicklung der Benutzeroberfläche wurde Angular[2] als Framework gewählt. Angular bietet eine robuste Basis für Single Page Applications und ermöglicht eine komponentenbasierte Entwicklung. Als Package Manager kommt Bun[3] zum Einsatz, welcher sowohl die Paketinstallation als auch das Bundling übernimmt. Für das Styling wird Tailwind CSS[4] verwendet, welches eine konsistente und effiziente Gestaltung der Benutzeroberfläche ermöglicht.

Für die Qualitätssicherung werden E2E-Tests mit Playwright[5] durchgeführt. Diese Tests stellen sicher, dass die gesamte Anwendung aus Benutzersicht korrekt funktioniert.

2.2.2 Backend-Technologien

Das Backend basiert auf Spring Boot, einem Java-Framework, das eine schnelle Entwicklung von produktionsreifen Anwendungen ermöglicht. Spring Boot wurde gewählt, da es umfangreiche Funktionalitäten out-of-the-box bietet und sich durch eine starke Community-Unterstützung auszeichnet. Als Build-Tool kommt Gradle zum Einsatz, welches flexiblere Konfigurationsmöglichkeiten als Maven bietet.

Für die Datenpersistierung wird PostgreSQL[6] verwendet.

2.3 Systemarchitektur

2.3.1 Frontend-Architektur

Das Frontend wurde als Single Page Application (SPA) konzipiert, um eine flüssige Benutzererfahrung zu gewährleisten. Die Architektur folgt Angulars modularem Ansatz und gliedert sich in verschiedene Bereiche: Feature-Module organisieren die Funktionalitäten nach Geschäftsbereichen wie Spiele und Einzahlungen. Wiederverwendbare UI-Komponenten wurden in einem Shared-Bereich zusammengefasst, um Code-Duplikation zu vermeiden.

[2] Angular - https://angular.io/
[3] Bun - https://bun.sh/
[4] Tailwind CSS - https://tailwindcss.com/
[5] Playwright - https://playwright.dev/
[6] PostgreSQL - https://www.postgresql.org/

Services übernehmen die Kommunikation mit dem Backend und kapseln die Geschäftslogik. HTTP-Interceptors behandeln globale Aspekte wie Fehlerbehandlung zentral.

2.3.2 Backend-Architektur

Das Backend implementiert eine klassische mehrschichtige Architektur, die eine klare Trennung der Verantwortlichkeiten gewährleistet. Die Controller-Schicht stellt die REST-API-Endpunkte bereit und behandelt HTTP-Anfragen. Die Service-Schicht enthält die Geschäftslogik und orchestriert verschiedene Use Cases.

Die Repository-Schicht abstrahiert den Datenzugriff und verwendet Spring Data JPA für die Kommunikation mit der Datenbank. Entity-Klassen repräsentieren die Domain-Modelle und bilden die Datenbankstrukturen ab.

2.4 Datenarchitektur

Die Datenbank folgt einem relationalen Design mit klar definierten Entitätsbeziehungen. Das Schema gliedert sich in mehrere Hauptbereiche: Der User Management Bereich verwaltet Benutzerkonten und Benutzerprofile. Spielbezogene Daten wie Spielstände, Wetten und Ergebnisse werden in separaten Tabellen gespeichert, um die Integrität der Spiellogik zu gewährleisten.

Der Financial Bereich behandelt alle monetären Transaktionen, Guthaben und Einzahlungen mit entsprechenden Audit-Trails für Compliance-Zwecke. Das Loot System verwaltet Lootboxen, deren Belohnungen und die zugehörigen Wahrscheinlichkeitsverteilungen, wobei besonderer Wert auf Transparenz und Fairness gelegt wird.

2.5 Deployment-Strategie

Die Deployment-Strategie wurde so konzipiert, dass sie sowohl lokale Entwicklung als auch produktive Umgebungen unterstützt. Für die lokale Entwicklung wird Docker Compose[7] eingesetzt, welches eine konsistente Entwicklungsumgebung über verschiedene Entwicklerrechner hinweg gewährleistet.

Frontend und Backend können unabhängig voneinander gebaut und deployed werden, was eine flexible Entwicklung und Wartung ermöglicht. Diese Entkopplung erlaubt es verschiedenen Teams, parallel zu arbeiten, ohne sich gegenseitig zu blockieren. Verschiedene Umgebungskonfigurationen durch Profile stellen sicher, dass entwicklungs- und produktionsspezifische Einstellungen sauber getrennt sind und automatisiert angewendet werden können.

[7] Docker Compose - https://docs.docker.com/compose/

3 Continuous Integration

Das Projekt verwendet Gitea Actions[8] als Continuous Integration/Continuous Deployment (CI/CD)-Pipeline, welche vollständig kompatibel mit GitHub Actions ist. Entsprechend den Qualitätsanforderungen soll eine hohe Code-Qualität durch automatisierte Tests gewährleistet werden.

3.1 Aufbau der CI-Pipeline

Die Haupt-Continuous Integration (CI)-Pipeline wird durch die Datei `ci.yml` definiert und bei Pull Requests ausgelöst. Aufgrund der separaten Frontend- und Backend-Komponenten wurde eine *Change Detection* implementiert, welche nur relevante Tests für geänderte Bereiche ausführt.

Ein initialer Job identifiziert geänderte Dateien und ermöglicht eine selektive Ausführung:

- Backend-Änderungen: `backend/**`
- Frontend-Änderungen: `frontend/**`
- Workflow-Änderungen: `.gitea/workflows/**`

3.1.1 Backend-Qualitätssicherung

Für Backend-Änderungen werden folgende Prüfungen durchgeführt:

- **Unit Tests:** Ausführung mit `./gradlew test` in OpenJDK 23 Container
- **Checkstyle:** Code-Style-Validierung mit Caching-Mechanismus
- **Docker Build:** Überprüfung der Build-Funktionalität

3.1.2 Frontend-Qualitätssicherung

Für Frontend-Änderungen wird eine umfassende Testsuite ausgeführt:

- **ESLint:** Code-Qualitätsprüfung mit `bun run lint`
- **Prettier:** Code-Formatierungsvalidierung
- **Build-Test:** Produktions-Build-Validierung mit `bun run build`
- **Playwright End-to-End (E2E) Tests:** End-to-End-Tests mit automatischem Backend-Start

[8] Gitea Actions - https://docs.gitea.com/usage/actions/overview

- **Docker Build:** Validierung der Container-Erstellung

3.2 Release-Management

Das Release-Management erfolgt automatisiert durch die `release.yml` Pipeline bei Pushes auf den `main`-Branch. Die Implementierung folgt *Semantic Versioning*[9] und *Conventional Commits*[10].

Die Release-Pipeline umfasst:

1. **Semantic Release:** Automatische Versionierung basierend auf Commit-Nachrichten
2. **Docker Image Build:** Parallele Erstellung von Backend- und Frontend-Images
3. **Registry Push:** Upload zur privaten Gitea Docker Registry

Die CI/CD-Pipeline implementiert Performance-Optimierungen wie intelligentes Caching, Concurrency Control und selektive Job-Ausführung. Diese Automatisierung gewährleistet eine hohe Software-Qualität bei effizienten Entwicklungsprozessen.

[9] Semantic Versioning - https://semver.org/
[10] Conventional Commits - https://www.conventionalcommits.org/

4 Dice

4.1 Was ist Dice?

Das Würfelspiel 'Dice' ist ein originelles Spiel der Casinoplattform Stake.com[11]. Das Spiel dreht sich um einen virtuellen 100-seitigen Würfel, bei dem Spieler die Parameter ihrer Wette beeinflussen können. Im Kern geht es darum, einen zuvor festgelegten 'Roll Over'- oder 'Roll Under'-Betrag zu unter- oder überschreiten, um eine Runde zu gewinnen. Spieler haben die Kontrolle über den Multiplikator und ihre Gewinnchancen: Durch die Anpassung des Zielwerts können sie das Verhältnis von Risiko und potenzieller Auszahlung steuern. Ein höherer Multiplikator verspricht zwar größere Gewinne, reduziert jedoch gleichzeitig die Wahrscheinlichkeit eines erfolgreichen Würfelwurfs.

4.1.1 Zufallszahlengenerierung

Zur Generierung des Würfelwurfs verwendet diese Implementierung die Standardklasse java.util.Random. Sie erzeugt eine pseudo-zufällige Zahl zwischen 1 und 100 (inklusive), die das Ergebnis des virtuellen 100-seitigen Würfels darstellt.

4.1.2 Spielablauf und Datenfluss

Der zentrale Controller steuert den Spielablauf und empfängt die Anfragen vom Frontend. Jede Anfrage enthält die Eckdaten des gewünschten Würfelwurfs:

- **Einsatz:** Der gesetzte Münzbetrag.
- **Wettart:** Soll der Würfel "über" oder "unter" einen Wert fallen?
- **Zielwert:** Der vom Spieler festgelegte Referenzwert (1-100).

Zuerst prüft der Controller das Guthaben des Spielers. Bei unzureichenden Mitteln wird der Vorgang abgelehnt. Andernfalls übergibt er die weitere Ausführung an die Dienstklasse.

Die Dienstklasse übernimmt die eigentliche Logik des Würfelspiels:

1. Zieht den Einsatz vom Spielerkonto ab.
2. Erzeugt einen zufälligen Würfelwurf (Wert zwischen 1 und 100).
3. Prüft, ob der Wurf die Gewinnbedingung erfüllt (entsprechend Wettart und Zielwert).
4. Berechnet die Gewinnwahrscheinlichkeit und den sich daraus ergebenden Multiplikator.

[11] Stake.com ist eine bekannte Online-Glücksspielplattform, die eine Vielzahl von Casinospielen und Sportwetten anbietet.

5. Schreibt bei einem Gewinn den entsprechenden Betrag (Einsatz $\times$ Multiplikator) dem Spielerkonto gut.

Das Ergebnis des Spiels wird an das Frontend zurückgesendet und enthält:

- Gewinnstatus (gewonnen/verloren)
- Auszahlungsbetrag
- Gewürfelten Wert

5 Coinflip

5.1 Was ist Coinflip?

Das Münzwurf-Spiel 'Coinflip' ist ein klassisches Glücksspiel, das in seiner digitalen Umsetzung den traditionellen Münzwurf simuliert. Das Spiel basiert auf dem einfachen Prinzip einer Münze mit zwei Seiten: Kopf und Zahl. Spieler setzen auf eine der beiden Seiten und haben eine 50%-ige Gewinnchance. Die Einfachheit des Spiels macht es zu einem idealen Einstiegsspiel für neue Nutzer der Casino-Plattform.

Im Gegensatz zu komplexeren Spielen wie Dice bietet Coinflip eine feste Gewinnwahrscheinlichkeit von 50% und einen konstanten Multiplikator von 2x. Dies bedeutet, dass Spieler bei einem Gewinn ihren Einsatz verdoppeln, während sie bei einer Niederlage ihren gesamten Einsatz verlieren.

5.1.1 Zufallszahlengenerierung

Die Implementierung verwendet die Standardklasse java.util.Random zur Generierung des Münzwurfs. Die Zufallsgenerierung erzeugt einen booleschen Wert, der anschließend einer der beiden Münzseiten zugeordnet wird. Diese binäre Entscheidung gewährleistet die faire 50:50-Verteilung, die für ein authentisches Münzwurf-Erlebnis erforderlich ist.

5.1.2 Spielablauf und Datenfluss

Der Spielablauf von Coinflip folgt einem strukturierten Datenfluss zwischen Frontend und Backend. Der Controller empfängt die Spielanfrage mit folgenden Parametern:

- **Einsatz:** Der gesetzte Münzbetrag.
- **Gewählte Seite:** Die vom Spieler gewählte Münzseite (Kopf oder Zahl).

Nach dem Erhalt der Anfrage führt der Controller eine Guthabenprüfung durch. Bei ausreichendem Guthaben wird die Anfrage an die Service-Schicht weitergeleitet, andernfalls wird eine entsprechende Fehlermeldung zurückgegeben.

Die Service-Klasse verarbeitet die Spiellogik in folgender Reihenfolge:

1. Abbuchung des Einsatzes vom Spielerkonto.
2. Generierung des zufälligen Münzwurfs (Kopf oder Zahl).
3. Vergleich zwischen gewählter Seite und Wurfergebnis.
4. Bei einem Gewinn: Gutschrift des doppelten Einsatzes auf das Spielerkonto.

5. Rückgabe des Spielergebnisses an das Frontend.

Das Spielergebnis wird strukturiert an das Frontend übermittelt und enthält:

- Gewinnstatus (gewonnen/verloren)
- Auszahlungsbetrag (bei Gewinn: 2x Einsatz)
- Geworfene Münzseite

Literaturverzeichnis

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase			**6 h**
1. Analyse des Ist-Zustands		2 h	
2. „Make or buy“-Entscheidung und Wirtschaftlichkeitsanalyse		1 h	
3. Erstellen eines „Use-Case“-Diagramms		2 h	
4. Erstellen des Lastenhefts		1 h	
Entwurfsphase			**14 h**
1. UML-Komponentendiagramm erstellen		1 h	
2. Datenbankentwurf		1 h	
2.1. M2 Migration für Tabelle definieren	1 h		
3. Erstellen des Schnittstellen Konzeptes für IX		4 h	
3.1. Erstellung Verbindung zur IX API	1 h		
3.2. Verarbeitung der JSON-Daten	1 h		
3.3. Verarbeitung der Konfigurationsdatei der Anwendung	2 h		
4. Benutzeroberflächen entwerfen und abstimmen		3 h	
5. Erstellen des Schnittstellen Konzeptes für M2		4 h	
5.1 Erstellung Schnittstelle für M2 Import Funktionen		2 h	
5.2 Erstellung Schnittstelle zur Verarbeitung der geholten IX JSON-Daten		2 h	
Implementierungsphase			**37 h**
1. Anlegen der Datenbank		4 h	
2. Einrichtung der Docker Umgebung zur lokalen Entwicklung		2 h	
3. Programmierung der PHP-Module für die Funktionen		29 h	
3.1 Erstellung eines Client Moduls zur Abfrage der IX API per **curl!** Requests	3 h		
3.2. Import der Produktdaten aus JSON-Dateien	2 h		
3.3. Import der Nutzerdaten aus JSON-Dateien	2 h		
3.4. Aufbereitung (Mapping) der geholten Daten für M2	3 h		
Einbindung der Konfigurationsdateien	4 h		
3.6. Import der aufbereiteten Daten von IX zu M2	5 h		
3.7. Einbindung der CRON-Job Möglichkeit	3 h		
3.8. Abdeckung der Module mit Unit-Tests	4 h		
4. Umsetzung der CLI Benutzeroberfläche		2 h	
Abnahmetest der Fachabteilung			**3 h**
1. Abnahmetest der NSD Mitarbeiter		3 h	
Einführungsphase			**1 h**
1. Einführung/Benutzerschulung		1 h	
Erstellen der Dokumentation			**9 h**
1. Erstellen der Entwicklerdokumentation		2 h	
2. Erstellen der Projektdokumentation		7 h	
Gesamt			**70 h**

A.2 Lastenheft (Auszug)

Im folgenden Auszug des Lastenheftes werden die Anforderungen definiert, die an die neu entwickelte Anwendung gestellt werden.

Die Anwendung muss folgende Anforderungen erfüllen:

1. Verarbeitung der empfangenen Daten von IX
   1.1. Die Anwendung muss Produkt- und Nutzerdaten von der IX-API abrufen können.
   1.2. Die geholten Daten sollen temporär zwischengespeichert werden, um den Import zu M2 von der Datenbeschaffung abzukapseln
   1.3. Die gespeicherten Daten werden getrennt von der IX-API an die M2 Import-Funktionen gegeben und so in das M2 System integriert
2. Sonstige Anforderungen
   2.1. Die Anwendung muss ohne das Installieren einer zusätzlichen Software über das Terminal erreichbar sein.
   2.2. Die Import der Daten muss jede Nacht bzw. nach jedem Fehlschlag automatisch aktualisiert werden.
   2.3. Die Anwendung soll jederzeit erreichbar sein.
   2.4. Da sich die Entwickler auf die Anwendung verlassen, muss diese korrekte Daten liefern und darf keinen Interpretationsspielraum lassen.
   2.5. Die Anwendung muss so flexibel sein, dass sie bei Änderungen im Entwicklungsprozess einfach angepasst werden kann.

A.3 Verwendete Ressourcen

Hardware

- Büroarbeitsplatz (Schreibtisch, ergonomischer Stuhl)
- Fujitsu Lifebook U-Series - Notebook
- Peripheriegeräte für Notebook (Tastatur, Maus)

Software

- Debian Derivat Ubuntu 18.04 - Betriebssystem
- JetBrains PHPStorm - Entwicklungsumgebung PHP
- git - Verteilte Versionsverwaltung
- Docker - Open Source Management von virtuellen Maschinen (Containerbasiert)
- mySQL - Open Source relationelles Datenbank Management System
- Composer - Open Source Package Management System auf Anwendungsebene für PHP
- Magento 2 Community Edition - Open Source eCommerce Platform
- TeXify IDEA - Open Source Plugin für PHPStorm als Distribution des Textsatzsystems TeX
- PHPUnit - Framework zur Durchführung von Unit-Tests
- LaTeX-Vorlage zur IHK-Projektdokumentation für Fachinformatiker Anwendungsentwicklung von Stefan Macke - `http://fiae.link/LaTeXVorlageFIAE`

Personal

- Mitarbeiter Softwareentwicklung der NSD - Festlegung der Anforderungen, Abnahme des Projektes & Review des Codes
- Auszubildender Fachinformatiker als Entwickler - Umsetzung des Projektes

A.4 Use Case-Diagramm

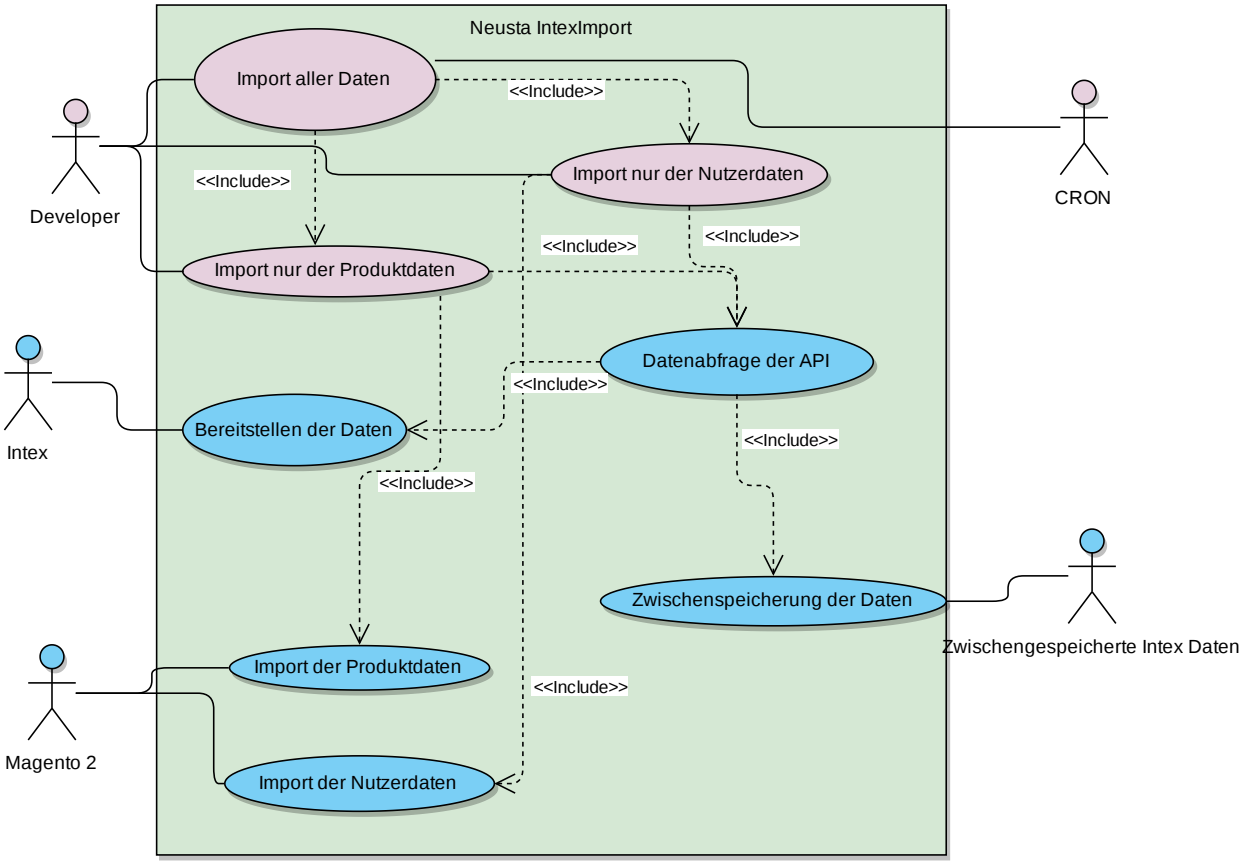

Abbildung 1: Use Case-Diagramm

A.5 Amortisation

Der Zeitpunkt der Amortisation wird als Schnittpunkt der beiden Geraden angegeben.

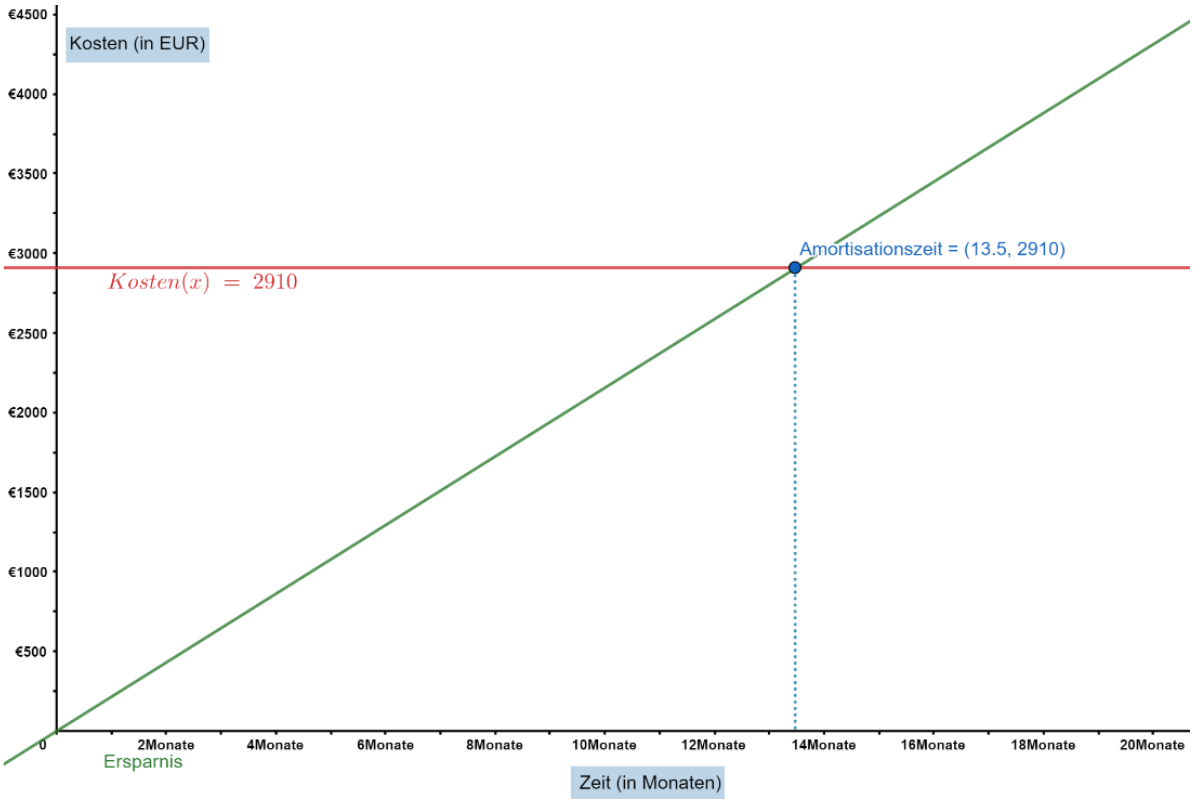

Abbildung 2: Grafische Darstellung der Amortisation

A.6 composer.json Konfiguration für neusta-m2-intex-client

```
{
  "name": "neusta/m2-intex-client",
  "license": "proprietary",
  "description": "Client for Magento2 Intex Import/Export",
  "type": "magento2-module",
  "version": "0.0.1",
  "require": {
    "php": "~7.1.0|~7.2.0|~7.3.0",
    "magento/framework": "~101.0",
    "neusta/m2-intex-base": "0.0.1",
    "guzzlehttp/guzzle": "6.3.*"
  },
  "autoload": {
    "files": [
      "registration.php"
    ],
    "psr-4": {
      "Neusta\\IntexClient\\": "src"
    }
  }
}
```

Listing 1: Konfiguration für neusta-m2-intex-client

A.7 Deklaration zur Anlage einer SQL Tabelle im Magento 2 Umfeld

```
<?xml version="1.0" ?>
<schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
      urn:magento:framework:Setup/Declaration/Schema/etc/schema.xsd">
    <table name="neusta_intex_import" resource="default" engine="innodb"
           comment="Neusta IntexImport Logging Table">
        <column xsi:type="int" name="value_id" padding="11" unsigned="false" nullable="false" identity="true"
                comment="Value ID"/>
        <column xsi:type="smallint" name="attribute_id" padding="5" unsigned="true" nullable="false" identity="
                false" default="0" comment="Attribute ID"/>
        <column xsi:type="smallint" name="store_id" padding="5" unsigned="true" nullable="false" identity="false
                " default="0" comment="Store ID"/>
        <column xsi:type="int" name="entity_id" padding="10" unsigned="true" nullable="false" identity="false"
                default="0" comment="Entity ID"/>
        <column xsi:type="datetime" name="value" on_update="false" nullable="true" comment="Value"/>
        <constraint xsi:type="primary" referenceId="PRIMARY">
            <column name="value_id"/>
        </constraint>
        <constraint xsi:type="foreign" referenceId="CAT_PRD_ENTT_DTIME_ATTR_ID_EAV_ATTR_ATTR_ID"
                table="catalog_product_entity_datetime" column="attribute_id" referenceTable="eav_attribute"
                referenceColumn="attribute_id" onDelete="CASCADE"/>
        <constraint xsi:type="foreign" referenceId="
                CAT_PRD_ENTT_DTIME_ENTT_ID_CAT_PRD_ENTT_ENTT_ID" table="
                catalog_product_entity_datetime" column="entity_id" referenceTable="catalog_product_entity"
                referenceColumn="entity_id" onDelete="CASCADE"/>
        <constraint xsi:type="foreign" referenceId="
                CATALOG_PRODUCT_ENTITY_DATETIME_STORE_ID_STORE_STORE_ID" table="
                catalog_product_entity_datetime" column="store_id" referenceTable="store" referenceColumn="
                store_id" onDelete="CASCADE"/>
        <constraint xsi:type="unique" referenceId="
                CATALOG_PRODUCT_ENTITY_DATETIME_ENTITY_ID_ATTRIBUTE_ID_STORE_ID">
            <column name="entity_id"/>
            <column name="attribute_id"/>
            <column name="store_id"/>
        </constraint>
        <index referenceId="CATALOG_PRODUCT_ENTITY_DATETIME_ATTRIBUTE_ID" indexType="btree">
            <column name="attribute_id"/>
        </index>
        <index referenceId="CATALOG_PRODUCT_ENTITY_DATETIME_STORE_ID" indexType="btree">
            <column name="store_id"/>
        </index>
    </table>
</schema>
```

Listing 2: Deklaration zur Anlage einer SQL Tabelle im Magento 2 Umfeld

A.8 Klasse: Factory

```php
<?php declare(strict_types=1);

namespace Neusta\IntexClient\Connection;

use Neusta\IntexClient\Connection\Model\Connection;

use function class_exists;
use function sprintf;
use function ucwords;
use function strtolower;

class Factory
{
    public static function create(string $connectionType): Connection
    {
        $connectionPath = sprintf(
            '\Neusta\IntexClient\Model\Connection\%sConnection',
            ucwords(strtolower($connectionType))
        );

        if (class_exists($connectionPath)) {
            return new $connectionPath();
        }

        throw new \RuntimeException('Invalid Connection Type given');
    }
}
```

Listing 3: Klasse: Factory

A.9 Klasse: CustomerConnection

```php
<?php declare(strict_types=1);

namespace Neusta\IntexClient\Connection\Model;


use Neusta\IntexClientConfig\Model\Config\Config;
use Neusta\IntexClientConfig\Provider\JsonConfigProvider;
use Psr\Log\LoggerInterface;

use function json_decode;

class CustomerConnection extends Connection
{
    /**
     * @var Config
     */
    private $customerConfig;

    public function __construct(JsonConfigProvider $configProvider, LoggerInterface $logger)
    {
        parent::__construct($configProvider, $logger);
        $this->customerConfig = $configProvider->getConfig('customer');
    }

    public function loadCustomerData(): array
    {
        $response = $this->createGet(
            $this->customerConfig->getCustomerUri()
        );

        return json_decode($response->getBody(), true);
    }
}
```

Listing 4: Klasse: CustomerConnection

A.10 Klasse: Connection

```php
<?php declare(strict_types=1);

namespace Neusta\IntexClient\Connection\Model;


use GuzzleHttp\Exception\RequestException;
use Psr\Http\Message\ResponseInterface;
use Psr\Log\LoggerInterface;
use Neusta\IntexClientConfig\Model\Config\Config;
use Neusta\IntexClientConfig\Provider\JsonConfigProvider;
use \GuzzleHttp\Client as GuzzleHttpClient;
use \GuzzleHttp\Event\BeforeEvent;
use \GuzzleHttp\Event\CompleteEvent;

use function sprintf;

abstract class Connection
{
    protected const HTTP_STATUS_CODE_OK = 200;

    /**
     * @var GuzzleHttpClient
     */
    protected $httpClient;

    /**
     * @var Config
     */
    protected $serverConfig;

    /**
     * @var LoggerInterface
     */
    protected $logger;

    public function __construct(JsonConfigProvider $configProvider, LoggerInterface $logger)
    {
        $this->logger = $logger;
        $this->serverConfig = $configProvider->getConfig('server');
        $this->httpClient = new GuzzleHttpClient(
            $this->getHttpClientOptions()
        );

        $this->initHttpEventEmitter();
    }

    protected function authenticate(): bool
    {
        $result = false;

```

```
51          $response = $this->createPost(
52              $this->serverConfig->getAuthUri()
53          );
54
55          $statusCode = $response->getStatusCode();
56
57          if ($statusCode === self::HTTP_STATUS_CODE_OK) {
58              $this->logger->info('authentication successful');
59              $result = true;
60          } else {
61              $this->logger->info(
62                  sprintf('authentication failed with Code: %s', $statusCode)
63              );
64          }
65
66          return $result;
67      }
68
69      protected function createPost(string $uri, array $options = []): ?ResponseInterface
70      {
71          $result = null;
72          $request = $this->httpClient->postAsync($uri, $options);
73
74          $request->then(
75              static function (ResponseInterface $response) {
76                  $result = $response;
77              },
78              function (RequestException $exception) {
79                  $this->logger->info(
80                      sprintf('Crucial Error: %s \n Trace: %u', $exception->getMessage(), $exception->
                            getTraceAsString())
81                  );
82              }
83          );
84
85          return $result;
86      }
87
88      protected function createGet(string $uri, array $options = []): ?ResponseInterface
89      {
90          $result = null;
91          $request = $this->httpClient->getAsync($uri, $options);
92
93          $request->then(
94              static function (ResponseInterface $response) {
95                  $result = $response;
96              },
97              function (RequestException $exception) {
98                  $this->logger->info(
99                      sprintf('Crucial Error: %s \n Trace: %u', $exception->getMessage(), $exception->
                            getTraceAsString())
100                 );
```

```
101             }
102         );
103
104         return $result;
105     }
106
107     protected function initHttpEventEmitter(): void
108     {
109         // initialize the EventEmitters to get a good Logging whether the Request was send
110         // or failed
111         $this->httpClient->getEmitter()->on('before', static function (BeforeEvent $event) {
112             $this->logger->info(
113                 sprintf('About to send Request: %s', $event->getRequest())
114             );
115         });
116
117         $this->httpClient->getEmitter()->on('complete', static function (CompleteEvent $event) {
118             $this->logger->info(
119                 sprintf('Request %s finished', $event->getRequest())
120             );
121         });
122
123         $this->httpClient->getEmitter()->on('error', static function (ErrorEvent $event) {
124             $this->logger->info(
125                 sprintf('Request %s failed', $event->getRequest())
126             );
127         });
128     }
129
130     protected function getHttpClientOptions(): array
131     {
132         // Configure Base URL for all ongoing Requests and set Cookies to true,
133         // so we can handle the auth cookie over and over again without using authenticate() multiple times
134         return [
135             'base_url' => $this->serverConfig->getBaseUrl(),
136             'cookies' => true
137         ];
138     }
139 }
```

Listing 5: Abstrakte Klasse: Connection

A.11 Klasse: CustomerDataController

```php
<?php declare(strict_types=1);


namespace Neusta\IntexCustomer\Controller;


use Neusta\IntexClient\Connection\Model\CustomerConnection;

class CustomerDataController
{
    /**
     * @var CustomerConnection
     */
    private $connection;

    public function __construct(\Neusta\IntexClient\Connection\Factory $connectionFactory, \Neusta\
        IntexCustomer\Handler\DataHandler $dataHandler)
    {
        $this->connection = $connectionFactory->create('customer');
        $this->customerDataHandler = $dataHandler;
    }

    public function getCustomerData(): void
    {
        $customerData = $this->connection->loadCustomerData();

        $this->customerDataHandler->saveCustomerData($customerData);
    }
}
```

Listing 6: Klasse: CustomerDataController

A.12 UnitTest: FactoryTest

```php
<?php declare(strict_types=1);

use PHPUnit\Framework\TestCase;
use Neusta\IntexClient\Connection\Factory;
use Neusta\IntexClient\Connection\Model\CustomerConnection;

class FactoryTest extends TestCase
{
    /**
     * @var Factory
     */
    private $subject;

    public function setUp()
    {
        $this->subject = new Factory();
    }

    /**
     * @test
     */
    public function createMustReturnInstanceOfCustomerConnectionWithValidIdentifierGiven()
    {
        $expectedInstance = $this->subject->create('customer');

        self::assertInstanceOf(CustomerConnection::class, $expectedInstance);
    }

    /**
     * @test
     * @throws RuntimeException
     */
    public function createMustThrowRuntimeExceptionIfInvalidIdentifierIsGiven()
    {
        $unexpectedInstance = $this->subject->create('fooBar');

        self::expectException(RuntimeException::class);
    }
}
```

Listing 7: Unit Test der Klasse: Factory