CaaS – Cart as a Service

Stellen Sie sich folgendes Beispiel vor: Sie sind Hobby-Fotograf*in und haben eine Website entwickelt, auf der ihre besten Fotos ausgestellt werden. Weil Sie merken, dass die Zugriffszahlen steigen, möchten Sie versuchen, ihr Hobby zu Geld zu machen: Besucher*innen sollen die Fotos gegen Bezahlung hochauflösend herunterladen und für den eigenen Bedarf (Wandbild, Plakat, etc.) verwenden können.

Es gibt zahlreiche hervorragende All-In-One-Lösungen am Markt, um eine Website inkl. Shop zu betreiben (als Name unseres Projekts war übrigens auch *kauFHaus hagenberg* im Gespräch, die Marke ist derzeit aber etwas negativ besetzt). Der Nachteil dieser Komplettlösungen: Man bindet sich sehr stark an den jeweiligen Anbieter und dessen Content-Management-System. Manchmal möchte man aber eigentlich nur eine bestehende Website um eine Shop-Funktionalität erweitern – und hier kommt CaaS ins Spiel.

Anstatt selbst ein Backend zu betreiben, entsprechende APIs zu entwickeln und einen Zahlungsanbieter auszuwählen, nützen Sie das vielversprechende Angebot von CaaS: einem Warenkorb „as a Service“. CaaS liefert eine API, mit der Warenkörbe erstellt, befüllt und bestellt werden können. Sie kümmern sich nur um das Frontend ihres Shops, CaaS kümmert sich um die Persistierung der Warenkörbe, die Preisberechnung und den Checkout-Prozess.

Im Zuge des SWK5/WEA5-Projekts entwickeln Sie nun genau diese CaaS-Lösung, um Hobby-Fotograf*innen (oder ganz anderen Websites) eine einfache Integration eines Warenkorbs zu ermöglichen.

Funktionale Anforderungen

Das Softwaresystem erfüllt im Wesentlichen folgende Aufgaben:

- Das Backend ermöglicht den Betrieb von mehreren voneinander isolierten Shops. Jede Shop-Betreiber*in führt seinen eigenen Produktkatalog und hat einen eigenen Kundenkreis. Die Shop-Betreiber*innen nutzen das Backend, um die erforderlichen Stamm- und Bewegungsdaten zu verwalten. Desweitern stellt das Backend typische Geschäftslogik zur Verfügung, die beim Betrieb eines Shops benötigt wird (z. B. Verwaltung von Warenkörben, Rabattregeln etc.).
- Jede Shop-Betreiber*in erstellt individuell eine Web-Anwendung für die Interaktion seiner Kund*innen mit dem Shop (z. B. der „Fotografie-Shop“). Für die Warenkorb-Funktionalität nützt diese Web-Anwendung das CaaS-Backend und bezieht von dort die erforderlichen Daten.
- Den Shop-Betreiber*innen steht eine Web-Anwendung zur Verfügung, über die sie gewisse Stammdaten verwalten und Statistiken visualisieren können.

Das zu entwickelnde Softwaresystem besteht aus den folgenden Komponenten:

- *CaaS.Core* ist die zentrale Komponente, welche die Geschäftslogik beinhaltet und für die Datenverwaltung zuständig ist. Sie stellt die von *CaaS.Web* benötigte Funktionalität zur Verfügung und kapselt den Zugriff auf die Datenbank. Auch die flexible Verwaltung von Rabattregeln und die Anwendung dieser Regeln bei der Preisberechnung gehören zu den Aufgaben dieser Komponente.

- *CaaS.Api* exportiert die von *CaaS.Web* benötigte Funktionalität von *CaaS.Core* in Form eines REST-basierten Web-Service.
- *CaaS.Web* besteht aus den Komponenten *CaaS.Shop* und *Caas.Admin*.
  - *CaaS.Shop* stellt die Implementierung eines (beispielhaften) Shops dar, der über *CaaS.Api* mit *CaaS.Core* kommuniziert.
  - Über *CaaS.Admin* kann ein Shop administriert werden. Die wichtigste Aufgabe dieser Komponente ist aber die grafische Visualisierung der in *CaaS.Core* gesammelten Daten.

(a) Vorbemerkungen

Zum besseren Verständnis dieser Spezifikation, werden einige der nachfolgend häufig verwendeten Begriffe näher erläutert.

(a.1) *Shop-Betreiber*in (Mandant/Tenant):* CaaS unterstützt die Verwaltung von mehreren Shops. Kund*innen, Produkte und Warenkörbe sind einem Shop zugeordnet. Für jeden Shop ist zumindest eine Bezeichnung zu speichern (z. B. Amazon, e-tec, …). Die Daten der verschiedenen Shops sind sauber voneinander zu trennen, können aber in einer gemeinsamen Datenbank gehalten werden. Jeder Shop-Betreiber*in ist genau ein Shop zugeordnet.

(a.2) *Kund*in (Customer):* Unter Kund*in ist eine Kund*in eines Shops und nicht eine Kund*in von *CaaS* – welche einer Shop-Betreiber*in entspricht – zu verstehen. Für Kund*innen ist deren Name und E-Mail-Adresse zu speichern.

(a.3) *Produkt (Product):* Für Produkte sind eine Bezeichnung, eine Kurzbeschreibung, ein Download-Link sowie der aktuelle Preis zu speichern. Sie können davon ausgehen, dass in einem Shop ausschließlich digitale Produkte, also Software, PDFs, Audiodateien, Lizenzen u. dgl. angeboten werden. Da digitale Produkte beliebig duplizierbar sind, kann die Lagerverwaltung (Lagerstand) entfallen.

(a.4) *Warenkorb (Cart):* In Warenkörben können Shop-Kund*innen eingekaufte Produkte ablegen. Die Kund*in kann angeben, wie viel Stück eines Produkts er oder sie bestellen möchte. Jedem Warenkorb ist ein eindeutiger Schlüssel zuzuweisen, über den er vom Frontend angesprochen werden kann.

(a.5) *Bestellung (Order):* Sobald die Kund*in den Einkauf bestätigt, wird aus dem Warenkorb eine Bestellung. Alle für die Rechnungslegung relevanten Daten (Datum der Bestellung, bestellte Produkte, Einzelpreise und Anzahl der Produkte, Summe der Rabatte) sind zu speichern.

(a.6) Gutschein (Coupon): Kund*innen, die im Besitz eines Gutscheins sind, können diesen bei einem Einkauf einlösen. Mit einem Gutschein ist ein fixer Betrag verbunden und einem Shop zugeordnet. Es ist zu gewährleisten, dass ein Gutschein nur einmal eingelöst wird.

(a.7) *App-Key:* Für die Durchführung von Operationen zur Administration eines Shops ist ein App-Key notwendig. Jedem Shop ist ein eindeutiger App-Key zugeordnet, den *CaaS.Admin* bei jedem Request mitsenden muss.

(b) CaaS.Core

CaaS.Core ist die zentrale Stelle im System, in der die Daten aller Shops verarbeitet, gespeichert und für das Frontend (*CaaS.Web*) aufbereitet werden. *Caas.Core* ist zusammen mit *CaaS.Api* so zu gestalten, dass der Frontendentwickler so gut wie möglich bei der Implementierung eines Shops unterstützt wird.

(b.1) *Produktverwaltung:* Die Shop-Betreiber*in kann neue Produkte hinzufügen und den Preis eines Produkts anpassen. Das Löschen eines Produkts soll zur Folge haben, dass es in Suchabfragen nicht mehr berücksichtigt wird und nicht weiter bestellt werden kann. Es soll jedoch keinen Einfluss auf bestehende Referenzen in Warenkörben und Bestellungen haben. Auf Basis eines Filterkriteriums soll auf effiziente Art und Weise eine Liste von Produkten erstellt werden, auf die dieses Kriterium zutrifft. In die Suche sind die Bezeichnung und die Kurzbeschreibung einzubeziehen.

(b.2) *Warenkorbverwaltung:* Der Warenkorb ist das zentrale Element eines jeden Shops. Für alle Aktivitäten von Kund*innen sind entsprechende Funktionen vorzusehen. So soll es möglich sein, einen neuen Warenkorb zu erstellen, Produkte mit einer bestimmten Menge zum Warenkorb hinzuzufügen, die Menge der im Warenkorb befindlichen Produkte anzupassen oder ein Produkt vollständig aus dem Warenkorb zu entfernen. Der Status des Warenkorbs muss nach jeder Aktion persistiert werden. Der Status des Warenkorbs muss zu jedem Zeitpunkt abgefragt werden können.

Abschließend können alle Artikel im Warenkorb bestellt oder der Warenkorb verworfen werden (Checkout). Bei einer Bestellung werden die relevanten Daten des Warenkorbs (Datum der Bestellung, bestellte Produkte, Einzelpreise und Anzahl der Produkte, Summe der Rabatte) gespeichert und der Kund*in eine Bestellnummer übermittelt, mit welcher diese auf die Bestelldaten zugreifen kann. Nach der Durchführung der Bestellung sind keine Änderungen am Warenkorb mehr möglich.

Erst beim Checkout müssen die für die Bestellung erforderlichen Kund*innen-Daten (Name, E-Mail-Adresse) erfasst werden.

Für Kund*innen sind auch die erforderlichen Kreditkartendaten zu erfassen.

Im Zuge der Bestellung ist unter Einbeziehung des Bezahlsystems der Gesamtbetrag der Bestellung von der Kreditkarte der Kund*in abzubuchen. Nur, wenn diese Operation erfolgreich durchgeführt werden kann, darf die Bestellung abgeschlossen werden.

Warenkörbe, auf die über einen konfigurierbaren Zeitraum hinweg weder lesend noch schreibend zugegriffen wurde, sind aus der Datenbank zu löschen.

(b.3) *Rabattsystem:* Shop-Betreiber*innen wollen ihre Plattform mit Rabatten attraktiv gestalten. Da sich die Rabattaktionen stark unterscheiden, soll ein flexibles und erweiterbares Rabattsystem geschaffen werden. Es soll eine Reihe von Rabattregeln (z. B. bei Kauf von einer bestimmten Mindestmenge eines Produkts oder bei Bestellung innerhalb eines bestimmten Zeitfensters wird ein Rabatt gewährt) und Rabattaktionen (z. B. prozentueller, fixer oder von anderen Kriterien abhängiger Preisnachlass) geben, aus denen Shop-Betreiber*innen auswählen können. Rabattregeln und -aktionen sind als .NET-Komponente zu implementieren und müssen flexibel miteinander kombinierbar sein. Die wesentlichen Parameter (z. B. Mindestbestellmenge, Rabattsatz etc.) müssen konfigurierbar sein.

Es müssen zumindest zwei Rabattregeln und zwei Rabattaktionen implementiert werden. *CaaS.Core* ist so auszulegen, dass weitere Regeln und Aktionen einfach hinzugefügt werden können.

- ☑ Sind die Komponenten für die Rabattregeln und -aktionen lose an die Anwendung gekoppelt und so einfach austauschbar? Welche Stellen im Code müssten geändert werden, wenn weitere Regeln und Aktionen hinzugefügt werden?
- ☑ Gibt es automatisierte Tests für die Logik, die einfach (ohne Datenbank) ausführbar sind?

(b.4) 👪 *Coupons:* Eine spezielle Rabattform stellen Gutscheine (Coupons) dar. Macht die Kund*in bei einer Bestellung einen Gutscheincode geltend, verringert sich der Preis um einen fixen an den Gutschein gekoppelten Betrag. Ein Gutschein kann nur einmal eingelöst werden.

(b.5) *Statistische Auswertungen:* Die Shop-Betreiber*in kann eine Reihe von statistischen Auswertungen durchführen, die einen Überblick über die Umsatzentwicklung und eventuell aufgetretener Anomalien geben. Es sind zumindest drei sinnvolle statistische Auswertungen zu realisieren. Dabei sind auch die Anforderungen aus *Caas.Web* zu berücksichtigen. Nachfolgend sind beispielhaft mögliche Auswertungen angeführt:

- Meistverkaufte Produkte in einem Monat / Jahr
- Gesamtstatistik (Umsatz pro Monat und Jahr, beliebtester und unbeliebtester Artikel, Anzahl der gekauften Artikel etc.)
- Durchschnittlicher Umsatz pro Warenkorb für einen auszuwählenden Zeitbereich
- Anzahl der Warenkörbe (bis zum Anfragezeitpunkt), d.h. offene und geschlossene Warenkörbe
- Anzahl eingelöster Gutscheine
- Zeitliche Verteilung von Warenkörben (z.B. Anzahl pro Wochentag) und deren Umsätze

(b.6) 👪 *Integration des Bezahlsystems:* Die Bezahlung einer Bestellung erfolgt über einen Drittanbieter, der die Abwicklung der Kreditkartenzahlung übernimmt. Machen Sie einen Vorschlag für die Schnittstelle des Bezahlsystems und erstellen Sie eine Implementierung, welche das Bezahlsystem simuliert. Berücksichtigen Sie dabei verschiedene Fehlermöglichkeiten (Kreditkartennummer falsch, Kreditkarte als verloren gemeldet, Betrag nicht gedeckt etc.) und simulieren Sie, dass die Zugriffsoperationen länger dauern können.

- ☑ Komponente ist sauber gekapselt, etwaige Anpassungen müssen nur an einer Stelle im Programm durchgeführt werden.

(b.7) *Autorisierung:* Für die eingehenden Anfragen, welche die Shop-Administration betreffen, ist zu überprüfen, ob Sie einen gültigen App-Key enthalten. Ist dies nicht der Fall, ist die Anfrage zu verwerfen und ein entsprechender Statuscode an den Sender zurückzuschicken.

(c) CaaS.Api

Die für *CaaS.Web* (*CaaS.Shop* und *CaaS.Admin*) erforderliche Funktionalität ist in Form eines REST-basierten Web-Service zu exportieren. Die für die (eingeschränkte) Autorisierung erforderliche Funktionalität ist zu berücksichtigen.

Die Web-Service-Schnittstellen werden maschinenlesbar als OpenAPI-Dokument ausgeliefert. Client-Entwickler erhalten eine übersichtliche Dokumentation in Form von HTML-Seiten.

- ☑ API-Dokumentation, Uniform Interface, schmale API-Schicht (keine Geschäftslogik im Web-Service)
- ☑ Sind die REST-Endpunkte übersichtlich strukturiert und verschiedene fachliche Aspekte sauber voneinander getrennt?

(d) CaaS.Web

Der Web-Client muss die nachfolgend näher ausgeführten Anforderungen erfüllen:

(d.1) Shop. Um die Fähigkeiten des Warenkorbs testen und demonstrieren zu können, entwickeln Sie mindestens zwei Shops, die die dem Shop zugeordneten Produkte auflistet und eine Kaufmöglichkeit bietet. Die Shop-Komponente muss auch bei sehr großen Datenmengen gut bedienbar sein, überlegen Sie sich hierzu geeignete Konzepte.

- ☑ Validierung bei Eingabefeldern, eigene Komponente für die Suche nach einem Produkt
- ☑ Kommunikation mit dem Backend, Observables
- ☑ Jeder Shop muss sich in den Produkten teilweise unterscheiden.

(d.2) *Warenkorb*. Entwickeln Sie eine Komponente für den Warenkorb, welche in den Shop integriert wird. Achten Sie auf ein ansprechendes Erscheinungsbild (Übersichtlichkeit, Funktionalität, der Warenkorb soll das Kaufvergnügen nicht beeinträchtigen). Der Warenkorb soll für eine gewisse Zeit persistiert werden. D.h., wenn eine Kund*in zu einem späteren Zeitpunkt den Shop nochmal besucht, soll ein vorheriger Warenkorb, in dem bereits Artikel abgelegt sind, auch wieder angezeigt werden. Preisliche Informationen (inkl. Rabatte) sollen übersichtlich angezeigt werden. Es sollte auch möglich sein, im Warenkorb die Menge eines einzelnen Produkts zu verändern (incl. Löschen). Implementieren Sie auch den Checkout Prozess (inkl. Anzeige der Bestellnummer, etc.) und sorgen Sie dafür, dass nach dem Checkout der Warenkorb nicht mehr editiert werden kann.

- ☑ Eigenständige Komponente, gespeicherten Warenkorb einer Kund*in anzeigen
- ☑ Kommunikation mit dem Backend, Observables

(d.3) *Authentifizierung mit OAuth2*. Shop-Betreiber*innen müssen sich für die Verwaltung ihrer Shops anmelden. Verwenden Sie hier bspw. den in der Übung eingesetzten Identity-Server, der von Manfred Steyer in der Azure-Cloud gehostet wird, oder Sie können auch andere Dienste wie Auth0 oder Open-Source-Lösungen wie z.B. Keycloak dafür nutzen.

Wenn Sie bspw. die Lösung von Manfred Steyer verwenden, dann steht Ihnen nur ein Benutzeraccount (max/geheim) zur Verfügung. Zur Vereinfachung darf dieser Benutzer mehrere Tenants verwalten.

- ☑ Guards, OAuth2

(d.4) *Verwaltung von Shops (= Tenant)*. Es soll zumindest möglich sein, neue Shops anlegen, bzw. bearbeiten zu können. Bitte beachten Sie, dass der eindeutige Applikationsschlüssel angezeigt werden muss, damit dieser in der Konfiguration des spezifischen Shops abgelegt werden kann.

- ☑ Validierung, Formular, Kommunikation mit dem Backend, Observables

(d.5) *Rabattregeln (nur für Sehr Gut notwendig)*. Nach dem Login sollen für einen Shop Rabattregeln bearbeitet werden können. Hier ist es ausreichend, wenn die Rabattregeln über ein statisches Formular konfiguriert werden (d.h. es ist keine dynamische Ermittlung, welche Regelkategorien es gibt, bzw. muss das Formular nicht dynamisch generiert werden).

- ☑ Validierung, Formular, Kommunikation mit dem Backend, Observables

(d.6) *Warenkorbstatistiken*. Die Shop-Betreiber*innen soll sich Statistiken zu den Warenkörben pro Shop anzeigen lassen können. Stellen Sie diese übersichtlich und grafisch ansprechend dar (z.B. mit Hilfe der Verwendung von Charts). Überlegen Sie sich hier sinnvolle Darstellungs- und Interaktionsmöglichkeiten (achten Sie darauf, dass z.B. eine Zeitachse nicht endlos wächst, etc.).

- ☑ Verwendung externer Komponenten

- ☑ Charts, übersichtliche Darstellung, sinnvolle Auswertungen (siehe (b.5))
- ☑ Kommunikation mit dem Backend, Observables

(e) 👥 Allgemeine Anforderungen für Zweierteams

Überlegen Sie sich geeignete Maßnahmen, die eine professionelle Anwendungsentwicklung in einem Team unterstützen, setzen Sie diese um und nehmen Sie diese auch in die Dokumentation auf. Dazu zählen beispielsweise der Einsatz eines Issue-Tracking-Systems (z. B. jenes von GitHub), automatisierte Builds und Vorkehrungen, die eine einfache Installation und Inbetriebnahme Ihres Softwaresystems ermöglichen (beispielsweise mit Docker und Docker Compose).

Hinweis: GitHub Classroom stellt nur sehr eingeschränkt Ressourcen für GitHub Actions zur Verfügung, die sich alle Mitglieder eines Classrooms, also alle Studierenden, teilen müssen.

Ergebnisse

Die in der Anforderungsdefinition festgelegten Funktionen sind in Einer- oder Zweierteams zu realisieren. Anforderungen, die mit dem Symbol 👥 gekennzeichnet sind, müssen nur von Zweierteams ausgearbeitet werden. Alle anderen Anforderungen gelten für beide Gruppen. Die WEA5 zuzuordnende Funktionalität ist in Form von Einzelarbeiten umzusetzen.

Die Entwicklung der Projektarbeit ist auf GitHub durchzuführen. Alle Teams bekommen ein Repository zugewiesen, auf welches das Team und die Übungsleiter Zugriff haben.

Erstellen Sie für alle Dokumente und Quelltext-Dateien eine übersichtliche Verzeichnisstruktur und packen Sie diese in eine ZIP-Datei. Dieses Archiv stellen Sie Ihrem Übungsleiter auf der E-Learning-Plattform in den Kursen zur SWK5-Übung bzw. zu WEA5 zur Verfügung. Achten Sie darauf, dass Sie nur die relevanten (keine generierten) Dateien in das Archiv geben. Große Archive mit unnötigen Dateien vergeuden Speicherplatz und können daher zu Punkteabzügen führen.

Große Binärdateien, wie Komponenten von Drittherstellern oder Datenbanken dürfen keinesfalls im Repository abgelegt werden. Checken Sie stattdessen Scripte zur Erzeugung des Datenbankschemas und zum Befüllen der Datenbank ein. Dokumentieren Sie, wie Ihr System in Betrieb genommen werden kann.

Konzentrieren Sie sich bei der Dokumentation dieser Projektarbeit auf die Darstellung der Architektur und des Designs Ihrer Anwendung. Legen Sie die Dokumentation so an, dass Personen, die Ihr Projekt weiterführen möchten, die Struktur Ihres Softwaresystems schnell erfassen und Designentscheidungen nachvollziehen können. Durch das unreflektierte Einfügen von generierten Diagrammen wird diese Anforderung nicht erfüllt. Quelltexte sind nur dann in die Dokumentation aufzunehmen, wenn Sie damit zentrale Aspekte Ihrer Implementierung erläutern wollen. Die Dokumente sind ausschließlich in Form von PDF-Dateien abzugeben.