Dokumentation der softwaretechnischen Umsetzung

Common Component Modelling Example

Philipp Wiegland & Lisa Wittmann

Fortgeschrittene Softwaretechnologie WS 21/22

Inhaltsverzeichnis

Architekturentscheidungen

Entwickelt wurde eine Trading Software nach dem Beispiel des Common Component Modelling Exemples in Form einer Webanwendung mithilfe des Web Application Frameworks ASP.NET Core 5 und des Typescript-basierten Frontend-Framework Angular. Wir setzen in unserer Dokumentation die Inhalte des CoCoME Dokumentes voraus und gehen daher nicht genauer auf die Anwendungsszenarien und weitere Voraussetzungen ein.

Die Dokumentation des CoCoMEs sieht ein System aus einzelnen Komponenten vor, das aus den Subsystemen Enterprise Server, Store Server und Cashdesk Computer besteht. Wir haben uns bei der Dekomposition des Systems dazu entschieden,die Komponenten des Systems in Form von Dependency Injection Components innerhalb einer einzigen Anwendung zu implementieren. Aufgrund dieser Architekturentscheidung haben wir eine rollenbasierte Autorisierung für die unterschiedlichen Bereiche sowohl im Frontend als auch im Backend vorgesehen. Sollte die Anwendung in unterschiedlichen Instanzen je Filiale laufen sollen, wären nach unserer Implementierung eine Synchronisierung der Datenbankserver zusätzlich von Nöten.

Da das System als Webanwendung implementiert wurde, ist davon auszugehen, dass sich die Kassengeräte nicht zwangsläufig im lokalen Netzwerk des Servers befinden. Aufgrund dessen erfolgt die Anbindung der Ein- (Barcode Scanner, Kartenlesegerät) und Ausgabegeräte (Drucker, Display) clientseitig.

Bei der Entwicklung haben wir vorausgesetzt, dass das System lediglich unternehmensintern zum Einsatz kommen würde und somit in unterschiedlichen Instanzen und Ausprägungen für unterschiedliche Kunden existieren könnte. Daher wurde systemintern auf die Differenzierung zwischen Unternehmen verzichtet, stattdessen wird davon ausgegangen, dass alle Filial Instanzen einem übergeordneten Träger angehören.

Die Entwicklung erfolgte aufgrund der Nutzungsszenarien “Desktop First”. Zudem haben wir uns dazu entschlossen, den Internet Explorer bei der Frontend-Entwicklung nicht zu berücksichtigen.

Backend

Fremdkomponenten

Die Webanwendung wurde nach Anforderung mithilfe des Web Application Frameworks ASP.NET Core 5 entwickelt. Die Datenbank der Anwendung basiert auf dem relationalen Datenbanksystem der SQLite-Bibliothek.

Zur Benutzerauthentifizierung wurde das OpenID Connect und OAuth 2.0 Framework IdentityServer 4 und die damit einhergehenden auto generierten Razor Views genutzt. Sämtliche Inhalte innerhalb des Projektordners Areas/Identity wurden daher automatisch generiert und sind lediglich nach den Bedürfnissen unserer Anwendung modifiziert worden.

Zur Generierung von PDF Dokumenten wird die Bibliothek DinkToPdf sowie RazorLight genutzt. Die Implementierung innerhalb des PrinterService richtet sich stark nach folgende Anleitung zur Generierung eines PDF Dokumentes anhand eines Razor Templates.

Das Testing der Schnittstellen und Services haben wir versucht mithilfe des Moq Packages umzusetzen. Leider haben die Beispiele der Microsoft ASP.NET Dokumentation zum Testen von Entity Framework Anwendungen in unserem Projekt nicht funktioniert. Deshalb mussten wir letztendlich schweren herzens auf eine vernünftige Testabdeckung verzichten.

Zur Dokumentation sowie zum Testen der Schnittstellen wurde zudem Swagger über das Package Swashbuckle.AspNetCore eingebunden.

Domänenmodell

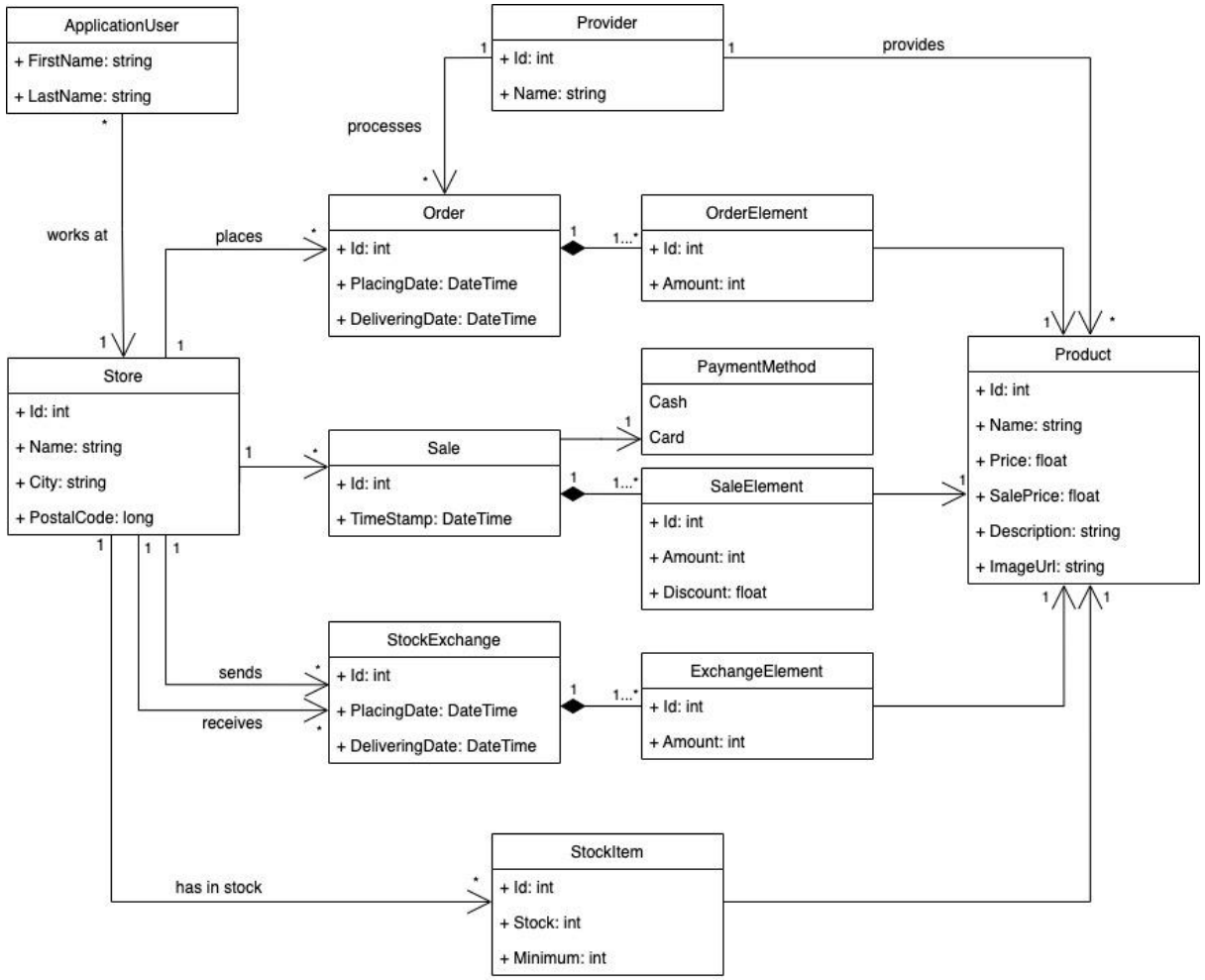

Abb. 1: Gesamtübersicht der Objektbeziehungen und -attribute

Benutzeridentifizierung

Zur Identifikation innerhalb der Anwendung dient die Klasse ApplicationUser, die die Basisklasse IdentityUser des genutzten Packages Microsoft.AspNetCore.Identity erweitert. Neben der zugeordneten Identity Role zur Autorisierung innerhalb der Anwendung, wird ein Benutzer einer Filiale zugeordnet und kann Interaktionen nur innerhalb seiner Filiale ausführen. Die ID der Filiale des Benutzers wird innerhalb des JWT Token als Claim ausgewiesen. Der Administrator benötigt diese Zuordnung nicht.
Neben einer E-Mail Adresse und einem einmaligen Nutzernamen (in unserer Applikation wird einfachhalber die E-Mail-Adresse als Benutzername verwendet), die die Basisklasse vorgibt, muss sich ein neuen Benutzer ebenfalls mit seinem Klarnamen registrieren.

Filialen

Eine Filiale wird identifiziert über eine automatisch generierte ID und bekommt einen Klartextnamen, der allerdings nicht eindeutig sein muss. Ebenfall können Adressinformationen in Form einer Stadt und einer Postleitzahl vermerkt werden. Nur mit Angabe einer Postleitzahl ist der automatische Produktaustausch zwischen Filialen möglich. Jede Filiale hat eine beliebige Menge an Stock Items, die den Lagerbestand und gleichzeitig das Sortiment definieren. Nur Produkte, die auf Lager sind können an Kunden weiterverkauft werden und allein Waren, die der Filiale als Stock Item zugeschrieben sind, können nachbestellt oder editiert werden.

Produkte

Da die generierte ID eines Produktes einmalig ist,haben wir auf ein weiteres Barcode-Attribut verzichtet und stattdessen mit der Produkt-ID verschmolzen. Weitere Informationen über das Produkt können anhand eines Namens, einer Beschreibung sowie einer ImageUrl gespeichert werden. Das Attribut Price definiert den Einkaufspreis des Produktes und bestimmt den Betrag einer Nachbestellung bei einem der Lieferanten, während das Attribut SalePrice den Verkaufspreis der Ware definiert. Dieses bestimmt entsprechend den Betrag für einen Verkauf an einen Kunden. Ein neu angelegtes Produkt muss eine Beziehung zu einem Lieferanten haben, der das Produkt an Filialen liefern kann, da sonst keine Nachbestellungen des Produktes möglich sind.

Warenaustausch

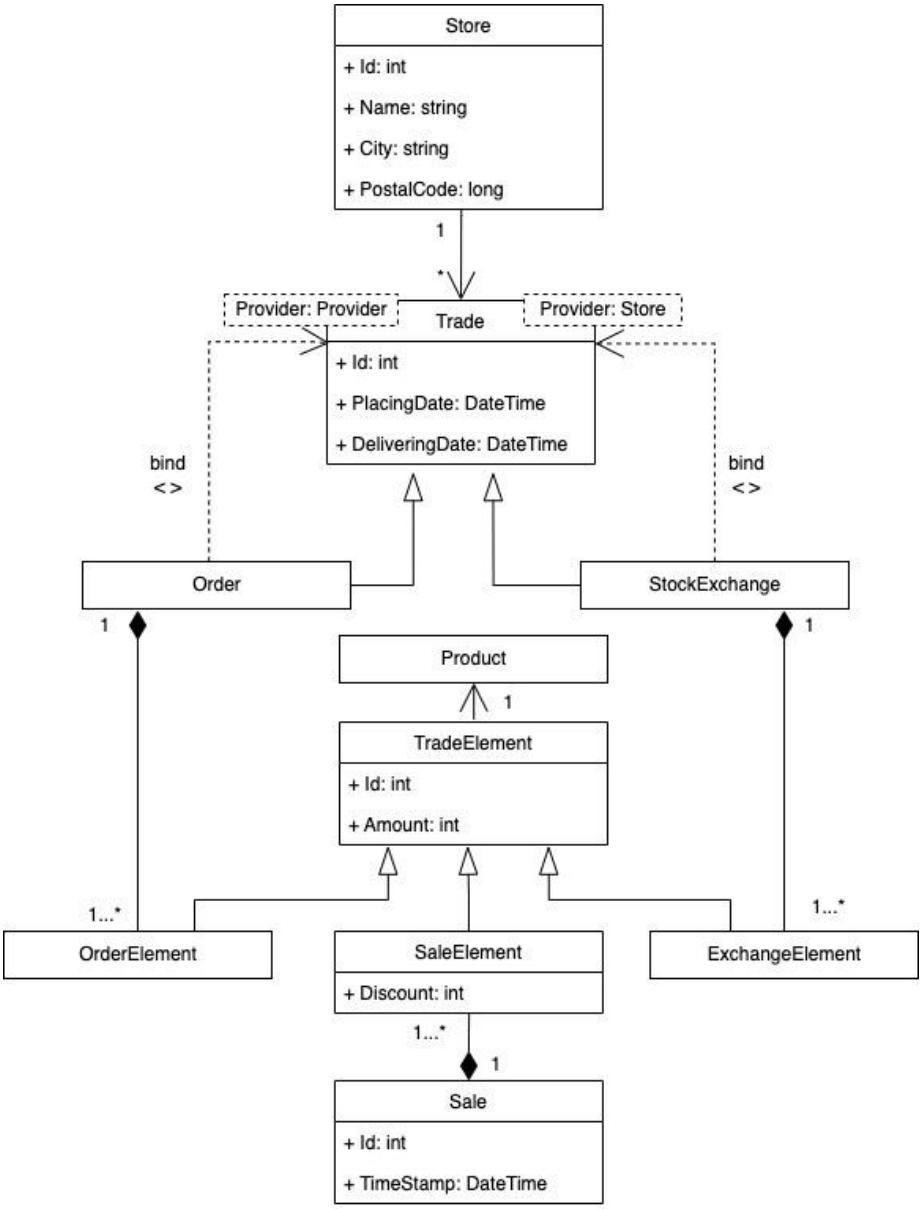

Abb. 2: Übersicht der Warenaustausch Entitäten

Die abstrakte generische Klasse Trade bildet einen Warenaustausch zwischen Filialen ab. Jeder Austausch kann einer Filiale zugeordnet werden, die den Austausch initiiert und Produkte, gekapselt in TradeElements, anfordert. Bei der Erweiterung der abstrakten Klasse wird der Typ des Lieferanten definiert, im Falle einer Bestellung ist es ein Provider Objekt und bei einem Produktaustausch ist der Lieferant eine andere Filiale. Ein Trade speichert zwei Zeitstempel, einerseits den Zeitpunkt zu dem die Bestellung aufgegeben wurde und andererseits den Zeitpunkt des Erhalts der Produkte. Ein TradeElement bietet die Möglichkeit der Kapselung eines Produktes mit der angeforderten Menge. Die Relation zu einer Trade Instanz muss in den Unterklassen implementiert werden.

Ein Verkauf bildet zwar genau wie eine Bestellung oder ein Produktaustausch wischen Filialen einen Warenaustausch ab, unterscheidet sich jedoch stark von den anderen Spezifikationen. Ein Verkauf erfolgt unmittelbar und anonym, dementsprechend ist die Beziehung von zwei Handelspartnern nicht abbildbar und die Speicherung von zwei Zeitstempeln redundant. Aufgrund dessen haben wir uns dazu entschieden einen Trade auf einen Handel zwischen einer Filiale und einen generischen Händler zu beschränken und den Verkauf ohne Abhängigkeit zu implementieren.

Warenaustausch zwischen Kunde und Filiale

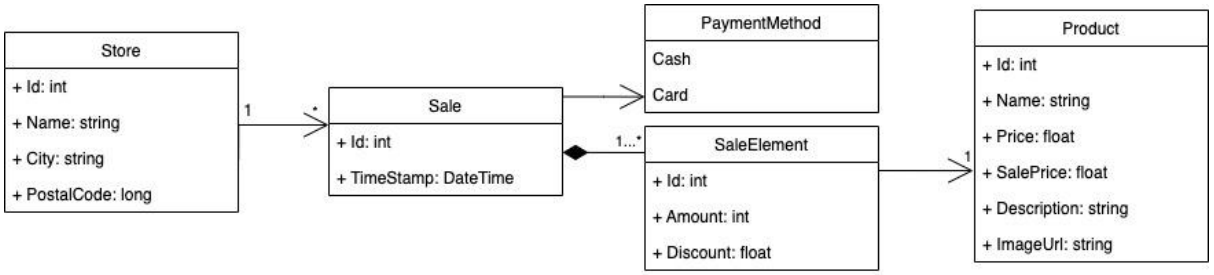

Abb. 3: Beziehungen der Entitäten innerhalb eines Verkaufsprozesses

Ein Verkaufsvorgang (Sale) kann eindeutig einer Filiale zugeordnet und über eine generierte ID identifiziert werden. Das Sale Objekt enthält den Zeitpunkt des Kaufabschlusses sowie Informationen über die gewählte Zahlungsmethode des Kunden. Die Enumeration PaymentMethod stellt dabei alle verfügbaren Zahlungsmethoden dar, im Falle unserer Anwendung Bar- und Kartenzahlung.

Produkte, die bei einem Verkauf erworben werden, werden in einem SaleElement Objekt gekapselt, das von der abstrakten TradeElement Klasse erbt. Dieses enthält neben der Referenz auf das erworbene Produkt auch die Anzahl der entsprechenden Ware sowie einen möglichen Rabatt in Prozent. Anhand dieser Informationen und Beziehungen lässt sich der Umsatz einer Filiale mit Möglichkeit der Spezifizierung eines Zeitraums auswerten.

Warenaustausch zwischen Lieferant und Filiale

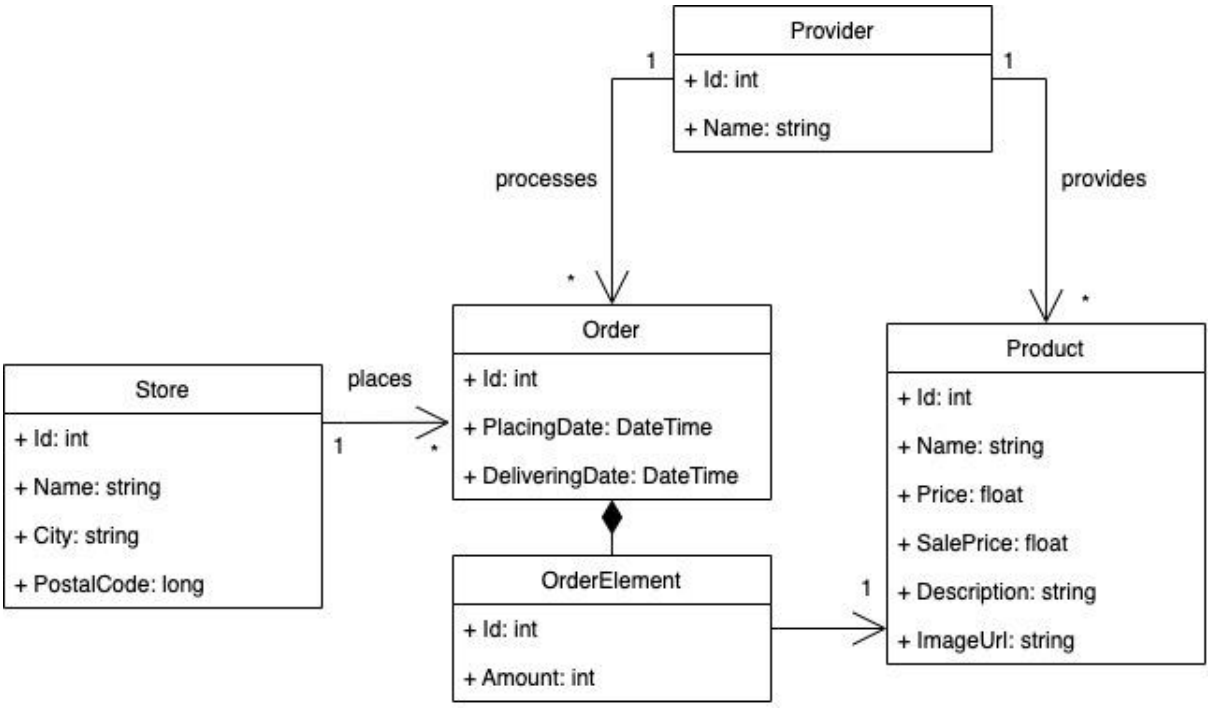

Abb. 4: Involvierte Entitäten in einem Bestellprozess

Eine Bestellung implementiert einen Warenaustausch zwischen einem Lieferanten und einer Filiale. Die Order Klasse erweitert daher die abstrakte Trade Klasse für den Provider-Typ Provider. Jede Produktinstanz der Bestellung wird in einer OrderElement Instanz unter der Angabe der bestellten Menge gekapselt. Die Klasse OrderElement erweitert daher die abstrakte Klasse OrderElement um eine Referenz zu der Bestellung, die für die Erzeugung der Instanz geführt hat.

Jede Produktinstanz hat eine eindeutige Referenz zu einem Lieferanten, von dem die Ware bezogen wird. Wenn eine neue Bestellung erzeugt werden soll, werden die gewünschten Produkte anhand ihres festen Lieferanten in einzelne Order Instanzen aufgeteilt. Der Lieferant der Bestellung wird daher aus den Produktinformationen abgeleitet.

Warenaustausch zwischen Filialen

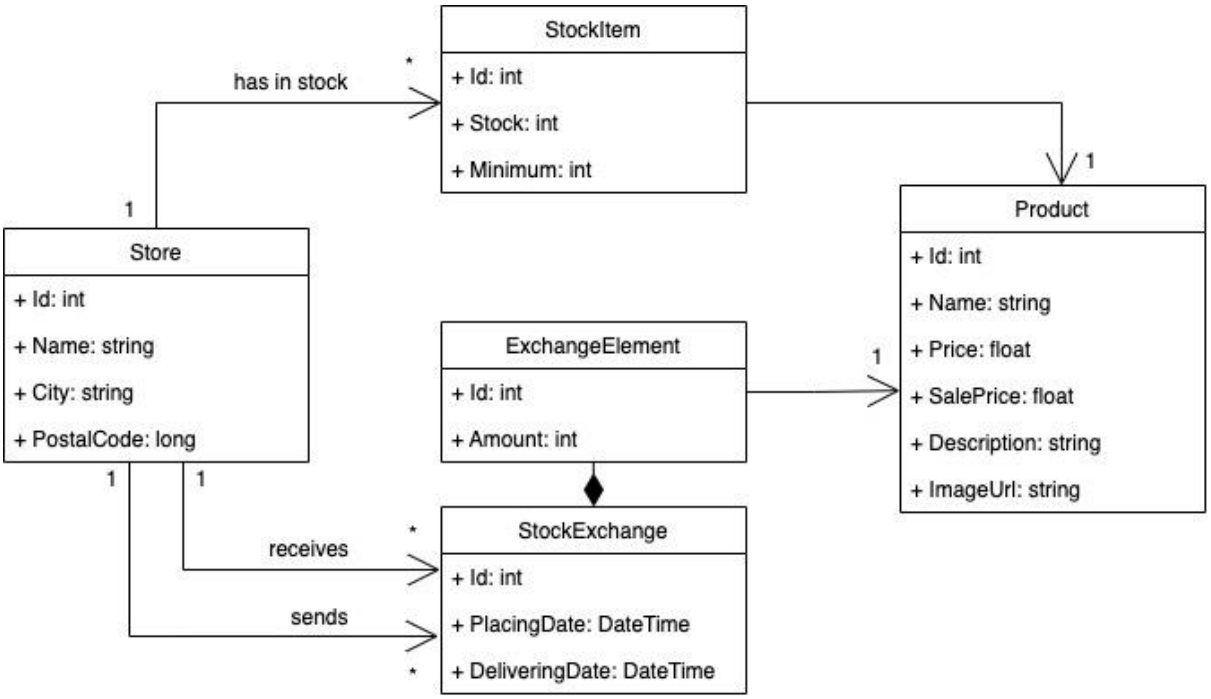

Abb. 5:Involvierte Entitäten in einem Produktaustausch Vorgang

Ein Produktaustausch ist eine Spezifikation des Warenaustausches zwischen zwei Filialen. Eine Filiale löst dabei ohne Einwirkungen des Nutzers einen Warenaustausch aus, wenn ein kritischer Warenbestand erkannt wird und sich eine geeignete weitere Filiale für einen Austausch in Reichweite befindet. Der kritische Bestand wird dabei anhand des Minimalen Bestands eines Stock Items bemessen. Ist der Minimalwert nicht gesetzt, wird kein Austausch getriggert. Es wird stattdessen davon ausgegangen, dass es sich nicht um Produkt des festen Sortiments handelt, das dringend neu beschafft werden muss.

Die StockExchange Klasse implementiert die abstrakte Klasse Trade mit dem Lieferantentyp Store, da eine Filiale die Rolle als Empfänger und eine weitere die des Lieferanten übernimmt. Das zugehörige ExchangeElement erweitert die abstrakte Klasse TradeElement, um ein Produkt mit der angeforderten Menge zu verknüpfen.

Hinweis: Die CoCoME Dokumentation sieht vor, dass die Rentabilität eines Produktaustausch anhand der Distanz zwischen zwei Filialen berechnet wird. Für diesen Ansatz hätten wir gerne eine API verwendet, die bei Erstellung einer Filiale die geografischen Koordinaten der Adressinformationen generiert. Diese könnten mit der Filiale persistiert werden, sodass eine tatsächliche Distanzbestimmung möglich wäre.

Da eine nennenswerte große Firma sich solche Schnittstellenanfragen gut bezahlen lässt und wir nicht abschätzen konnten, wie viele Anfragen während der Entwicklung generiert werden, haben wir uns dazu entschieden eine konkrete Distanzbestimmung außer Acht zu lassen. Stattdessen werden bei der Kalkulation die ersten beiden Ziffern der Postleitzahl verwendet, was zumindest innerhalb von Deutschland eine Form des Indikators darstellt. So kann es aber natürlich sein, dass ein Produktaustausch zwischen Mainz und Mainz-Amöneburg von dem System als nicht rentabel eingestuft wird, eine Lieferung von Idar-Oberstein nach Mainz jedoch schon. Für eine Verwendung außerhalb des Hochschulkontext wäre das natürlich unzureichend und müsste unter der Verwendung der genannten Schnittstelle implementiert werden.

Komponenten und Schnittstellen

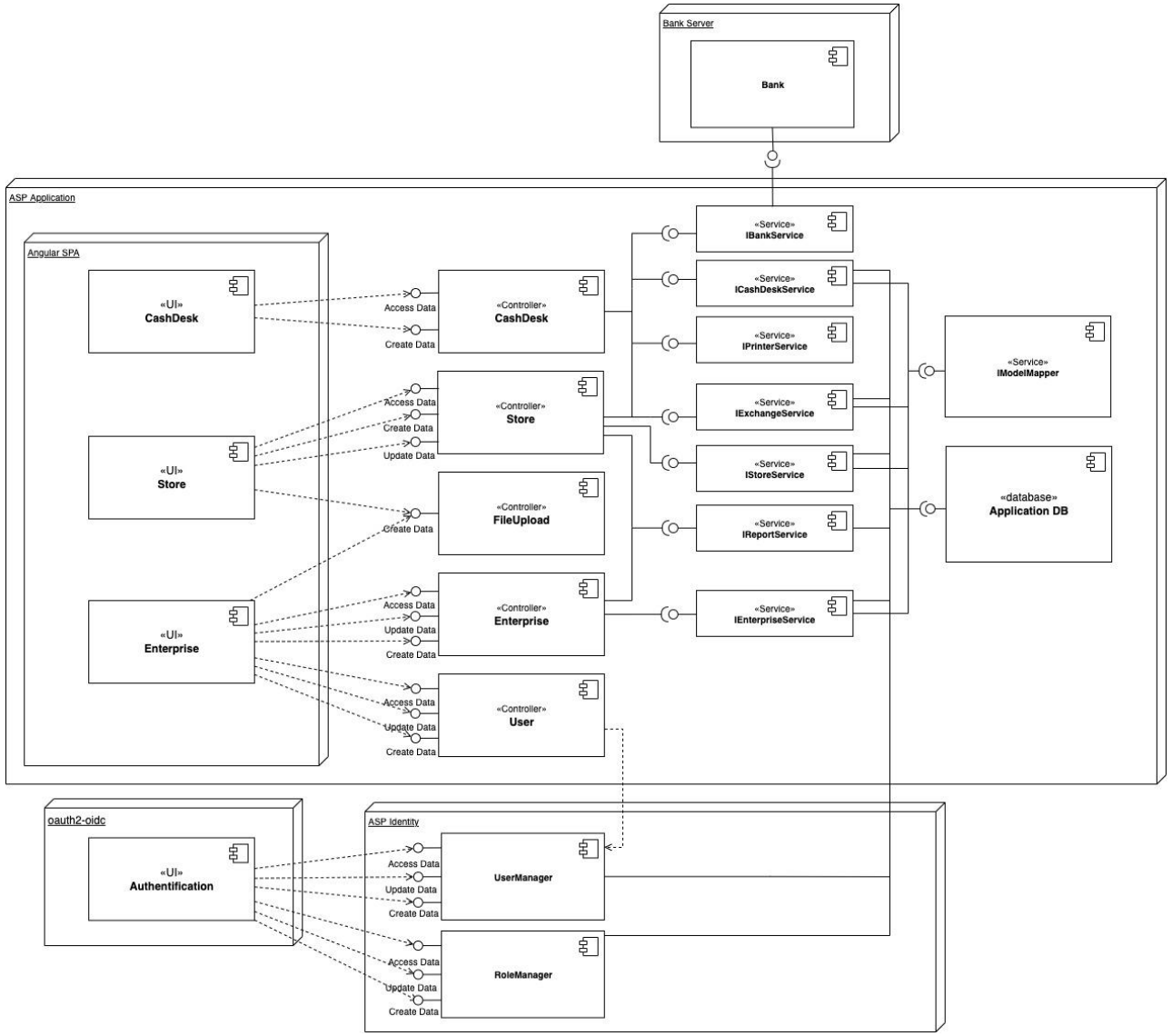

Abb. 6: Gesamtübersicht über die Komponenten und Schnittstellen der Anwendung

Da die Autorisierung und Authentifizierung durch Fremdkomponenten umgesetzt wurde, wird die Funktionsweise im folgenden Abschnitt nicht weiter erläutert.

Zur Dokumentation der API Schnittstellen des Systems wurde Swagger genutzt. Das Swagger Interface ist nach Starten der Anwendung im Development Modus unter “/swagger” einzusehen.

Cash Desk System

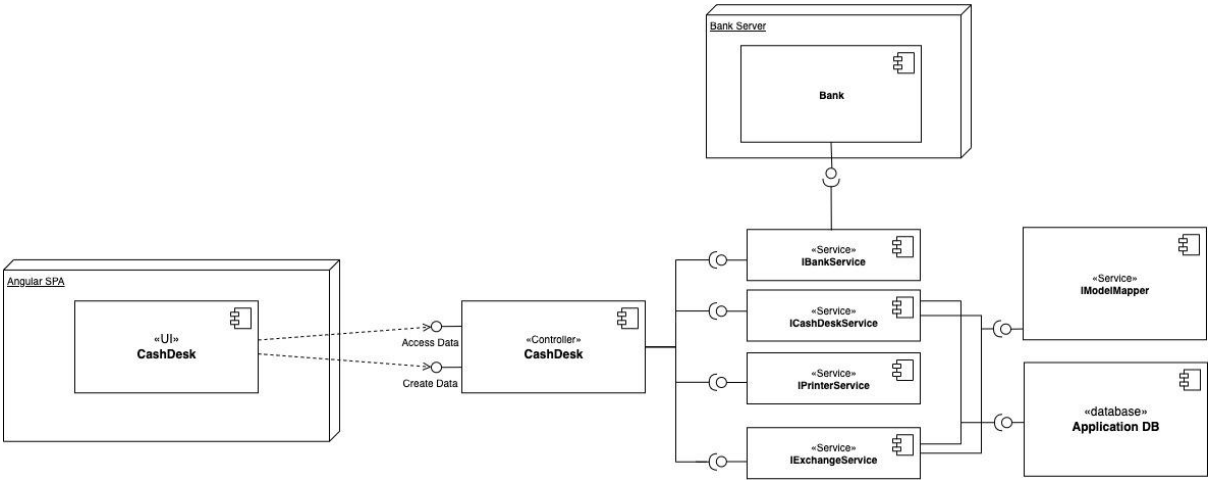

Abb. 7: Komponenten des Kassensystems

Der Cash Desk Controller stellt alle benötigten Schnittstellen für die Funktionsweise einer Kasse bereit. Das Frontend kommuniziert daher nur mit der Cash Desk API.
Die Cash Desk API stellt Schnittstellen zur Verfügung um die verfügbaren Produkte einer Filiale abzufragen, ein Produkt Objekt anhand seiner ID zu erhalten, wenn es unter den verfügbaren Produkten ist, eine Kartenzahlung durchzuführen sowie einen neuen Verkauf zu registrieren. Innerhalb des Controllers werden die Aufgaben an die zuständigen Services delegiert.

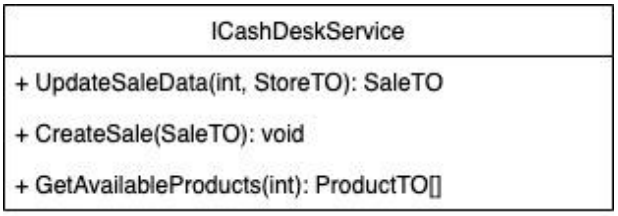

Abb. 8: Komponente ICashDeskService im Detail

Der Cash Desk Service stellt die Haupt Funktionalitäten der Kasse bereit: Die verfügbaren Produkte der Filiale abfragen, Informationen zur Persistierung der einkommenden Sale Transfer Objekte ergänzen sowie einen Verkauf persistieren und dabei den Filialbestand zu aktualisieren. Um die Transfer Objekte in Datenbankmodelle aufzulösen verwendet der Service das Interface IModelMapper, das Funktionen zur Umwandlung von und zu Transfer Objekten bereitstellt.

Nach der Aktualisierung des Produktbestands bei erfolgreichem Abschluss eines Verkaufs, wird der Exchange Service (Siehe Abb. 12) angestoßen. Dieser ermittelt, ob ein Produkt einen kritischen Bestand erreicht hat und ein Produktaustausch ausgelöst werden sollte.

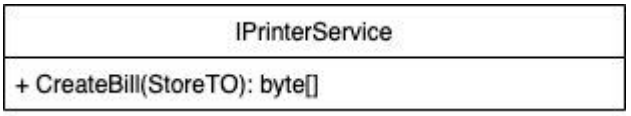

Abb. 9: Komponente IPrinterService im Detail

Der Printer Service wird zur Generierung eines Verkaufbeleges verwendet. Anhand der erhaltenen Informationen des Sale Transfer Objects wird eine Razor Template Vorlage zu einem PDF-Dokument umgewandelt und bei erfolgreicher Persistierung des Verkaufs als Antwort der Schnittstelle auf die Verkaufsanfrage zurückgegeben.

Abb. 10: Komponente IBankService im Detail

Das IBankInterface ist für die Kommunikation mit dem externen Bank Server zuständig. Dazu erhält es die einkommenden Kreditkarten Informationen, die es zu überprüfen gilt. Die Anbindung an einen Bank Server haben wir im Rahmen des Projektes zeitlich leider nicht mehr umsetzen können.

Store System

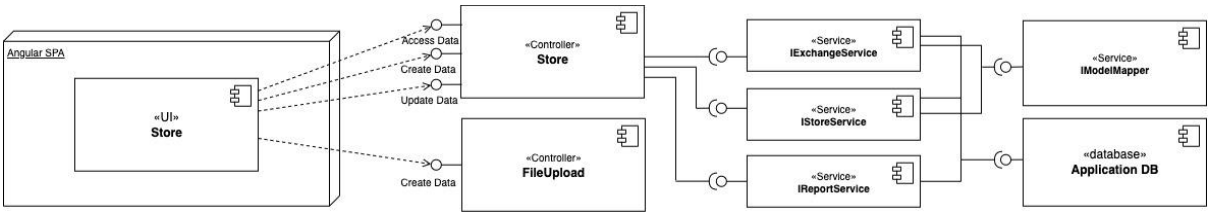

Abb. 11: Komponenten des Store Systems

Die Schnittstellen mit den wesentlichen Funktionen des Store Systems sind alle innerhalb des Store Controllers untergebracht. Zusätzlich haben wir die Möglichkeit geschaffen, Produktbilder einzufügen und hochzuladen. Diese zusätzliche Funktion ist in den separaten FileUpload Controller ausgelagert, da die Funktion sowohl dem Filialleiter als auch dem Enterprise Manager zu Verfügung stehen soll. Der Controller stellt nur eine einzige Schnittstelle zum Hochladen einer Datei bereit. Die Dateien werden nicht in der Anwendungsdatenbank persistiert, sondern im Public Folder des Projektes abgelegt. Der Store Controller ist ähnlich aufgebaut wie der Cash Desk Controller. Er stellt alle Funktionen als Schnittstellen bereit und delegiert die Anfragen an die zuständigen Services.

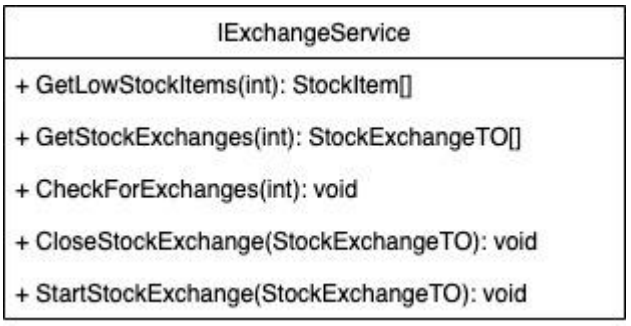

Abb. 12: Komponente IExchangeService im Detail

Der Exchange Service verwaltet die Produktaustausche zwischen Filialen. Neben der Funktionalität den Bestand einer Filiale zu überprüfen und, sofern rentabel, einen neuen Austausch zu veranlassen, lassen sich die offenen Austausche einer Filiale anfragen, starten oder schließen. Ein erzeugter Exchange muss erst von der liefernden Filiale bestätigt werden, bevor die Waren aus dem Inventar entfernt werden (Start). Erst dann kann die empfangende Filiale den Austausch abschließen und als geliefert markieren, wobei die empfangenen Produkte automatisch dem Filialbestand hinzugefügt werden. Der Exchange Service konvertiert mit hilfe des ModelMappers die einkommenden Transfer Objekte zur Verarbeitung in Datenbankobjekte und bei Rückgabe erneut in Transfer Objekte.

Abb. 13: Komponente IReportService im Detail

Zur Erzeugung der Store Reports wird der gleichnamige Report Service genutzt. Dieser stellt sowohl die Funktionen zur Generierung von Berichten für den Store Client als auch für den Enterprise Client zur Verfügung und wird daher auch innerhalb des Enterprise Controllers verwendet. Für das Store System werden die Filialinternen Funktionen zur Ermittlung der Umsätze der jeweiligen Filiale verwendet. Dem Enterprise Manager ist es möglich neben der Generierung von Liefer Statistiken, sich den aktuellen Jahresumsatz aller Filialen gesammelt ausgeben zu lassen.

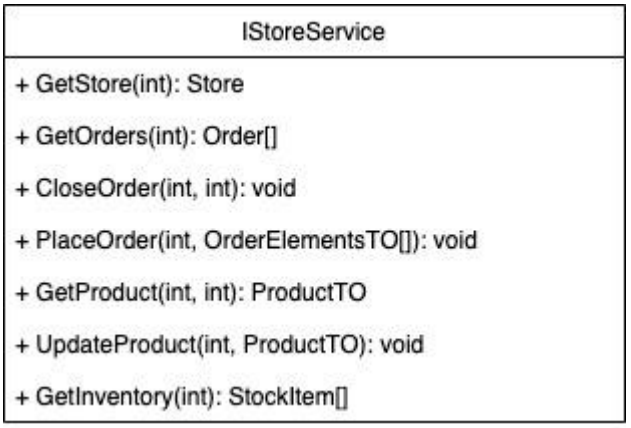

Abb. 14: Komponente IStoreService im Detail

Der Store Service stellt weitere Funktionen bereit um Daten einer Filiale zu erfragen und beschränkt zu modifizieren. So ist es dem Store Manager erlaubt Produktinformationen zu aktualisieren, sofern das Produkt innerhalb seiner Filiale angeboten wird. Zudem können Bestellungen abgeschlossen werden, wobei die bestellten Waren dem Warenbestand der Filiale gutgeschrieben werden. Sonst verfügt der Store Manager nur über Lesezugriff auf das eigene Inventar, ausstehende Bestellungen und kann Informationen über eine Filiale anhand der ID erfragen.

Enterprise System

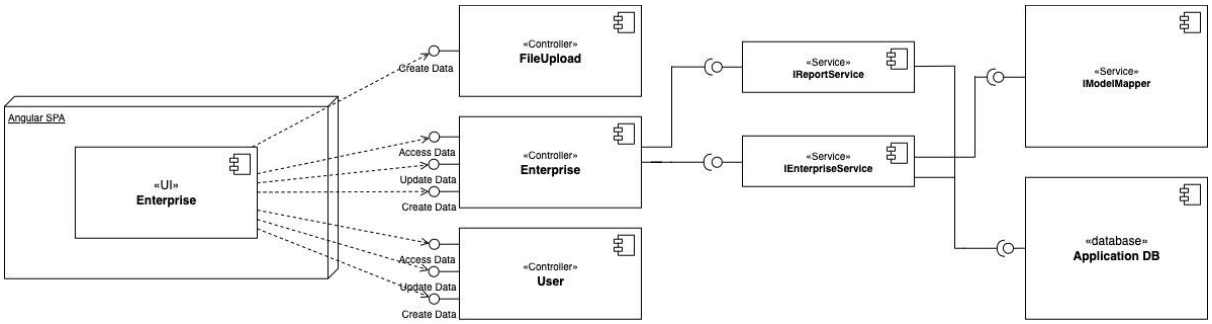

Abb. 15: Komponenten des Enterprise Systems

Das Enterprise Frontend bietet die meisten Funktionalitäten der Anwendung und kommuniziert daher mit drei unterschiedlichen Controllern. Genau wie der Store Manager ist der Enterprise Administrator befugt Dateien innerhalb der Anwendung hochzuladen und daher an die FileUpload Schnittstelle angebunden. Der Enterprise Controller bündelt erneut die wesentlichen Funktionalitäten der Enterprise Subanwendung. Zusätzlich haben wir uns dazu entschieden, dem Administrator die Möglichkeit zur Verfügung zu stellen Benutzer hinzuzufügen, einer Filiale zuzuordnen und Anwendungsrechte bzw. -Rollen zu vergeben. Diese Funktionalitäten werden als Schnittstellen innerhalb des User Controllers zur Verfügung gestellt. Dieser greift auf die Schnittstellen des ASP Identity Packages zu und wird daher im weiteren Abschnitt nicht genauer erläutert.

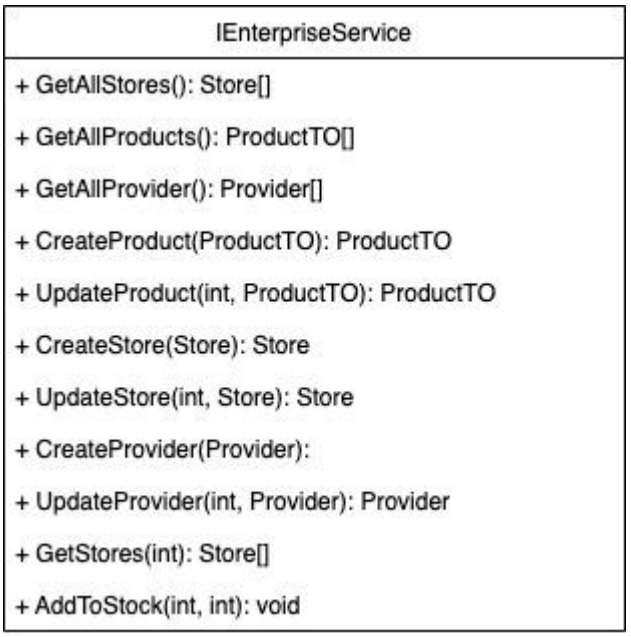

Abb. 16: Komponente IEnterpriseService im Detail

Neben der Möglichkeit Anwendungsdaten abzufragen, stellt der Enterprise Service Funktionen zur Verfügung neue Daten in die Anwendungsdatenbank einzupflegen sowie

Datensätze zu aktualisieren. Der Administrator kann daher Filialen, Produkte und Lieferanten abfragen, hinzufügen und editieren. Zusätzlich kann ein Produkt dem Bestand einer Filiale und somit dem Sortiment hinzugefügt werden. Der Service greift auf den die Model Mapper Komponente zurück um Transfer Objekte in Datenbankobjekte umzuwandeln und umgekehrt.

Zudem delegiert der Enterprise Controller Anfragen bezüglich Lieferstatistiken an Report Service (Siehe Abb.13), Zusätzlich hat der Administrator die Möglichkeit die aktuellen Jahresumsätze aller Filialen in Form eines Berichts abzufragen.

Frontend

Fremdkomponenten

Das Frontend der Applikation wurde Typescript basiert in Angular Version 8 angelegt. Zur anschaulichen Darstellung der Reports für die Filialleiter Ansicht und den Adminbereich wurden die Bibliotheken Chart.js und D3.js verwendet.

Zur Benutzerautorisierung wurde das package angular-oauth2-oidc eingebunden. Die benötigten Services, Module und Komponenten in dem Frontend-Package api-authorization wurden anhand der ASP Net Core Projektvorlag automatisch generiert und wurden lediglich auf die Bedürfnisse der Applikation angepasst.

Architektur

Um das mehrschichtige System am Besten abzubilden, haben wir die drei Bereiche der Anwendung in Form von Angular Moduls umgesetzt. Der Zugriff auf die Subanwendungen wird durch RouteGuards überwacht und ist nur nach vorheriger Authentifizierung und Nachweis der entsprechenden Identity Role möglich. Jedes Modul definiert eigene Unterpfade, die innerhalb des App-Moduls zusammengeführt werden.

Um globale Datenhaltung zwischen Komponenten zu ermöglichen und die Zahl der REST-Anfragen zu reduzieren, ist innerhalb der einzelnen Module ein Injectable StateService verfügbar. Der State Service stellt die benötigten Daten innerhalb des Moduls als Observable allen Komponenten zu Verfügung, die daher automatisch bei Änderung ihre Darstellung der Daten aktualisieren.

Module

Cash Desk

Das Kassen Interface stellt die Funktionen zur Verfügung Verkaufsprozesse zu starten, durchzuführen und den Bezahlvorgang abzuschließen. Zudem soll eine Kasse automatisch in den Express Modus übergehen, wenn die Voraussetzungen dazu erfüllt sind. Die Subanwendung ist nur nach Autorisierung der Kassierer-Rolle möglich, die anhand des JWT-Tokens geschieht. Zudem enthält das Token Information über die Filiale des Benutzers und ermöglicht nur Aktionen innerhalb der eigenen Filiale.

Aufgrund der Umsetzung des Systems als Webapplikation, wird der Express Modus einer Kasse clientseitig verwaltet. Im Cash Desk Service wird der globale State der Kassen Anwendung gespeichert und verwaltet. Zudem verwaltet der State den Warenkorb des aktuellen Verkaufs Vorganges und stellt die Anbindungen zu den backend seitigen REST-Schnittstellen bereit.

Die Anbindung der Kassengeräte erfolgt ebenfalls clientseitig. Bei dem Barcode Scanner sowie dem Kartenlesegerät handelt es sich um Eingabegeräte, die die eingelesenen Daten automatisch in Formularelemente übertragen. Zur Nutzung wurden daher entsprechende Formulare angelegt, die sowohl mit als auch ohne die verwiesenen Eingabegeräte die Funktionalitäten abdecken. Nach erfolgreichem Abschluss eines Verkaufs quittiert die Schnittstelle das Ergebnis mit einem Kassenbon Dokument, das an einen lokalen Drucker weitergeleitet werden kann.

Das Kassen Interface ist unterteilt in drei Views. Der Startscreen bietet die Möglichkeit einen neuen Vorgang zu starten oder den aktivierten Express Modus zu beenden. Bei Start eines neuen Vorganges erfolgt die Weiterleitung zur Checkout Component, in der der eigentliche Verkaufsprozess stattfindet. Neben einem Eingabefeld um den Barcode eines Produktes einzulesen und das zugehörige Produkt dem Warenkorb hinzuzufügen, gibt es eine zusätzliche Anzeige der verfügbaren Produkte in der Filiale, die ebenfalls zur Eingabe genutzt werden kann. Die Produktliste kann zudem nach Namen und Barcodes durchsucht werden. Nach Bestätigung des Kaufabschlusses wird der Kassierer zur Payment Component weitergeleitet. Durch die Auswahl der Bezahlmethode öffnet sich ein Modal zur Eingabe weiterer Zahlungsinformationen. Bei einer Barzahlung kann der übergebene Betrag eingeben werden und der Kauf abgeschlossen werden. Im Falle der Kartenzahlung können die Karteninformationen eingeben werden und im Anschluss durch eine REST-Anfrage überprüft werden. Erst nach der Verifizierung der Informationen kann der Verkauf abgeschlossen werden.

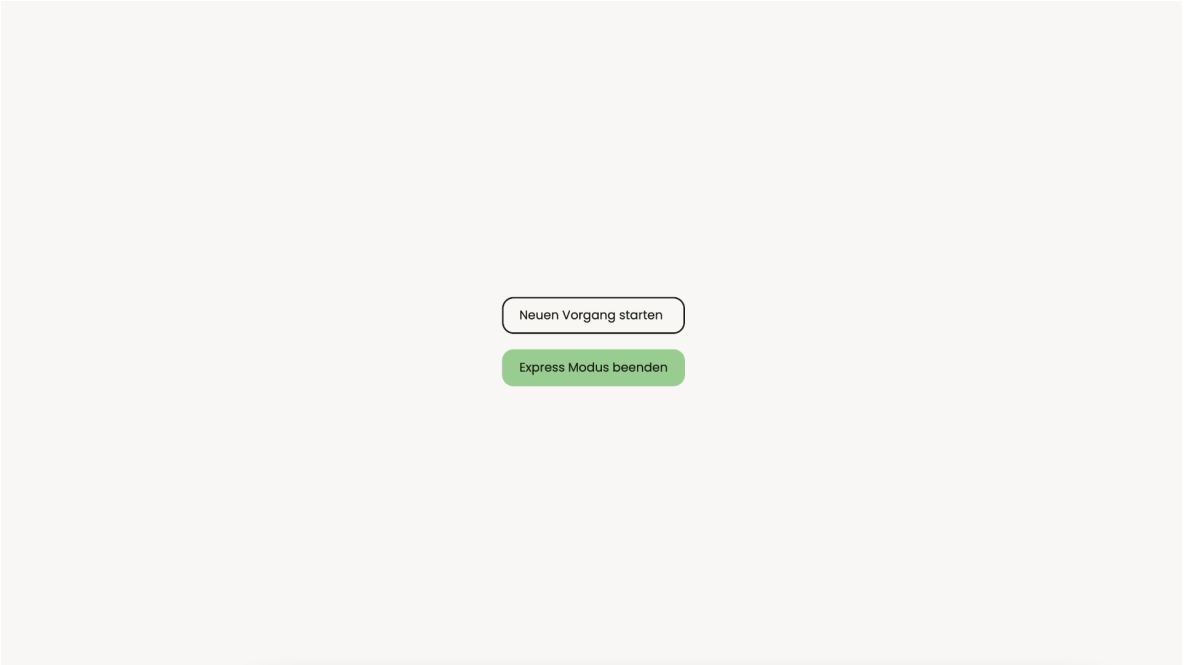

Abb. 17: Wireframe Einstiegsseite

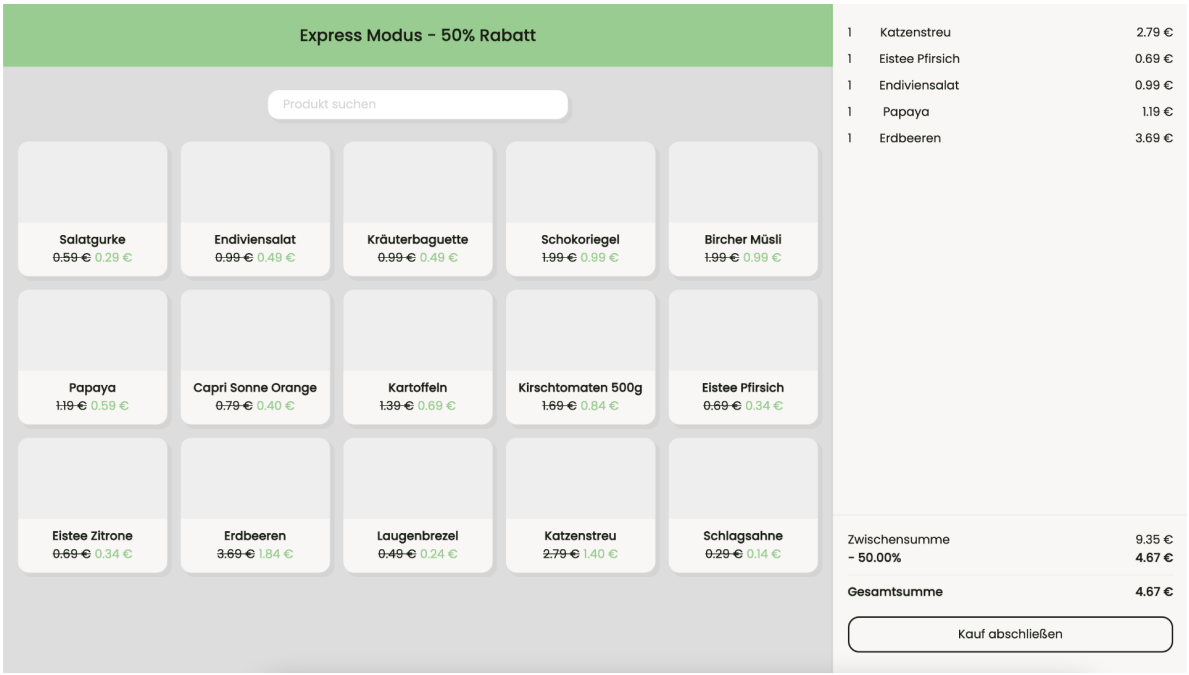

Abb. 18: Wireframe Ansicht eines Verkaufsvorgangs

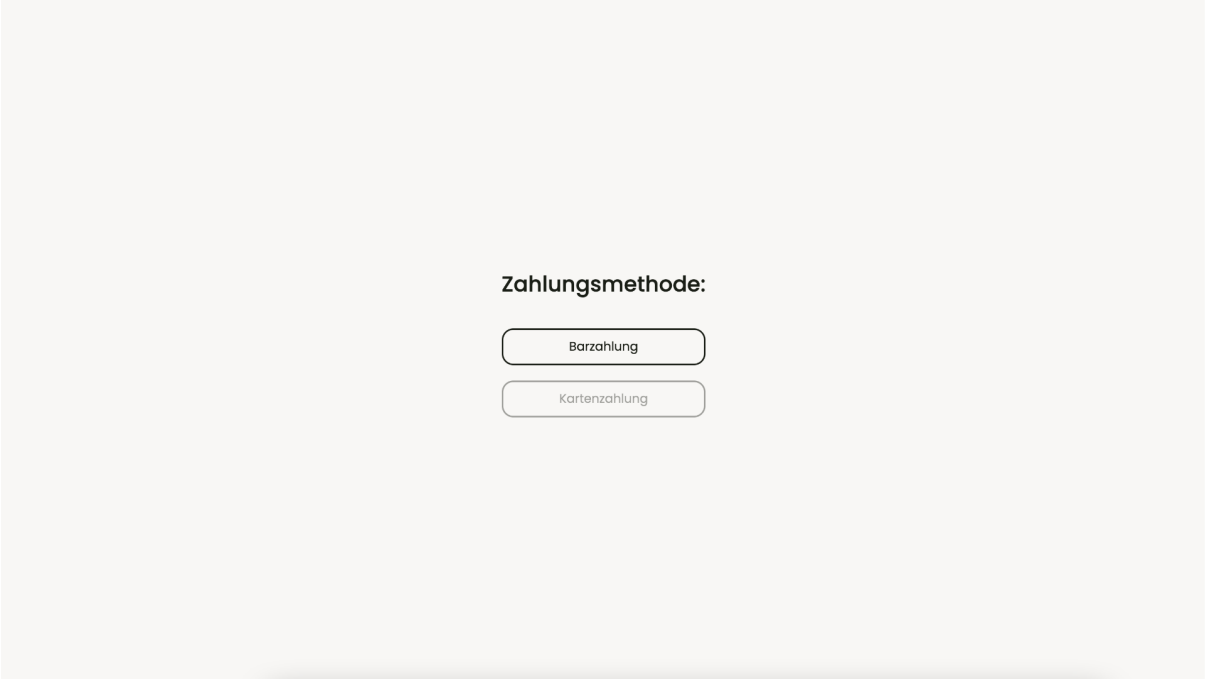

Abb. 19: Wireframe: Ansicht Zahlungsmethode auswählen

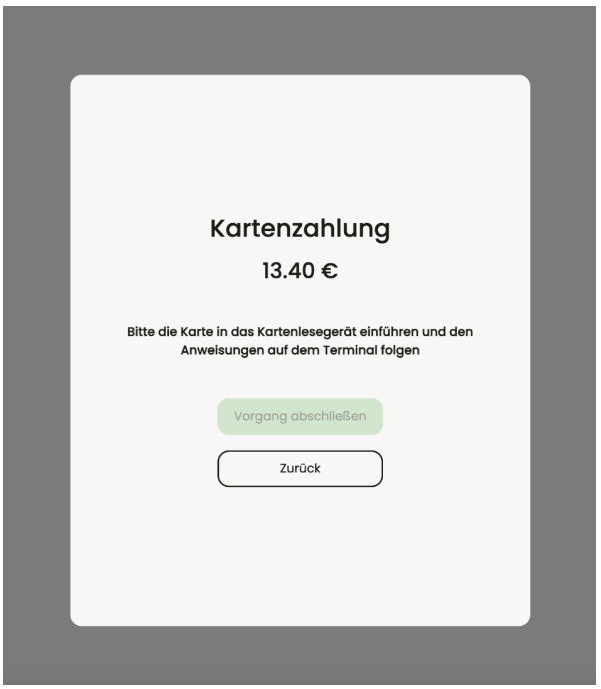

Abb. 20 und 21: Wireframe: Ansicht nach Auswahl einer Zahlungsmethode

Store

Das Store Interface soll dem Filialleiter dazu dienen Geschäftsinterne Prozesse zu überwachen, den Filialbestand zu überprüfen und neue Waren zu bestellen. Zudem soll der Leiter die Möglichkeit haben den Verkaufspreis eines Produkts anzupassen.

Die einzelnen Komponenten der Anwendung greifen auf den StoreStateService zu, der den modul internen State als Observables nach außen bereitstellt und nur selbst Änderungen auf dem State vornehmen kann. Zudem beinhaltet der Service alle Schnittstellenfunktionen zur Kommunikation mit der API. Um die aktuelle Filiale zu ermitteln, greift der Service auf die aus dem JWT Token ausgelesenen Nutzerinformationen zu. Wenn das Token eine Store ID enthält, werden die nach außen bereitgestellten Filial Informationen mit den Schnittstellendaten synchronisiert,

Der interne State verwaltet den Filialbestand, die getätigten offenen Bestellungen und Produktaustausche mit anderen Filialen sowie, um eine neue Bestellung initiieren zu können, ähnlich wie bereits der Cash Desk State einen Warenkorb.

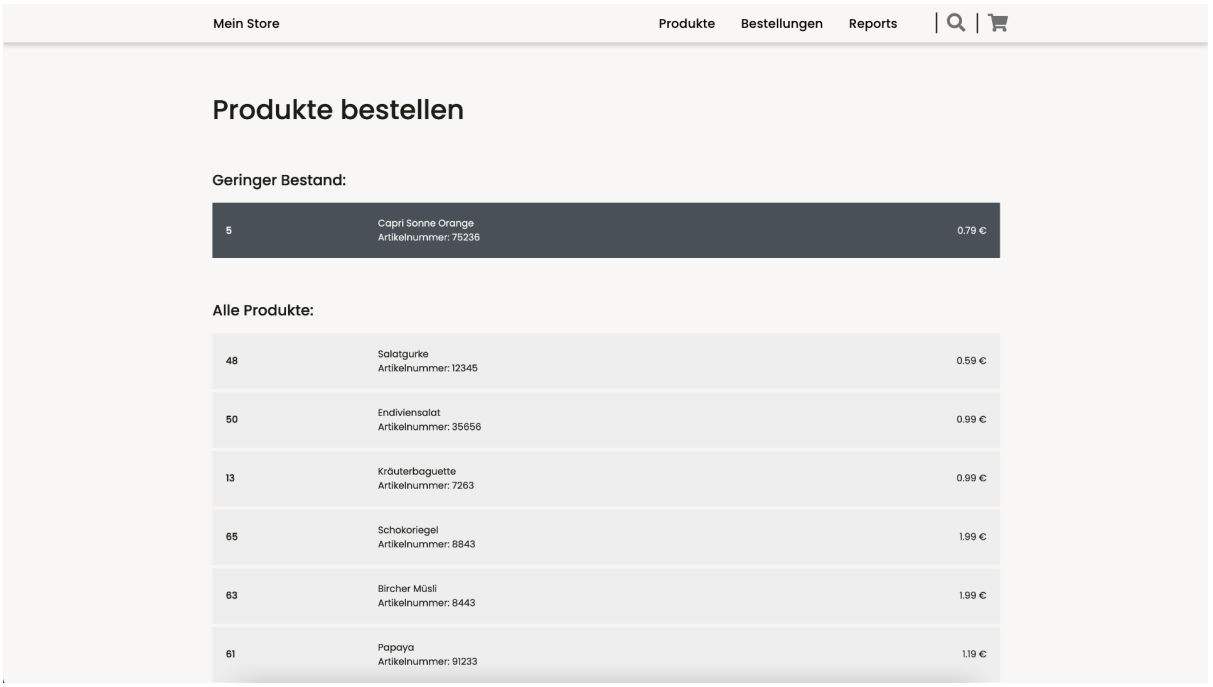

Abb. 22: Wireframe: Ansicht Produkte bestellen und geringen Bestand einsehen

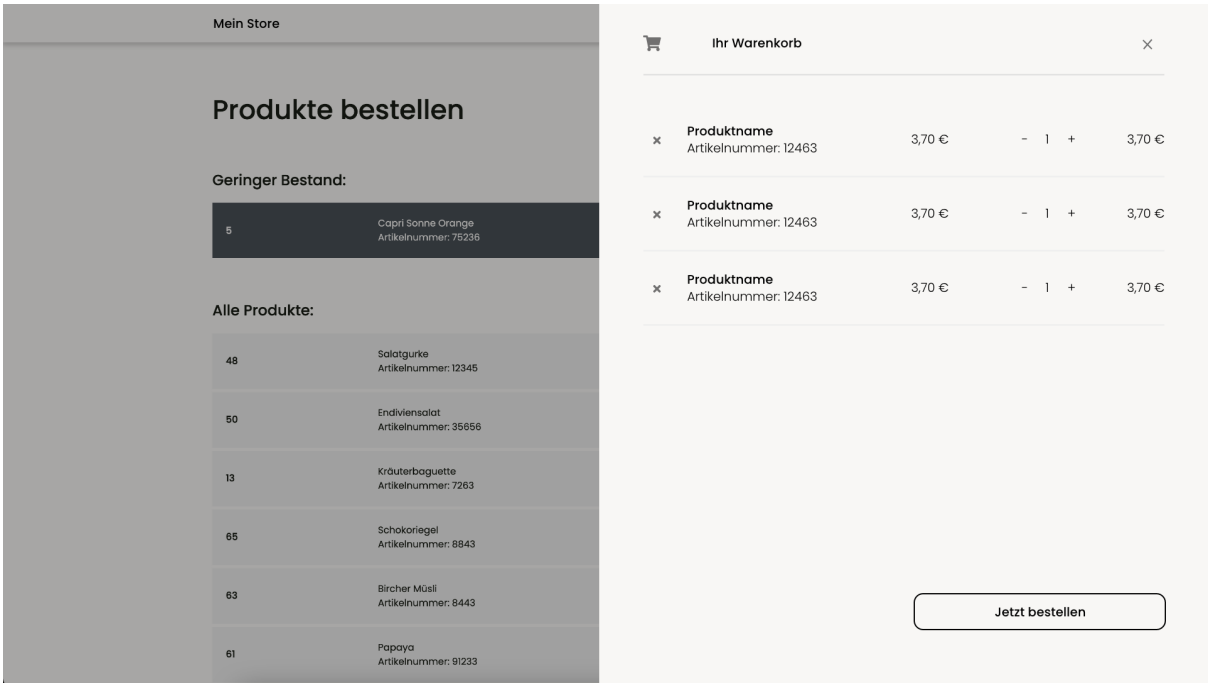

Abb. 23: Wireframe: Ansicht Produkte in den Warenkorb legen

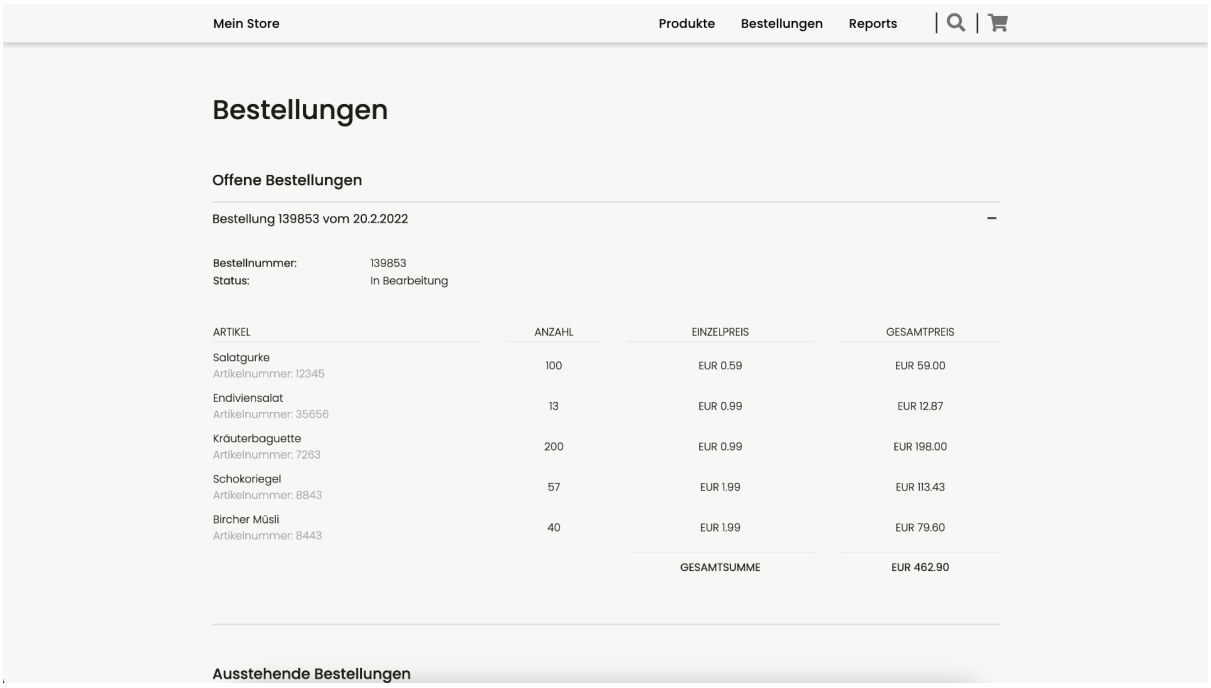

Abb. 24: Wireframe: Getätigte Bestellungen ansehen

Enterprise

Das Interface des Enterprise Clients soll nach der Vorlage des CoCoME dem Administrator die Funktionen bereitstellen Reports abzufragen, die die Lieferzeiten der einzelnen Lieferanten aggregiert.Neben dieser einzigen Funktion, die das Common Component Example für das Enterprise System vorsieht, haben wir für eine geschlossene, sinnhafte Anwendung weitere Anwendungsfälle hinzugefügt. So ist es dem Administrator möglich neue Lieferanten, Filialen und Produkte hinzuzufügen und Einträge zu bearbeiten. Zu einem Produkt kann abgefragt werden, in welchen der Filialen es verfügbar ist und als neues Stock Item einer Filiale hinzugefügt werden. Neben den Reports über die Lieferzeiten der Lieferanten kann der Enterprise Manager den aktuellen Jahresumsatz alles Filialen einsehen.

Zudem haben wir ein User Management ergänzt, sodass es dem Administrator ermöglicht wird, manuell Benutzer hinzuzufügen und Rechte zu verteilen. Innerhalb des Bereiches können jedoch nur die Zugehörigkeit zu einer Filiale sowie die Nutzungsrechte der Applikation geändert werden. Sonstige Änderungen personenbezogener Daten sollte der Benutzer selbst übernehmen.

Auch das Enterprise Modul ist sehr ähnlich zu dem Cash Desk und Store Modul aufgebaut. Ein StateService stellt den Komponenten innerhalb des Moduls Anbindungen zur Schnittstellen bereit und verwaltet einen inneren Status.