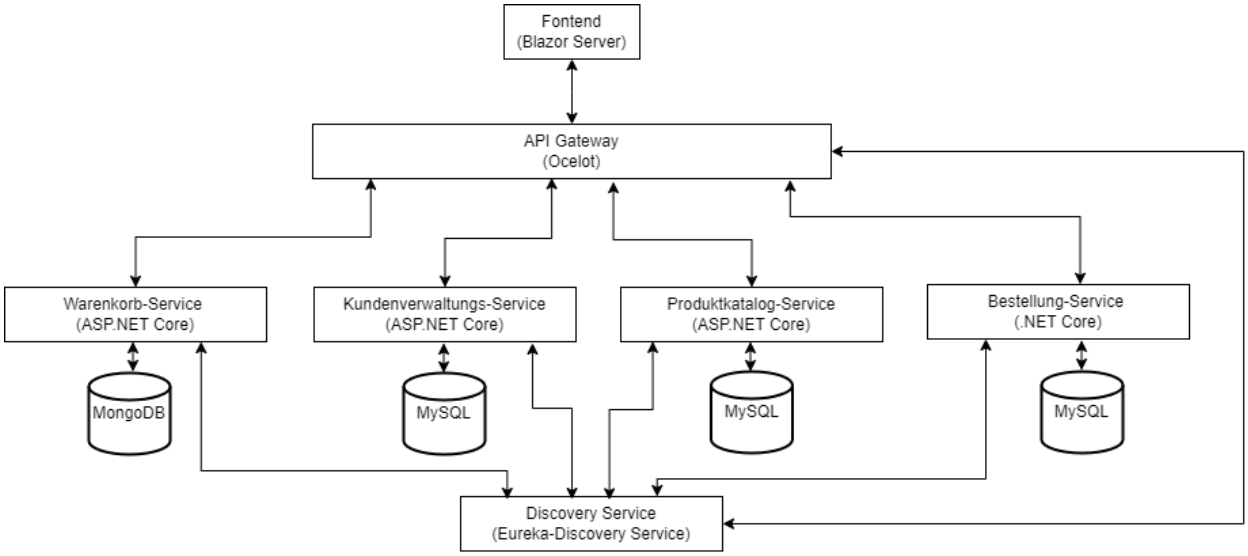
Projektdefinition

In diesem Dokument werde ich beschrieben, wie meine Microservice Shop Anwendung aufgebaut sein wird. Ich werde die Applikation nicht mit Java und Springboot umsetzten, da ich im Betrieb mit C-Sharp programmiere und auch schon an einer Microservice Applikation gearbeitet habe, die in C-Sharp umgesetzt wird. Zudem werde ich kein Messaging mit Kafka umsetzten, da sich, dass Dokument mit den Vorgaben selbst widerspricht. Somit werden alle Microservices über REST-APIs kommunizieren.

Das Architekturdiagramm sieht wie folg aus:



Frontend

Der Frontend-Service wird mit Blazor Server umgesetzt und kommuniziert über das API-Gateway mit den Backend-Services. Die folgenden Seiten werden verfügbar sein:

- Login/Registrierung
- Produktübersicht (Seite mit den zu kaufenden Produkten)
- Warenkorb
- Bestellübersicht (Seite, auf der Bestellungen eingesehen werden können)

Falls noch genügend Zeit zur Verfügung steht, werde ich das Frontend ausbauen.

API-Gateway

Beim API-Gateway habe ich mich für Ocelot entschieden, da es in C-Sharp geschrieben ist und für Services in C-Sharp ausgerichtet ist. Das Gateway dient zur Kommunikation zwischen dem Frontend und den Backend Services.

Discovery Service

Der Discovery Service ermöglicht es, dass sich die anderen Services automatisch finden und miteinander kommunizieren können, ohne dass die spezifischen Netzwerkadressen der Services hartkodiert sein müssen. Hierfür müssen sich alle Backend Services bei Eureka regrisitrieren.

Produktkatalog-Service

Der Produktkatalog-Service wird mit ASP.NET Core umgesetzt und verwaltet Informationen zu Produkten (Namen, Beschreibungen, Preise und Lagerbestände). Der Service wird eine eigen MySQL Datenbank zur verfüg haben, in der die Informationen zu den Produkten abspeichert. Der Service wird folgende Endpunkte haben:

Methode	**Endpunkt**	**Beschreibung**
GET	/api/products	Abrufen der gesamten Produktliste
GET	/api/products/{id}	Abrufen eines einzelnen Produkts anhand der ID
POST	/api/products	Hinzufügen eines neuen Produkts
PUT	/api/products/{id}	Aktualisieren eines Produkts anhand der ID
DELETE	/api/products/{id}	Entfernen eines Produkts anhand der ID

Bestellung-Service

Der Bestellverwaltung-Service wird mit ASP.NET Core umgesetzt und übernimmt die Verarbeitung von Kundenbestellungen. Dabei werden Details, wie Datum der Bestellung, Totaler Preis und Artikel der Bestellung in einer MySQL Datenbank gespeichert. Der Service wird folgende Endpunkte haben:

Methode	**Endpunkt**	**Beschreibung**
POST	/api/orders	Erstellen einer neuen Bestellung
GET	/api/orders/{id}	Abrufen der Details einer einzelnen Bestellung anhand der ID
PUT	/api/orders/{id}	Aktualisieren des Status einer Bestellung anhand der ID

Kundenverwaltungs-Service

Der Kundenverwaltungs-Service wird ebenfalls mit APS.NET Core umgesetzt und übernimmt die Verwaltet Kundeninformationen wie Namen, Adressen und Kontaktinformationen. Die Daten werden einer MySQL Datenbank abgespeichert. Die Sicherheit des Services wird durch den Einsatz von JWT (JSON Web Token) für Authentifizierung und Autorisierung gewährleistet. Beim Login-Request sendet der Kunde seinen **Benutzernamen** und sein **Passwort**. Diese Anmeldedaten werden überprüft, und bei erfolgreicher Authentifizierung wird ein JWT-Token generiert und zurückgegeben. Dieses Token wird bei nachfolgenden API-Anfragen in der Autorisation-Header gesendet, um den Zugriff auf geschützte Endpunkte zu erlauben. Der Service hat folgende Endpunkte:

Methode	**Endpunkt**	**Beschreibung**
POST	/api/auth/login	Authentisiert Kunde und generiert ein JWT
GET	/api/users/{id}	Abrufen von Informationen eines einzelnen Kunden anhand der ID
PUT	/api/users/{id}	Aktualisieren der Daten eines vorhandenen Kunden

Warenkorb-Service

Der Warenkorb-Service wird ebenfalls mit APS.NET Core umgesetzt und ermöglicht Kunden das Hinzufügen und Entfernen von Produkten in einem virtuellen Warenkorb. Die Daten werden einer MongoDB Datenbank abgespeichert. Der Service hat folgende Endpunkte:

Methode	**Endpunkt**	**Beschreibung**
POST	/api/orders	Erstellen einer neuen Bestellung
GET	/api/orders/{id}	Abrufen der Details einer einzelnen Bestellung anhand der ID
PUT	/api/orders/{id}	Aktualisieren des Status einer Bestellung anhand der ID