Photo Office

Für
Fotografen und
Fotografinnen
(vice versa)

Betriebshandbuch

Dokument Version: 1.0, 25.09.2018
Autor: Dirk Goldbach

Inhaltsverzeichnis

1. Über dieses Dokument

Dieses Dokument beschreibt die Zusammenhänge und Abhängigkeiten der Komponenten von photocrm.

2. Hardwareanforderungen

- 32 GB RAM
- Plattenkapazität abhängig vom Datenvolumen; mind. 16 GB für photocrm
- Intel Core i5 7th Generation oder vergleichbares – idealerweise mind. 4 Kerne

Linux wird empfohlen, ist aber nicht zwingend notwendig.

3. Grundsätzliches

Das Back-End von photocrm basiert auf Microservices (SOAP). Dabei wird in *Domain Microservices* und *Entity Microservices* unterschieden. Die *Domain Mircoservices* nutzen die Web-Aktionen der *Entity Microservices*, sind also abhängig von ihnen.

Die Daten werden in einer Relationalen Datenbank verwaltet.

4. Datenbank

Datenzugriffe erfolgen ausschließlich durch die Microservices, also das Back-End. Die Zugriffe basieren auf einem Abstraktionslayer (JPA) und sind – theoretisch – datenbankunabhängig.
Bisher wurde mit **mariadb** gearbeitet und getestet.

Die Datenbankadministration ist nicht Teil dieses Dokuments, da lediglich gewährleistet sein muss, dass das Backend Zugriff auf die DB hat und die im Backend konfigurierten Benutzer/Passwörter gültig sind.

5. Microservices (Back-End)

5.1. Start

Die Services existieren als Java-Jar-Dateien und werden mit der Java Laufzeitumgebung gestartet.
Format: java -jar <service.jar>

Zusätzlich können Parameter zum Start angegeben werden, mit denen zum Beispiel die Datenbank der Port des Dienstes oder die URL eines verwendeten anderen Dienstes verändert werden.
Format: java -jar <service.jar> --<Parameter>

Alternativ könnten die Services als War-Dateien zur Verfügung gestellt und in einem Webserver deployed werden. Dies würde aber möglicherweise zu Lasten der Flexibilität und Skalierbarkeit des Gesamtsystems gehen.

Welche Parameter zur Verfügung stehen, ist im Absatz Konfiguration nachzulesen.

5.2. Abhängigkeiten

Zwischen den Microservices bestehen Abhängigkeiten. *Domain Microservices* greifen auf die *Entity Microservices* zu. Damit das Gesamtsystem korrekt arbeitet, müssen alle Dienste gestartet sein. Die Reihenfolge ist dabei nicht entscheidend, da die Abhängigkeit sich erst mit dem Aufruf der Web-Aktionen, also bei Nutzung des Systems auswirkt.

Allerdings muss die Erreichbarkeit der *Entity Microservices* für die *Domain Microservices* gewährleistet sein. Dafür müssen die entsprechenden Parameter für den Start der Domain Microservices korrekt angeben werden (s. Konfiguration).

5.2.1. Aufschlüsselung der Abhängigkeiten

Domain Microservice… abhängig von →	**Entity Service**
domain-service-organization	entitiy-service-organization entity-service-person entity-service-address entity-service-communication

5.3. Konfiguration

Die wichtigsten Parameter dienen
- der Erreichbarkeit der Microservices selbst
- dem Zugriff der Microservices auf andere Microservices
- dem Zugriff auf die Datenbank

5.3.1. Alle Services

Erreichbarkeit der Microservices selbst:
Wichtig ist, dass für jeden Service ein eigener Port anzugeben ist, auf den der Service lauscht. Identische Ports führen zu Konflikten bzw. lassen sich nicht ausführen.

--server.port=<Port>
Gibt den Port an, auf dem die Endpoints (Web-Actions) erreichbar sind.

Beispiel: --server.port=8080

5.3.2. Entity Microservices

Zusätzlich zur Port-Einstellung muss für die Daten Services die Datenbankverbindung konfiguriert werden.

Datenbankkonfiguration:
--spring.datasource.url=<url db incance und schema>
--spring.datasource.username=<user>
--spring.datasource.password=<user password>
--spring.datasource.driver-class-name=<database driver> (s.a. Maven POM)
--spring.jpa.hibernate.ddl-auto=<database action>

Beispiel:
--spring.datasource.url=jdbc:mariadb://localhost:3306/db_example
--spring.datasource.username=horst
--spring.datasource.password=horsts-passwort
--spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
--spring.jpa.hibernate.ddl-auto=update

Achtung! Der Konfigurationswert *spring.jpa.hibernate.ddl-auto* steuert, welche Aktion der Service beim Start auf die Datenbanktabellen ausübt. Mit diesem Parameter kann bei falscher Konfiguration die Tabelle bei Beenden des Service gelöscht werden.
Der Parameter *update* sorgt dafür, dass Schema-Änderungen in der Code-Basis auch auf die Datenbank angewendet werden.
Grundsätzlich sind diese Werte nur nach exakter Planung und Abstimmung mit dem Operations-Verantwortlichen durchzuführen.

5.3.3. Domain Microservices

Die Domänen Services greifen nicht direkt auf die Datenbank zu, sondern bedienen sich der Daten Services. Um die Datenservices zu erreichen sind diese Parameter zusätzlich zum Port zwingend und korrekt einzurichten:

--webclient.<entity>.port=<port>
--webclient.<entity>.host=<host>

Dabei sind diese Werte für alle verwendeten Entity Microservices einzutragen.

Beispiel domain-service-organization:
--server.port=8040

--webclient.person.port=8050
--webclient.person.host=localhost

--webclient.organization.port=8060
--webclient.organization.host=localhost

--webclient.communication.port=8070
--webclient.communication.host=localhost

--webclient.address.port=8080
webclient.address.host=localhost

6. Web-Frontend

Ist noch komplett offen

7. Skalierbarkeit

Da Skalierbarkeit in der Regel der Lastverteilung dient, kann vorab gesagt werden, dass es bei entsprechend hoher Auslastung des Systems eine Aufteilung des Web-Frontends

7.1. Skalierung der Microservices

Die Microservices des Back-Ends können mehrfach und auf verschiedenen Servern laufen. Ihre Erreichbarkeit wird durch die beim Start angegebenen Parameter gewährleistet.

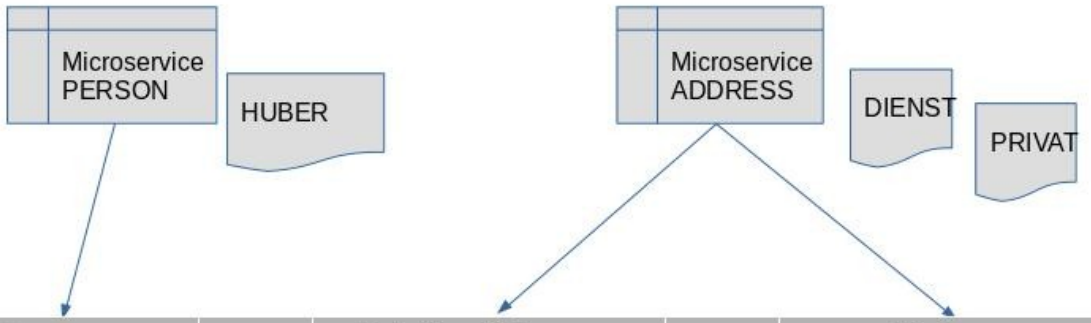
7.2. Skalierung der Datenbanken

Die Daten der Services sind hart voneinander getrennt, es existiert keine **direkte** relationale Abhängigkeit untereinander. Grundsätzlich ist es daher möglich, dass die Services unterschiedliche Datenbank-Instanzen nutzen oder sogar Services des gleichen Typs auf verschiedene Datenbank-Instanzen zugreifen.

Die zweite Variante – *Entity Microservices* gleichen Typs verwenden verschiedene Datenbank-Instanzen – ist aber nur mit Einschränkungen möglich, da zwar keine direkte (relationale Integrität) Abhängigkeit besteht aber sehr wohl Beziehungen zwischen den Entitäten (zum Beispiel Adressen und Personen). Diese Zuordnungen können nicht hergestellt werden, wenn die *Entity Microservices* falsch konfiguriert sind, also auf eine ‚falsche' Datenbank-Instanz zugreifen.

Das selbe gilt für den Umzug eines *Entity Microservice* auf eine andere Datenbank-Instanz bzw. ein anderes Datenbank-Schema. In diesem Falle ist für die Migration der Tabellen, die die Beziehungen darstellen, unbedingt Sorge zu tragen.

Beispielhafte Darstellung der Beziehung zwischen Adressen und Personen und deren Realisierung in den *Entity Microservices* (MS):



Personen			Relation Adressen			Addressen		
ID	Name		ID Person	ID Addresse		ID	Typ	Ort
5	Huber		5	77		77	PRIVAT	Hagen
12	Schmitt		12	61		78	DIENST	Köln
...			5	78		90	P-BOX	Wien
...			...	...		61	PRIVAT	Neuss

Würde der *Entity Microservice* Address auf eine andere Datenbank-Instanz ‚umziehen', wäre die Beziehung zwischen den Personen und Adressen nicht mehr reproduzierbar, solange nicht auch eine Migration der Relations-Tabelle erfolgt.
Ähnliches gilt für zwei parallel aktive Instanzen des Microservice Address. Daraus folgt, dass eine Skalierung der Datenbank nur sinnvoll ist, solange die Zuordnung *Entity Microservice* und Datenbank-Instanz (und -Schema!) nicht verändert wird.

Abhilfe könnte hier eine Echtzeit-Synchronisation zwischen den Datenbanken schaffen, die die Abhängigkeitstabellen stets auf dem aktuellen Stand hält.

7.3. Skalierung des Web-Frontends

Offen

8. Sicherheit

Der Datentransfer zwischen Web-Clients und Web-Service erfolgt verschlüsselt.

Für den Zugriff auf die Web-Dienste muss der Web-Client autorisiert sein (personenbezogene Anmeldung). Die Autorisierung (Anmeldung) ist zeitlich begrenzt beziehungsweise erlischt, sobald sich der Web-Client vom Dienst abmeldet.

Passwörter von Benutzern werden verschlüsselt abgelegt und sind auch durch Administratoren nicht lesbar.

Alle durch das System (photocrm) verwalteten Daten werden in einer Datenbank abgelegt. Da es sich bei diesen Daten teilweise um sensible und personenbezogene Daten der Nutzer selbst und derer Kunden handelt, dürfen ausschließlich autorisierte Administratoren Zugriff auf die Datenbank haben. Die Autorisierung setzt eine Sicherheitsüberprüfung der Person sowie die schriftliche Kenntnisnahme ihrer Verpflichtungen voraus.

Das Hosting des Systems muss in einem entsprechend zertifiziertem Rechenzentrum erfolgen, das alle im Zusammenhang mit sensiblen und personenbezogenen Daten stehenden rechtlichen Verpflichtungen erfüllt.

Um das System vor Angriffen von außen zu schützen können diverse Maßnahmen ergriffen werden. Die einfachste ist, das Web-Frontend, das ja ‚offen‘ im Web steht, in eine Demilitarisierte Zone zu stellen und die Datenbank sowie die Microservices in anderen Subnetzen zu installieren.

Zusätzlich könnte eine Absicherung der Web-Dienste erfolgen, indem auffällige Aufrufe der Schnittstellen überwacht und bei Verdacht die Verbindung zum Client unterbrochen wird bzw. die IP des aufrufenden Client in eine Blacklist überführt wird.