DAM 2.0

(Arbeitstitel)

Microservices

Autor: Dirk Goldbach

Konzept

Grundlagen

Services

Konzept

Architektur

Der Client (Smartphone) nutzt Funktionen, die auf einem oder mehreren Servern zur Verfügung gestellt werden. Die Funktionen werden auf Microservices aufgeteilt.

Der Zugriff auf die Funktionen erfolgt über einen zentralen Service (Gateway). Der Client hat keinen Zugriff auf die Microservices mit Ausnahme des Gateways → serviceProvider

Für einen ersten Überblick s.a. Grafik.

Vorteil: Das Gateway kann auf einem Server installiert werden, der als Demilitarisierte Zone gilt. Somit können Angriffe von außen ausschließlich auf das Gateway erfolgen.

Unter Verwendung von Containern (Docker + Kubernetes) ist das Gesamtsystem bei Bedarf skalierbar.

API, Funktionale Zugriffe

Die Summe der Microservices, gebündelt durch den zentralen Service → serviceProvider, bildet ein API. Die Microservices wurden als Webservices implementiert. Zum Austausch der Informationen zwischen Client und Server (Request/Response) wird das Datenformat **Json** genutzt.

Achtung, zur Vereinfachung und Vereinheitlichung wurden alle Methoden der Webservices als **POST** realisiert.

Datenhaltung

Die Persistierung von Daten erfolgt in einem relationen Schema, aktuell MariaDB.

Durch den Einsatz eines ORB (Hibernate) wurde der Datenzugriff abstrahiert, möglicherweise sind aber Anpassungen bei Wechsel des DBMS notwendig.

Authorisierung, Benutzer-Sicherheit

Mit Ausnahme weniger Web-Methoden (PING, Login, …), muss der Benutzer, in dessen Name eine Aktion ausgeführt wird, registriert und angemeldet sein (username, password).

API

Zugriff

Die Webservices bilden in Summe eine API in Form einer synchronen Schnittstelle ab.

Der Zugriff auf die API erfolgt über eine URL und einen vorgegebenen Port.

Für erste Tests empfiehlt sich ein geeignetes Tool, wie z.B. Insomnia, mit dem sehr einfach Json-formatierte Nachrichten an den serviceProvider gesendet werden können.

Der Datenaustausch zwischen Client und Server erfolgt im Json-Format.

Nutzung

Wie bereits erläutert, erfolgen alle Zugriffe über einen zentralen Service → serviceProvider.

Der Aufruf der Web-Methoden verläuft nach folgendem Schema:

1. Anmeldung des Clients mit user und password
   Das System liefert ein Token für die nachfolgenden Aktionen zurück
2. Nutzung einer oder mehrerer Web-Methode unter Angabe des Token und des users
3. Logout mit Token und user

Erfolgt über einen bestimmten (im System festgelegten) Zeitraum kein Zugriff auf die API, verfällt das Token und der Benutzer muss neu angemeldet werden.

Für die Verschlüsselung des Passworts ist der Client verantwortlich. Unverschlüsselte Passwörter dürfen nicht übertragen werden.

Services

In dieser ersten Version wird zunächst lediglich der serviceProvider im Sinne einer API beschrieben. Kenntnisse über die übrigen Services sind für die Nutzung der Schnittstelle nicht relevant, sollten aber natürlich im Sinne eines Wissenstransfers bei mehreren Entwicklern vorhanden sein.

Daher empfiehlt sich dringend die Vervollständigung dieser Dokumentation.

ServiceProvider, Gateway

Aufgaben

- Zentraler Dienst für den Client (nur ein Port).
- Bietet die vom Client benötigten Web-Methoden des Gesamt-Systems an.
- Fungiert als Gateway zum Schutz gegen Angriffe von außen

Allgemeine Rückgabewerte

Die meisten Methoden liefern Ergebniswerte in numerischer und/oder textueller Form zurück. Aktuell basieren die numerischen Results auf reiner Willkür, d.h. eine entsprechende Tabelle ist noch zu erstellen und die Werte müssen in die Software übernommen werden.

Die Tabelle mit den numerischen Ergebniswerten ist idealerweise Bestandteil dieses Dokuments.

Web-Methoden

Die URL setzt sich zusammen aus dem Schema
<host-address>:<port><path>

Beispiel: localhost:9090/user/create

Login

Notwendig für die Nutzung der übrigen Web-Methoden. Mit der Antwort erhält der Client ein Token, das für nachfolgende API-Aufrufe benötigt wird.

Path

/login

Request

Key	Value	Erläuterung
userName	Benutzername	z.B. minimaster
password	Passwort	Muss vom Client verschlüsselt werden

Beispiel Request

```
{
  "userName": "Rudi",
  "password": "test"
}
```

Response

Key	Value	Erläuterung
returnCode	<0 … 9999>	Numerischer Wert
description	Ausführliche Info	Status des Aufrufs (Erfolg/Nichterfolg)
result	Kurzinfo	Status des Aufrufs
tokenId	Token	Token für spätere Aufrufe
userName	Benutzername	Kontrollmöglichkeit
userId	ID des Nutzers in Persistenz	Kontrollmöglichkeit

Beispiel Response

```
{
  "returnCode": 0,
  "description": "User or Token validated",
  "result": "OK",
  "tokenId": "0000000003",
  "userName": "Rudi",
  "userId": 8
}
```

Logout

Meldet den Benutzer ab und macht das Token ungültig.

Path

/logout

Request

Key	Value	Erläuterung

userName	Benutzername	z.B. minimaster
tokenId	Token	

Beispiel Request

```
{
  "userName": "Rudi",
  "token": "0000000003"
}
```

Response

Key	Value	Erläuterung
returnCode	<0 … 9999>	Numerischer Wert
description	Ausführliche Info	Status des Aufrufs (Erfolg/Nichterfolg)
result	Kurzinfo	Status des Aufrufs
tokenId	Token	Bisher verwendetes Token
userName	Benutzername	Kontrollmöglichkeit
userId	ID des Nutzers in Persistenz	Kontrollmöglichkeit

Beispiel Response

```
{
  "returnCode": 0,
  "description": "Rudi",
  "result": "Logout successfull",
  "tokenId": "0000000003"
}
```

PING

Liefert Informationen über das Gateway und die Microservices. Listet dazu alle Services und ihren aktuellen Status auf.

Path

/ping

Request

Key	Value	Erläuterung

Beispiel Request

```
{ }
```

Response

Key	Value	Erläuterung
returnCode	<0 … 9999>	Numerischer Wert
description	Ausführliche Info	Status des Systems
message	Kurzinfo	Status des Systems (OK, WARNING, ERROR)
serviceInfos	Array	Liste mit Zuständen der Services (s.u.)

Liste mit Service-Informationen

Key	Value	Erläuterung
service	Name des Microservice	
uptime	Laufzeit des Dienstes	In Millisekunden
systime	Datum des Servers	… auf dem der Dienst läuft
hostName	Name des Servers	… auf dem der Dienst läuft
hostAddress	Netzadresse des Servers	… auf dem der Dienst läuft
status	Zustand des Dienstes	

Es können zusätzliche Daten veräußert werden, die allerdings im Kontext mit dem Ping nicht relevant sind.

Beispiel Response

```
{
  "message": "WARNING",
  "description": "Not all required Microservices are reachable",
  "returnCode": 111,
  "serviceInfos": [
    {
      "hostName": "localhost-live.fritz.box",
      "service": "ServiceProvider",
      "systime": "Fri Jul 26 16:57:02 CEST 2019",
      "hostAddress": "2001:4dd6:d5d:0:79ed:b936:828f:7355",
      "status": "OK",
      "uptime": "18307ms"
    },
    {
```

```
      "hostName": "localhost-live.fritz.box",
      "service": "UserRepository",
      "systime": "Fri Jul 26 16:57:02 CEST 2019",
      "hostAddress": "2001:4dd6:d5d:0:79ed:b936:828f:7355",
      "status": "OK",
      "uptime": "13457ms",
      "tokenId": null
    },
    {
      "service": "AuthenticationService",
      "status": "NO RESPONSE"
    }
  ]
}
```

CreateUser

Dient dem Anlegen eines Users.

Path

/user/create

Request

Key	Value	Erläuterung
userName	Benutzername	z.B. minimaster
password	Passwort	Muss vom Client verschlüsselt werden
givenName	Vorname	
lastName	Nachname	

Beispiel Request

```
{
  "user": {
    "userName": "superuser",
    "password": "!Super1Password2",
    "givenName": "Gucci",
    "lastName": "Guccio"
  }
}
```

Response

Key	Value	Erläuterung
returnCode	<0 … 9999>	Numerischer Wert
description	Ausführliche Info	Status des Aufrufs (Erfolg/Nichterfolg)
result	Kurzinfo	Status des Aufrufs
userName	Benutzername	Kontrollmöglichkeit
userId	ID des Nutzers in Persistenz	Kontrollmöglichkeit
givenName	Vorname	
lastName	Nachname	
password	Gewähltes Passwort	Durch den Client zu verschlüsseln

Beispiel Response

```
{
  "returnCode": 0,
  "description": "User created",
  "result": "OK",
  "user": {
    "userId": 1003,
    "userName": "supiusi",
    "givenName": "Dirk",
    "lastName": "Goldbach",
    "password": "test"
  }
}
```

UpdateUser

Dient dem Ändern der User-Daten.

Zum Ändern werden die Anmeldedaten benötigt sowie der zu ändernde user. Die Daten können aktuell nur durch den user selbst geändert werden.

Path

/user/update

Request

Key	Value	Erläuterung
tokenId	Token	Erhalten bei Login
userName	Angemeldeter Benutzer	Wurde angemeldet
userStored	Persistenter Benutzer	
userStored.userName	Persistierter Benutzername	Der zu ändernde Benutzer
userUpdate	Die neuen Daten	
userUpdate.userName	Der neue userName	
userUpdate.password	Das neue Passwort	
userUpdate.givenName	Der neue Vorname	
userUpdate.lastName	Der neue Nachname	

Beispiel Request

```
{
  "tokenId": "0000000003",
  "userName": "supiusi",
```

```
  "userStored": {
    "userName": "supiusi"
  },
  "userUpdate": {
    "userName": "superuser",
    "password": "test",
    "givenName": "Dirk",
    "lastName": "Goldbach"
  }
}
```

Response

Key	Value	Erläuterung
returnCode	<0 … 9999>	Numerischer Wert
description	Ausführliche Info	Status des Aufrufs (Erfolg/Nichterfolg)
result	Kurzinfo	Status des Aufrufs
user	Die neuen persistenten Daten	
user.userName	Benutzername	
user.givenName	Vorname	
user.lastName	Nachname	

Beispiel Response

```
{
  "returnCode": 0,
  "description": "User updated",
  "result": "OK",
  "user": {
    "userName": "superuser",
    "givenName": "Dirk",
    "lastName": "Goldbach"
  }
}
```

DropUser

Dient dem Löschen des Users.

Path

/user/drop

Request

Key	Value	Erläuterung
tokenId	Token	Erhalten bei Login
userName	Angemeldeter Benutzer	Wurde angemeldet
user	Persistenter Benutzer	Der zu löschende Benutzer
user.userName	Persistierter Benutzername	

Beispiel Request

```
{
  "tokenId": "0000000003",
  "userName": "Rudi",
  "user": {
    "userName": "Rudi"
  }
}
```

Response

Key	Value	Erläuterung
returnCode	<0 … 9999>	Numerischer Wert
description	Ausführliche Info	Status des Aufrufs (Erfolg/Nichterfolg)
result	Kurzinfo	Status des Aufrufs

Beispiel Response

```
{
  "returnCode": 0,
  "description": "User dropped",
  "result": "OK",
  "tokenId": "0000000003"
}
```