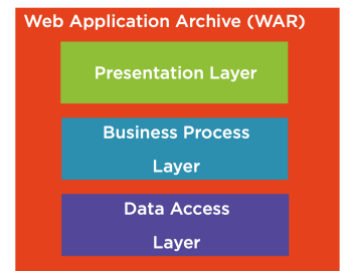


SpringBoot - Microservices – from Monolith to Microservices (Praxis)

Beschreibung

Wir werden nun hands on anlegen und eine Application von einer monolithic Arch. Zu einer MSA umschreiben.



2 Lektionen

Zeitbedarf:

Hilfsmittel:

Methode/Sozialform:

SpringBoot Monolith to MS



Lernziele:

- ✓ Monolithic Arch. Verstehen
- ✓ Microservice Arch. umschreiben
- ✓ Aufgaben

Legende:



Einzelarbeit,



Partnerarbeit,



Dokumentation,



Code,



Präsentation

Links:

Inhaltsverzeichnis

1	DECOMPOSITION STEPS	2
1.1	STEPS	2
1.2	BUGTRACKER - AUSGANGSLAGE	2
1.3	BUGTRACKER-BEFORE23	3
2	AUFGABEN	4
2.1	ENTWICKLUNG	4
2.2	COMPLEX RELATIONSHIPS BETWEEN MS (KNACKNUSS)	4
2.2.1	Define a FeignClient	5
2.3	TESTS	5

1 Decomposition Steps

Use the domain-driven design pattern to decompose a system!

1.1 Steps



Application
catalog



Ticket
management



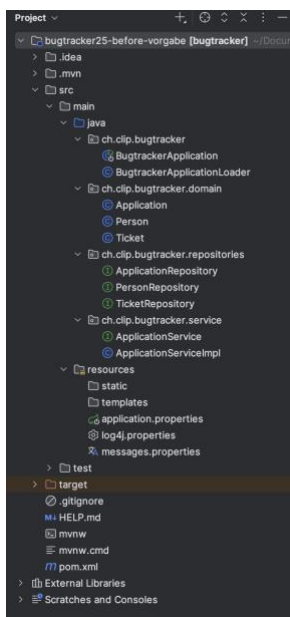
User
management

1. List of applications
 - Application Management Service
 - List of people
 - User Management Service
2. List of tickets by application
 - Ticket Management Service
3. List of assigned tickets by person
 - Ticket Management Service

1.2 Bugtracker - Ausgangslage

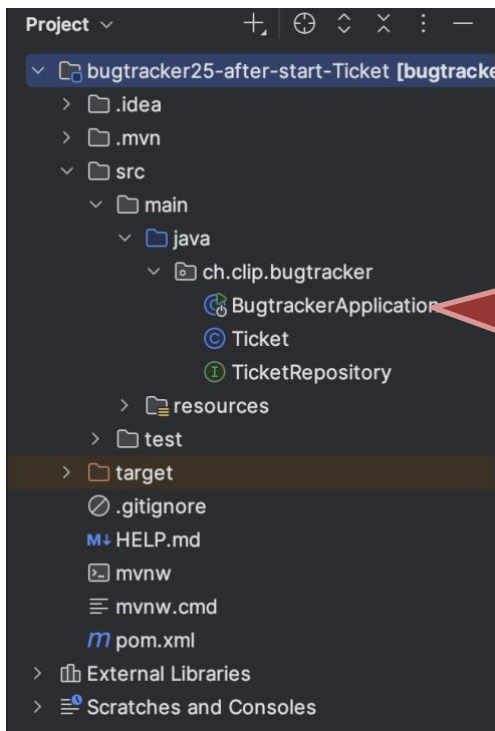
(TrackZilla)

Bugtracker-before definiert die alte monolithische Anwendung, es benutzt schon SpringBoot 3.3 und Java 17:



Domains: Applications, Persons, Tickets

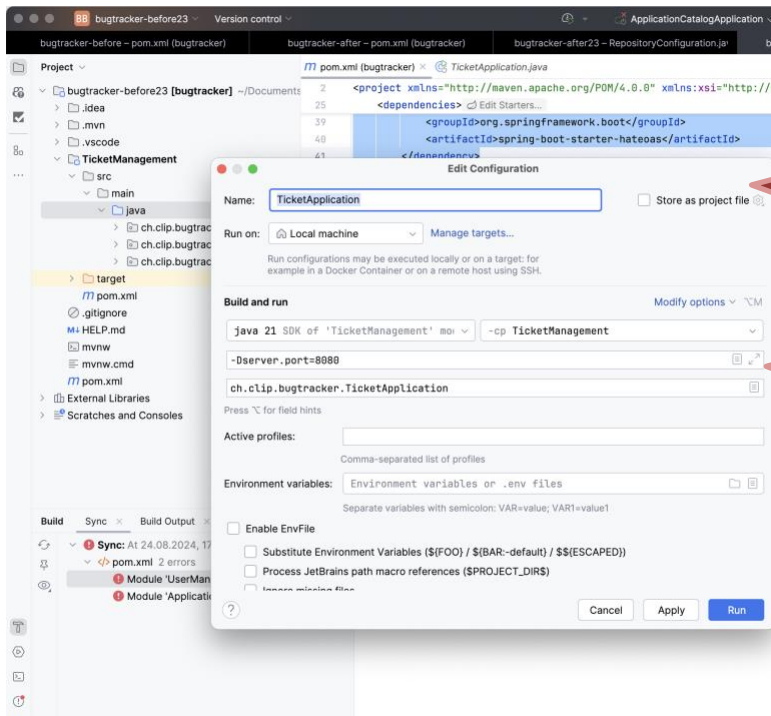
Bugtracker-after-start ist der Start für die neue MSA – TrackZilla !



Domains: Tickets

1.3 Bugtracker-before

Die Ausgangslage für die MSA der TrackZilla App:



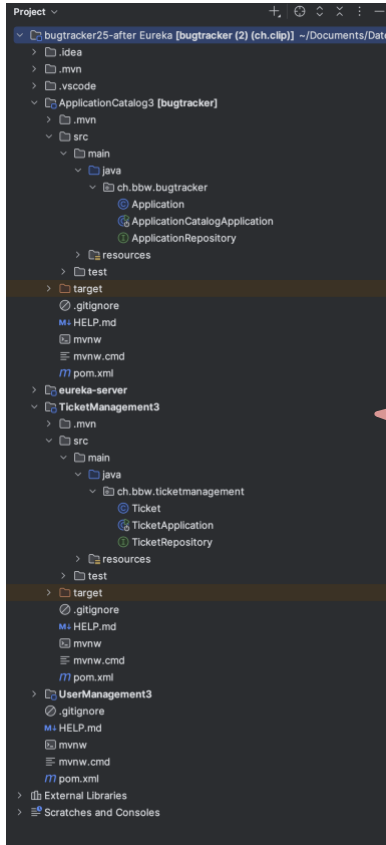
Wir erstellen jeden MS in einem IntelliJ Module mit eigener Run Configuration so können wir alle 3 MS in einem Projekt haben!

Jeder MS runnt auf einem anderen Port 8080, 81, 82

2 Aufgaben

2.1 Entwicklung

Erstellt nun für die anderen 2 MS ApplicationManagement und UserManagement analog zum TicketManagement diese MS in einem IntelliJ Module mit eigener Run Config!

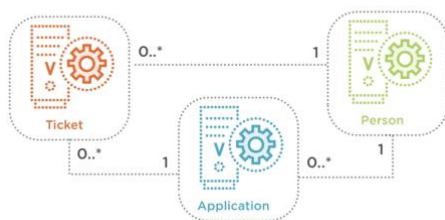


Neue MS Arch. Mit IntelliJ Modulen in einem Projekt!

2.2 Complex Relationships between MS (Knacknuss)

ApplicationCatalog has to call UserManagementService to . . .

Relationships Between Entities

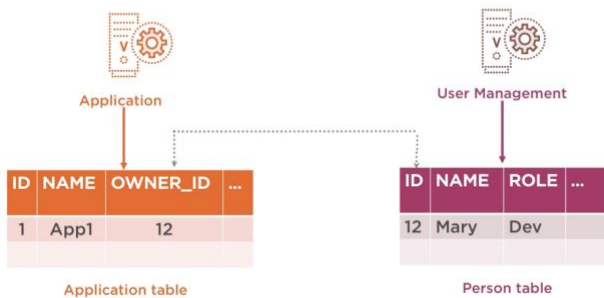


- Monolith Kommen wir nun zu einem Punkt in welchem wir eine Abfrage eines anderen Services machen müssen, in einem Monolith ist das kein Problem, wenn aber die DB's und MS getrennt sind kommt dies zu einer Herausforderung ;)

Synchronous Lookup between Application and UserManagement

Schaut euch dann den Feign Client an, das ist eine gute Lösung

Synchronous Lookup



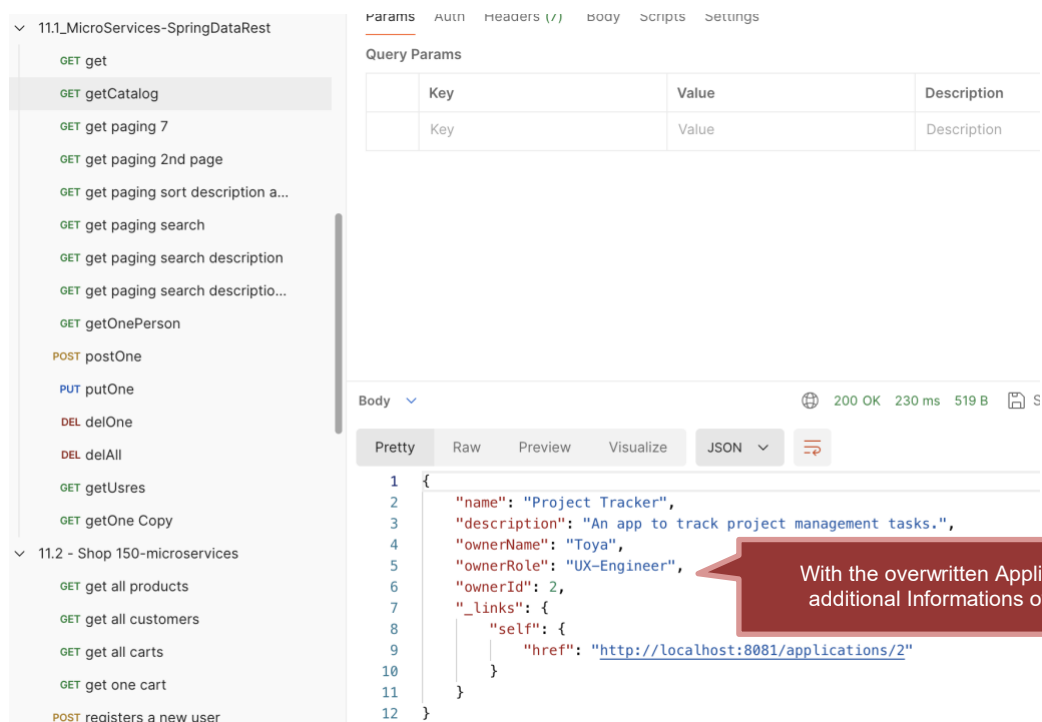
We want to retrieve the Name and the Role of the User with the Owner_id!

2.2.1 Define a FeignClient

Dieses wird später erläutert!

2.3 Tests

Nehmt die PostMan Collection 11.1_MicroServices-SpringDataRest.postman_collection
Und testet ausführlich die 3 MicroServices!



The screenshot shows the Postman interface. On the left, a collection named '11.1_MicroServices-SpringDataRest' is expanded, showing various endpoints. The 'GET getCatalog' endpoint is selected. On the right, the 'Query Params' section is empty. Below it, the 'Body' section shows a JSON response in 'Pretty' format. The response is a JSON object with the following structure:

```

1 {
2   "name": "Project Tracker",
3   "description": "An app to track project management tasks.",
4   "ownerName": "Toya",
5   "ownerRole": "UX-Engineer",
6   "ownerId": 2,
7   "_links": {
8     "self": {
9       "href": "http://localhost:8081/applications/2"
10    }
11  }
12 }

```

A red callout box points to the 'ownerName' and 'ownerRole' fields in the JSON response, stating: "With the overwritten ApplicationController you get the additional Informations ownerName and ownerRole".