Projekt: EPRO

Artur Mustaf, Nikita Stephan,

Norman Dobrovsky

RESTful Webservice

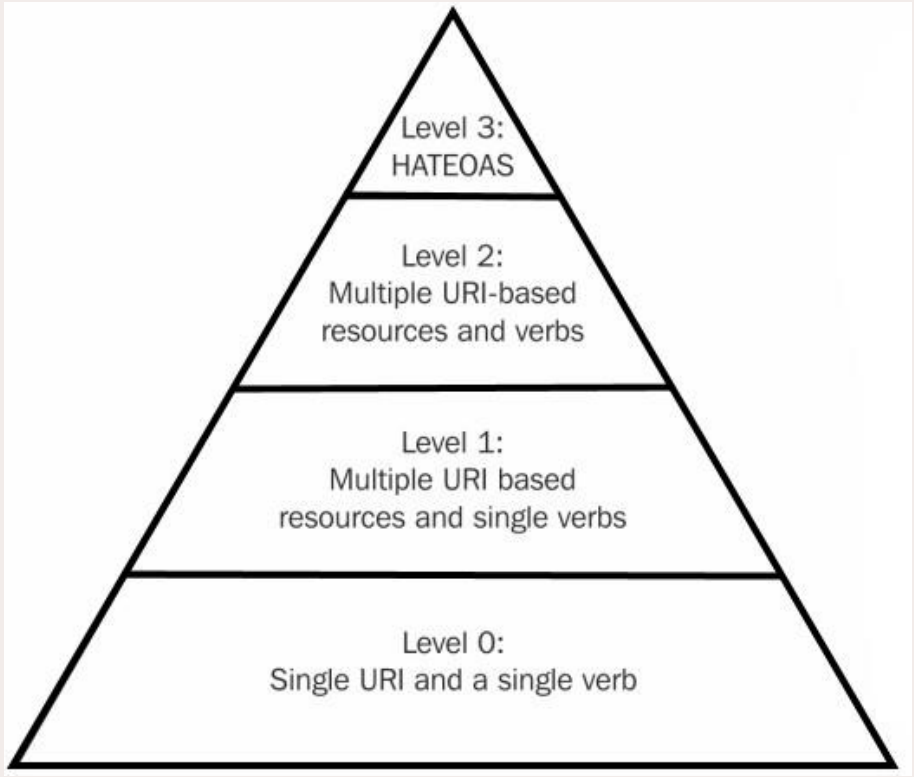
- REST = Paradigma für Softwarearchitektur verteilter Systeme
- Webservices, die diese Richtlinien befolgen, sind *RESTful*

- Schwerpunkt war die Entwicklung eines solchen Webservices
  - Wie RESTful ist unser Webservice?

  ➡ Richardson Maturity Model

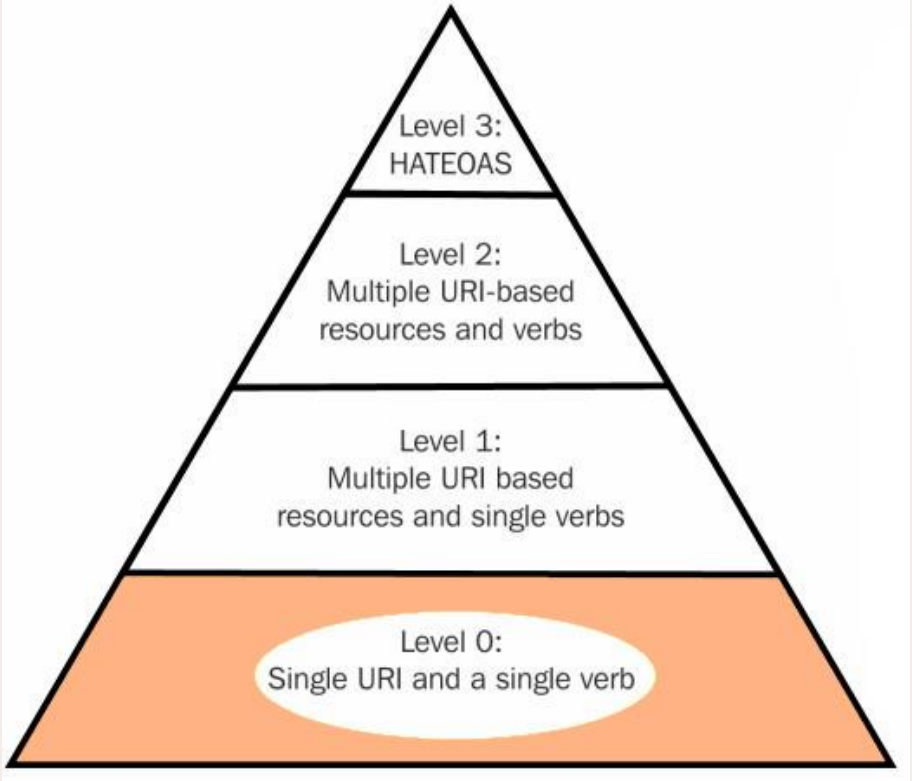
Richardson Maturity Model

- Wie REST-konform ist ein Webservice
- 4 Levels
- Jedes unterliegende Level Voraussetzung zum Erreichen des nächsthöheren



Richardson Maturity Model – Level 0

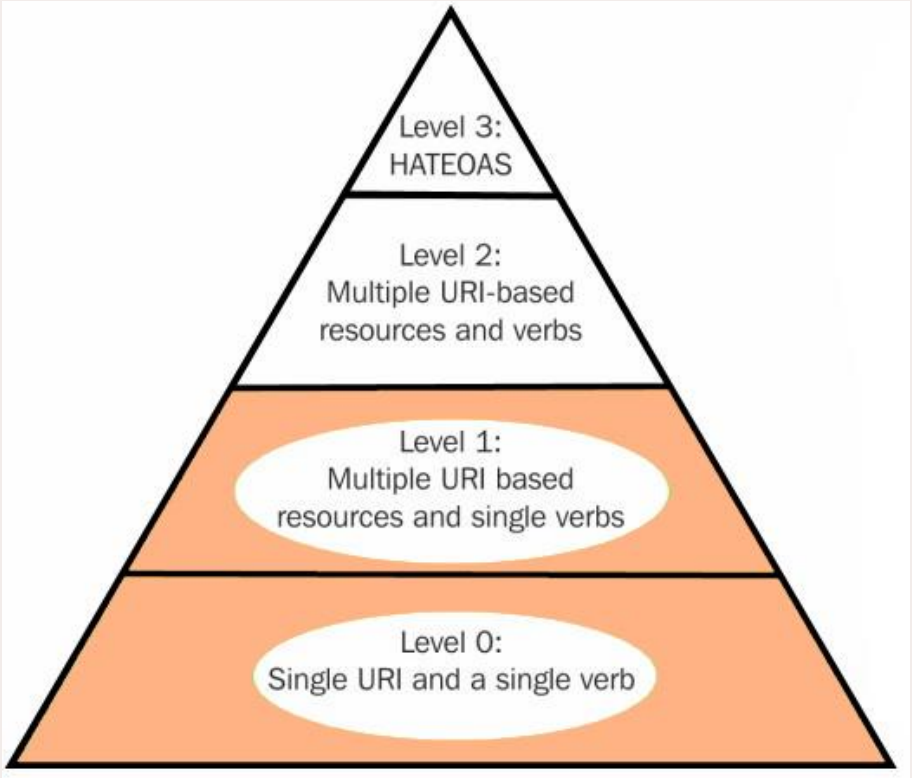
- HTTP als Kommunikationsprotokoll
  - GET oder POST
- Kommunikatoin gegen einen Endpoint



Richardson Maturity Model – Level 1

- Mehrere URIs verwendet
- Konzept “Ressource”

- /business-unit-objectives/0
- /business-unit-objectives
- /dashboard
- /company-objectives/1/company-key-results/2/changes
- /business-unit-objectives/0/business-unit-key-results/link

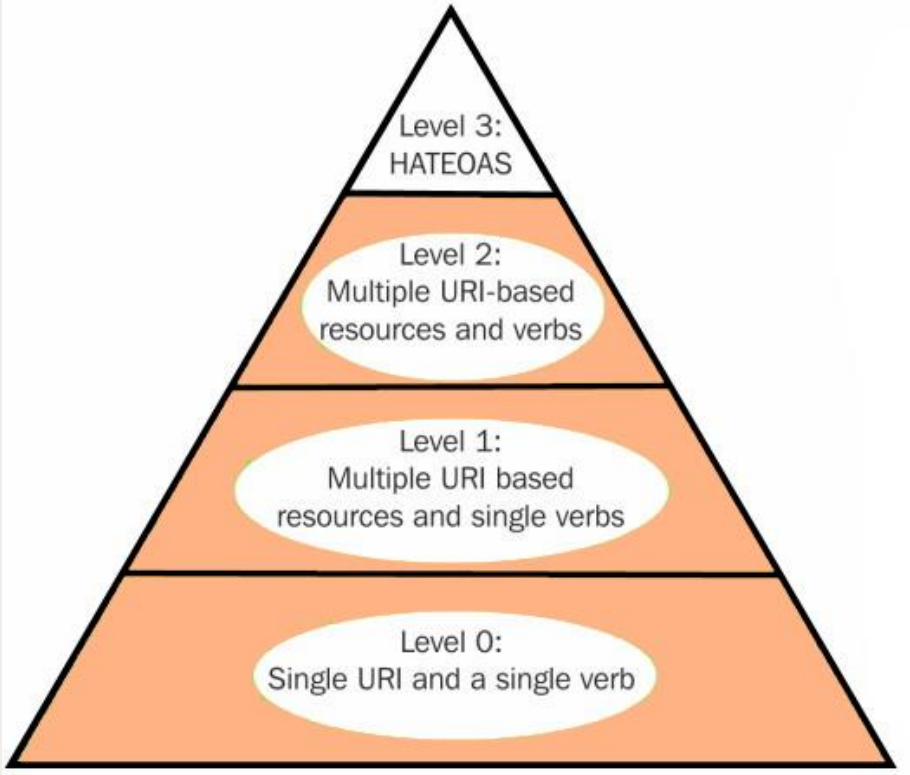


Richardson Maturity Model – Level 2

- Alle HTTP-Methoden zum Einsatz
- Status Codes korrekt verwendet

- Objectives und Key Results jeweils:
  - GET, PUT, PATCH, POST, DELETE

- GET auf existierende Ressource: 200 OK
- Erfolgreiches Anlegen einer neuen Ressource: 201 Created
- Zugriff auf nicht existierende Ressource: 404 Not Found
- Nicht authentifizierter Zugriff: 403 Forbidden

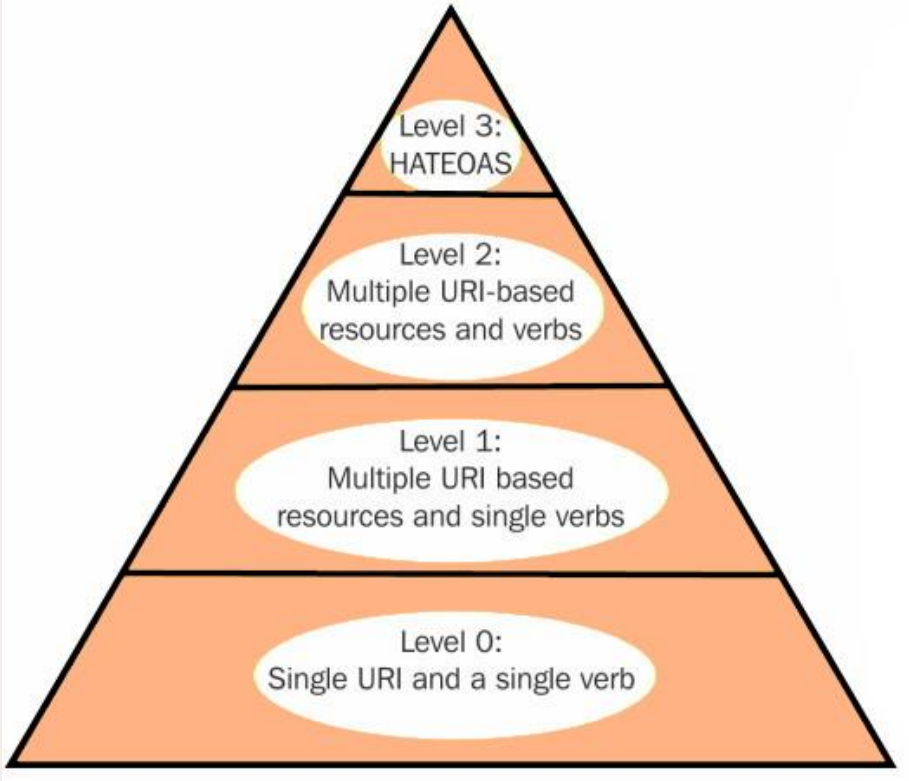


Richardson Maturity Model – Level 3

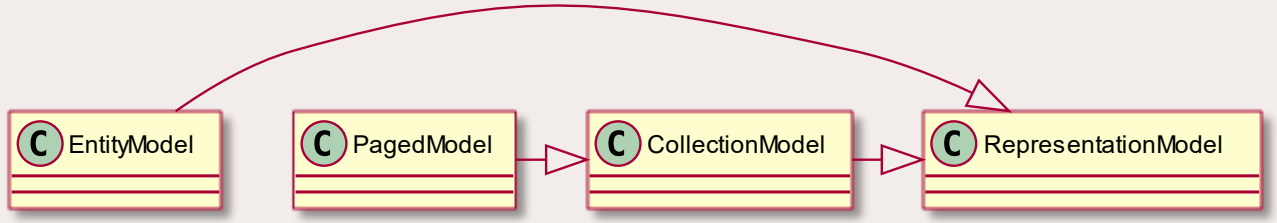
- Server-Response selbsterklärend
- Navigieren der API über URLs

  => Externe Dokumentation nicht zwingend erforderlich

```
{
  "name": "Business Unit Key Result 0",
  "current": 0.0,
   ...
  "_embedded": {
    "businessUnitObjective": {
      "name": "Business Unit Objective 0",
      "_links": { "self": { "href": "http://localhost/business-unit-objectives/0" } } },
    "companyKeyResult": {
      "name": "Company Key Result 0",
      "overall": 0.7,
      "_links": { "self": { "href": "http://localhost/company-objectives/0/company-key-results/0"
   ...
```



Spring-Projekt: HATEOAS



- Objekt + Links = Ressource
  - RepresentationModel als DTO

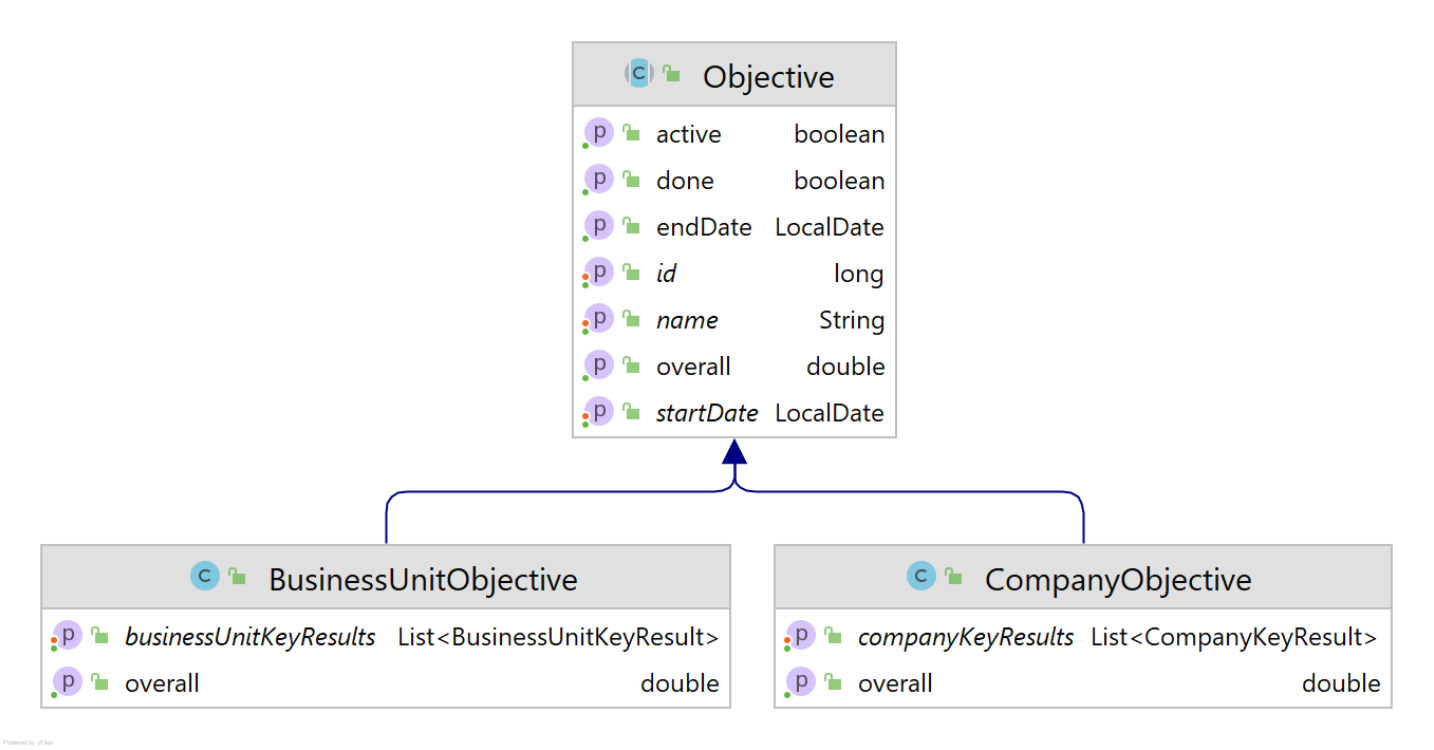
```
@GetMapping("/{id}")
public ResponseEntity<RepresentationModel<BusinessUnitObjectiveModel>>
                findOne(@PathVariable("id") long id)
```

- Affordances: Nicht nur Daten liefern, sondern auch Controls auflisten
  - Mithilfe von `application/prs.hal-forms+json`

```
GET  /business-unit-objectives/0/business-unit-key-results/1
...
  "method": "PUT",
  "properties": [
   { "name": "confidence",  "type": "number„ },
   { "name": "current",  "type": "number„ },
   { "name": "goal",  "type": "number„ },
   { "name": "name",  "type": "text„ } ]
...
```

Domain

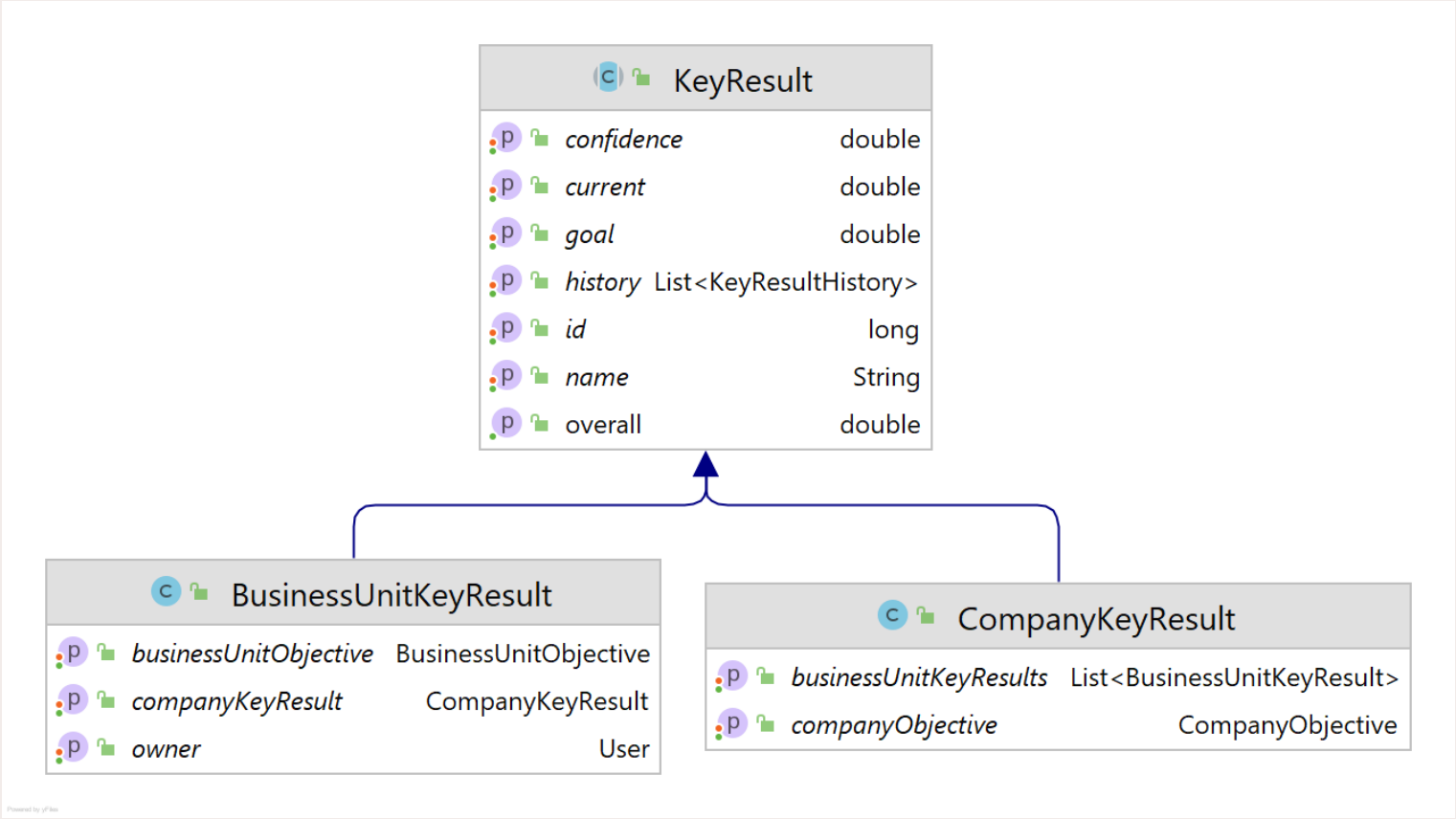
Objectives


Objective
active boolean
done boolean
endDate LocalDate
id long
name String
overall double
startDate LocalDate
BusinessUnitObjective
businessUnitKeyResults List<BusinessUnitKeyResult>
overall double
CompanyObjective
companyKeyResults List<CompanyKeyResult>
overall double

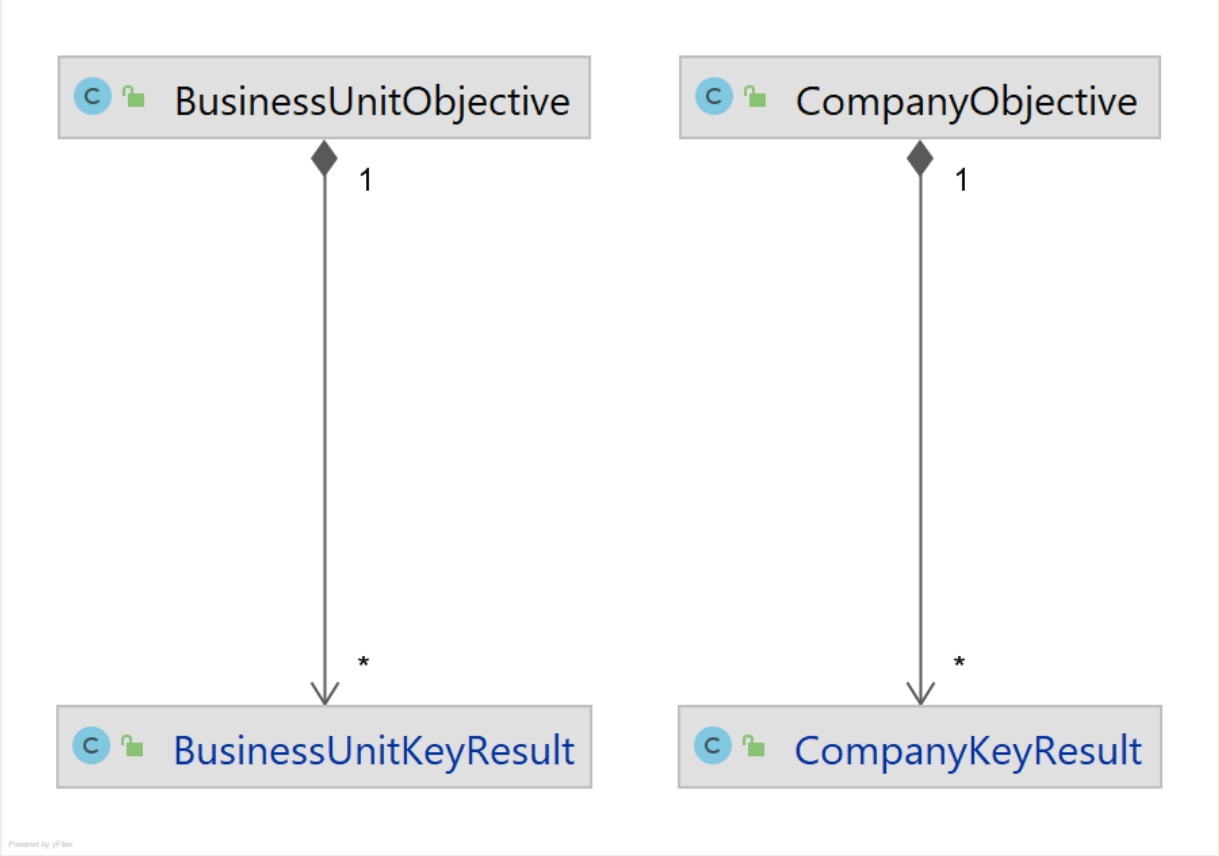
Objectives

```java
public LocalDate getEndDate() {
    return startDate.plusDays(DURATION_IN_DAYS);
}

public boolean isActive() {
    var today :LocalDate = LocalDate.now();
    return today.isAfter(startDate) &&
            today.isBefore(getEndDate());
}

public boolean isDone() {
    var today :LocalDate = LocalDate.now();
    return today.isAfter(getEndDate());
}
```

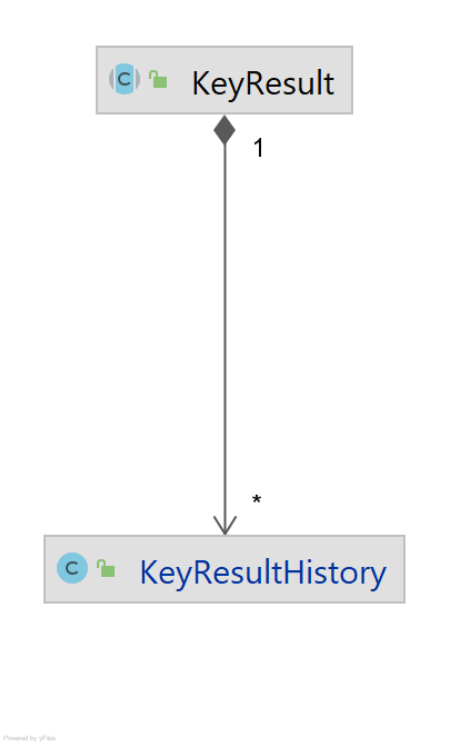
Key Results

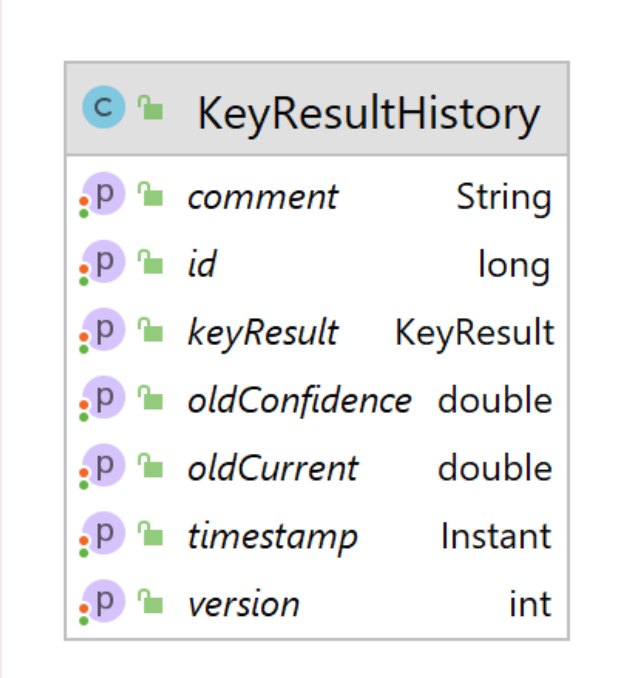

KeyResult
confidence double
current double
goal double
history List<KeyResultHistory>
id long
name String
overall double
BusinessUnitKeyResult
businessUnitObjective BusinessUnitObjective
companyKeyResult CompanyKeyResult
owner User
CompanyKeyResult
businessUnitKeyResults List<BusinessUnitKeyResult>
companyObjective CompanyObjective

Objectives ↔ Key Results



Key Result History

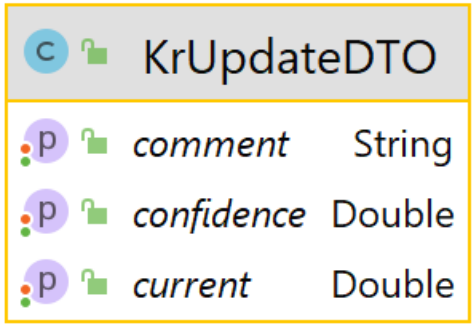




Key Result History

```
public interface CustomKeyResultRepository {

                ⋮

    public KeyResult updateWithDto(long keyResultId, KrUpdateDTO keyResultUpdate);
}
```

- altes current und confidence merken
- Versionsnummer aus History hochzählen
- History mit Kommentar speichern
- KeyResult updaten



JSON Patch

- PATCH-Request
  - Content-Type: application/json-patch+json
- Liste von Operationen
  - z.B. add, replace, remove
- Pfad zur Property
- Wert

```
[
    {
        "op"    :"replace",
        "path"  :"/current",
        "value" :"2.1"
    },
    {
        "op"    :"add",
        "path"  :"/comment",
        "value" :"ein Kommentar"
    }
]
```

JSON Patch

```
<dependency>
    <groupId>com.github.java-json-tools</groupId>
    <artifactId>json-patch</artifactId>
    <version>1.13</version>
</dependency>
```

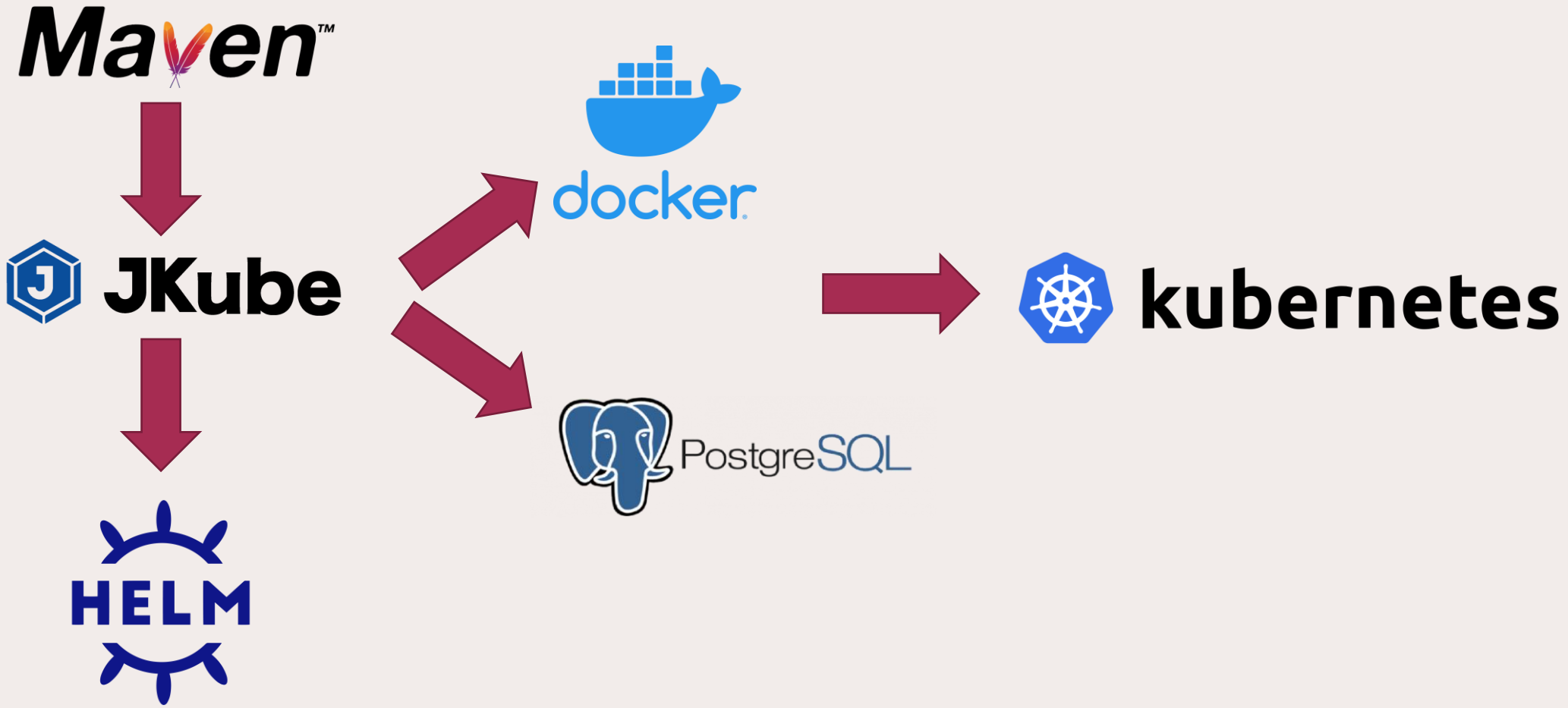
```
public T applyPatch(T obj, JsonPatch patch) {…}
```

Autorisierung

- Klassen User und Role
- drei Rollen
  - Readonly, CO Admin, BuO Admin
- Autorisierung über Antmatchers
- zusätzlich für Business Unit Key Results:
  - Setzen des Owners bei POST
  - Prüfen des Owners bei PUT, PATCH, DELETE

Deployment

CI/CD



CI/CD

- Maven Profiles
  - zu Spring Profiles
- JKube
  - Vollautomatisierung des Deployments

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <activatedProperties>dev</activatedProperties>
    </properties>
  </profile>
  <profile>
    <id>prod</id>
    <properties>
      <activatedProperties>prod</activatedProperties>
    </properties>
  </profile>
</profiles>
<build>
...
    <plugin>
      <groupId>org.eclipse.jkube</groupId>
      <artifactId>kubernetes-maven-plugin</artifactId>
      <version>${jkube.version}</version>
    </plugin>
...
```

Spring Profiles

- application.yaml
  - .Values von Chart
  - Umgebungsvariablen
- -dev.yaml
  - h2 Databank
  - Dev tools
- -prod.yaml
  - PostgreSQL
  - Validierung der Tables

```
spring:
  config:
    activate:
      on-profile: prod
  jpa:
    hibernate:
      ddl-auto: validate
  datasource:
    driverClassName: org.postgresql.Driver
    url: jdbc:postgresql://${DB_CONNECT}/${POSTGRES_DB}
    username: ${POSTGRES_USER}
    password: ${POSTGRES_PASSWORD}

server:
  error:
    include-stacktrace: never
```

Dockerfile

- Standard Image
- Non-root Container
  - Security
- Umgebungsvariable
  - Spring Profile

```
FROM openjdk:11-jdk-slim
RUN groupadd -r spring && useradd --no-log-init -r -g spring spring
USER spring:spring
COPY maven/target/*.jar app.jar
ENV spring_profiles_active="prod"
ENTRYPOINT ["java","-jar","/app.jar"]
```

Kubernetes object

- Image von Dockerfile
- Umgebungsvariablen
  - .Values von Chart
  - oder Secret
- PostgreSQL
  - Persistenter Speicher

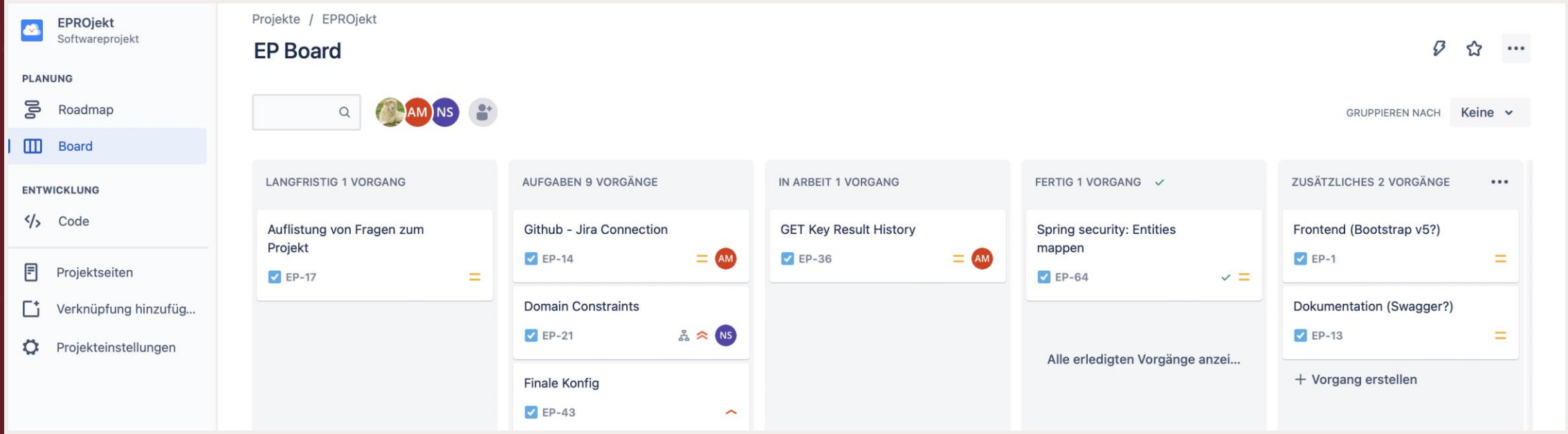
```
- name: DB_CONNECT
  value: {{ .Release.Name }}-postgresql
- name: POSTGRES_DB
  value: {{ .Values.global.postgresql.auth.database }}
- name: POSTGRES_USER
  value: postgres
- name: POSTGRES_PASSWORD
{{- if .Values.global.postgresql.auth.postgresPassword }}
  value: "{{ .Values.global.postgresql.auth.postgresPassword }}"
  {{ else }}
  valueFrom:
    secretKeyRef:
      name: {{ .Release.Name }}-postgresql
      key: postgres-password
      {{- end }}
```

Helm Chart

- Umgebungsvariablen
- Dependencies/Subcharts
  - Bitnami PostgreSQL
    - Alternative für ARM
- Initialisierung Schema

```
global:
  postgresql:
    auth:
      database: ###
      postgresPassword: ###
postgresql:
  # remove hashtags for aarch64 image
  #image:
  #  registry: ghcr.io
  #  repository: zcube/bitnami-compat/postgresql
  #  tag: 14
  primary:
    initdb:
      scripts:
        schema.sql: | ...
```

Jira


EPROjekt
Softwareprojekt
PLANUNG
Roadmap
Board
ENTWICKLUNG
Code
Projektseiten
Verknüpfung hinzufüg...
Projekteinstellungen
Projekte / EPROjekt
EP Board
AM
NS
GRUPPIEREN NACH
Keine
LANGFRISTIG 1 VORGANG
Auflistung von Fragen zum Projekt
EP-17
AUFGABEN 9 VORGÄNGE
Github - Jira Connection
EP-14
Domain Constraints
EP-21
Finale Konfig
EP-43
IN ARBEIT 1 VORGANG
GET Key Result History
EP-36
FERTIG 1 VORGANG
Spring security: Entities mappen
EP-64
Alle erledigten Vorgänge anzei...
ZUSÄTZLICHES 2 VORGÄNGE
Frontend (Bootstrap v5?)
EP-1
Dokumentation (Swagger?)
EP-13
Vorgang erstellen