Softwaretechnik 2 (ST2)

Informatik Bachelor, SoSe 2025

Prof. Dr.-Ing. Stefan Bente

Workshop M3:

(De-Facto-)Standards für REST-API-Design

Technology
Arts Sciences
TH Köln

Stil-Regeln für URI-Design (1)

1. Forward slash separator (/) must be used to indicate a hierarchical relationship
2. A trailing forward slash (/) should not be included in URIs
   - **Falsch:** `http://canvas.org/shapes/`
   - **Richtig:** `http://canvas.org/shapes`
3. File extensions should not be included in URIs
   - **Falsch:** `http://college.org/students/3248234/address.json`
   - **Richtig:** `http://college.org/students/3248234/address`

Quelle: [Massé], p. 11ff. Beispiele leicht gekürzt.

Stil-Regeln für URI-Design (2)

4. Translation of entity names into URI

   Assume you have an entity **`MutualObligationAgreement`**

   a. ☺ Lower Camel Case: **`/mutualObligationAgreements/1234`**

   b. ☺ Kebab (Spinal) Case: **`/mutual-obligation-agreements/1234`**

- Was man **nicht** macht: Underscores!
  - ☹ `/mutual_obligation_agreement/…`

- Es gibt hier keine “harten” Regeln
  - In der Praxis gibt es beides
  - Im Praktikum verwenden wir 5.a. (Lower Camel Case)
  - Wie immer Sie es machen: Machen Sie es konsistent!

Aggregates => REST-APIs

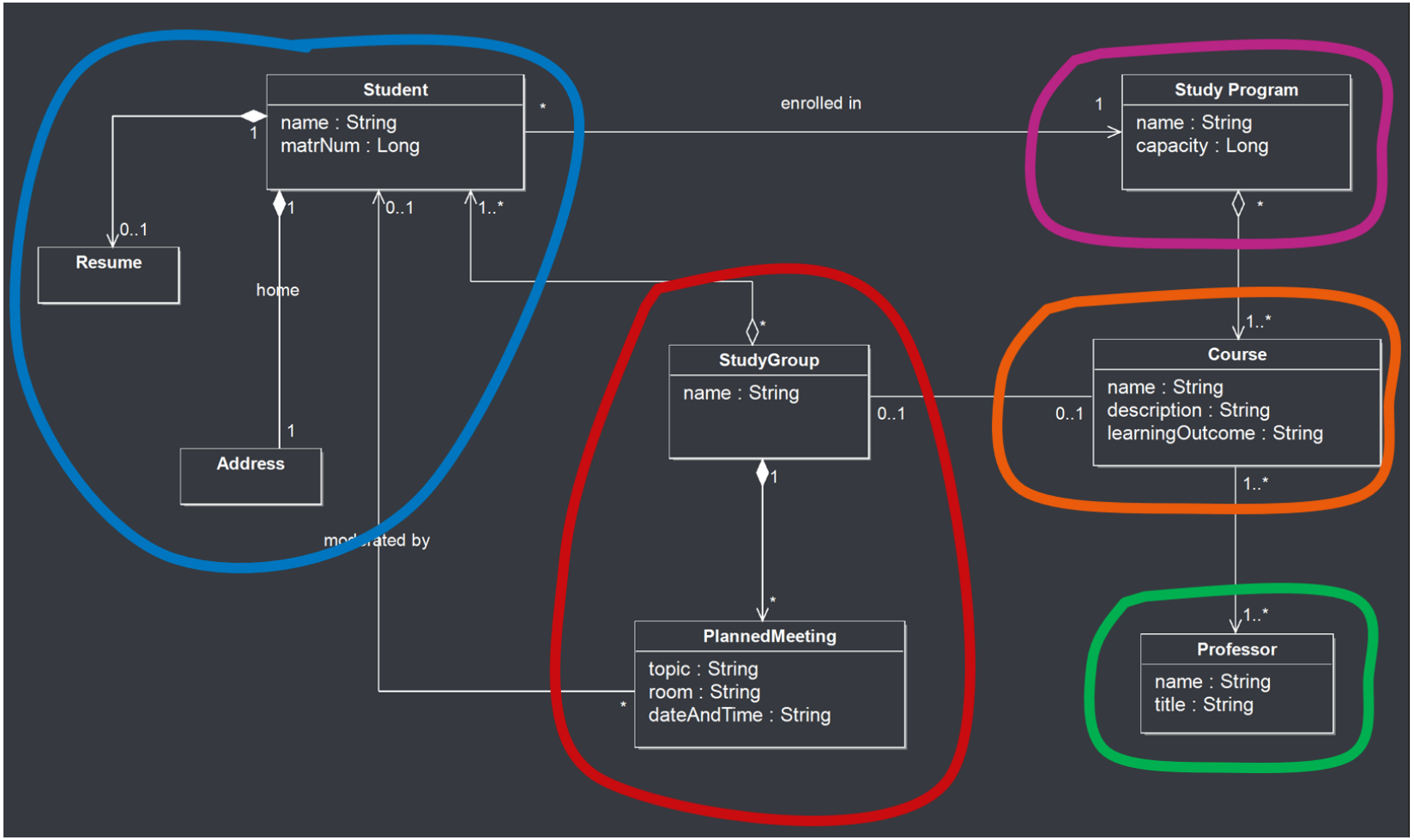
Endpoint

=

Aggregate

- … und es gelten alle Aggregate-Regeln
- z.B. keine Möglichkeit des direkten Zugriffs auf innere Entities
- siehe nächste Seiten

Ein kleines Beispiel-Datenmodell



- Hier unser Beispiel für die Spezifikation von REST-Endpoints – das Campus-Management-System.
- Farbig umkringelt: Die Aggregates.

Aufbau der URIs in REST-APIs

- URIs bilden Hierarchien in den Daten ab
  - Nicht alle Pfade durchs Datenmodell müssen im REST-API verfügbar sein!
- Drei Kernelemente in der URI:

	URI-Spezifikation Konkrete Beispiel-URL
1. Collections ○ Sammlungen von Ressourcen ○ im Plural benannt	**/students** https://hs-mgmt.de/students
2. IDs ○ Selektieren Entity aus Collection ○ Kann Zahl oder Text sein	**/students/{s-id}** https://hs-mgmt.de/students/123456
3. Entities ○ Steht für **einzelne** Ressource – Annahme: Student => Address 1:1 ○ im Singular benannt	**/students/{s-id}/address** https://hs-mgmt.de/students/123456/address

Anwendung REST-Verben (1)

- HTTP-Verben werden auf das **letzte (rechteste) Element** des URI-Pfads angewendet
- Sie wirken **bezüglich des jeweiligen Parent**

- `GET /studyPrograms`
  - gebe alle Studiengänge zurück *(zu entscheiden: will man das im API?)*
- `GET /studyPrograms/{s-id}`
  - gebe einen bestimmtes SG zurück
- `GET /studyPrograms/{s-id}/courses`
  - gebe alle Fächer zu einem Studiengang zurück

- `DELETE /studyPrograms`
  - lösche alle Studiengänge *(will man das im API?)*
- `DELETE /studyPrograms/{s-id}/courses`
  - lösche alle Zuordnungen von Fächern zu einem Studiengang *(ist das gewollt?)*
    - Wirkt auf Parent => nur Beziehung wird gelöscht, die Fächer selbst bleiben bestehen

Anwendung REST-Verben (2)

- **`POST /studyPrograms`**
  - Lege neuen Studiengang an
    - System legt neuen Studiengang an
    - Details des Studiengangs müssen im Request-Body mit übertragen werden
    - z.B. als JSON oder XML

- **`PUT /studyPrograms/{s-id}/courses/{c-id}`**
  - Ordne ein Fach einem Studiengang zu
    - Fach muss schon existieren

- **`PATCH /studyPrograms/{s-id}`**
  - Mache einen Update des Studiengangs
    - Details (z.B. neuer Name) muss im Request-Body übertragen werden

Query-Design

Query String

```
GET /mypath?att1=value1&att2=value2
```

- Folgende Operationen werden im Query-String ausgedrückt:
  - Searching
  - Paging
  - Filtering
  - Sorting

- Beispiel: Filtern
- *alle Studierenden mit Matrikel-Nummern, die mit 1,2 oder 3 beginnen*

```
GET /students?
        matrNumFrom=10000000&matrNumUntil=39999999
```