

# Miracum Mapper 2.0

Henrik Herzig und Nina Mücke

Studiengang Informatik

Projektarbeit: Implementierung eines web-basierten Mapping-Tools

Betreut von: PD Dennis Toddenroth, Noemi Deppenwiese M.Sc.

## 01 Motivation

## 02 Implementierung

- Datenbank

- API

- Backend

- Frontend

- Keycloak/Deployment

## 03 Live Demo

## 04 Ausblick

---

# Motivation

## Was ist ein Mapping?

- Mappen: Konzepte aus einer Terminologie mit Konzepten aus anderer Terminologie in Beziehung setzen
- Ziel: Semantische Interoperabilität
  - Forschung über mehrere Institutionen hinweg unterstützen
  - Datenaustausch zwischen verschiedenen Informationssystemen vereinfachen
- Grund: medizinische Daten oft in verschiedenen Terminologien erfasst

Source Labor Codesystem		Target Loinc		Equivalence ↑↓
Code ↑↓	Meaning ↑↓	Code ↑↓	Meaning ↑↓	
<input type="checkbox"/>	<input type="text" value="Search by code"/> <input type="text" value="Search by meaning"/>	<input type="checkbox"/>	<input type="text" value="Search by code"/> <input type="text" value="Search by meaning"/>	<input type="text" value="Any"/>
<input type="checkbox"/>	0000 Auftragskommentar	56850-1 Interpretation and review of laboratory results		Equivalent
<input type="checkbox"/>	4SOLI Oligoklonale AK im Serum	48422-0 Oligoclonal bands [Presence] in Serum or Plasma		Broader

Beispiel-Mapping

- Mappen ist sehr zeitaufwendiger Prozess
- Erfordert Expertenwissen
- In nicht Domänen-spezifischen Anwendungen möglich (Microsoft Access oder Excel)
- Unterstützung des Mappingprozesses mittels Tools (Mapper) möglich

Mapping Table Edit project

+ Add Delete Keyword Search Table Options

Target: Loinc Test2

Code  $\updownarrow$  Meaning  $\updownarrow$  2 Status  $\updownarrow$  Equivalence  $\updownarrow$  Comment  $\updownarrow$  1

Search by code Search by meaning Pending  $\otimes$   $\downarrow$  Any  $\downarrow$  Search by comment

	hemoglobin	Pending $\downarrow$	Not Related $\downarrow$	Test	$\checkmark$ $\otimes$
83250-1	Hemoglobin A [Units/volume] in Blood by Electrophoresis 10346-5		Equivalent	Test Comment to show the table adjusting the width for long comments	$\otimes$ $\otimes$
56300-7	Hemoglobin [Moles/volume] in Venous cord blood 103750-6		Related To	Test Comment to show the table adjusting the width for long comments	$\otimes$ $\otimes$
	Basic metabolic with hemoglobin and hematocrit panel - Blood 104076-5				
	Hemoglobin gastrointestinal lower [Procent] in Stool				

<< < 1 > >> Showing 1 to 3 of 3 mappings 10  $\downarrow$

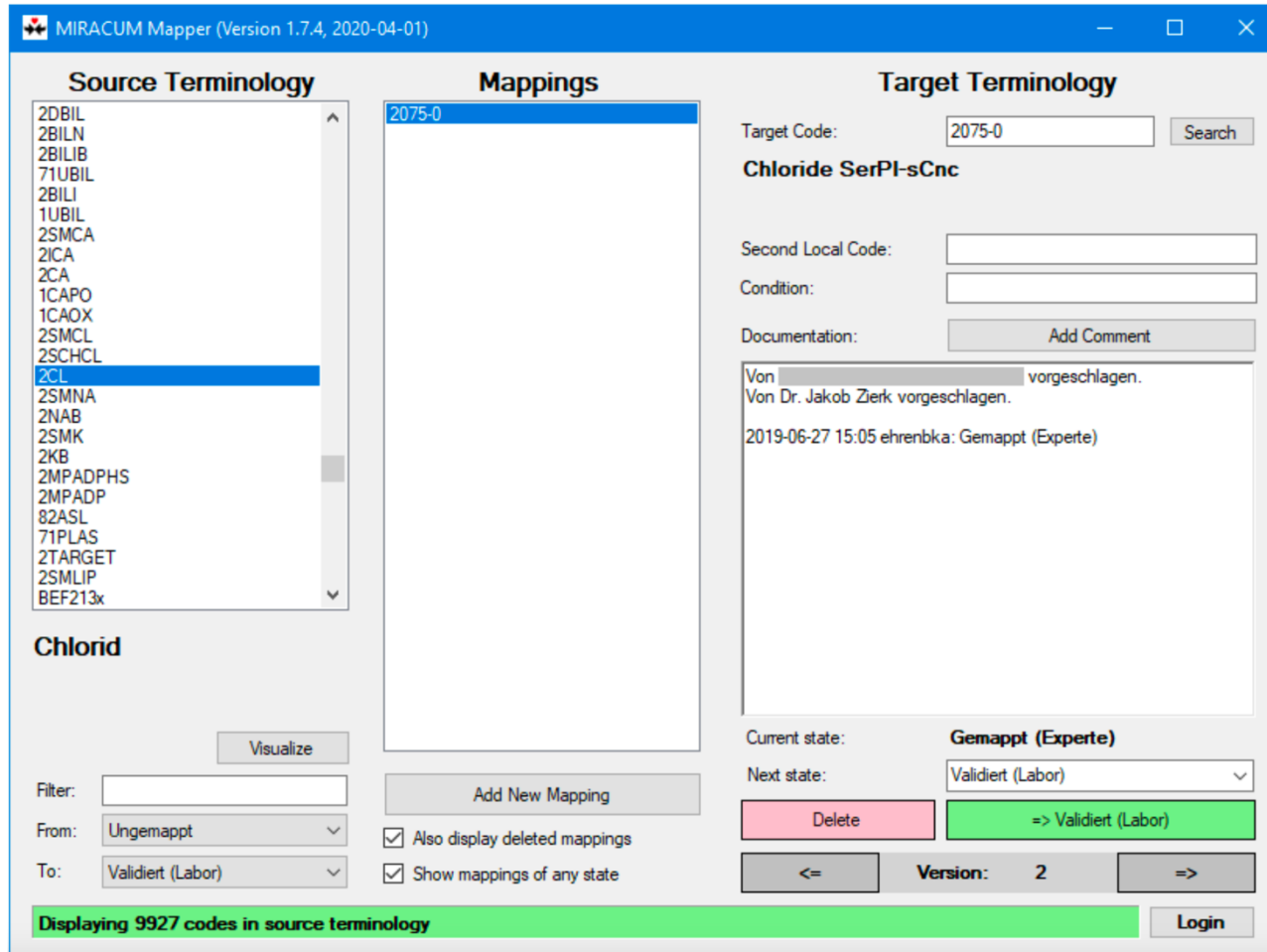


Bild 1: Nutzer Interface (Mate et al.)

- Workflow Unterstützung des Mapping-Prozesses
- Einschränkungen:
  - Lokale Anwendung (Installation nötig)
  - “Nur“ 1:1 Mappings möglich
  - Nutzer-Rollen systemübergreifend

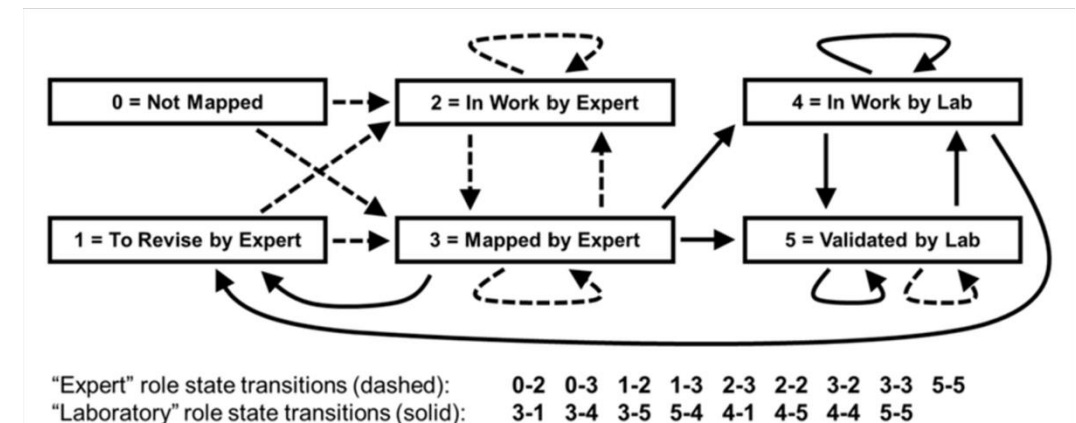
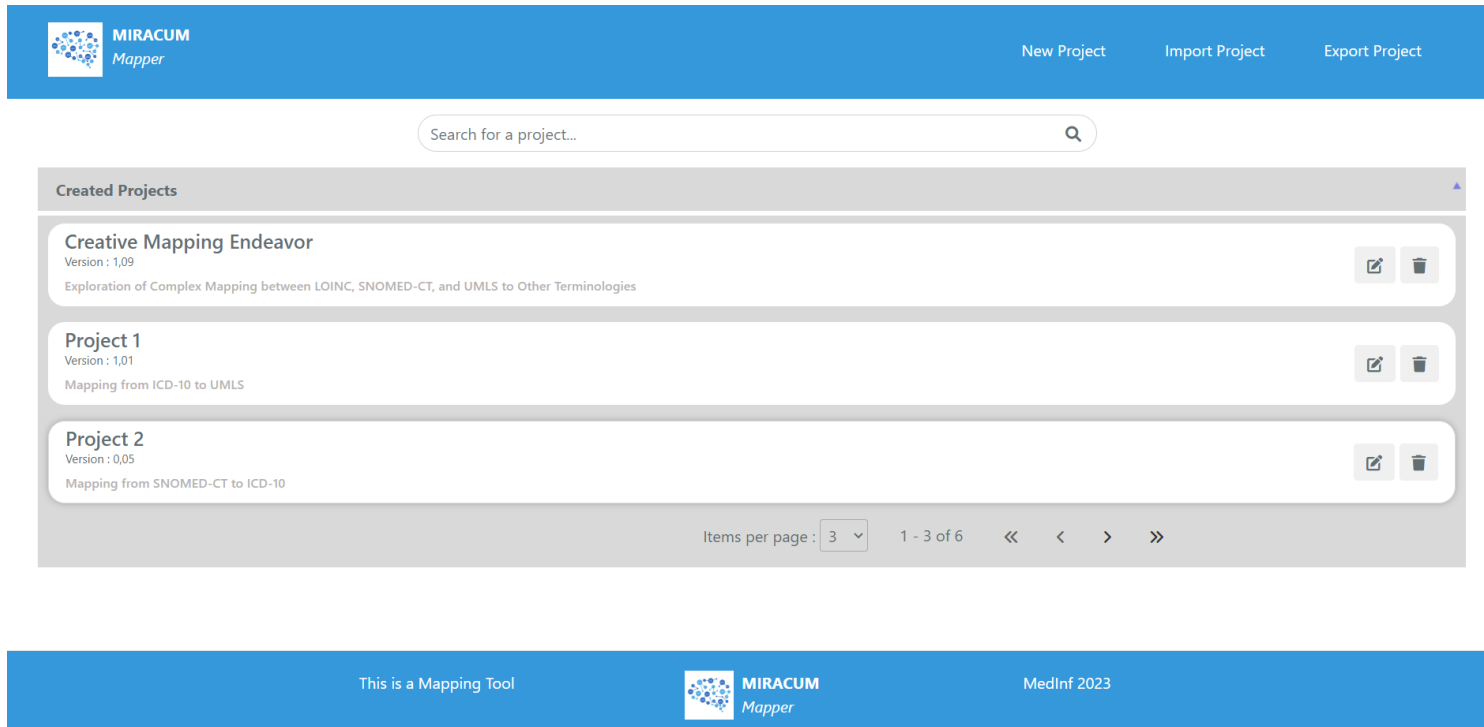


Bild 2: mögliche Workflows (Mate et al.)



- Frontend Prototyp
- In C# entwickelt
- Grundaufbau der UI übernommen und angepasst
- Projekt selbst nicht verwendet, da schwierig mit API und externem Backend zu verwenden

Bild 3: Miracum Mapper 2.0 Frontend Prototyp  
<https://github.com/steve-237/MIRACUM-Mapper-2.0>

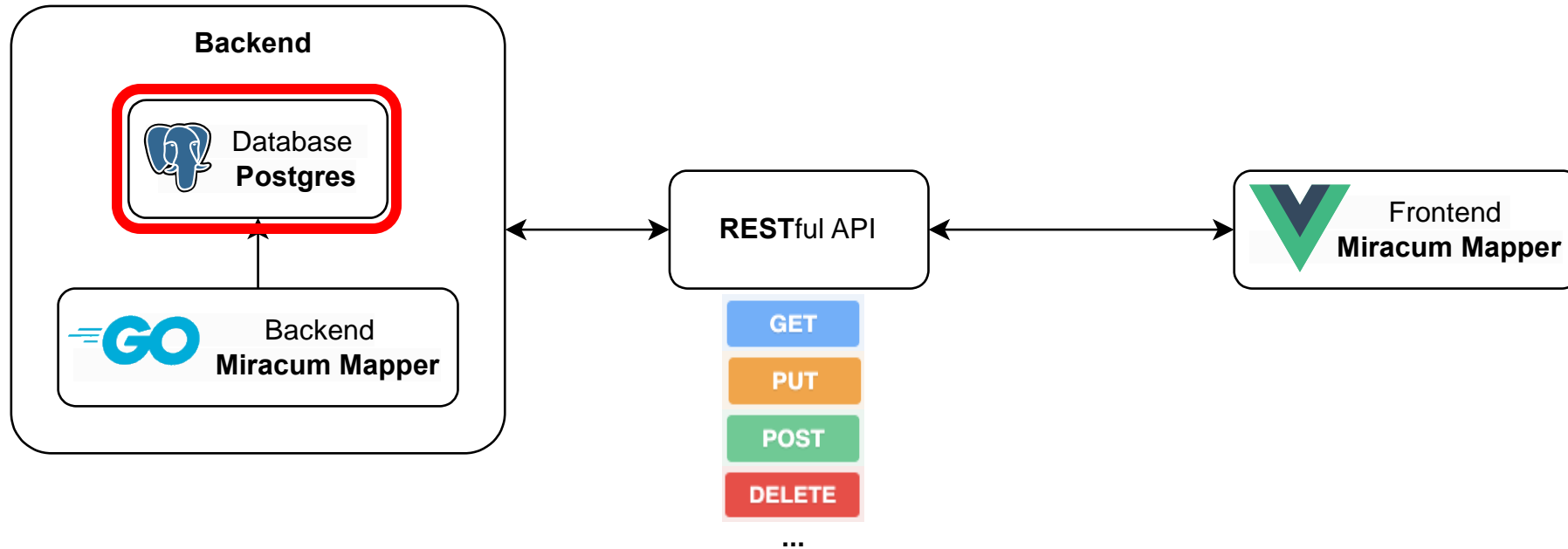
- Frontend als Webanwendung, separates Backend mit Datenbank
- Vorteile Webanwendung:
  - Keine Installation nötig
  - Lokal keine Datenbank
  - Möglichkeit an verschiedenen Geräten synchronisiert zu arbeiten
- Vorteile eines Servers (im Vergleich zu lokaler Anwendung):
  - Autovervollständigung eines Codesystem-Elements (schnelle Durchsuchung einer großen Datenbank mit allen möglichen Codes)
  - zentraler Ort für alle Mapping-Daten
  - Nutzerverwaltung mit verschiedenen Rechten pro Nutzer je Mapping-Projekt



# MiracumMapper

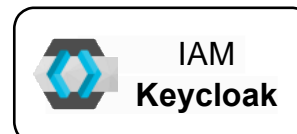
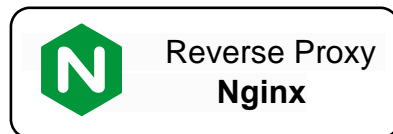


# Implementierung

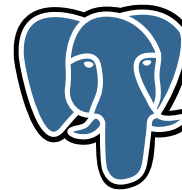



- Separates Backend und Frontend
- Kommunikation mittels RESTful API

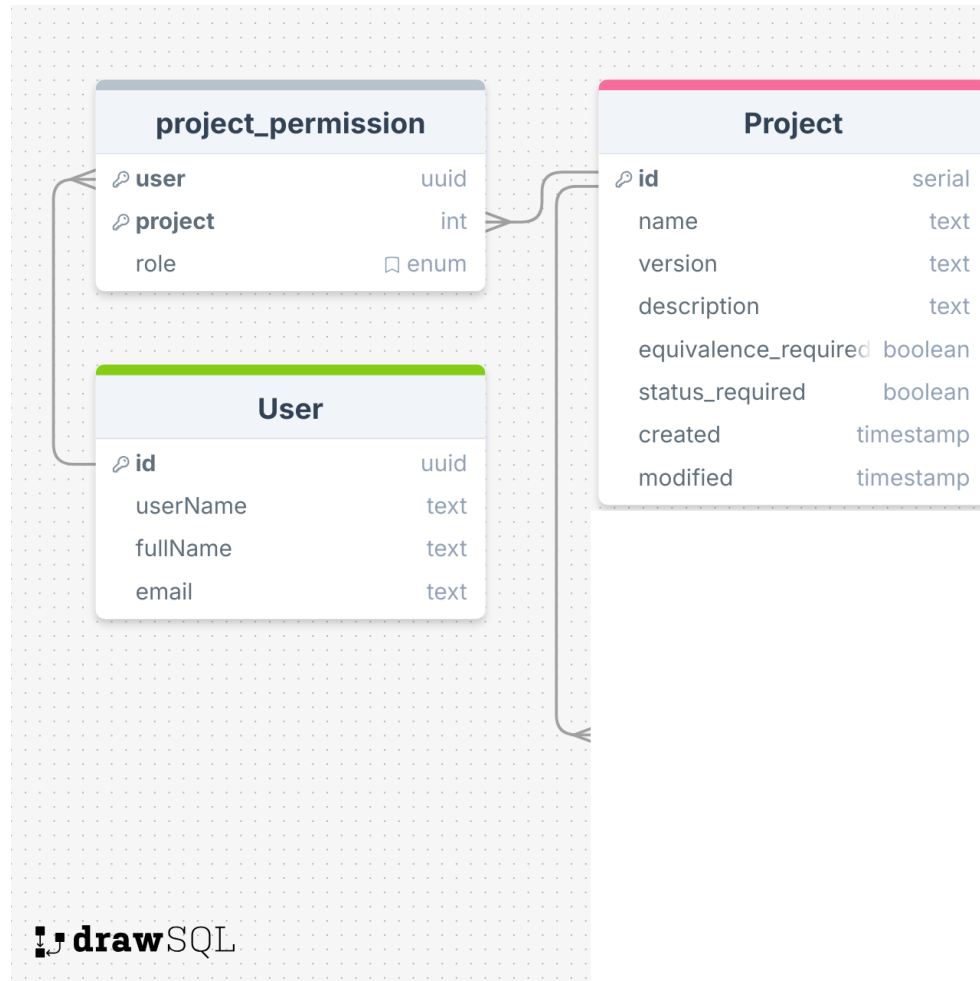
- Andere Dienste:



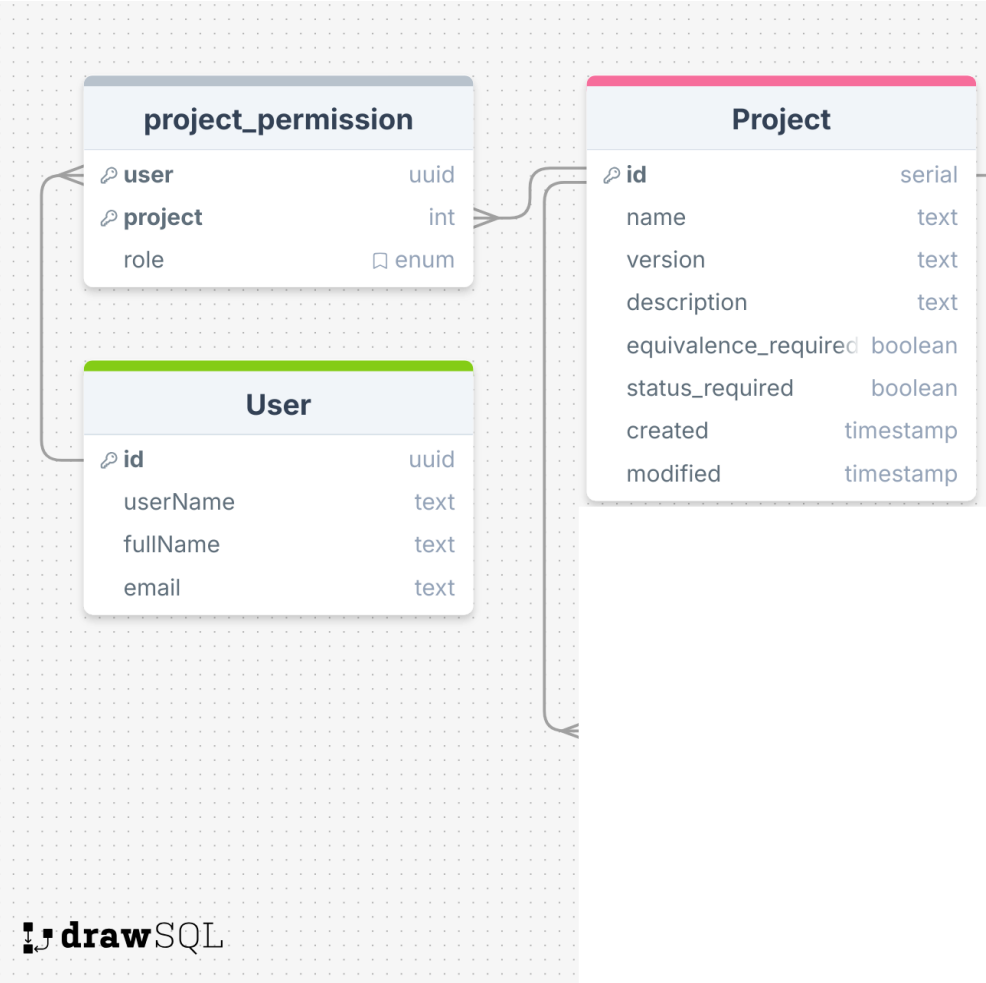
- Mapper muss Daten speichern (Projekte, Mappings, Codesysteme, Benutzerrechte, ...)
- Anforderungen: platzsparende Speicherung, schnelle Suche (z.B. Elemente in Codesystemen)
- Verwenden einer relationalen Datenbank → Postgres
  - Open Source
  - Docker-Container verfügbar
  - Weitere Funktionen: z.B. Volltextsuche, Ähnlichkeitssuche



Project	
 <b>id</b>	serial
name	text
version	text
description	text
equivalence_required	boolean
status_required	boolean
created	timestamp
modified	timestamp

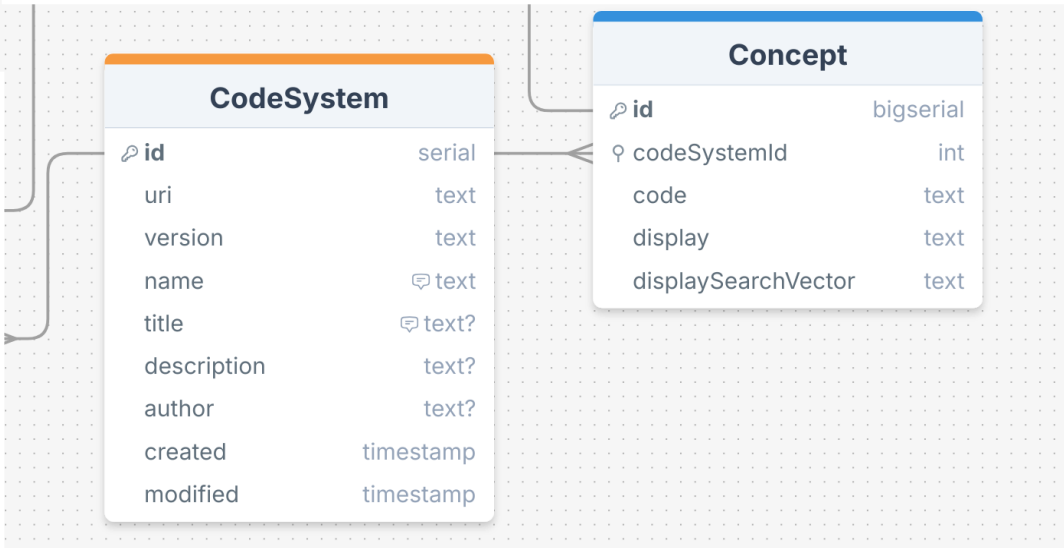


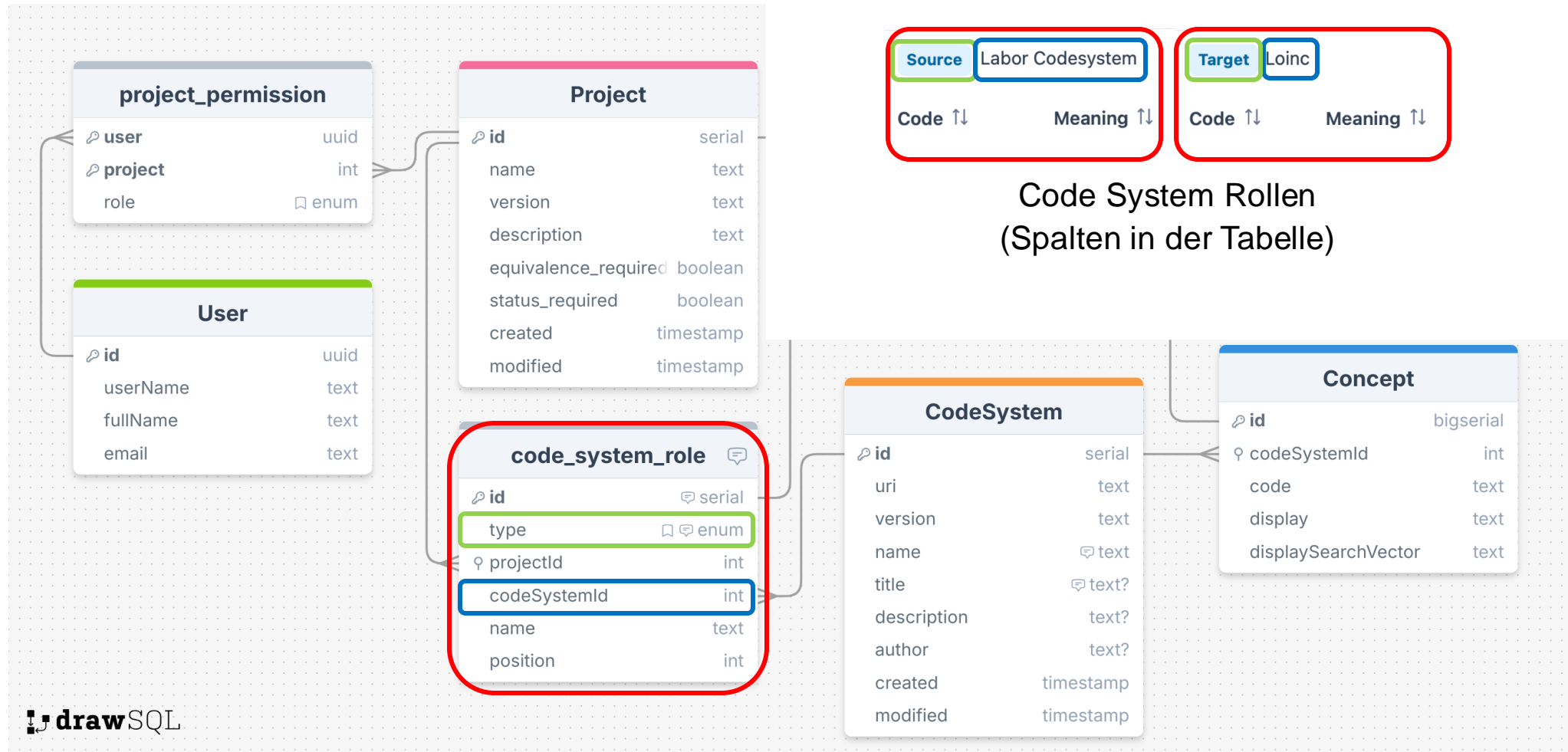
```
[ reviewer, project_owner, editor ]
```

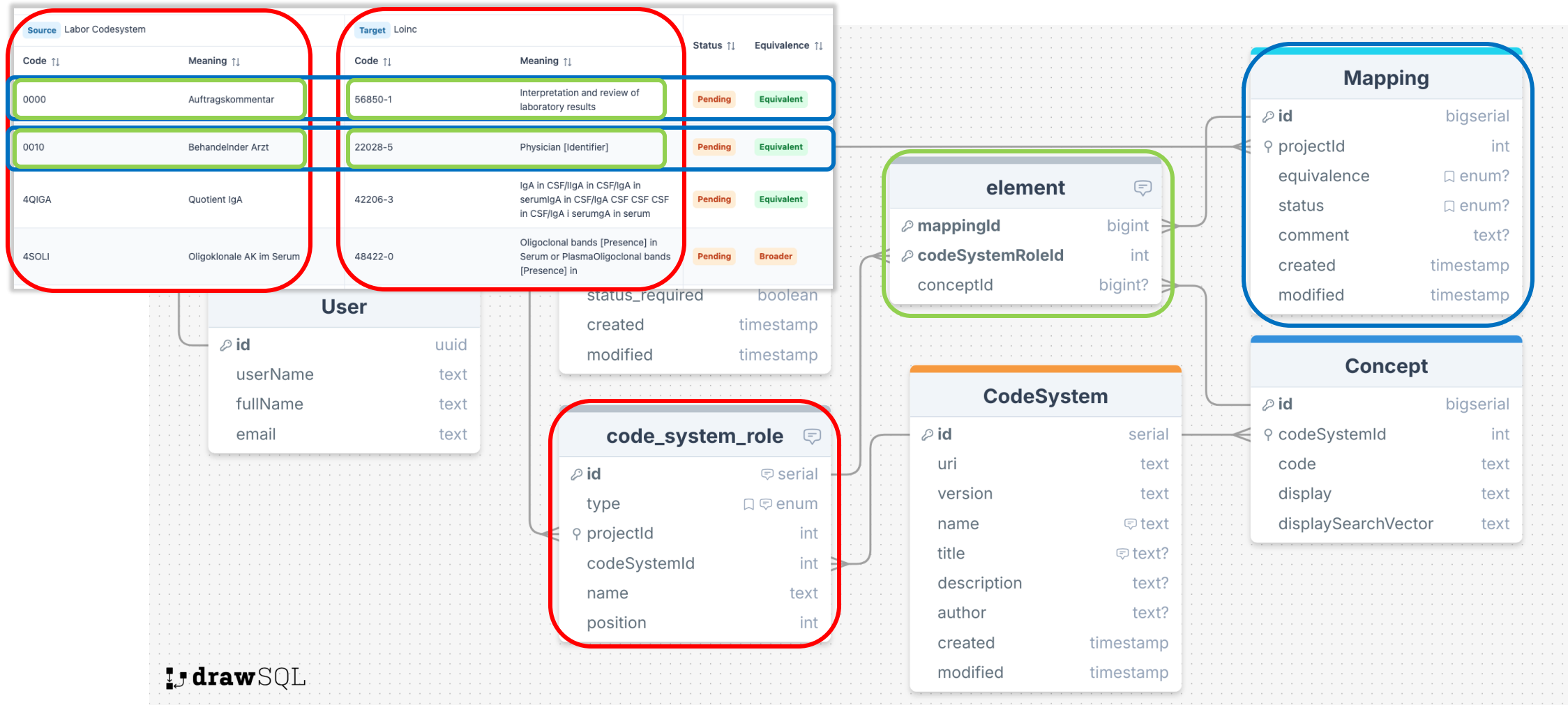


Code ↕	Meaning ↕
100002-5	Specimen care is maintained
10346-5	Hemoglobin A [Units/volume] in Blood by Electrophoresis
100309-4	Activity support person

## Loinc Concepts

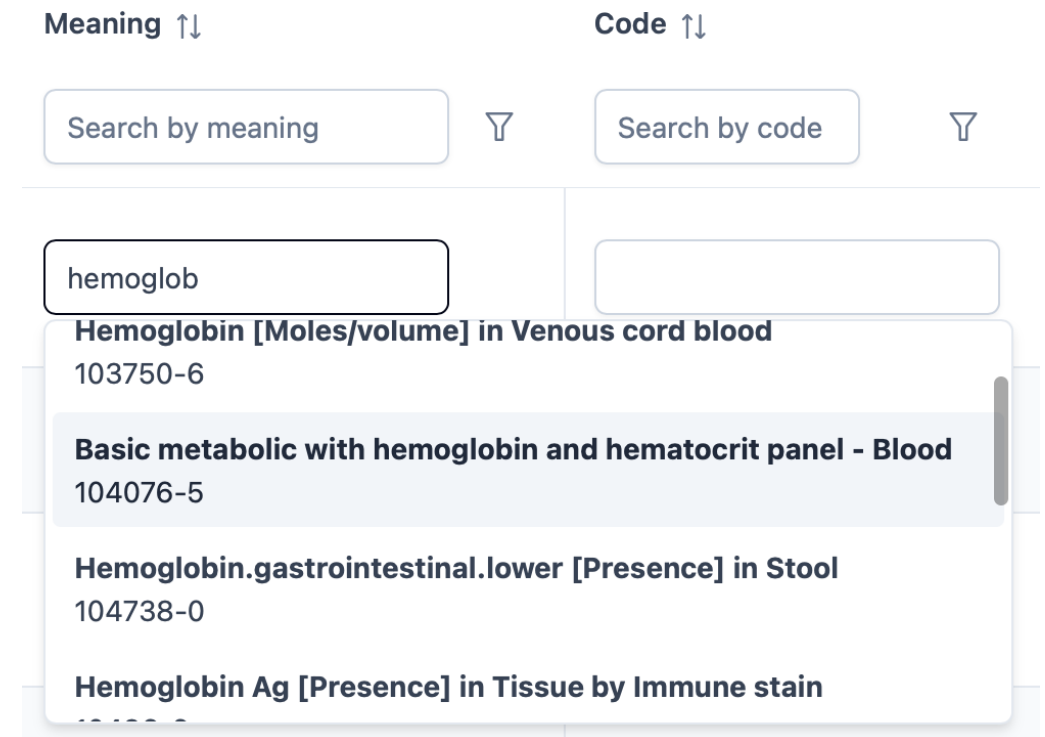








- Unterstützung einer schnellen Suche in Codesystemen
- **Codes:** Alle die mit Suchmuster anfangen  
Beispiel: **123** → Ergebnis: **12345, 12321**, Nicht: **45123**
- **Meaning:** Volltextsuche, Postgres bietet Unterstützung hierfür  
Beispiel: **hemoglobin blood meta**  
→ Basic metabolic with **hemoglobin** and hematocrit panel - **Blood**



Meaning ↑↓ Code ↑↓

Search by meaning Search by code

hemoglob

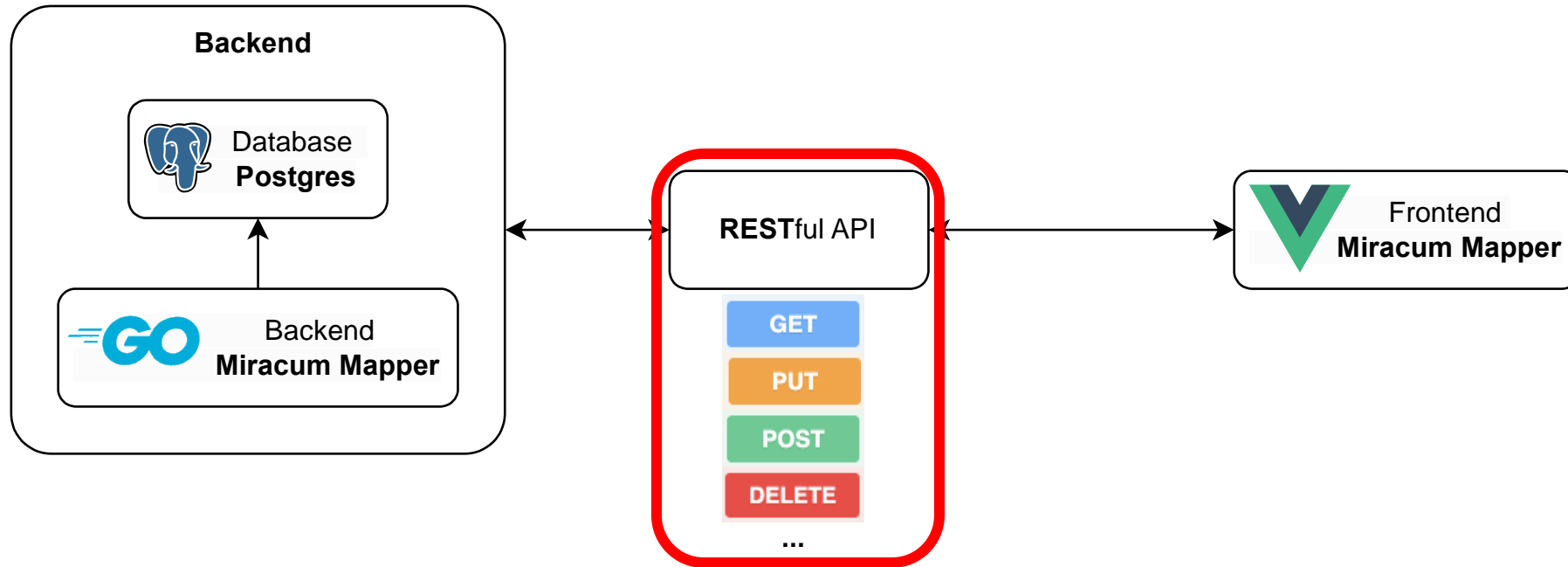
Hemoglobin [Moles/volume] in Venous cord blood  
103750-6

**Basic metabolic with hemoglobin and hematocrit panel - Blood**  
104076-5

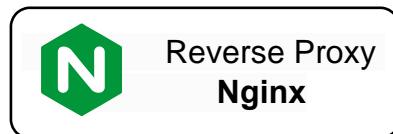
Hemoglobin.gastrointestinal.lower [Presence] in Stool  
104738-0

Hemoglobin Ag [Presence] in Tissue by Immune stain

Suche eines Loinc Codes im Frontend



– Andere Dienste:



- API-Definition mittels OpenAPI Spezifikation (Version 3.1.0) in YAML
- Definition von Endpunkten:



mapping Operations about mappings			^
GET	/projects/{project_id}/mappings	Get all mappings for a project by project ID	✓ 🔒
POST	/projects/{project_id}/mappings	Create a new mapping for a project	✓ 🔒
PUT	/projects/{project_id}/mappings	Update a mapping and its elements by their concept IDs	✓ 🔒
PATCH	/projects/{project_id}/mappings	Update a mapping and its elements by their concept IDs	✓ 🔒
GET	/projects/{project_id}/mappings/{mapping_id}	Get a mapping with its elements by ID	✓ 🔒
DELETE	/projects/{project_id}/mappings/{mapping_id}	Delete a mapping and its elements by ID	✓ 🔒

Visualisierung mittels [Swagger Preview](#)

- Definition von Query- und Path- Parametern
- Typen können angegeben werden

GET

/projects/{project\_id}/mappings

Get all mappings for a project by project ID

^

🔒

Get all mappings for a project by project ID. The mappings include all elements with concepts. Paging and sorting can be specified by query parameters.

Parameters

Try it out

Name	Description
<b>project_id</b> <small>★ required</small> integer(\$int32) (path) minimum: 1	The ID of the project
sortBy string (query)	Field to sort by  Available values : id, equivalence, status, comment, created, modified  Default value : id

project\_id

id

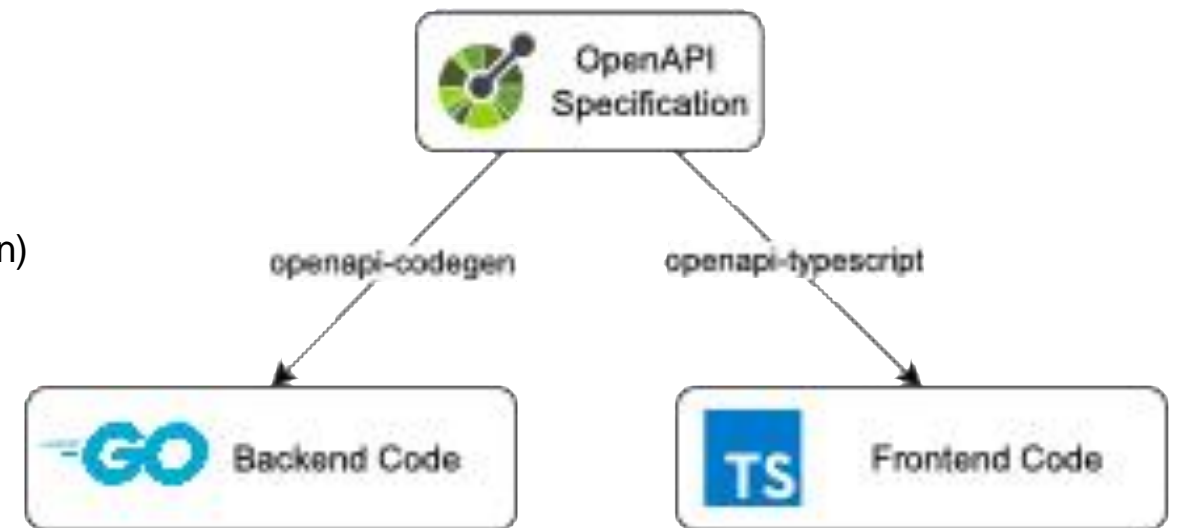
▼

Path und Query Parameter

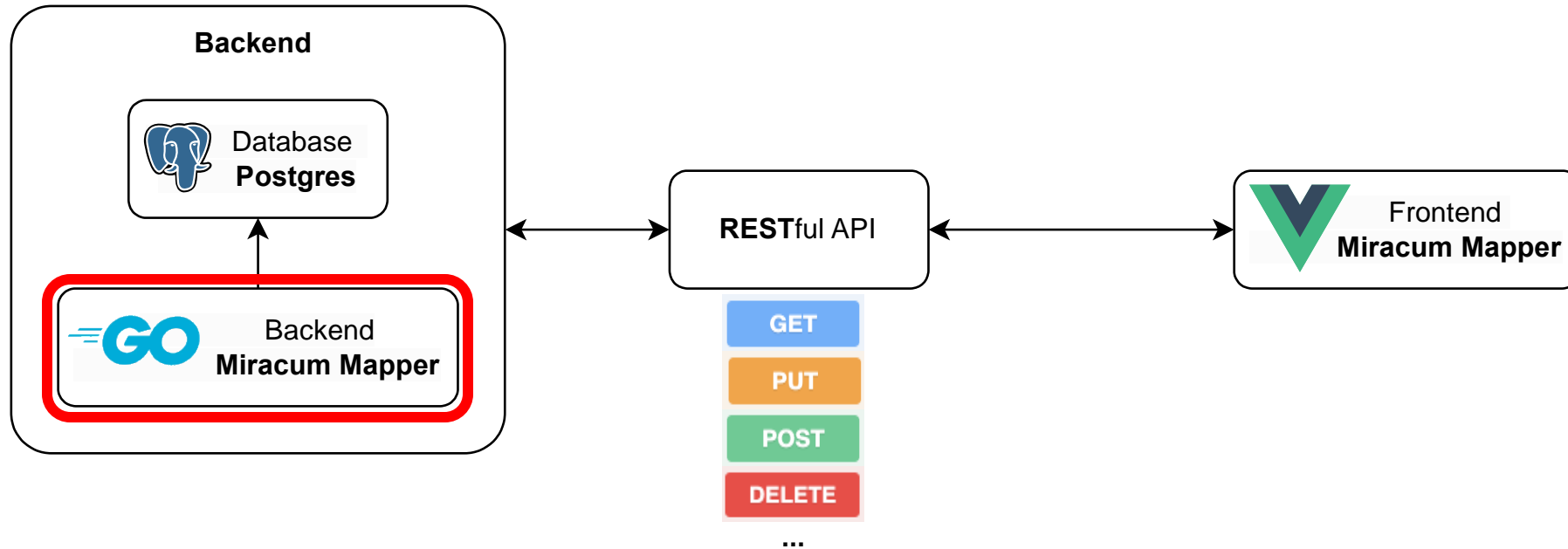
## Beispielschema "Mapping"

## Beispiel Response Body

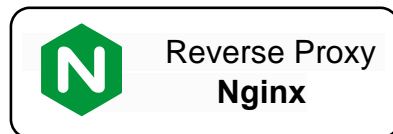
- Aus OpenAPI Spezifikation kann automatisch Client- und Server-Code generiert werden
- Vorteile:
  - Endpunkte automatisch dokumentiert
  - Typsicherheit durch Compiler
  - einfachere Erweiterbarkeit
  - kein doppelter Code für Client und Server (z.B. Definition der Typen)
  - weniger Code zu schreiben (z.B. für Validierung)
  - stellt Code-Grundgerüst für Backend bereit
- Nachteile:
  - höhere Komplexität
  - Verhalten des Generators schwierig zu ändern



TODO pixelig



– Andere Dienste:



- Implementierung in Go (Gründe: kompilierte Sprache: schnell, Typsicherheit)
- Tools/Dependencies:
  - oapi-codegen: Codegenerator beispielhafte Implementierung eines Endpunktes



POST

/projects/{project\_id}/mappings Create a new mapping for a project



```
func (s *Server) CreateMapping(ctx context.Context, request api.CreateMappingRequestObject) (api.CreateMappingResponseObject, error) {
    projectId := request.ProjectId
    createMapping := request.Body

    permissions, err := getUserPermissions(ctx, s, request.ProjectId)
    if err != nil {
        // [...]
    }
    if !checkUserHasPermissions(MappingCreatePermission, permissions) {
        // [...]
    }

    mapping := transform.ApiCreateMappingToGormMapping(*createMapping, projectId)
    if err := s.Database.CreateMappingQuery(&mapping, checkFunc); err != nil {
        // [...]
    }
    return api.CreateMapping200JSONResponse(transform.GormMappingToApiMapping(mapping)), nil
}
```

Beispielhafte Implementierung eines Endpunktes



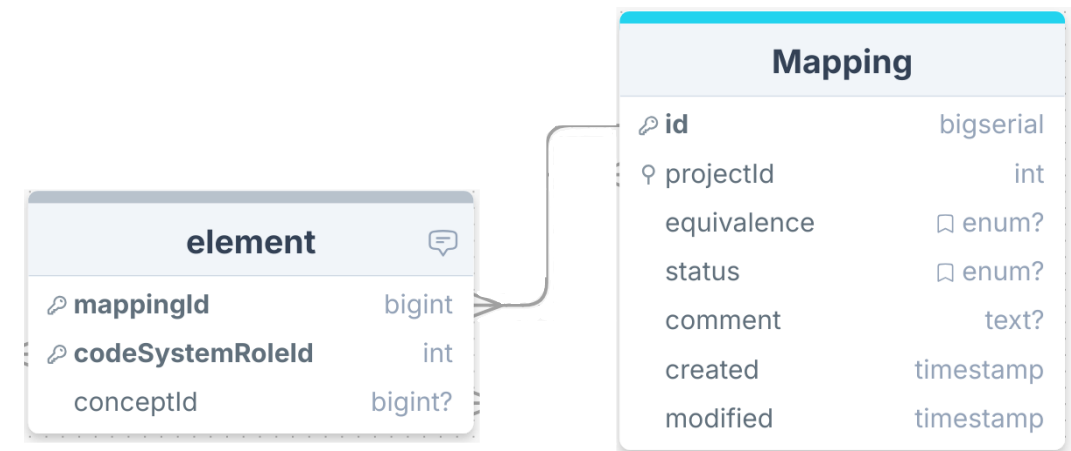
- Implementierung in Go (Gründe: compilierte Sprache: schnell, Typsicherheit)
- Tools/Dependencies:
  - gorm: ORM, um auf Datenbank mit Abstraktionsschicht zuzugreifen (weniger Fehleranfällig, wartbarer)



```
type Mapping struct {
    ModelBigId
    ProjectID    uint32    `gorm:"index"`
    Equivalence  *Equivalence `gorm:"type:Equivalence"`
    Status       *Status    `gorm:"type:Status"`
    Comment      *string
    Elements     []Element `gorm:"constraint:OnDelete:CASCADE"`
}
```

```
type Element struct {
    MappingID      uint64 `gorm:"primaryKey;index"`
    CodeSystemRoleID uint32 `gorm:"primaryKey"`
    ConceptID      *uint64
    Concept        Concept
}
```

Gorm Tabellen Definitionen



Datenbank-Schema

- Implementierung in Go (Gründe: kompilierte Sprache: schnell, Typsicherheit)
- Tools/Dependencies:
  - gorm: ORM, um auf Datenbank mit Abstraktionsschicht zuzugreifen (weniger Fehleranfällig, wartbarer)



```
func (gq *GormQuery) GetAllMappingsQuery(mappings []*models.Mapping, projectId int,
    pageSize int, offset int, sortBy string, sortOrder string) error {
    return gq.Database.
        Where("project_id = ?", projectId).
        Preload("Elements.Concept.CodeSystem").
        Order(fmt.Sprintf("%s %s", sortBy, sortOrder)).
        Offset(offset).Limit(pageSize).
        Find(&mappings).Error
}
```

### Gorm Query für Mappings

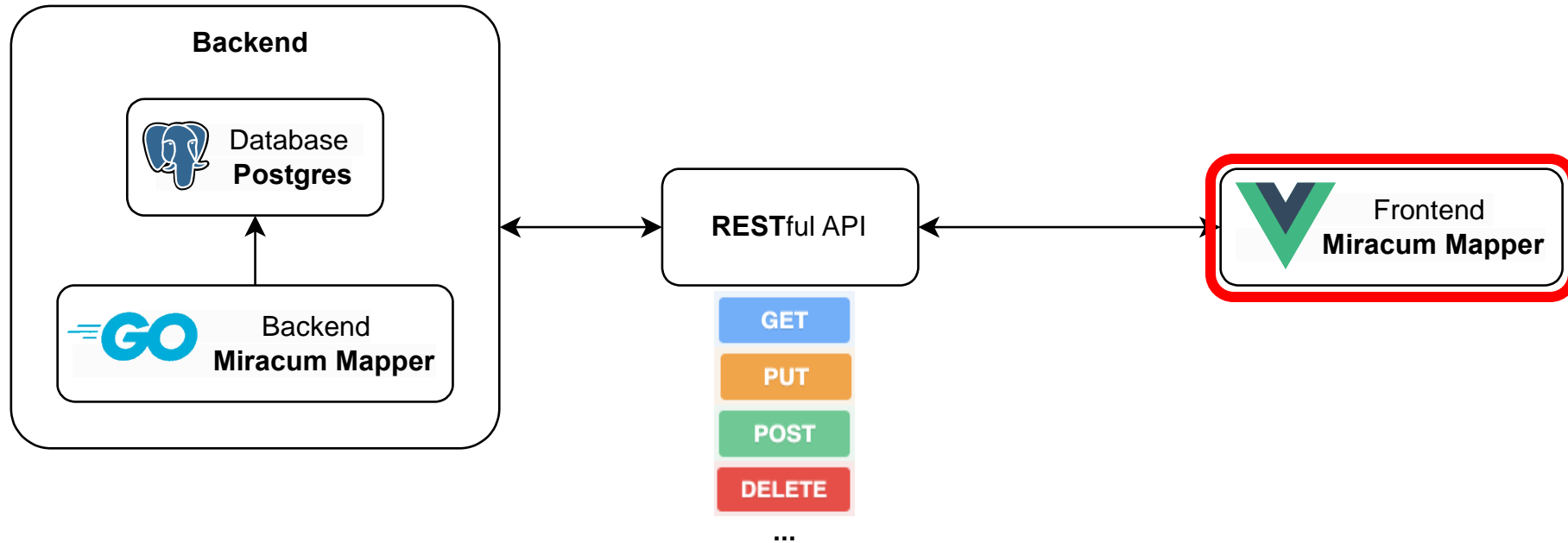
```
2024/11/28 15:01:00
[0.324ms] [rows:1] SELECT * FROM "code_systems" WHERE "code_systems"."id" = 1

2024/11/28 15:01:00 /app/internal/database/gormQuery/mappingQuery.go:86
[2.054ms] [rows:10] SELECT * FROM "concepts" WHERE "concepts"."id" IN (66525,86049,34860,8597,3829,5412,345,49740,4,22)

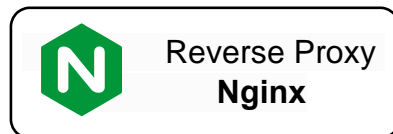
2024/11/28 15:01:00 /app/internal/database/gormQuery/mappingQuery.go:86
[3.914ms] [rows:10] SELECT * FROM "elements" WHERE "elements"."mapping_id" IN (3,40,41,42,43)

2024/11/28 15:01:00 /app/internal/database/gormQuery/mappingQuery.go:86
[16.222ms] [rows:5] SELECT * FROM "mappings" WHERE project_id = 4 ORDER BY ID ASC LIMIT 100000
[GIN] 2024/11/28 - 15:01:00 | 200 | 20.0356ms | 192.168.65.1 | GET | "/projects/4/mappings"
```

### Äquivalente SQL Statements



– Andere Dienste:



– Umsetzung Frontend als Webanwendung

– TypeScript: Typsicherheit



– Vue.js: OpenSource



– Komponentenbibliothek PrimeVue



– viele vorgefertigte UI-Komponenten

– besonders interessant: DataTable

Mapping Table

[+ Add](#) [Delete](#)  [Table Options](#) [Edit project](#)

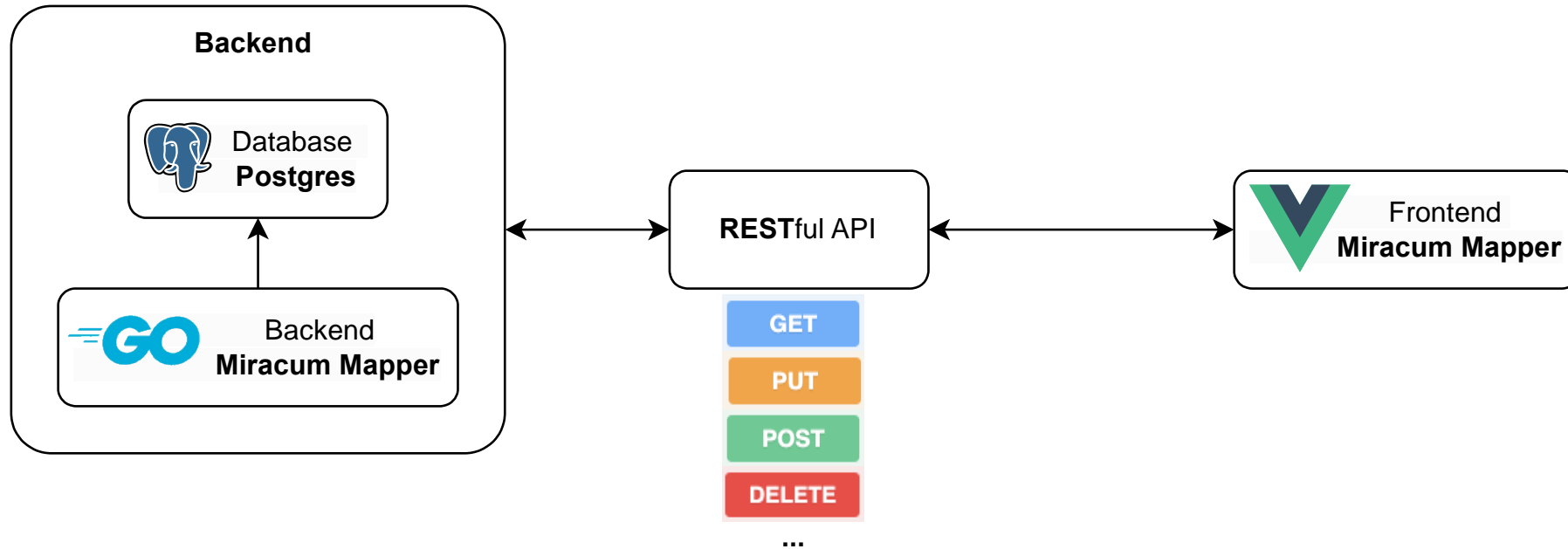
Target	Loinc test	Status	Equivalence	Comment
Code	Meaning			
<input type="text" value="100000-9"/>	<input type="text" value="Health informatics pnone"/>	<input type="text" value="Active"/>	<input type="text" value="Narrower"/>	<input type="text" value="11456"/>
60001-5	Insulin-like growth factor binding protein 3 [Mass/volume] in Serum or Plasma --3rd specimen post XXX challenge	Active	Related To	successfully mapped
<input type="text" value="30001-2"/>	<input type="text" value="Microalbumin/Creatinine"/>	<input type="text" value="Active"/>	<input type="text" value="Equivalent"/>	<input type="text" value="successfully mapped"/>
20001-4	Oxygen [Partial pressure] in Capillary blood by Transcutaneous O2 monitor -- during treatment	Active	Related To	hallo

Showing 1 to 4 of 4 mappings 10

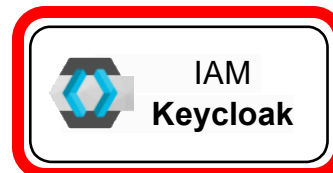
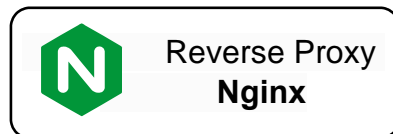
Beispiel-DataTable

- **Pinia**: zentraler Store (im Code überall auf einen globalen State zugreifen)
  - z.B. welches Projekt ist aktuell ausgewählt, welche Mappings
  - Keycloak-Token
- **keycloak-js**: Kommunikation mit Keycloak
  - beim Login auf Keycloak SSO weiterleiten
  - beim Redirect Access und Refresh Token holen
  - Nutzerinformationen wie Rolle, Nutzernamen, Email usw. holen
- **openapi-typescript**: Codegenerator für das Frontend
  - Typen, die an die API gesendet/empfangen werden als getypte JavaScript-Objekte generieren
  - Unterstützt korrekte Nutzung der API(-Typen)
  - einfach aktualisierbar, indem OpenAPI-Spezifikation anpasst

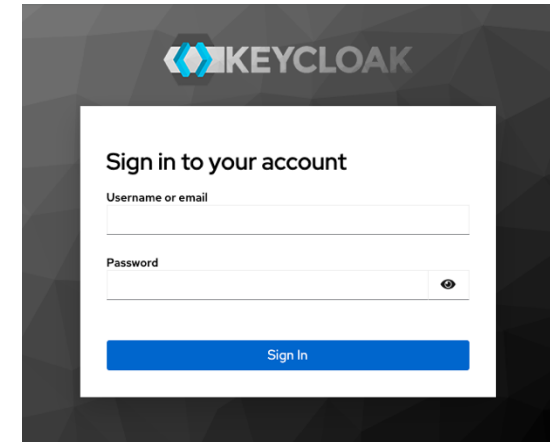




– Andere Dienste:



- Authentifizierung: Backend schützen, damit nicht jeder Zugriff auf alles hat
- Mapper nutzt den OAuth 2.0 Standard, Flow: Authorization Code Flow
- Keycloak implementiert den Standard plus extra Features
  - System in Nutzermanagement der Uni integrieren -> kein extra Login notwendig
  - Miracum Mapper muss keine Passworthashes speichern -> geringeres Sicherheitsrisiko
  - Frontend bekommt Nutzerinformationen direkt von Keycloak
- Backend nutzt Roles von Keycloak, um zwischen normalem Nutzer und Admin zu unterscheiden

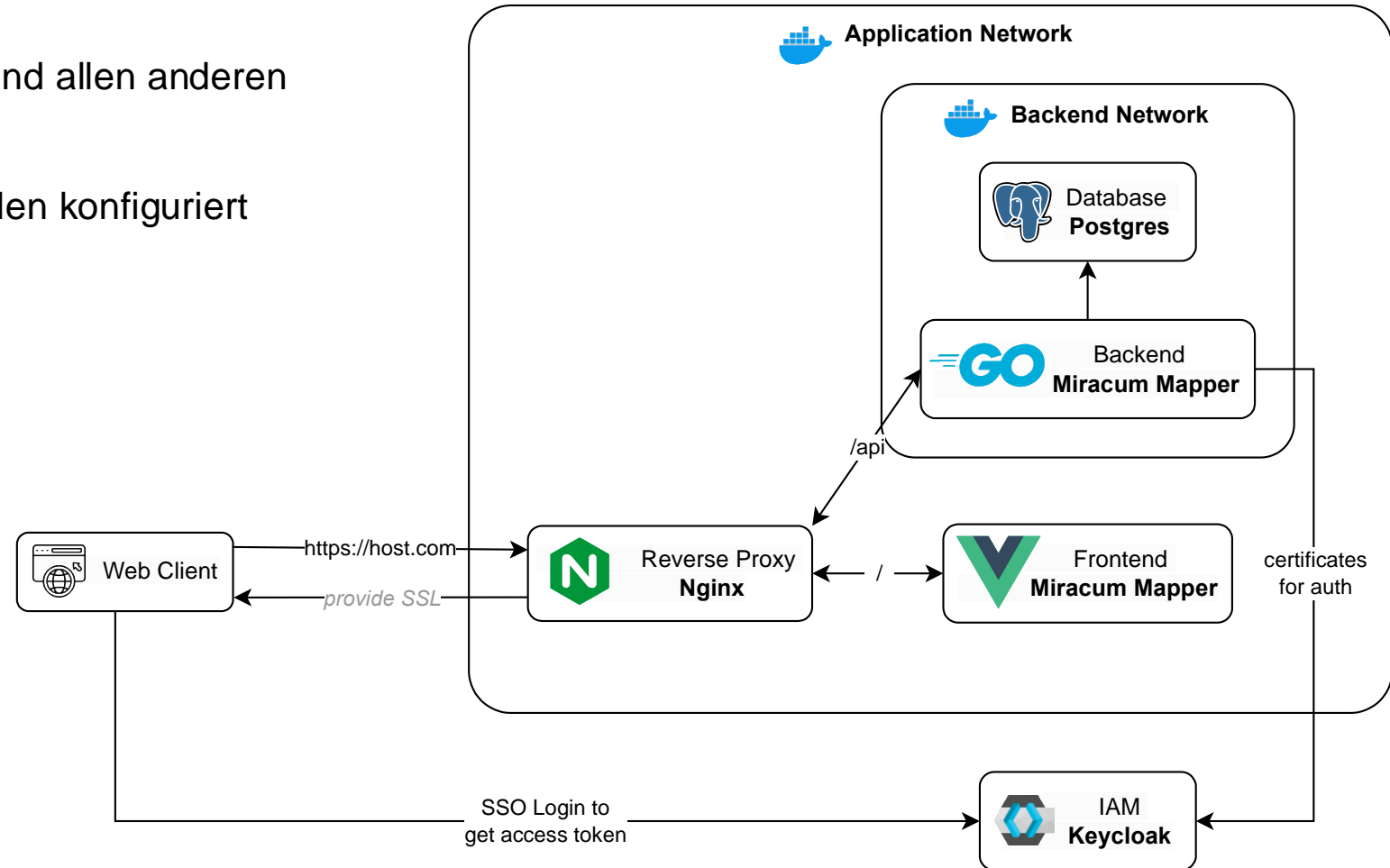


### SSO mit Keycloak

```
▼ state
  authenticated : true
  ▼ user : Reactive
    username : testuser
    token : eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSM...
    refToken : eyJhbGciOiJIUzUxMiIsInR5cCIgOiIiZWN...
  ▼ userInfo : Reactive
    id : 63ffdf30-3494-4447-a630-6fefc7e83042
    username : testuser
    firstName : Fleißiger
    lastName : Mapper
    email : test.user@miracum.de
    emailVerified : false
  ► userProfileMetadata : Object
    isAdmin : true
```


### Nutzerobjekt in Pinia

- Docker Container von Frontend, Backend und allen anderen verwendeten Diensten stehen bereit
- Dienste können mittels Environment Variablen konfiguriert werden



Zusammenspiel der verschiedenen Komponenten

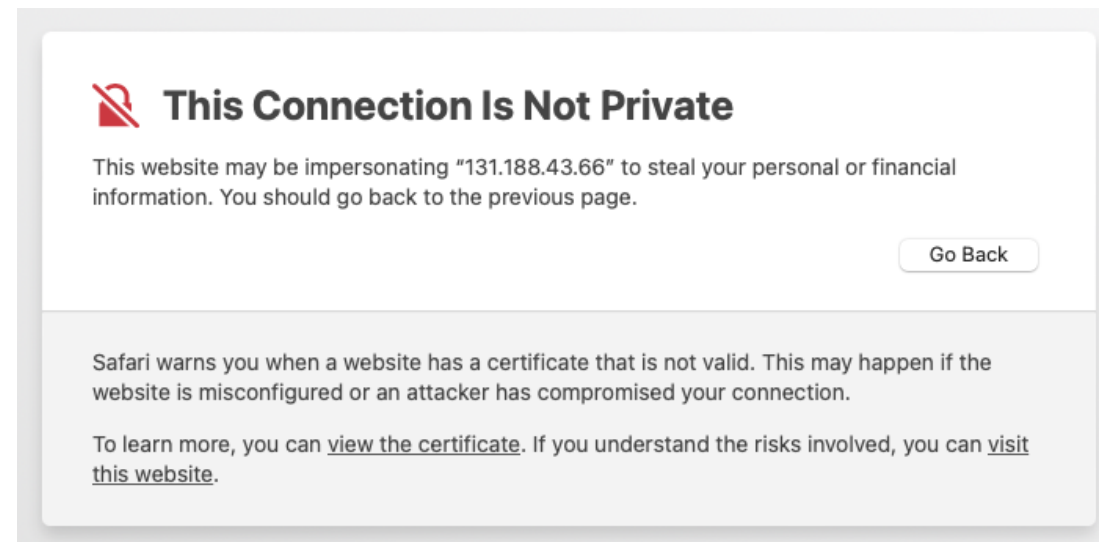


- Dokumentation: READMEs in Frontend- und Backend-Repositories
- DevContainer: Entwicklung in einheitlicher Umgebung 
- Python Skripte
  - Loinc in CSV umwandeln (kann dann über API in die Datenbank importiert werden)
  - Aus SQL-Dump des alten Systems Codesystem in CSV umwandeln



# Live Demo

- Testsystem URL: <https://131.188.43.66>
- Nutzernamen: **demo**
- Passwort: **project2024**



Warnung aufgrund von Selbst Signierten Zertifikaten

---

# Ausblick

- Erweiterung des Tools
  - in weiterem Projekt durch Jonas Hüttinger (Versionierung von Codesystemen)
  - in Masterarbeit von Henrik Herzig (Vorschläge automatisch generieren beim Mappen mit KI-Methoden)
- Weitere Features / Möglichkeiten zur Entwicklung:
  - Automatisierte Tests
  - Anbindung des Tools an Terminologie-Server
  - Arbeitslisten
  - Änderungshistorie von Mappings
  - ...

**Vielen Dank  
für Ihre Aufmerksamkeit!**

# Anhang

- 
- Backend: <https://github.com/miracum/MIRACUM-Mapper-2.0-backend.git>
  - Frontend: <https://github.com/miracum/MIRACUM-Mapper-2.0-frontend.git>



- Konfiguration des Backends mittels:
  - Konfigurations-Datei
  - Environment-Variablen

```
# Version of the config file
version: 1.0.0
database:
  # How many times should the service try to connect to the database when starting
  retry: 30
  # How long should the service sleep between each try to connect to the database
  sleep: 5
keycloak:
  # How many times should the service try to connect to the database when starting
  retry: 30
  # How long should the service sleep between each try to connect to the database
  sleep: 5
cors:
  # Allowed origins for the CORS policy
  allowed_origins:
    - "*"

```

config.yaml Datei

```
version: "3.3"
services:
  miracum-mapper:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: miracum-mapper
    ports:
      - "8080:8080"
    environment:
      - KEYCLOAK_URL=http://keycloak:8080/auth
      - KEYCLOAK_REALM=master
      - KEYCLOAK_CLIENT_ID=miracum-mapper
      - DB_HOST=miracum-postgres
      - DB_NAME=miracum_db
      - DB_USER=miracum_user
      - DB_PASSWORD=miracum_password

```

Environment Variablen

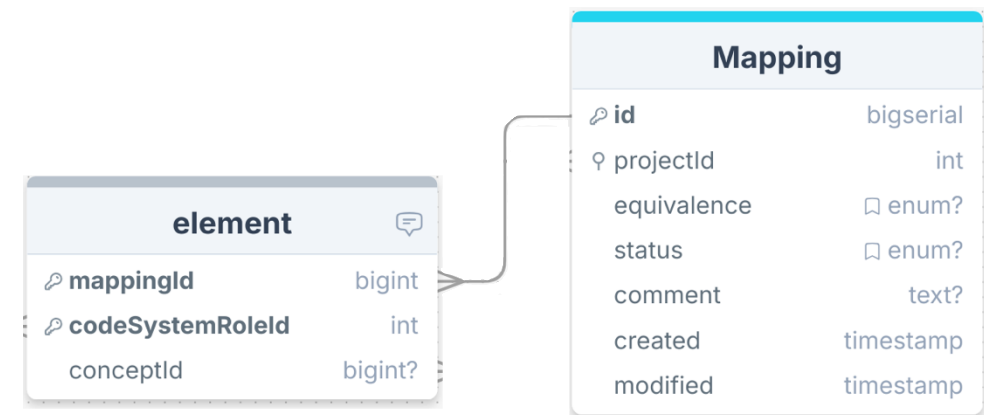
- Implementierung in Go (Gründe: compilierte Sprache: schnell, Typsicherheit)
- Tools/Dependencies:
  - gorm: ORM, um auf Datenbank mit Abstraktionsschicht zuzugreifen (weniger Fehleranfällig, wartbarer)



```
type ModelBigId struct {  
    ID          uint64    `gorm:"primaryKey" // implicitly autoIncrement  
    CreatedAt   time.Time `gorm:"<=:create"`  
    UpdatedAt   time.Time  
}
```

```
type Mapping struct {  
    ModelBigId  
    ProjectID   uint32    `gorm:"index"`  
    Equivalence *Equivalence `gorm:"type:Equivalence"`  
    Status      *Status    `gorm:"type:Status"`  
    Comment     *string  
    Elements    []Element `gorm:"constraint:OnDelete:CASCADE"`  
}
```




```
type Element struct {  
    MappingID      uint64 `gorm:"primaryKey;index"`  
    CodeSystemRoleID uint32 `gorm:"primaryKey"`  
    ConceptID      *uint64  
    Concept        Concept  
}
```

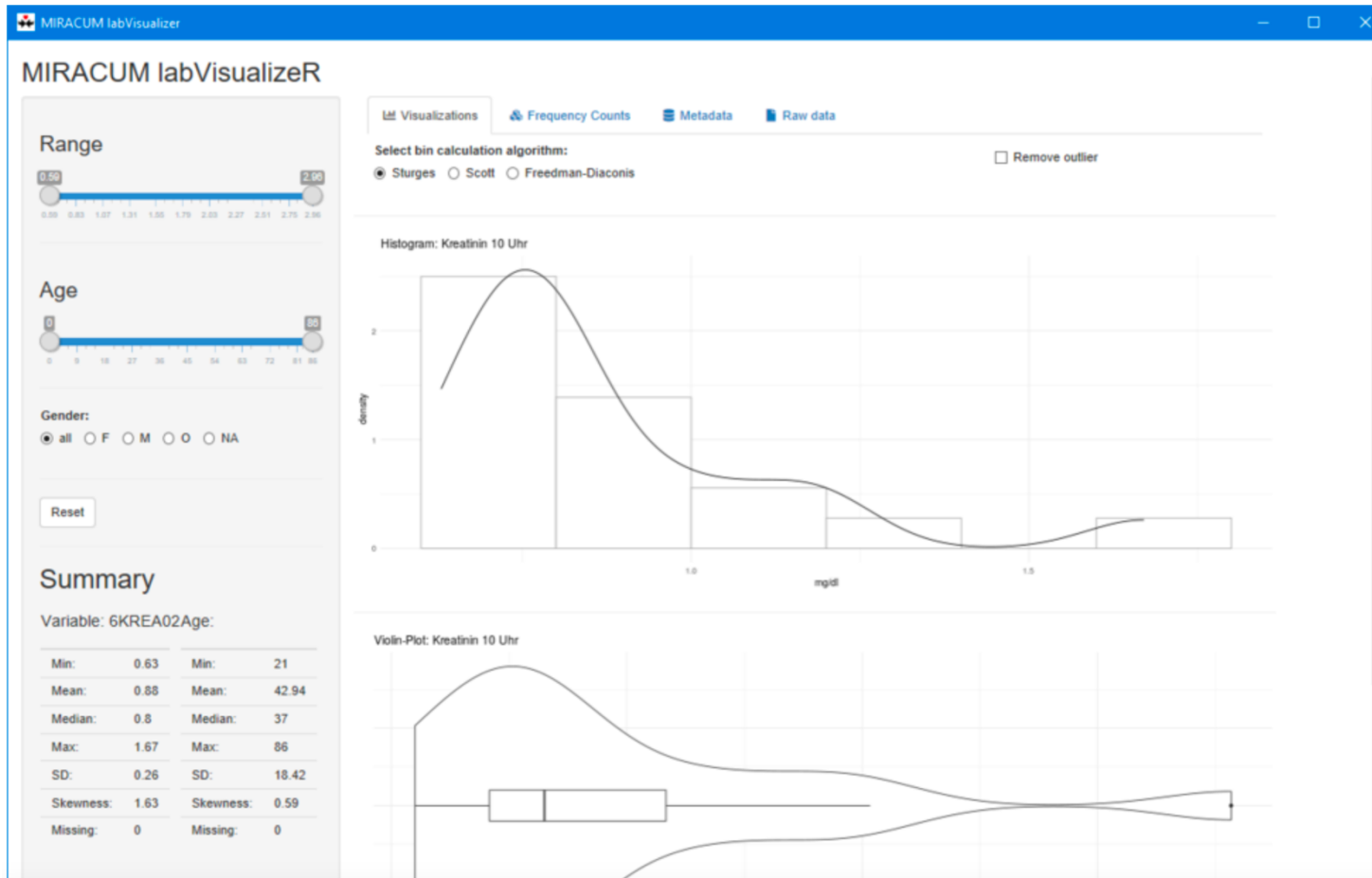


## Gorm Tabellen Definitionen

- Implementierung in Go (Gründe: compilierte Sprache: schnell, Typsicherheit)
- Tools/Dependencies:
  - Gin als Webserver Framework um z.B. middlewares (Authentifizierung) umzusetzen

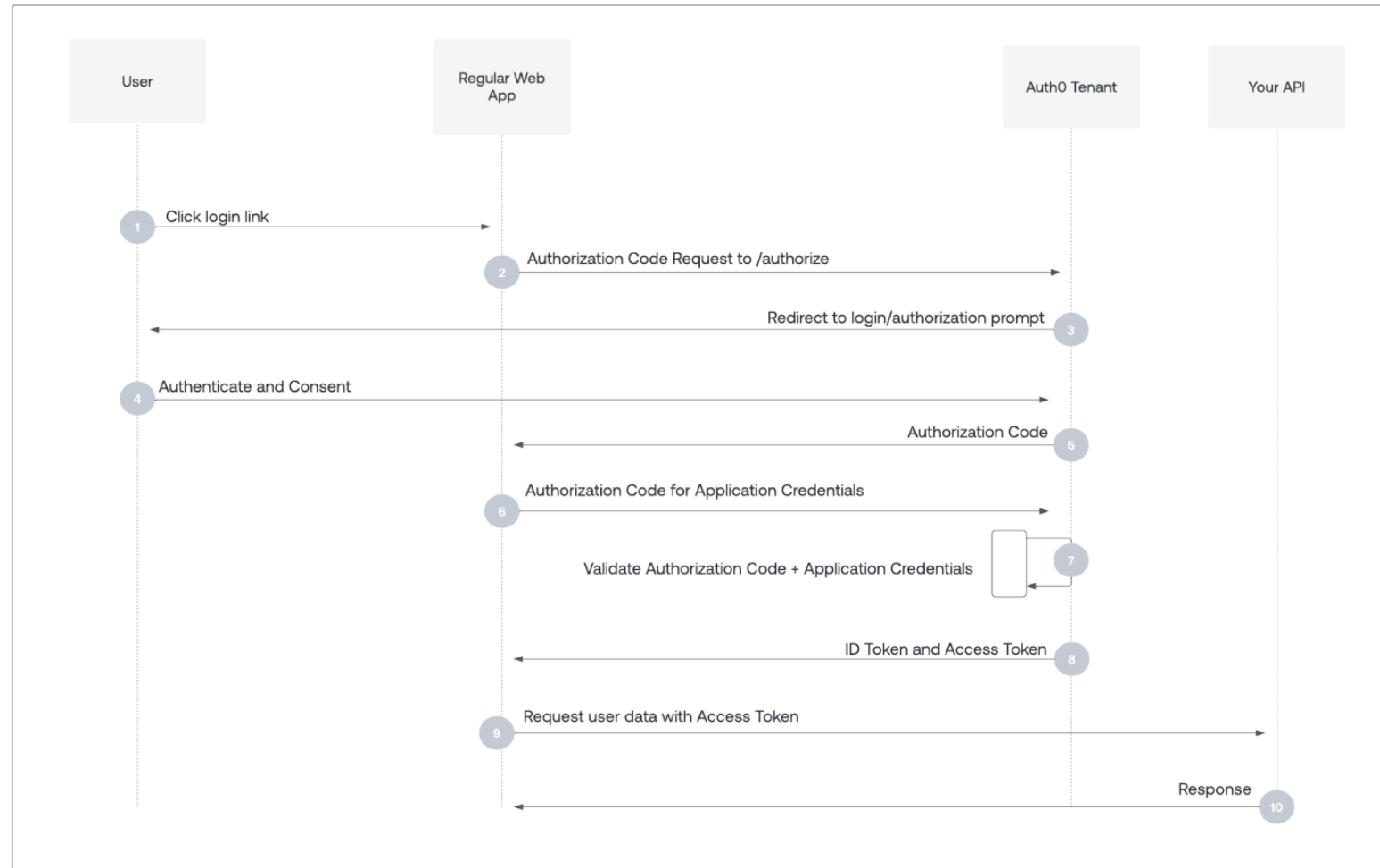


- **Vite:** Build Tool 
  - Konfiguration mittels .env Datei (z.B Keycloak Endpunkt, Backend API Endpunkt)
- **Pinia:** zentraler Store (im Code überall auf einen globalen State zugreifen) 
  - z.B. welches Projekt ist aktuell ausgewählt, welche Mappings
  - Keycloak-Token
- **keycloak-js:** Kommunikation mit Keycloak 
  - beim Login auf Keycloak SSO weiterleiten
  - beim Redirect Access und Refresh Token holen
  - Nutzerinformationen wie Rolle, Nutzernamen, Email usw. holen
- **openapi-typescript:** Codegenerator für das Frontend
  - Typen, die an die API gesendet/empfangen werden als getypte JavaScript-Objekte generieren
  - Unterstützt korrekte Nutzung der API(-Typen)
  - einfach aktualisierbar, indem OpenAPI-Spezifikation anpasst



- Evtl. weglassen
- In R programmiert
- schnell Informationen zu Laborcodes erhalten
  - Metadaten
  - Anonymisierte Beispiel Patientendaten
  - Statistiken
- In Miracum Mapper 1.0 integriert

Bild 2: Miracum Lab VisualizeR Nutzer Interface (Mate et.al)



Authorization Code flow

<https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow>