Dokumentation zur Studienarbeit

Web-Programmierung

Projektbeschreibung

Snippets - Twitter meets Stackoverflow

Snippets ist eine kleine “Social Media”-Anwendung, welche das Teilen von kurzen Codesnippets ermöglicht.
Ähnlich wie beispielsweise im Feed von Twitter werden dem Nutzer Posts in einer wählbaren Sortierung (aktuell: “Meistgelikete”, “Neueste”, “Meistgelikte der letzten 24h”, “Meistgelikte der letzten 7 Tage”) anzeigt.
Neben der Wahl der Sortierung ist auch eine (eingeschränkt) fehlertolerante Suche nach den Tags der Posts möglich.
Wie für eine Social Media Anwendung üblich können für Posts Likes vergeben und Kommentare unter den Posts hinterlassen werden. Die Kommentare können dabei ebenfalls Likes erhalten.
Der Hauptinhalt der Anwendung sind die Codesnippets:
Für diese ist beim Erstellen eines Posts eine Programmiersprache wählbar (aktuell: javascript, html, sql, python, c#), aufgrund derer die Posts im Feed ein passendes Syntax-Highlighting erhalten.
Die Anwendung besitzt außerdem ein Accounting System, welches das Erstellen und Bearbeiten von Accounts erlaubt. Das Feed ist dabei auch ohne Anmeldung einsehbar, für Interaktionen wie Likes oder Kommentare ist eine Anmeldung nötig.

Features:

	Unterstützte Features	Erweiterungsmöglichkeiten
Accounts	Account erstellen, bearbeiten Login, Logout Möglichkeit zur Interaktion nur nach Autorisierung	Unterschiedliche Nutzerrollen (z.B. Admins) Profilbilder Folgen von Accounts OAuth z.B. GitHub
Posts	Posts erstellen mit: Titel Codesnippet Programmiersprache Tags Eigene Posts löschen	Nachträgliches Bearbeiten von Posts Abonnieren von Tags, “personalisiertes” Feed

	Syntaxhighlighting je nach Programmiersprache Liken von Posts Kommentieren unter Posts Sortieren der Posts im Feed nach: Meistgeliked Neueste Meistgelikte Heute Meistgelikte diese Woche Fehlertolerante Suche von Posts nach Tags	Persönliches Feed mit eigenen Posts Posts in 50er Schritten ausliefern
Kommentare	Erstellen von Kommentaren Liken von Kommentaren	Nachträgliches Bearbeiten von Kommentaren Subkommentare (Antwort auf Kommentar) Wahlmöglichkeit Plain-Text-Kommentar oder Codesnippet Kommentare in 50er Schritten ausliefern Sortierungen für Kommentare

Technologie Stack

Vue.js (Erweiterung: prism.js)
Express
Postgres (Erweiterungen: bcrypt, pg_trgm)

Aufgabenteilung

Design: Wireframes: Moritz Kronberger
Komponenten: Lea Jell
CSS: Martin Kohnle

Anwendung: Frontend: Martin Kohnle
Backend: Lea Jell
Datenbank: Moritz Kronberger

Geteilte Komponenten:

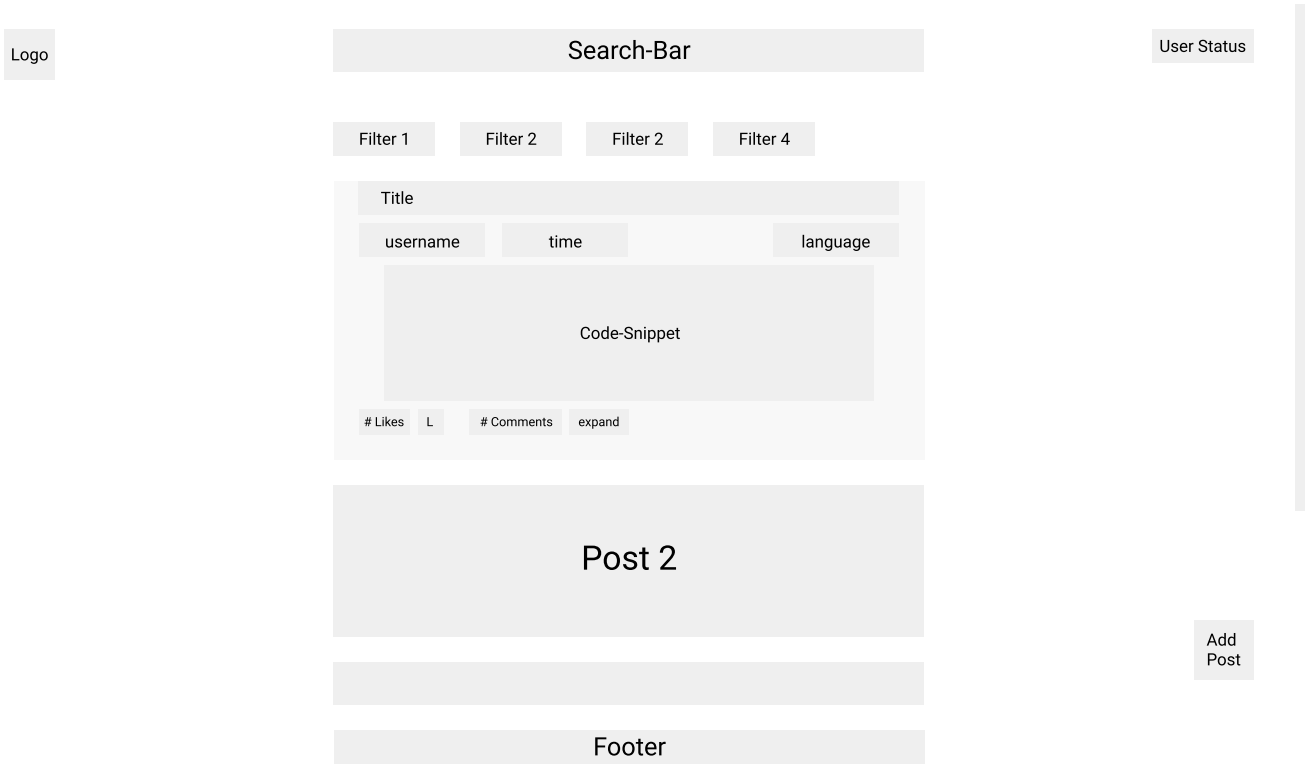
Backend: db-Dateien: Lea Jell & Moritz Kronberger
(Javascript & Postgres Queries)

Frontend: Stores: Martin Kohnle und Lea Jell (Actions)

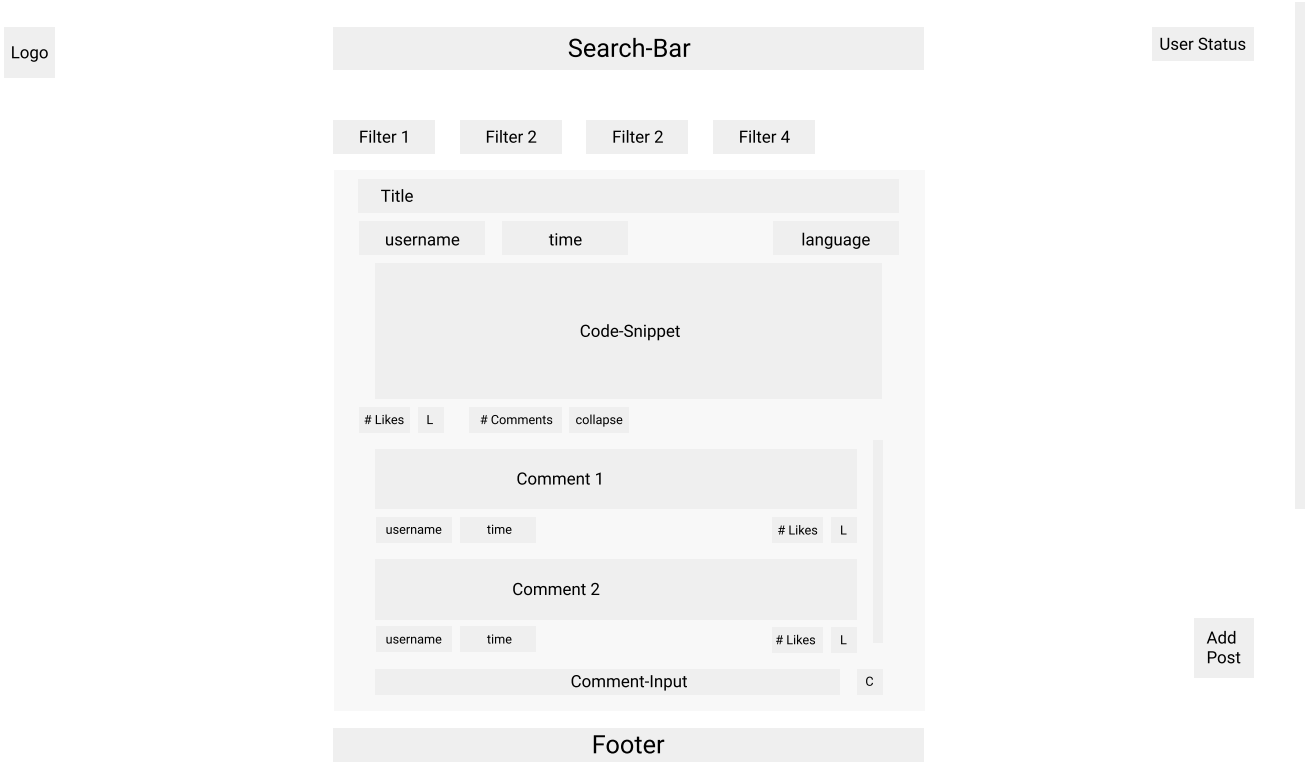
Diagramme

Seitenaufbau Frontend

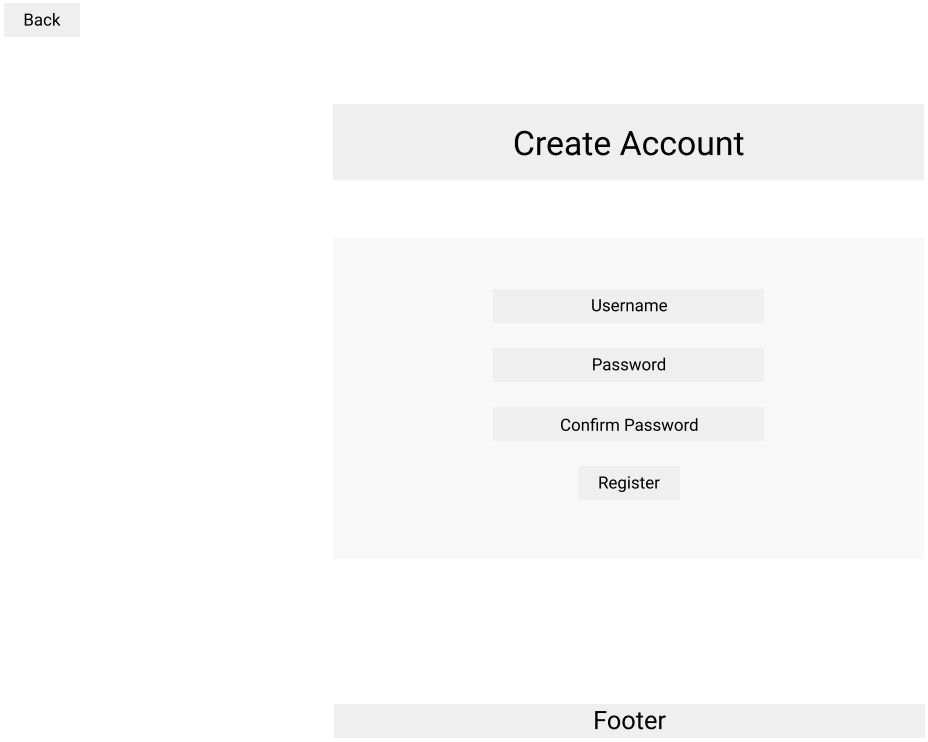
Haupt-Feed:



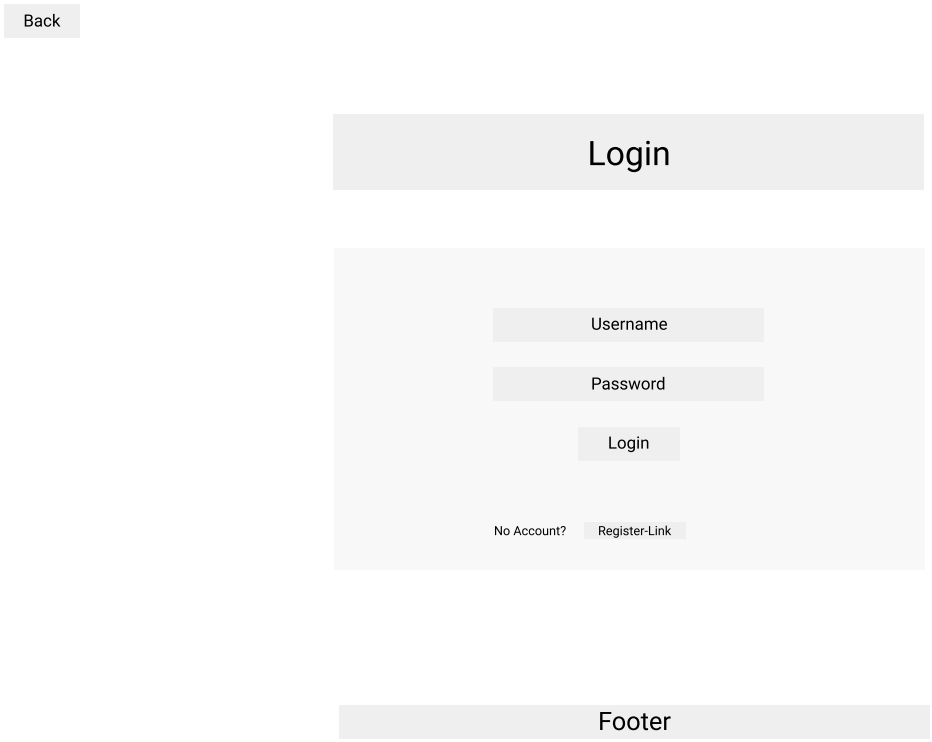
Kommentare ausgeklappt:



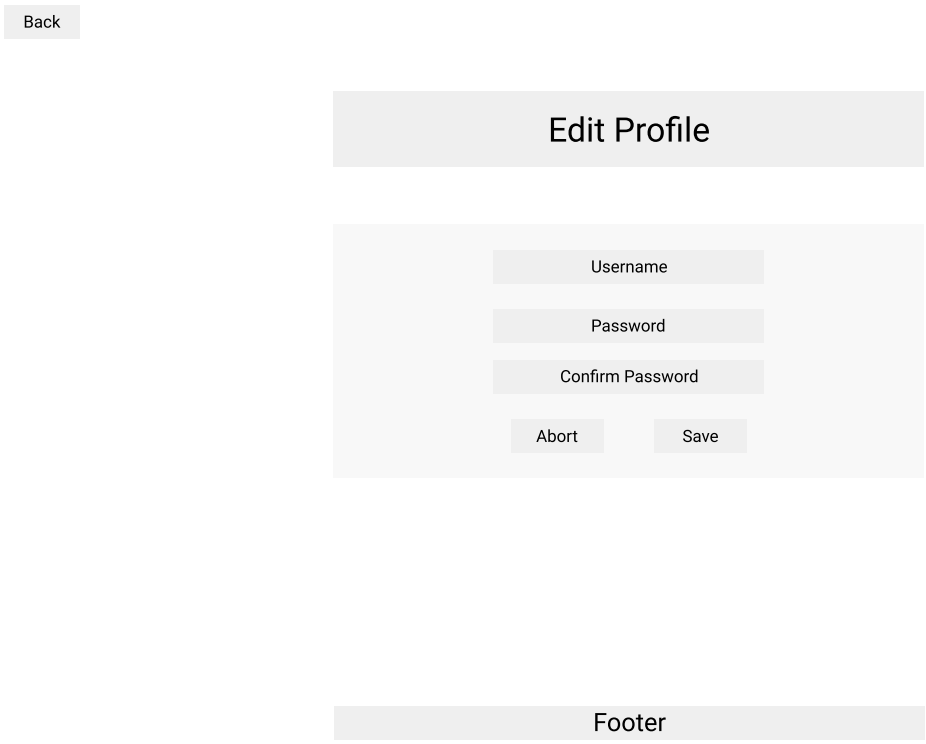
Register:



Login:



Profil Bearbeiten:



Post erstellen:

REST-API

Auth

method	route	status	routes	example	definition
POST	/v1/register	201, 400	isNotAuthorized	Request: { "username": "tallgiraffe667", "password": "imagiraffe" } Response: "3f66443e-abb9-423c-92b3-a80de18eb8df"	Neuer Account wird erstellt, alle Attribute müssen ausgefüllt werden.
POST	/v1/login	200, 401	isNotAuthorized	Request: { "username": "tallgiraffe667", "password": "imagiraffe" } Response: { "message": "logged in" }	Bei korrekten Zugangsdaten wird ein neuer Token übermittelt.

Accounts

method	route	status	routes	example	definition
GET	/v1/accounts	200	isAuthorized	Response: [{ "id": "18c40035-e459-45a1-ae3c-e50e71296526", "username": "tinykoala648"}]	Gibt alle Accounts zurück
POST	/v1/accounts	201, 400	isAuthorized	Request: { "username": "tallgiraffe667", "password": "imagiraffe" } Response: { "id": "d1214eb3-b0eb-4e81-9707-73bc7c112237" }	Gibt URL des neuen Accounts zurück
GET	/v1/accounts/:id	200, 404	isAuthorized	Response: { "id": "b8b5fba4-c295-4497-9a67-c2d8112119c4", "username": "a" }	Kann nur sich selbst zurückgeben, ID muss mit der Benutzer-ID übereinstimmen
PATCH	/v1/accounts/:id	200, 404, 400	isAuthorized	Request: { "username": "tallgiraffe664" } Response: { "id": "b8b5fba4-c295-4497-9a67-c2d8112119c4", "data": null, "status": 200, "message": null, "pgstate": "00000", "constraint": null }	Bei übereinstimmender ID wird das Account Tupel des eingeloggten Benutzers geändert. Nur zu verändernde Attribute werden angegeben.
DELETE	/v1/accounts/:id	200, 404	isAuthorized	Response: { "id": "b8b5fba4-c295-4497-9a67-c2d8112119c4", "data": null, "status": 200, "message": null, "pgstate": "00000", "constraint": null }	Nur der eingeloggte Account kann gelöscht werden.

Categories

method	route	status	routes	example	definition
GET	/v1/categories	200		Response:[{ "id": "297b44f4-cdfa-406a-893a-bb8518f2245f", "name": "web" }, { "id": "2b36a775-eaad-4076-8d0b-9031183e00bc", "name": "js" }]	Gibt alle aktuell vorhandenen Categories zurück, kann durch eine Query erweitert werden, um nach Namen oder ID zu suchen.
POST	/v1/categories	201, 400	isAuthorized	Request: { "name": "errormessage" } Response: { "id": "a7ba3d69-f3f5-4c66-9535-d09361f0af86", "data": null, "status": 201, "message": null, "pgstate": "00000", "constraint": null }	Erstellt eine neue Category und gibt deren ID zurück. (Diese kann für die Erstellung der Relation zwischen Post und Category genutzt werden)
GET	/v1/categories/:id	200, 404		Response: { "id": "c001a2aa-8c1d-414d-8046-7aee717570dc", "name": "errormessage" }	Gibt alle Attribute der angeforderten Category zurück.

Comments

method	route	status	routes	example	definition
GET	/v1/comments	200		Response: [{ "id": "9ebc80cb-b91a-4cee-b5e6-d09607fbb504", "creation_time": "2021-06-15T19:43:43.671Z", "content": "Nice post!", "post_id": "1e76a7b1-8109-4bec-8fac-c00b5abc9764", "user_id": "da258f6f-514d-48b6-9d47-fb9ae22c6dee", "profile_picture": null, "num_likes": "0" }]	Gibt alle Comments zurück, kann um einen Query erweitert werden, der nach ID oder User-ID filtert.
POST	/v1/comments/:post_id	201, 400	isAuthorized	Request: { "content": "Great!" } Response: { "id": "4ed5ac5a-b240-441e-9468-627ff29dd67c", "data": null, "status": 201, "message": null, "pgstate": "00000", "constraint": null }	Gibt die ID des neuen Comments zurück. Erwartet nur den Content, die ID des Posts, unter den kommentiert wird, gibt das Frontend an. Die ID des Users wird über den Request geholt.
GET	/v1/comments/:id	200, 404		Response: { "id": "fc37f330-1581-4230-aa43-9d64d763f7a9", "creation_time": "2021-06-17T09:13:05.560Z", "content": "Nice post!", "post_id": "d23bec1a-9895-405d-8763-cf6e441f9619", "user_id": "7492ccbe-c2b2-49a0-88c0-40460508b01d", "profile_picture": null, "num_likes": "0" }	Gibt alle Informationen zum gesuchten Comment zurück.
PATCH	/v1/comments/:id	200, 404, 400	isAuthorized	Request: { "content": "Great code!" } Response: { "id": "4ed5ac5a-b240-441e-9468-627ff29dd67c", "data": null, "status": 200, "message": null, "pgstate": "00000", "constraint": null }	Bei übereinstimmender ID wird der Comment abgeändert.
DELETE	/v1/comments/:id	200, 404	isAuthorized	Response: { "id": "4ed5ac5a-b240-441e-9468-627ff29dd67c", "data": null, "status": 200, "message": null, "pgstate": "00000", "constraint": null }	Nur der Verfasser des Comments kann diesen wieder löschen.

has Categories

method	route	status	routes	example	definition
POST	/v1/has-categories	201, 400	isAuthorized	Request: { "category_id": "a4d8a25d-2613-4379-9943-4f97fc541c3e", "post_id": "cd6f81d9-1860-420f-a051-1c1b59b2d3ba" } Response: { "id": null, "data": { "id1": "902a8543-7842-4f14-a953-969566cb1198", "id2": "ec9b62ff-d5c1-41ca-9f60-ce5b6a2fcf49" }, "status": 201, "message": null, "pgstate": "00000", "constraint": null }	Nur der Ersteller des Posts kann hier die Relation zwischen Category und Post knüpfen und muss dafür die ID des Posts und die ID der Category übergeben. Wenn beim Erstellen des Posts eine Category übergeben wird, wird diese Methode automatisch aufgerufen, um die Relation direkt zu verknüpfen, damit im Frontend nicht für den Post mehrere Methoden aufgerufen werden müssen.
DELETE	/v1/has-categories	200, 404	isAuthorized	Request: { "category_id": "a4d8a25d-2613-4379-9943-4f97fc541c3e", "post_id": "cd6f81d9-1860-420f-a051-1c1b59b2d3ba" } Response: { "id": null, "data": { "id1": "902a8543-7842-4f14-a953-969566cb1198", "id2": "ec9b62ff-d5c1-41ca-9f60-ce5b6a2fcf49" }, "status": 201, "message": null, "pgstate": "00000", "constraint": null }	Nur der Ersteller des Posts kann eine Category Relation wieder löschen. Die Category wird automatisch in der Datenbank gelöscht, falls es keinen anderen Post mit der gleichen Category gibt.

Languages

method	route	status	routes	example	definition
GET	/v1/languages	200		Response: [{ "id": "39719c4d-558d-404b-8298-03a49cdb83ea", "name": "javascript" }]	Gibt alle Languages mit ID und Name zurück.
GET	/v1/languages/:id	200, 404		Response: { "id": "9474ae3f-2d2e-4db1-85ba-db389286b9fe", "name": "c++" }	Gibt nur ID und Name der Language mit der übereinstimmenden ID zurück.

Posts

method	route	status	routes	example	definition
GET	/v1/posts?sorting_id={id}&query_string={category}	200		Response: [{ "id": "77ab8d3d-2b96-4e75-be6f-1fa8d6496327", "creation_time": "2021-06-17T09:13:05.560Z", "title": "A Hello World Post", "content": "Hello World!", "language": "javascript", "user_id": "ce3f2ba0-9be3-4397-b713-b79f4d87ac43", "username": "smallladybug804", "profile_picture": null, "num_likes": "1", "num_comments": "2", "categories": "{code,es6}" }]	Gibt alle Posts zurück, die zu den Queries passen, auch unauthorisiert, da man auf der Startseite auch Posts sehen soll. Die Posts werden mit allen Daten angezeigt, auch mit verknüpften Eigenschaften wie Likes oder Categories. Vom Frontend wird immer eine gültige Sorting-ID übergeben und wahlweise ein String für die Suche nach bestimmten Categories.
GET	/v1/posts/categories	200, 404	isAuthorized	Request: { "id": "18e40bb6-cbe5-4528-9a76-39d593d9bf53" } Response: [{ "category_id": "ec9b62ff-d5c1-41ca-9f60-ce5b6a2fcf49", "name": "web", "post_id": "18e40bb6-cbe5-4528-9a76-39d593d9bf53" }, { "category_id": "df207d09-ae69-4323-b656-88392f3d8aff", "name": "js", "post_id": "18e40bb6-cbe5-4528-9a76-39d593d9bf53" }	Erwartet eine gültige Post ID und gibt die ID und alle Categories mit Category-ID, Namen und der Post ID zurück. Wird in der patch-Methode von Post benutzt, um die neuen Categories mit den alten Categories zu vergleichen.
POST	/v1/posts	201, 400	isAuthorized	Request: { "title": "Timer", "content": "Timer in Javascript", "language_id": "39719c4d-558d-404b-8298-03a49cdb83ea", "categories": ["js"] } Response: { "id": "4fae52bb-b6c9-4601-be49-ef95ee87ab7b", "data": null, "status": 201, "message": null, "pgstate": "00000", "constraint": null }	Gibt Id des neuen Posts zurück. Die Categories werden ebenfalls direkt hier erstellt und verknüpft. Dazu wird zuerst überprüft, ob es die Category schon gibt. Gibt es sie noch nicht, wird diese automatisch erstellt und die ID gespeichert, ansonsten wird die bereits vorhandene ID der Category geholt. Diese wird dann über hasCategory mit dem Post verknüpft. Es kann sowohl keine als auch beliebig viele Categories übergeben werden.
GET	/v1/posts/:id	200, 404		Response: { "id": "62751e6f-6453-4f2f-afa4-6ac357b0e2dc", "user_id": "1f54d171-3dd8-404a-bfa3-60ebbb5428a1" }	Gibt Id des Posts und die UserId des Erstellers zurück.
PATCH	/v1/posts/:id	200, 404, 400	isAuthorized	Request: { "title": "Timer with millis()", "content": "A timer with function millis() in Javascript", "categories": ["js", "timer"] } Response: { "id": "4fae52bb-b6c9-4601-be49-ef95ee87ab7b", "data": null, "status": 200, "message": null, "pgstate": "00000", "constraint": null }	Nur der Ersteller des Posts kann diesen bearbeiten. Die Categories können ebenfalls über diese Route angepasst werden. Dafür werden zuerst die Categories des alten Posts geholt und mit den neuen Categories verglichen. Nicht mehr vorhandene werden gelöscht. Bei neuen Categories wird zuerst überprüft, ob diese bereits in der DB sind und eine ID haben, und wenn nicht, wird die Category erstellt und die ID zurückgegeben. Anschließend wird mit dieser ID die Verknüpfung über hasCategories erstellt.
DELETE	/v1/posts/:id	200, 404	isAuthorized	Response: { "id": "4fae52bb-b6c9-4601-be49-ef95ee87ab7b", "data": null, "status": 200, "message": null, "pgstate": "00000", "constraint": null }	Nur der Ersteller des Posts kann diesen wieder löschen.

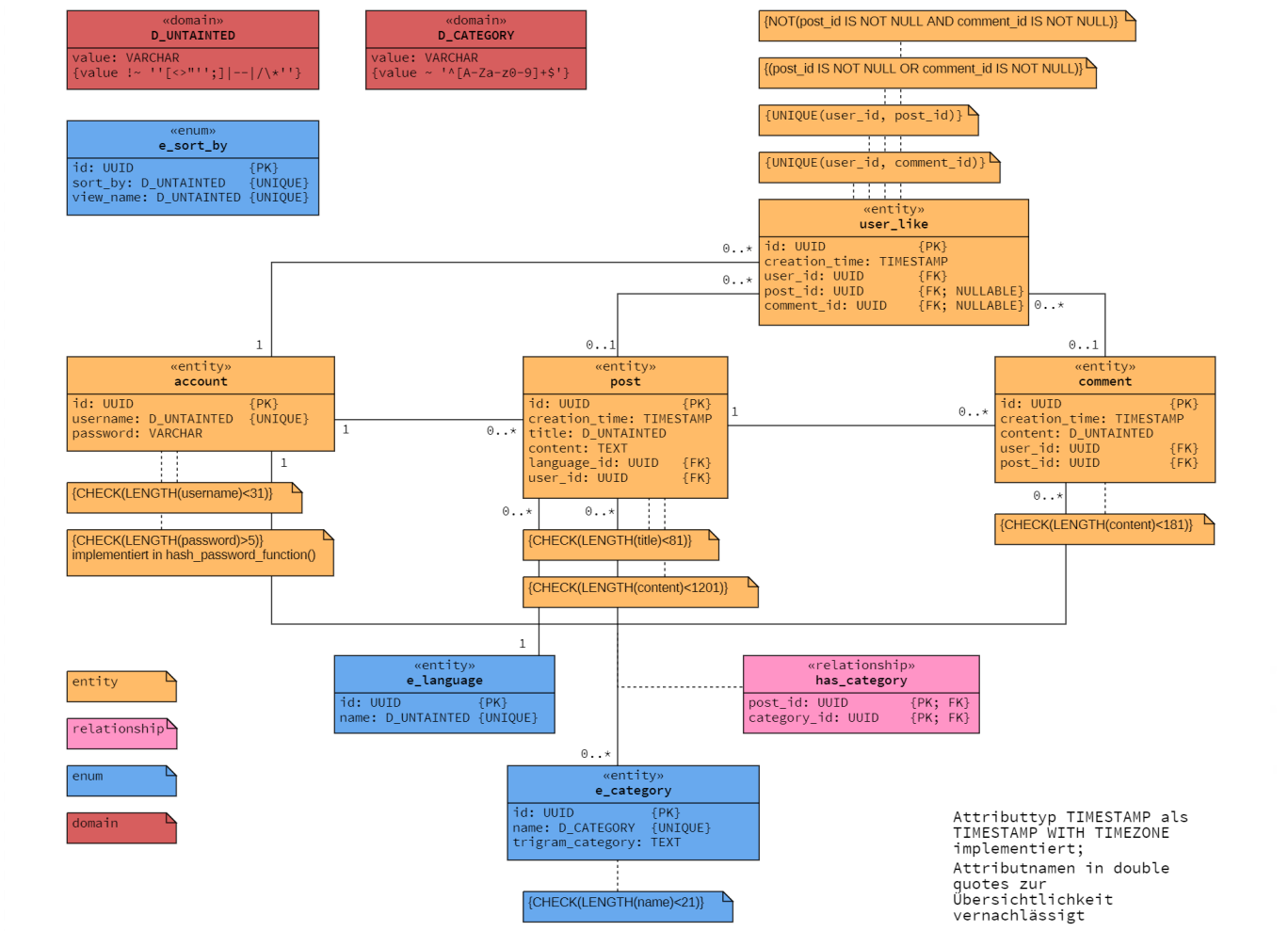
Sorting

method	route	status	routes	example	definition
GET	/v1/sorting	200		Response: [{ "id": "2284ce2f-2e18-4483-9fde-984a9c488189", "sort_by": "most liked" }, { "id": "a0031e55-4b00-486c-bb78-e00f44b8701e", "sort_by": "newest" }, { "id": "10ff83c4-3ab1-4d8c-a167-97e1588e6dd1", "sort_by": "best of today" }, { "id": "ce132445-a64d-4b01-aaab-f7e32ab63e69", "sort_by": "best of this week" }]	Gibt alle Sortings mit ID und Namen der Sortierung zurück.
GET	/v1/sorting/:id	200, 404		Response: { "id": "2284ce2f-2e18-4483-9fde-984a9c488189", "sort_by": "most liked" }	Gibt nur die Sortierung an, die per ID angefordert wurde.

UserLikes

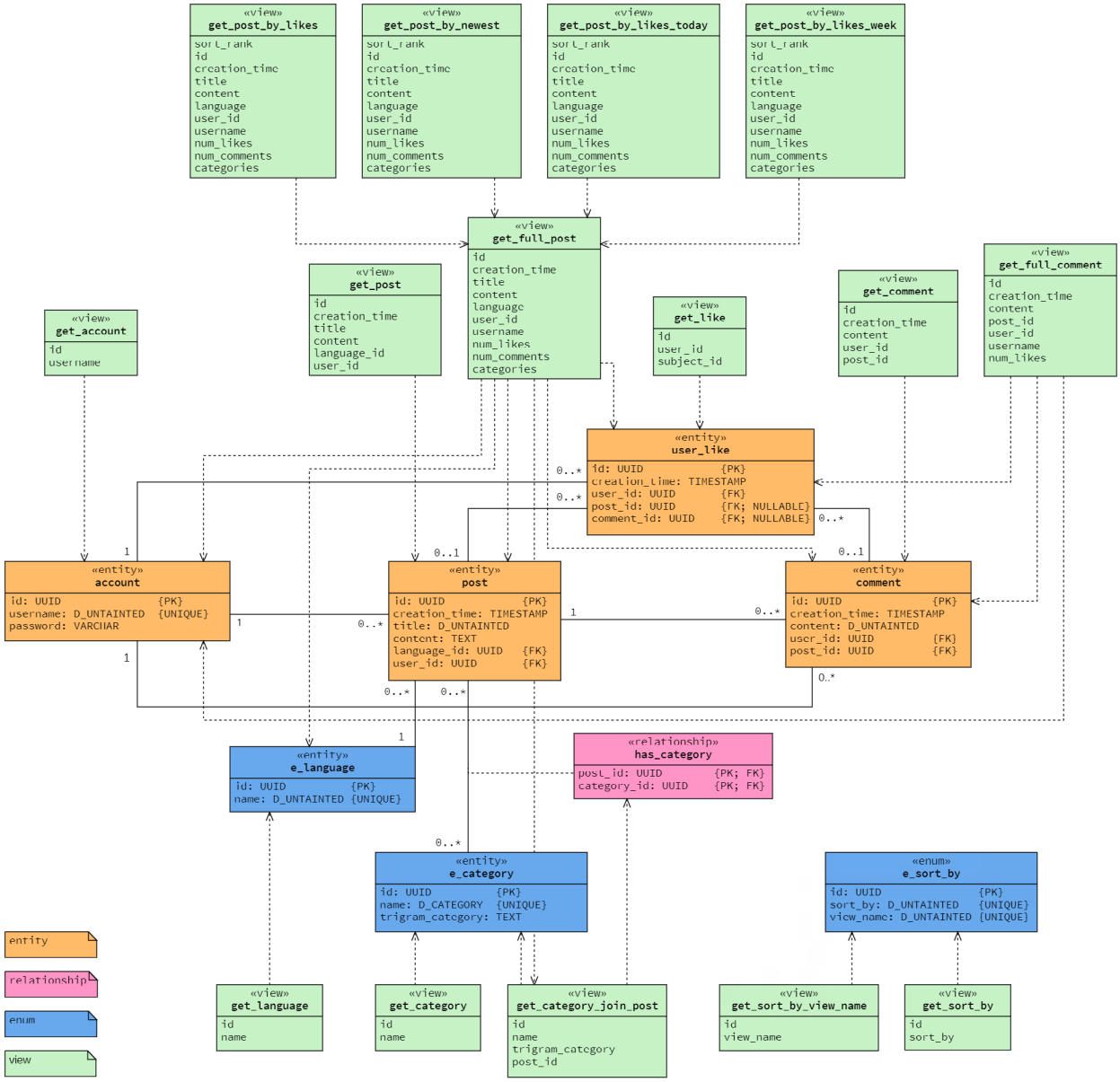
method	route	status	routes	example	definition
GET	/v1/user-likes	200		Response: [{ "id": "e4ed8073-c954-49b2-bc24-d3dd77d7e70f", "user_id": "7492ccbe-c2b2-49a0-88c0-40460508b01d", "subject_id": "d23bec1a-9895-405d-8763-cf6e441f9619" }]	Gibt alle UserLikes zurück
POST	/v1/user-likes	201, 400	isAuthorized	Request: { "comment_id": "6f1e755c-3594-4a29-aab6-1d09144477f4" } Response: { "id": "422abdab-1589-400c-a0bd-708569f68753", "data": null, "status": 201, "message": null, "pgstate": "00000", "constraint": null }	Gibt ID des neuen Likes zurück, User-ID muss nicht im Json gesetzt werden, da diese automatisch vom Request geholt wird. Es wird entweder comment_id oder post_id übergeben, nie beides.
GET	/v1/user-likes/:id	200, 404		Response: [{ "id": "e4ed8073-c954-49b2-bc24-d3dd77d7e70f", "user_id": "7492ccbe-c2b2-49a0-88c0-40460508b01d", "subject_id": "d23bec1a-9895-405d-8763-cf6e441f9619" }]	Gibt den erwarteten UserLike mit ID, User-ID und Subject-ID zurück. Die Subject-ID fasst comment_id und post_id zusammen.
DELETE	/v1/user-likes	200, 404	isAuthorized	Request: { "comment_id": "5c70fc5d-c073-4326-b8b2-aa7de1662559", "post_id": null } Response: { "id": "422abdab-1589-400c-a0bd-708569f68753", "data": null, "status": 200, "message": null, "pgstate": "00000", "constraint": null }	Ein bereits existierender User-Like wird gelöscht. Damit nur der aktuell eingeloggte User seine Likes löschen kann, wird die User-ID nicht übergeben, sondern aus der Request-ID geholt. Der User übergibt entweder comment_id oder post_id, die Datenbank bekommt diese als subject_id und sucht nach dem passenden Like und löscht dieses.

Datenmodell



Datenmodell mit Views:

Domains, Attribut-Typen der Views
und Constraints der Klassen zur
Übersichtlichkeit ausgeblendet;
Attributnamen in double quotes
ebenfalls vernachlässigt



Relationales Datenmodell:

```
e_language: „id“, „name“
{PRIMARY KEY: „id“}
{UNIQUE: „name“}

e_category: „id“, „name“, „trigram_category“
{PRIMARY KEY: „id“}
{UNIQUE: „name“}
{CHECK (LENGTH(„name“)<21)}

e_sort_by: „id“, „sort_by“, „view_name“
{PRIMARY KEY: „id“}
{UNIQUE: „sort_by“}
{UNIQUE: „view_name“}

account: „id“, „username“, „password“
{PRIMARY KEY: „id“}
{UNIQUE: „username“}
{CHECK (LENGTH(„username“)<31)}

post: „id“, „creation_time“, „title“, „content“, „language_id“, „user_id“
{PRIMARY KEY: „id“}
{FOREIGN KEY: „language_id“ REFERENCES e_language(„id“)}
{FOREIGN KEY: „user_id“     REFERENCES account(„id“)}
{CHECK (LENGTH(„title“)<81)}
{CHECK (LENGTH(„content“)<1201)}

has_category: „post_id“, „category_id“
{PRIMARY KEY: „post_id“, „category_id“}
{FOREIGN KEY: „post_id“     REFERENCES post(„id“)}
{FOREIGN KEY: „category_id“ REFERENCES e_category(„id“)}

comment: „id“, „creation_time“, „content“, „user_id“, „post_id“
{PRIMARY KEY: „id“}
{FOREIGN KEY: „user_id“ REFERENCES account(„id“)}
{FOREIGN KEY: „post_id“ REFERENCES post(„id“)}
{CHECK (LENGTH(„content“)<181)}

user_like: „id“, „creation_time“, „user_id“, „post_id“, „comment_id“
{PRIMARY KEY: „id“}
{FOREIGN KEY: „user_id“    REFERENCES account(„id“)}
{FOREIGN KEY: „post_id“    REFERENCES post(„id“)}
{FOREIGN KEY: „comment_id“ REFERENCES comment(„id“)}
{UNIQUE: „user_id“, „post_id“}
{UNIQUE: „user_id“, „comment_id“}
{CHECK („post_id“ IS NOT NULL OR „comment_id“ IS NOT NULL)}
{CHECK (NOT(„post_id“ IS NOT NULL AND „comment_id“ IS NOT NULL))}
```