1. DB API specsheet

Time: 17-03-2024 @ 11:43
Author: Sebastian Lindau-Skands @ GNUF | Backend
Reciever: GNUF | DB Team
Version: 1.5.2

2. Overview

DB: sqlite
API: C# (IMC for frequent values)
Env: alpine/sqlite:3.48.0
Dedicated com-port for API: 9012
API Auth Method: Token: 64-bit

3. Database Schema

3.1 Community

Column name	Type	Constraints	Description
ID	INT	PRIMARY KEY	Unique Identifier
NAME	TEXT	UNIQUE. NOT NULL. CHECK (LENGTH(NAME) \< 100)	Community Name
Description	TEXT	CHECK (LENGTH(description) \< 2000)	Community description
IMG_PATH	TEXT		URL to community image
MEMBER_COUNT	INT	DEFAULT 0. NOT NULL	Number of members
TAGS	INT\[\]	NOT NULL	Content tags
POST_IDs	INT\[\]		Array of post ID\'s (FK to posts.id)

3.2 User

Column Name	Type	Constraints	Description
ID	INT	PRIMARY KEY	Unique Identifier
EMAIL	TEXT	UNIQUE. NOT NULL	User\'s Email
USERNAME	TEXT	UNIQUE. NOT NULL. CHECK (LENGTH(USERNAME) \< 100)	Unique

```
Username
PASSWORD        TEXT      NOT NULL. CHECK (LENGTH(PASSWORD) \< 1000)          Hashed
Password (sha256)
IMG_PATH        TEXT                                                          URL TO
PP
POST_IDs        INT\[\]                                                       Array of
post IDs (FK to posts.id)
COMMUNITY_IDs   INT\[\]                                                       Array of
communities (FK to communities.id)
ADMIN           BOOL      NOT NULL. DEFAULT FALSE                             ADMIN
FLAG
TAGS            INT\[\]                                                       Array of
tags for content recommendation
```

3.3 Posts

```
Column Name    Type        Constraints                         Description
-------------- ----------- ----------------------------------- ------------------------
---------------------------------
POST_ID        INT         UNIQUE. NOT NULL                    Unique Identifier
TITLE          TEXT        NOT NULL. CHECK (LENGHT \< 1000)    Post title
MAIN_TEXT      TEXT        CHECK (LENGTH \< 100k)              Post body
AUTH_ID        INT         NOT NULL                            Author ID (FK. User.ID)
COM_ID         INT         NOT NULL                            Community ID (FK.
Community.ID)
TIMESTAMP      INT         NOT NULL                            Time of post
LIKES          INT         NOT NULL. DEFAULT 0                 Likes on post
DISLIKES       INT         NOT NULL. DEFAULT 0                 Dislikes on post
POST_ID_REF    INT                                             Refference to original
post (for reposts) (FK. Post.ID)
COMMENT_FLAG   BOOL        NOT NULL                            Indicates comment
instead of post
COMMENT_CNT    INT         NOT NULL. DEFAULT 0                 Comment count
Comments       INT\[\]                                         Array of post id\'s for
comments (FK. Post.ID)
```

if POST_ID_REF is set, this indicates repost by default UNLESS COMMENT_FLAG is set too. COMMENT_FLAG cannot be set without POST_ID_REF

4. API Endpoints

4.1 Authentication

4.1.1 User Login

Endpoint: `POST /api/auth/login`
Desc: Auth user and return access token
Request body:

```
{
  "username": "string",
  "password": "string"
}
```

Response:

```
{
  "user_id": "INT"
}
```

if user not found, refer to "5.1 HTTP codes"

4.1.2 User Registration

Endpoint: `POST /api/auth/register`
Desc. Register a new user, and return access token
Request body:

```
{
  "email": "string",
  "username": "string",
  "password": "string"
}
```

Response:

```
{
  "user_id": "INT"
}
```

4.2 User Management

4.2.1 Get User Profile

Endpoint: `GET /api/user/profile/{user_id}`
Desc: Retrieve user data
Response:

```
{
  "id": "INT",
  "email": "string",
  "username": "string",
  "img_path": "string",
  "post_ids": ["INT"],
  "community_ids": ["INT"],
  "tags": ["INT"],
  "admin": "boolean"
}
```

4.2.2 Delete User Account

Endpoint: `DELETE /api/user/remove/{user_id}`
desc: Deletes the account
Response: `200 (ok)`

4.2.3 Update User Profile (user)

Endpoint: `PUT /api/user/update/user/{user_id}`
desc: Updates user profile
Request body:

```
  {
    "img_path": "string",
    "password": "string",
  }
```

Response: `200 (OK)`

4.2.4 Update User Profile (backend)

Endpoint: `PUT /api/user/update/backend/{user_id}`
desc: update backend parameters
Request body:

```
  {
    "communities" : [INT],
    "POST_IDs" : [INT],
    "TAGS" : [INT]
  }
```

Response: `200 (OK)`

4.3 Community management

4.3.1 Create community

Endpoint: `POST /api/community/create`

desc: Creates a community

Request body:

```
  {
    "name": "string",
    "description": "string",
    "img_path": "string",
    "tags": ["INT"]
  }
```

Response:

```
  {
    "community_id": "INT"
  }
```

4.3.2 Get community

Endpoint: `GET /api/community/details/{community_id}`
desc: Fetches details of community
Response:

```
  {
    "id": "INT",
    "name": "string",
    "description": "string",
    "img_path": "string",
    "member_count": "int",
    "tags": ["INT"],
    "post_ids": ["INT"]
  }
```

4.3.3 Update community (user)

Endpoint: `PUT /api/community/update/details/{community_id}`
desc: updates details and image of community (not membercount)
request body:

```
  {
    "description": "string",
    "img_path": "string"
  }
```

Response: `200 (ok)`

4.3.4 Update community (backend)

Endpoint: `PUT /api/community/update/backend/{community_id}`
desc: Updates member count of community
Request body:

```
  {
    "member_count": "int",
    "TAGS": ["INT"],
    "POST_IDs": ["INT"]
  }
```

Response: `200 (ok)`

4.3.5 Delete community

Endpoint: `DELETE /api/community/remove/{community_id}`
desc: deletes the community
Response: `200 (ok)`

4.4 Post management

4.4.1 Create post

Endpoint: `POST /api/post/create`
desc: Create a new post
Request body:

```
{
  "title": "string",
  "main_text": "string",
  "auth_id": "INT",
  "com_id": "INT",
  "post_id_ref": "INT", //optional
  "comment_flag": "boolean"
}
```

Response:

```
{
  "post_id": "INT"
}
```

4.4.2 Get post

Endpoint: `GET /api/post/view/{post_id}`
desc: fetch all details about post
Response:

```
{
  "id": "INT",
  "title": "string",
  "main_text": "string",
  "auth_id": "INT",
  "com_id": "INT",
  "timestamp": "timestamp",
  "likes": "INT",
  "dislikes": "INT",
  "post_id_ref": "INT",
  "comment_flag": "boolean",
  "comment_count": "INT",
  "comments": ["INT"]
}
```

4.4.3 Update post (user)

Endpoint: `PUT /api/post/update/user/{post_id}`
desc: Edits post
Request:

```
  {
    "title": "string",
    "main_text": "string",
  }
```

4.4.4 Update Post (backend)

Endpoint: `PUT /api/post/update/backend/{post_id}`
desc: Update backend post descriptors
Request body:

```
  {
    "COMMENT_CNT": "INT",
    "LIKES": "INT",
    "DISLIKES": "INT",
    "COMMENTS": "[string]"
  }
```

Response: `200 (ok)`

4.4.5 Delete post

Endpoint: `DELETE /api/post/remove/{post_id}`
Desc: Deletes post
Response: `200 (ok)`

4.5 Interactions

4.5.1 Like / Dislike

Endpoint `PUT /api/post/vote/{post_id}`
Desc: allows backend to modify like/dislike
Request body:

```
  {
    "likes": "int",
    "dislikes": "int"
  }
```

deletion of like and dislike will be handled via same endpoint

Response: `200 (ok)`

4.5.2 Comments

Endpoint `PUT /api/post/comments/{post_id}`
Desc: Adds ref to comment post to origin post
Request body:

```
{
  "Comments": "INT[]"
}
```

Response: `200 (ok)`

4.6 Search

Search flags and filters are handled backend

4.6.1 Search Posts

Endpoint: `GET /api/search/posts`
Desc: searches accross all posts
Query parameter: `?q={keywords}`
Response:

```
{
  "results": [
    {
      "post_id": "INT",
      "title": "string",
      "main_text": "string",
      "timestamp": "timestamp"
    }
  ]
}
```

4.6.2 Search Communities

Endpoint: `GET /api/search/communities`
Desc: searches accross all communities
Query parameter: `q?={keywords}`
Response:

```
{
  "results": [
    {
      "community_id": "INT",
      "name": "string",
      "description": "string",
      "IMG_PATH": "string"
    }
  ]
}
```

4.6.3 Search user

Endpoint: `GET /api/search/users`
Desc: Searches across all users
Query parameter: `q?={keywords}`
Response:

```
{
  "results": [
    {
      "user_id": "INT",
      "username": "string",
      "IMG_PATH": "string"
    }
  ]
}
```

5. Error handling

5.1 HTTP codes used

- 200 (ok) | generic response when no response body is required
- 204 (no content) | use when data entry is empty (so does exist, but only contains NULL)
- 400 (bad request) | use when request body is malformed
- 401 (unauthorised) | use when user is a) not found, or b) does not have privilege to perform action
- 404 (not found) | use when data entry not found
- 500 (internal server error) | use for all other exceptions

5.2 Security

- Tokens are generated backend, and stored temporarilæy till their expiration time (3600). Tokens are linked to user id's (stored locally in browser cookie) to validate session

5.3 Versioning

Versioning will be done via headers in suggested formmat

```
Description
Date of last edit @ time of last edit
project name @ current version [stable / not stable / not tested]

Responsible editor @ Responsible team
```

6. Closing remarks

Due to the extensive nature of the API, DB- and BE team should be cooperating on making it a reality