1. API specsheet

Time: 21/05/25 @ 13:11 - @Lynet_101
Author: @Lynet_101 (Sebastian Lindau-Skands)
Version: 1.7 - Final

2. Overview

DB: sqlite
API: C#
env: Bare metal
Dedicated com-port for API: 5000

3. Database Schema

3.1 Community

```
Column name    Type         Constraints
Description                           Note
-------------- ------------ ---------------------------------------------- ----------
---------------------------- --------
ID             INT          PRIMARY KEY                                    Unique
Identifier
NAME           TEXT         UNIQUE. NOT NULL. CHECK (LENGTH(NAME) \< 100)  Community
Name
Description    TEXT         CHECK (LENGTH(description) \< 500)             Community
description
IMG_PATH       TEXT                                                        URL to
community image                 UNUSED
MEMBER_COUNT   INT          DEFAULT 0. NOT NULL                            Number of
members
TAGS           STR\[CSV\]   NOT NULL                                       Content
tags                          UNUSED
POST_IDs       STR\[CSV\]                                                  Array of
post ID\'s (FK to posts.id)
```

3.2 User

```
Column Name     Type         Constraints
Description
Note
--------------- ------------ ------------------------------------------------- -----
-------------------------------------------------------------------------------------
--------------- --------
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 (argon2)
SALT            TEXT         NOT NULL
Hashing salt (argon2)
IMG_PATH        TEXT                                                          URL
TO PP
UNUSED
POST_IDs        STR\[CSV\]                                                    Array
of post IDs (FK to posts.id)
LIKE_IDs        STR\[CSV\]                                                    Array
of post IDs (FK to posts.id)
DISLIKE_IDs     STR\[CSV\]                                                    Array
of post IDs (FK to posts.id)
COMMENT_IDs     STR\[CSV\]                                                    Array
of post IDs (FK to posts.id)
UNUSED
COMMUNITY_IDs   STR\[CSV\]                                                    Array
of community IDs and names \[id. name. id. name.\] (FK to communities.id) (FK to
communities.name)   UNUSED
ADMIN           INT          NOT NULL. DEFAULT FALSE                          ADMIN
FLAG
TAGS            STR\[CSV\]                                                    Array
of tags for content recommendation
UNUSED
```

3.3 Posts

```
Column Name    Type         Constraints                       Description
Notes
-------------- ------------ --------------------------------- ------------------------
--------------------------------- --------
POST_ID        INT          UNIQUE. NOT NULL                  Unique Identifier
TITLE          TEXT         NOT NULL. CHECK (LENGHT \< 100)   Post title
MAIN_TEXT      TEXT         CHECK (LENGTH \< 10k)             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   INT          NOT NULL                          Indicates comment
instead of post
COMMENT_CNT    INT          NOT NULL. DEFAULT 0               Comment count
Comments       STR\[CSV\]                                     Array of post id\'s for
comments (FK. Post.ID)
IMG_PATH       TEXT                                           URL to post image
```

```
TAGS           STR\[CSV\]                                    Array of tags for
content recommendation                  UNUSED
```

3.4 Feedback

```
Column name   Type     Constraints                  Description
------------ -------- ---------------------------- ---------------------------------
---------
id            INT      PRIMARY KEY AUTO_INCREMENT   Unique identifier for each
feedback entry
worked        STRING   NOT NULL                     what worked
didnt         STRING   NOT NULL                     what didnt work
Other         STRING                                additional feedback
rating        INT      NOT NULL                     rating from 1 to 5
timestamp     INT      NOT NULL                     timestamp of feedback entry
```

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 (image_path can be undefined)
Request body:

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

Response:

```
{
  "user_id": "INT",
  "username: "string",
  "email": "string",
  "image_path": "string",
  "token": "string"
}
```

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. `img_path` , `post_ids` , `community_ids` , `display_name` and `description` are optional
Response:

```
{
  "id": "UUID",
  "email": "string",
  "username": "string",
  "img_path": "string",
  "post_ids": ["UUID"],
  "like_ids": ["UUID"],
  "dislike_ids": ["UUID"],
  "comment_ids": ["UUID"],
  "community_ids": ["UUID"],
  "tags": ["UUID"],
  "admin": "boolean"
}
```

4.2.2 Delete User Account

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

4.2.3 Update User Profile (user)

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

Note: **Not Implemented**
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:

```
{
  "post_ids": ["UUID"],
  "like_ids": ["UUID"],
  "dislike_ids": ["UUID"],
  "comment_ids": ["UUID"],
  "community_ids": ["UUID"],
  "tags": ["UUID"],
  "admin": "boolean"
}
```

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:

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

Response: `200 (ok)`

4.3.4 Update community (backend)

Endpoint: `PUT /api/community/update/backend`
desc: Updates member count, tags and post_ids of community

Query parameters:

```
member_count: "int" // required
TAGS: ["INT"]
postID: int
```

Response: `200 (ok)`

Notes:

- All query parameters ecepe from id are optional
- postID values will be appended to existing post IDs (if any).
- tags should be passed as a single string (e.g., "tech,science").

Examples for using it:

- update POST_IDs: `/api/community/update/backend?id=123&PostID=1234`

- update member count: `/api/community/update/backend?id=123&member_count=1234568`

4.3.5 Delete community

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

4.3.6 Get All Communities

Endpoint: `GET /api/community/all`
desc: fetches ID and name of all communities
response:

```
{
  "id": ["int"],
  "name": ["string"]
  "description": ["string"]
}
```

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",
  "timestamp": "timestamp",
  "likes": "INT",
  "dislikes": "INT",
  "post_id_ref": "INT",
  "comment_flag": "boolean",
  "comment_count": "INT",
  "comments": ["INT"],
  "author": {
    "auth_id": "INT",
    "username": "string",
    "imagePath": "string",
    "isAdmin": "INT",
  },
  "community": {
    "com_id": "INT",
    "name": "string",
  }
}
```

4.4.3 Update post (user)

Endpoint: `PUT /api/post/update/user/{post_id}`
desc: Edits post
Note: **Unused**
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
Note: **Unused**
Response: `200 (ok)`

4.4.6 Get Multiple Posts

Endpoint: `GET /api/posts`
Desc: Fetch multiple posts based on various filter parameters

Query parameters:

```
communityId: INT                // Filter by community
userId: INT                     // Filter by author
timestampStart: INT             // Filter by post time (start)
timestampEnd: INT               // Filter by post time (end)
limit: INT                       // Max number of posts to return (default: 20, max:
100)
offset: INT                      // Pagination offset (default: 0)
sortBy: string                  // Options: "timestamp", "likes", "comments"
(default: "timestamp")
sortOrder: string               // Options: "asc", "desc" (default: "desc")
tags: [INT]                      // Filter by specific tags (optional)
getComments: boolean            // If true, get comments (default: false)
parentPostId: INT              // If used, only get comments from this parent post.
Only does something if get_commets is true.
getPosts: boolean               // If true, get posts (default: true)
```

Response:

```
{
  "posts": [
    {
      "post_id": "INT",
      "title": "string",
      "main_text": "string",
      "auth_id": "INT",
      "com_id": "INT",
      "timestamp": "INT",
      "likes": "INT",
      "dislikes": "INT",
      "post_id_ref": "INT",
      "comment_flag": "boolean",
      "comment_count": "INT"
    }
  ],
  "total_count": "INT",
  "next_offset": "INT"
}
```

Notes:

- All query parameters are optional
- The response includes a `total_count` field indicating the total number of posts matching the filters
- The response includes a `next_offset` field for scrolling/pagination (null if no more results)

Examples for using it:

- Get top posts from community: `/api/posts?community_id=123&sort_by=likes&sort_order=desc`
- User profile posts: `/api/posts?user_id=45`
- Recent popular posts: `/api/posts?sort_by=likes&timestamp_start=1714503600`

4.4.7 Get Multible posts by id

Endpoint: `GET /api/post/postsids`
Desc: Fetch multiple posts based on various filter parameters

Query parameters:

```
ids:   string               // filter by a csv string of post_ids
```

Response:

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

Examples for using it:

- Get posts by the id 1,2,3 and 4: `/api/post/postsids?ids=1,2,3,4`

4.4.8 Upload Image

Endpoint `POST /api/upload/image
Request body:

```
  {
    "file": "FILE",
```

```
  }
```

Response:

```
{
  "imageUrl": "string"
}
```

4.5 Interactions

4.5 Interactions

4.5.1 Like / Dislike

Endpoint `PUT /api/post/vote/{post_id}`
Request body:

```
  {
    "user_id": "INT",
    "vote_type": "string"  // "like", "dislike", or "none" ("none" removes the vote
for the user)
  }
```

Response: `200 (ok)`

4.5.2 Get User Vote Status

Endpoint: `GET /api/post/{post_id}/vote/{user_id}`
Description: Check if a user has liked/disliked a post
Note: **Not implemented**
Response:

```
  {
    "vote_type": "string"  // "like", "dislike", or "none"
  }
```

4.5.3 Create Comment

Endpoint `POST /api/post/comments/{post_id}`
Desc: Create a comment on a post
Request body:

```
  {
    "user_id": "INT",
    "text": "string"
  }
```

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
Note: **Not implemented**
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
Note: **Not implemented**
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
Note: **Not implemented**
Query parameter: `q?={keywords}`
Response:

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

4.7 Feedback

4.7.1 Create feedback

Endpoint: `POST: /api/feedback/submit`
Desc: creates a feedback entrance
Request:

```
{
  "worked": "string",
  "didnt": "string",
  "feedback": "string?",
  "rating": "int",
  "timestamp": "int"
}
```

4.7.2 Retrieve feedback

Endpoint: `GET: /api/feedback/all`
Desc: Retrieve all feedback
Note: **Unused**
Response:

```
{
  "results": [
    {
      "worked": "string",
      "didnt": "string",
      "feedback": "string?",
      "rating": "int",
      "timestamp": "int"
    }
  ]
}
```

5. Error handling

5.1 HTTP codes used

- 200 (ok) | generic response when no response body is required
- 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