Judul Proyek: Sistem Manajemen Database Stugether (Kolaborasi Tugas Kelompok, Berbagi Catatan Dan Forum Diskusi + Reminder)

Mata Kuliah: Manajemen Basis Data Client Server
Dosen Pengampu: Donny Sanjaya, M.Kom.

LAPORAN KONTRIBUSI INDIVIDU

Nama Anggota: Muhammad Farid Yamin

NIM: 2305181063

Kelompok: 2

Program Studi: Teknologi Rekayasa Perangkat Lunak

Politeknik Negeri Medan

2025

BAB I: PENDAHULUAN

1.1 Deskripsi Proyek

a. Nama Sistem

Stugether - Student Together Collaboration Platform

b. Tujuan Sistem

Stugether dibangun sebagai ruang kolaborasi yang aman, rapi, dan terstruktur untuk mahasiswa, tempat berkumpulnya forum akademik, komunitas, proyek kelompok, hingga task management sederhana. Sistem ini memudahkan mahasiswa dalam:

- Bergabung ke forum melalui kode undangan
- Berdiskusi
- Berbagi materi
- Mengatur kolaborasi tim

c. Ruang Lingkup

- UI/UX Design (mockup design)
- Frontend (responsive layout design & API connection plan)
- Backend (API CI4, routing, controller, model, business logic)
- Database (desain tabel, relasi, migration, seeder)
- OpenAPI Documentation (rencana integrasi)

1.2 Peran dan Tanggung Jawab Individu

a. Posisi Dalam Tim

Database Designer dan Backend Developer

b. Deskripsi Tugas yang Diampu
- Merancang ERD utama untuk sistem
- Mendesain tabel users, forums, dan rencana tabel forum_members
- Membuat struktur tabel MySQL
- Menentukan tipe data, constraint, dan relasi
- Membuat migration CI4 untuk implementasi database
- Mendesain dan mengimplementasikan API pada CodeIgniter 4
- Membuat routing, controller, dan model
- Implementasi proses pembuatan forum & login/register
- Validasi input dan business logic
- Menentukan struktur dokumentasi API (OpenAPI)

c. Tools dan Teknologi
- Database: MySQL
- Framework Backend: CodeIgniter 4
- Bahasa: PHP
- Tools: Cursor, Git, phpMyAdmin
- Dokumentasi: Swagger (OpenAPI)

BAB II: DESAIN SISTEM

2.1 Gambaran Umum Sistem

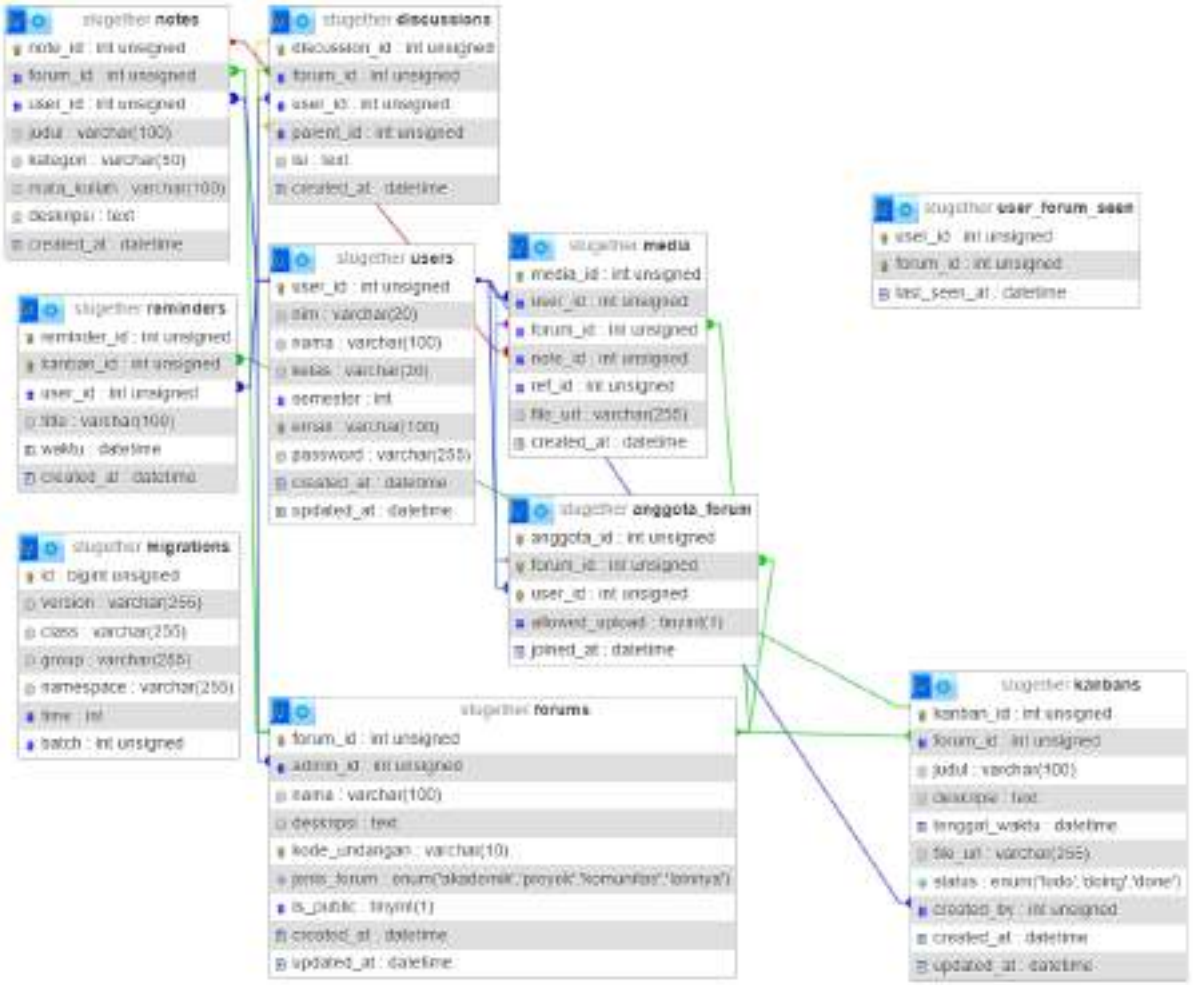
Stugether adalah sistem yang menyatukan kolaborasi mahasiswa melalui forum terstruktur. User dapat membuat forum, mengundang anggota melalui kode unik, serta mengelola informasi, tugas, dan diskusi.

Fitur Utama:

- Registrasi & login user
- Manajemen forum
- Bergabung melalui kode undangan
- Kategori forum: akademik, proyek, komunitas, lainnya
- Manajemen anggota

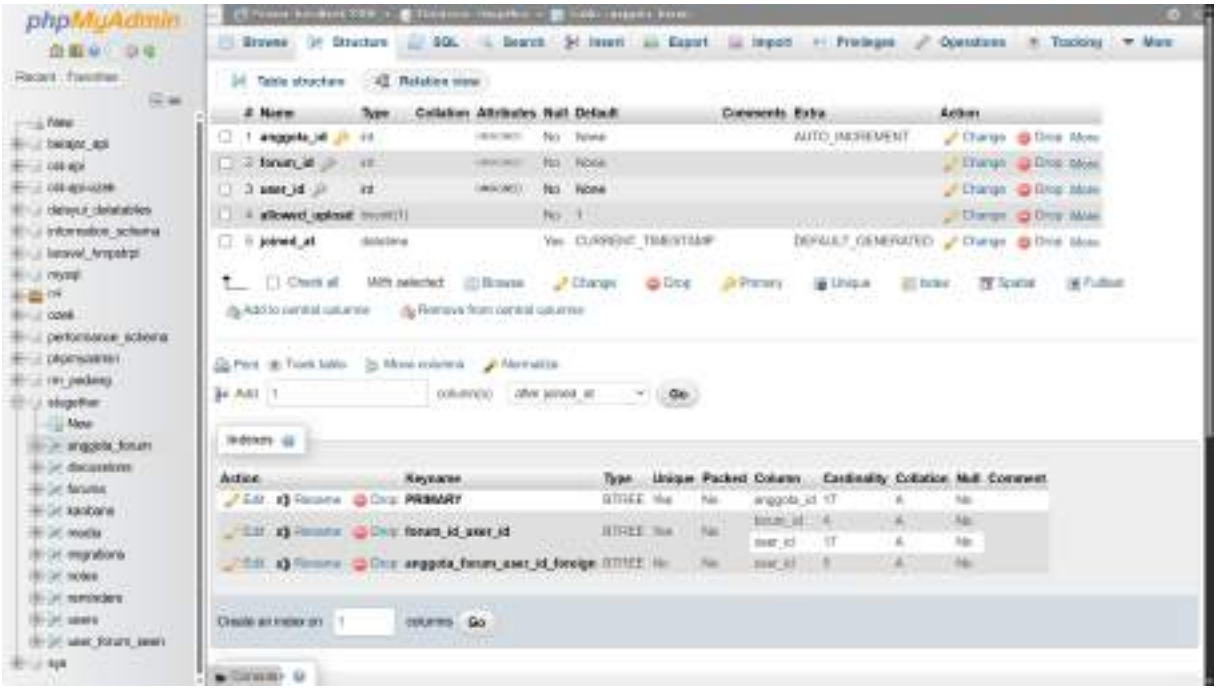
2.2 Desain yang Menjadi Tanggung Jawab

a. ERD (Entity Relationship Diagram)

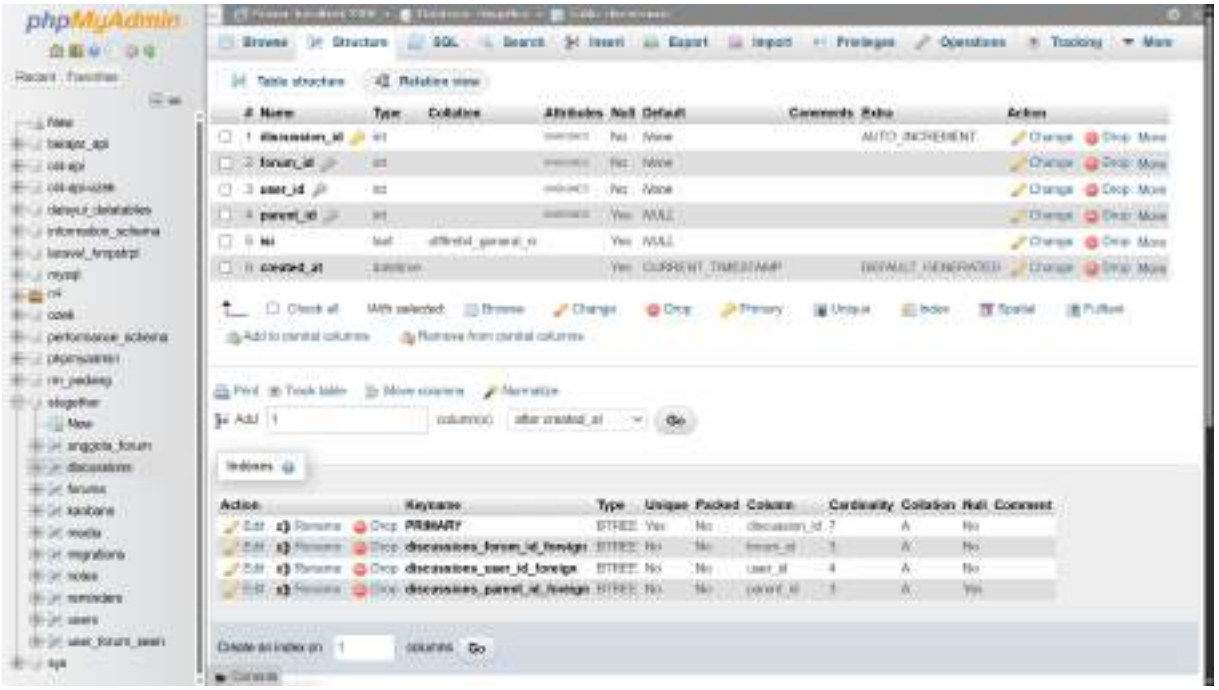


b. Struktur Tabel Database

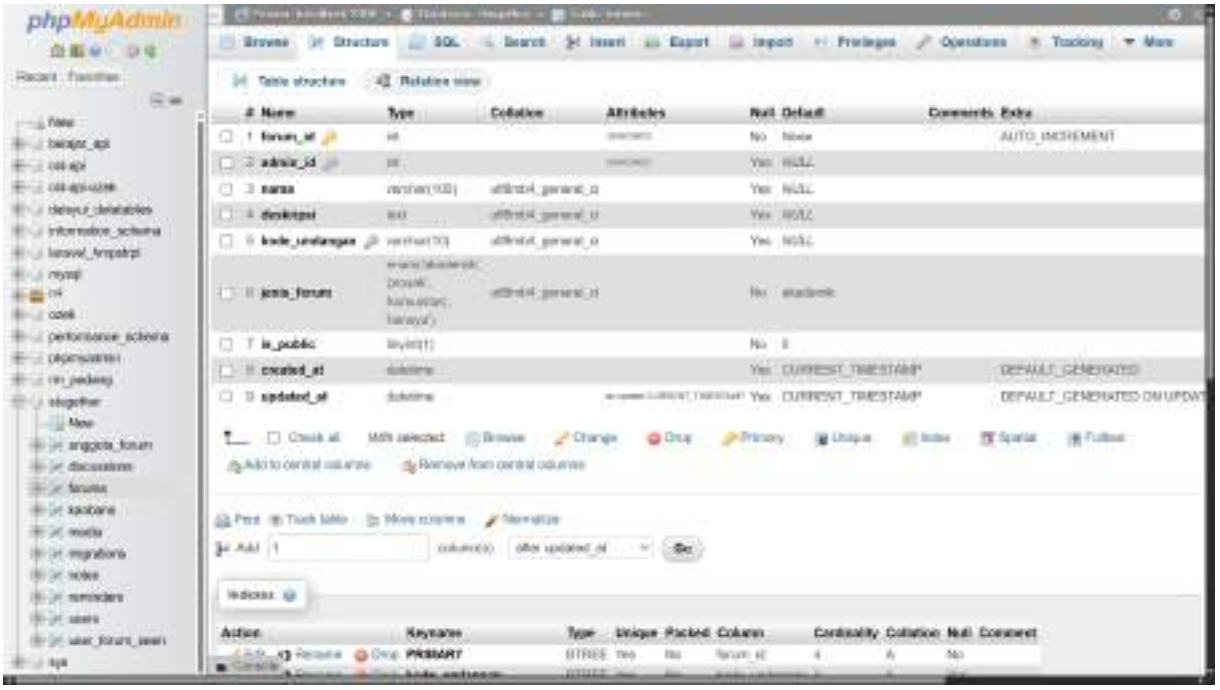
anggota_forum:



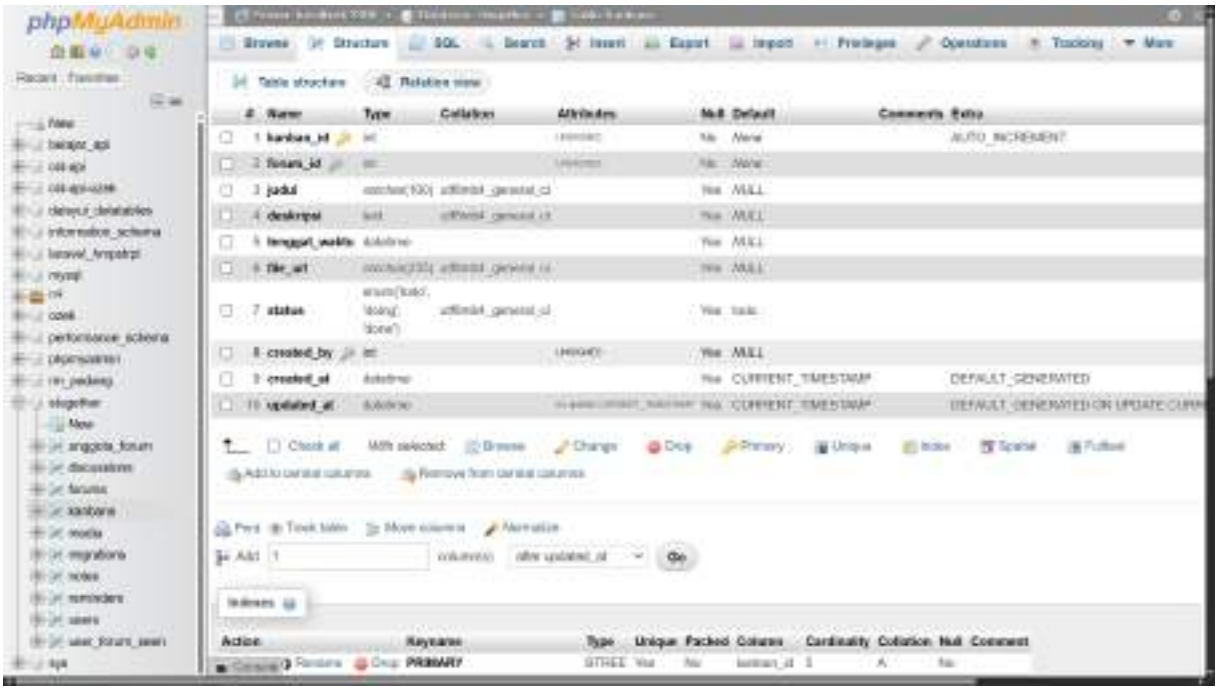
discussions:



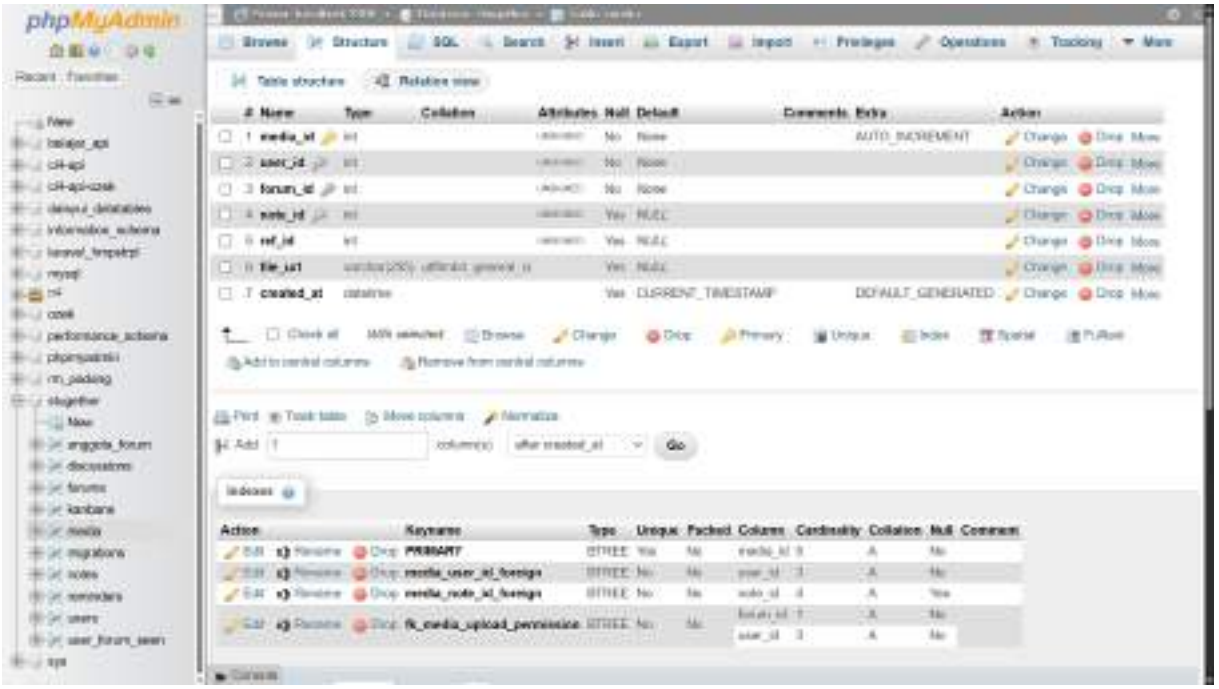
forums:



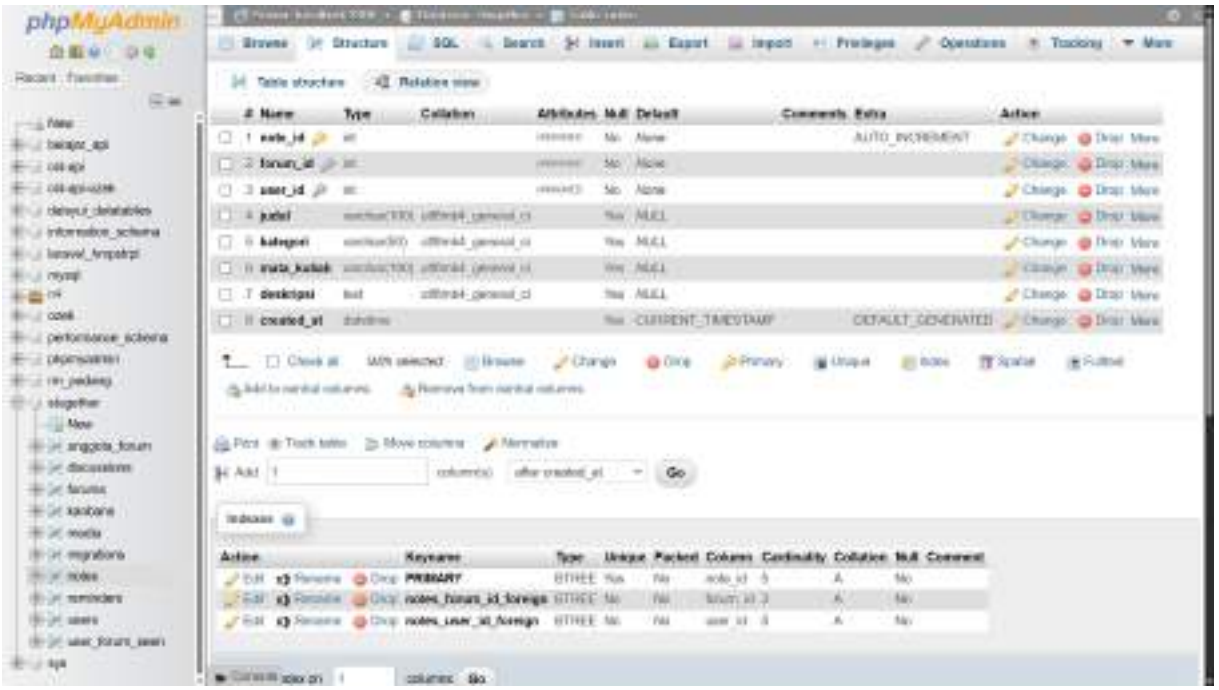
kanbans:



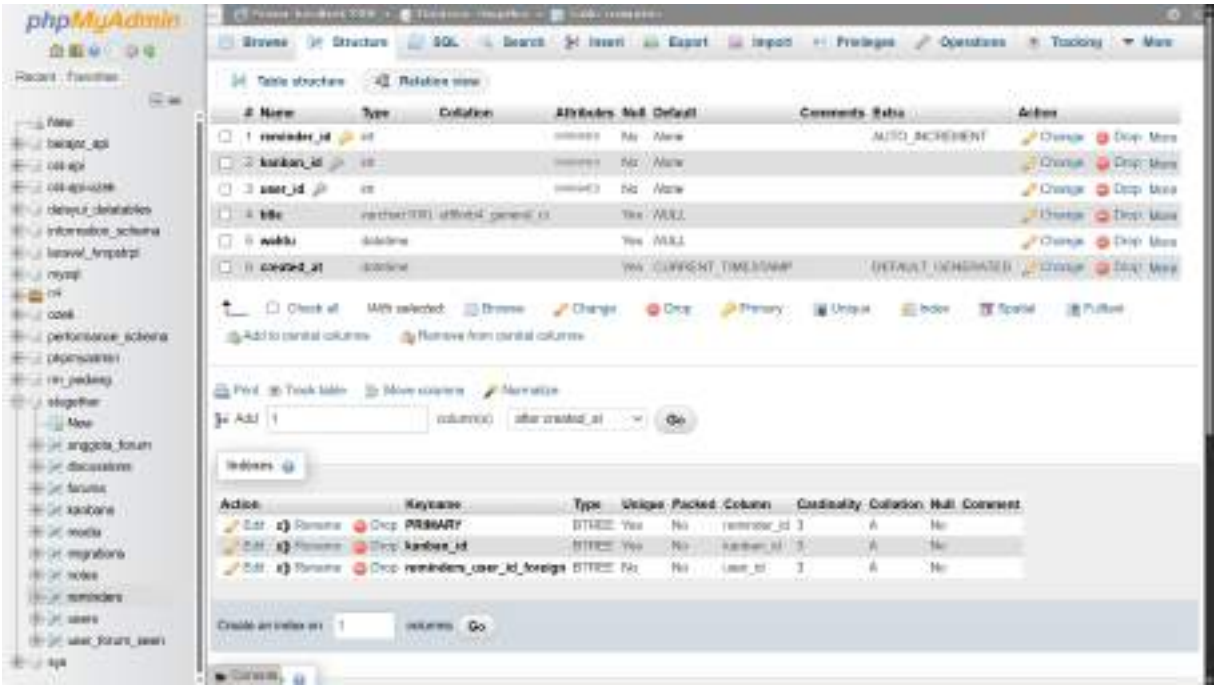
media:



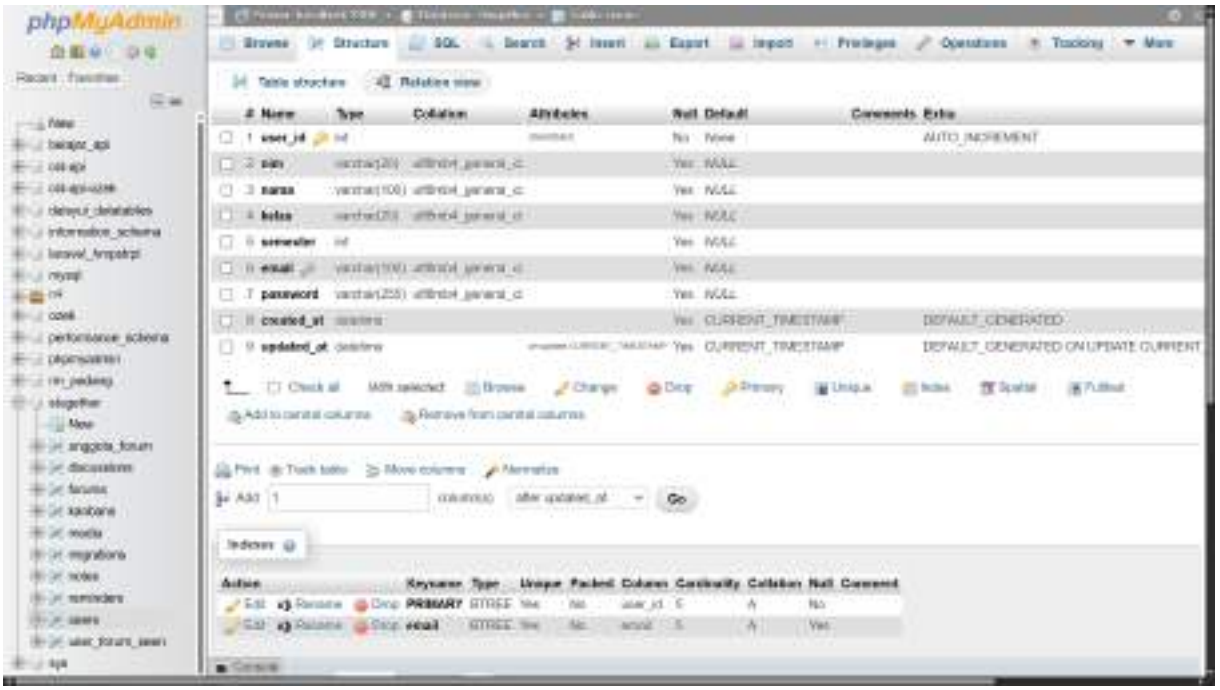
notes:



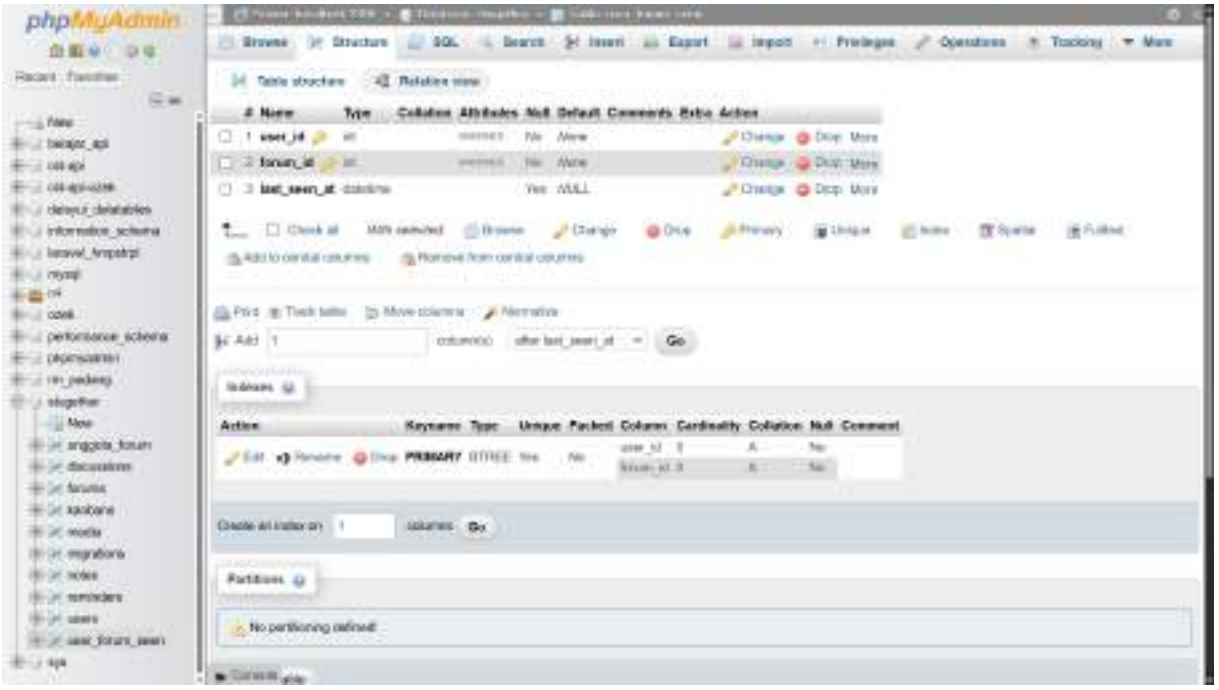
reminders:



users:



user_forum_seen:



c. Relasi Antar Tabel

Relasi	Tipe	Penjelasan Singkat
Users → Forums	1:M	User membuat banyak forum
Users ↔ Forums via Anggota Forum	M:M	User bisa join banyak forum
Forums → Kanbans	1:M	Forum punya banyak task
Kanbans → Reminders	1:1	Satu kanban satu reminder
Forums → Discussions	1:M	Forum punya banyak diskusi
Discussions → Discussions	Hierarki	Thread komentar
Forums → Notes	1:M	Forum punya banyak catatan
Media → (Users, Forums)	1:M	Media terhubung ke forum dan user
Media → Anggota Forum	Composite FK	Upload hanya untuk anggota

d. Arsitektur Sistem
- CI4 modular structure
- Controllers → Models → Database
- Routing sistem RESTful
- Rencana dokumentasi memakai Swagger

e. Struktur API Endpoint

Authentication:

Endpoint	Method	Deskripsi
`/auth/register`	POST	Registrasi user baru
`/auth/login`	POST	Login user dan mendapatkan JWT token
`/auth/logout`	POST	Logout user (stateless)
`/auth/me`	GET	Mendapatkan informasi user yang sedang login

Users:

Endpoint	Method	Deskripsi
`/users/{id}`	GET	Menampilkan detail user berdasarkan ID
`/users/{id}`	PUT	Update profil user (hanya user sendiri)

Forums:

Endpoint	Method	Deskripsi
`/forums`	POST	Membuat forum baru
`/forums`	GET	Mendapatkan daftar forum
`/forums/recommended`	GET	Mendapatkan daftar forum yang direkomendasikan
`/forums/{id}`	GET	Menampilkan detail forum berdasarkan ID
`/forums/{id}`	PATCH	Update forum (hanya admin forum)
`/forums/{id}`	DELETE	Hapus forum (hanya admin forum)

Forum Members:

Endpoint	Method	Deskripsi
`/forums/{id}/join`	POST	Bergabung ke dalam forum
`/forums/{id}/leave`	POST	Keluar dari forum (hanya anggota forum)
`/forums/{id}/members`	GET	Mendapatkan daftar anggota forum
`/forums/{id}/members/{member_id}`	PATCH	Update status anggota forum (hanya admin forum)

Tasks (Kanban):

Endpoint	Method	Deskripsi
`/forums/{id}/tasks`	POST	Membuat task baru di forum (hanya anggota forum)
`/forums/{id}/tasks`	GET	Mendapatkan daftar task di forum
`/tasks/{id}`	GET	Menampilkan detail task berdasarkan ID
`/tasks/{id}`	PATCH	Update task
`/tasks/{id}`	DELETE	Hapus task
`/tasks/{id}/attachments`	POST	Upload attachment ke task (hanya anggota forum)

Reminders:

Endpoint	Method	Deskripsi
`/tasks/{id}/reminder`	POST	Membuat reminder untuk task
`/reminders`	GET	Mendapatkan daftar reminder user
`/reminders/{id}`	DELETE	Hapus reminder

Discussions:

Endpoint	Method	Deskripsi
`/forums/{id}/discussions`	POST	Membuat diskusi baru di forum (hanya anggota forum)
`/discussions/{id}/replies`	POST	Membalas diskusi (hanya anggota forum)
`/forums/{id}/discussions`	GET	Mendapatkan daftar diskusi di forum
`/discussions/{id}`	GET	Menampilkan detail diskusi berdasarkan ID
`/discussions/{id}`	PATCH	Update diskusi
`/discussions/{id}`	DELETE	Hapus diskusi

Notes:

Endpoint	Method	Deskripsi
`/forums/{id}/notes`	POST	Membuat catatan baru di forum (hanya anggota forum)
`/forums/{id}/notes`	GET	Mendapatkan daftar catatan di forum
`/notes/{id}`	GET	Menampilkan detail catatan berdasarkan ID
`/notes/{id}`	PATCH	Update catatan
`/notes/{id}`	DELETE	Hapus catatan

Media:

Endpoint	Method	Deskripsi
`/media`	POST	Upload media/file
`/forums/{id}/media`	GET	Mendapatkan daftar media di forum
`/media/{id}`	GET	Menampilkan detail media berdasarkan ID
`/media/{id}`	DELETE	Hapus media

Search:

Endpoint	Method	Deskripsi
`/search`	GET	Pencarian global (forum, task, discussion, notes)

Notifications:

Endpoint	Method	Deskripsi
`/notifications`	GET	Mendapatkan daftar notifikasi user

Documentation:

Endpoint	Method	Deskripsi
`/docs`	GET	Dokumentasi API (OpenAPI)

f. Alur Logika Bisnis

- Validasi input
- Generate kode undangan otomatis
- Cek duplikasi email
- Relasi admin_id ke tabel forum

BAB III: IMPLEMENTASI

3.1 Daftar Pekerjaan yang Dikerjakan

No	Nama Task	Deskripsi	Status
1	Perancangan Database	Membuat struktur tabel & relasi	Selesai
2	Menyusun Migration CI4	Membuat file migration untuk users & forums	Selesai
3	Menyusun Seeder	Membuat data dummy awal	Selesai
4	Menentukan struktur API endpoint	Menyusun rute dasar untuk autentikasi & forum	Selesai
5	Menyiapkan Swagger	Menentukan integrasi dengan CI4	Selesai

3.2 Detail Implementasi

a. Database

```
-- ========================
-- 1. TABEL USERS
-- ========================
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    nim VARCHAR(20),
    nama VARCHAR(100),
    kelas VARCHAR(20),
    semester INT,
    email VARCHAR(100) UNIQUE,
    password VARCHAR(255),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- ========================
-- 2. TABEL FORUMS
-- ========================
CREATE TABLE forums (
    forum_id INT AUTO_INCREMENT PRIMARY KEY,
    admin_id INT,
    nama VARCHAR(100),
    deskripsi TEXT,
    kode_undangan VARCHAR(10) UNIQUE,
     jenis_forum ENUM('akademik', 'proyek', 'komunitas', 'lainnya') DEFAULT
'akademik',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            updated_at   DATETIME   DEFAULT   CURRENT_TIMESTAMP   ON   UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (admin_id) REFERENCES users(user_id)
);

-- ========================
```

```sql
-- 3. TABEL ANGGOTA_FORUM
-- ========================
CREATE TABLE anggota_forum (
    anggota_id INT AUTO_INCREMENT PRIMARY KEY,
    forum_id INT,
    user_id INT,
     allowed_upload BOOLEAN DEFAULT TRUE, -- hanya anggota & admin yg punya
izin upload
    joined_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (forum_id) REFERENCES forums(forum_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    UNIQUE (forum_id, user_id)
);

-- ========================
-- 4. TABEL KANBANS
-- ========================
CREATE TABLE kanbans (
    kanban_id INT AUTO_INCREMENT PRIMARY KEY,
    forum_id INT,
    judul VARCHAR(100),
    deskripsi TEXT,
    tenggat_waktu DATETIME,
    file_url VARCHAR(255),
    created_by INT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            updated_at   DATETIME   DEFAULT   CURRENT_TIMESTAMP   ON   UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (forum_id) REFERENCES forums(forum_id) ON DELETE CASCADE,
    FOREIGN KEY (created_by) REFERENCES users(user_id)
);

-- ========================
-- 5. TABEL REMINDERS
-- ========================
CREATE TABLE reminders (
    reminder_id INT AUTO_INCREMENT PRIMARY KEY,
    kanban_id INT UNIQUE,
    user_id INT,
    title VARCHAR(100),
    waktu DATETIME,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (kanban_id) REFERENCES kanbans(kanban_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- ========================
-- 6. TABEL DISCUSSIONS
-- ========================
CREATE TABLE discussions (
    discussion_id INT AUTO_INCREMENT PRIMARY KEY,
    forum_id INT,
    user_id INT,
    parent_id INT DEFAULT NULL,
    isi TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (forum_id) REFERENCES forums(forum_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
     FOREIGN KEY (parent_id) REFERENCES discussions(discussion_id) ON DELETE
SET NULL
);

-- ========================
-- 7. TABEL NOTES
```

```
-- ========================
CREATE TABLE notes (
    note_id INT AUTO_INCREMENT PRIMARY KEY,
    forum_id INT,
    user_id INT,
    judul VARCHAR(100),
    kategori VARCHAR(50),
    mata_kuliah VARCHAR(100),
    deskripsi TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (forum_id) REFERENCES forums(forum_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- ========================
-- 8. TABEL MEDIA
-- ========================
CREATE TABLE media (
    media_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    forum_id INT,
    note_id INT DEFAULT NULL,
    ref_id INT DEFAULT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (forum_id) REFERENCES forums(forum_id) ON DELETE CASCADE,
    FOREIGN KEY (note_id) REFERENCES notes(note_id) ON DELETE SET NULL,
    CONSTRAINT fk_media_upload_permission
        FOREIGN KEY (user_id, forum_id)
        REFERENCES anggota_forum(user_id, forum_id)
        ON DELETE CASCADE
);
```

b. Backend/API

AuthController.php

```
<?php

namespace App\Controllers\API;

use App\Models\UserModel;
use App\Entities\User;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class AuthController extends BaseAPIController
{
  #[OAT\Post(
    path: "/auth/register",
    tags: ["Auth"],
    summary: "Register user",
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["nama", "email", "password"],
        properties: [
          new OAT\Property(property: "nama", type: "string"),
                new OAT\Property(property: "email", type: "string", format:
"email"),
              new OAT\Property(property: "password", type: "string", format:
"password")
        ]
      )
    ),
```

```
    responses: [
      new OAT\Response(response: 201, description: "Registered"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
  public function register()
  {
    $rules = config('Validation')->authRegister;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }

    $data = $this->request->getJSON(true) ?? $this->request->getPost();
    $model = new UserModel();

    $userData = [
      'nama'     => $data['nama'] ?? null,
      'email'    => $data['email'],
      'password' => password_hash($data['password'], PASSWORD_BCRYPT),
    ];
    $userId = $model->insert($userData, true);
    $user   = $model->find($userId);

    $token = service('jwt')->issueToken($user);

    return $this->success(['token' => $token, 'user' => $user], 'Registered',
null, 201);
  }

  #[OAT\Post(
    path: "/auth/login",
    tags: ["Auth"],
    summary: "Login",
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["email", "password"],
        properties: [
                new OAT\Property(property: "email", type: "string", format:
"email"),
              new OAT\Property(property: "password", type: "string", format:
"password")
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 200, description: "Logged in"),
      new OAT\Response(response: 400, description: "Bad Request"),
      new OAT\Response(response: 401, description: "Unauthorized")
    ]
  )]
  public function login()
  {
    $rules = config('Validation')->authLogin;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }

    $data  = $this->request->getJSON(true) ?? $this->request->getPost();
    $email = $data['email'];
    $pass  = $data['password'];

    $model = new UserModel();
    $user  = $model->where('email', $email)->first();
```

```
    if (! $user || ! password_verify($pass, (string) $user->password)) {
      return $this->fail('Invalid credentials', 401);
    }

    $token = service('jwt')->issueToken($user);
    return $this->success(['token' => $token, 'user' => $user], 'Logged in');
  }

  #[OAT\Post(
    path: "/auth/logout",
    tags: ["Auth"],
    summary: "Logout (stateless)",
    security: [["bearerAuth" => []]],
    responses: [new OAT\Response(response: 200, description: "Logged out")]
  )]
  public function logout()
  {
    return $this->success(['ok' => true], 'Logged out');
  }

  #[OAT\Get(
    path: "/auth/me",
    tags: ["Auth"],
    summary: "Current user",
    security: [["bearerAuth" => []]],
    responses: [
      new OAT\Response(response: 200, description: "User"),
      new OAT\Response(response: 401, description: "Unauthorized")
    ]
  )]
  public function me()
  {
    return $this->success($this->currentUser());
  }
}
```

BaseAPIController.php

```
<?php

namespace App\Controllers\API;

use CodeIgniter\Controller;
use CodeIgniter\HTTP\ResponseInterface;
use App\Entities\User;

abstract class BaseAPIController extends Controller
{
   protected function success($data, ?string $message = null, ?array $meta =
null, int $status = 200)
  {
    $payload = ['data' => $data];
    if ($meta !== null) {
      $payload['meta'] = $meta;
    }
    if ($message !== null) {
      $payload['message'] = $message;
    }
    return $this->response->setStatusCode($status)->setJSON($payload);
  }

  protected function fail(string $message, int $code = 400)
  {
    return $this->response->setStatusCode($code)->setJSON([
      'error' => [
```

```
        'code'    => $code,
        'message' => $message,
      ],
    ]);
  }

  /**
   * Get the authenticated user from the request lifecycle.
   */
  protected function currentUser(): ?User
  {
    return service('authUser')->getUser();
  }
}
```

DiscussionController.php

```
<?php

namespace App\Controllers\API;

use App\Models\DiscussionModel;
use App\Models\ForumModel;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class DiscussionController extends BaseAPIController
{
  #[OAT\Post(
    path: "/forums/{id}/discussions",
    tags: ["Discussions"],
    summary: "Create discussion",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: true,
          content: new OAT\JsonContent(required: ["isi"], properties: [new
OAT\Property(property: "isi", type: "string")])
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
  public function store(int $forumId)
  {
    $rules = config('Validation')->discussionStore;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $data    = $this->request->getJSON(true) ?? $this->request->getPost();
    $current = $this->currentUser();
    $model   = new DiscussionModel();
    $id = $model->insert([
      'forum_id'  => $forumId,
      'user_id'   => $current->user_id,
      'parent_id' => null,
      'isi'       => $data['isi'],
    ], true);
    return $this->success($model->find($id), 'Created', null, 201);
  }

  #[OAT\Post(
    path: "/discussions/{id}/replies",
```

```
    tags: ["Discussions"],
    summary: "Reply to discussion",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: true,
          content: new OAT\JsonContent(required: ["isi"], properties: [new
OAT\Property(property: "isi", type: "string")])
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
  public function reply(int $discussionId)
  {
    $rules = config('Validation')->discussionReply;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $parent = (new DiscussionModel())->find($discussionId);
    if (! $parent) {
      return $this->fail('Discussion not found', 404);
    }
    $data    = $this->request->getJSON(true) ?? $this->request->getPost();
    $current = $this->currentUser();
    $model   = new DiscussionModel();
    $id = $model->insert([
      'forum_id'  => $parent->forum_id,
      'user_id'   => $current->user_id,
      'parent_id' => $discussionId,
      'isi'       => $data['isi'],
    ], true);
    return $this->success($model->find($id), 'Created', null, 201);
  }

  #[OAT\Get(
    path: "/forums/{id}/discussions",
    tags: ["Discussions"],
    summary: "List discussions (threaded by default)",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer")),
         new OAT\Parameter(name: "threaded", in: "query", required: false,
schema: new OAT\Schema(type: "boolean")),
       new OAT\Parameter(name: "q", in: "query", required: false, schema: new
OAT\Schema(type: "string")),
       new OAT\Parameter(name: "page", in: "query", required: false, schema:
new OAT\Schema(type: "integer")),
         new OAT\Parameter(name: "per_page", in: "query", required: false,
schema: new OAT\Schema(type: "integer"))
    ],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index(int $forumId)
  {
      $threaded = filter_var($this->request->getGet('threaded') ?? 'true',
FILTER_VALIDATE_BOOLEAN);
    $q        = trim((string) ($this->request->getGet('q') ?? ''));
    $model    = new DiscussionModel();
                  $builder         =     $model->builder()->where('forum_id',
$forumId)->orderBy('created_at', 'DESC');
```

```
    if ($q !== '') {
      $builder->like('isi', $q);
    }
    if ($threaded) {
      $rows = $builder->get()->getResultArray();
      $data = service('discussionTree')->buildTree($rows);
      return $this->success($data);
    }
    $page    = max(1, (int) ($this->request->getGet('page') ?? 1));
     $perPage = min(100, max(1, (int) ($this->request->getGet('per_page') ??
10)));
    $total   = (clone $builder)->countAllResults(false);
    $rows    = $builder->get(($page - 1) * $perPage, $perPage)->getResult();
    $meta    = service('paginationSvc')->buildMeta($page, $perPage, $total);
    return $this->success($rows, null, $meta);
  }

  #[OAT\Get(
    path: "/discussions/{id}",
    tags: ["Discussions"],
    summary: "Show discussion",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "OK"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function show(int $discussionId)
  {
    $disc = (new DiscussionModel())->find($discussionId);
    if (! $disc) {
      return $this->fail('Not found', 404);
    }
    return $this->success($disc);
  }

  #[OAT\Patch(
    path: "/discussions/{id}",
    tags: ["Discussions"],
    summary: "Update discussion",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: false,
        content: new OAT\JsonContent(properties: [new OAT\Property(property:
"isi", type: "string")])
    ),
    responses: [
      new OAT\Response(response: 200, description: "Updated"),
      new OAT\Response(response: 403, description: "Forbidden")
    ]
  )]
  public function update(int $discussionId)
  {
    $model = new DiscussionModel();
    $disc  = $model->find($discussionId);
    if (! $disc) {
      return $this->fail('Not found', 404);
    }
    if (! $this->canManage($disc->forum_id, $disc->user_id)) {
      return $this->fail('Forbidden', 403);
```

```
    }
    $data = $this->request->getJSON(true) ?? $this->request->getRawInput();
    $model->update($discussionId, ['isi' => $data['isi'] ?? $disc->isi]);
    return $this->success($model->find($discussionId), 'Updated');
  }

  #[OAT\Delete(
    path: "/discussions/{id}",
    tags: ["Discussions"],
    summary: "Delete discussion",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "Deleted"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function destroy(int $discussionId)
  {
    $model = new DiscussionModel();
    $disc  = $model->find($discussionId);
    if (! $disc) {
      return $this->fail('Not found', 404);
    }
    if (! $this->canManage($disc->forum_id, $disc->user_id)) {
      return $this->fail('Forbidden', 403);
    }
    $model->delete($discussionId);
    return $this->success(['ok' => true], 'Deleted');
  }

  private function canManage(int $forumId, int $ownerId): bool
  {
    $current = $this->currentUser();
    if (! $current) {
      return false;
    }
    if ($ownerId === (int) $current->user_id) {
      return true;
    }
    $forum = (new ForumModel())->find($forumId);
    return $forum && (int) $forum->admin_id === (int) $current->user_id;
  }
}
```

ForumController.php

```
<?php

namespace App\Controllers\API;

use App\Models\ForumModel;
use App\Models\AnggotaForumModel;
use App\Models\KanbanModel;
use CodeIgniter\Database\BaseBuilder;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class ForumController extends BaseAPIController
{
  #[OAT\Post(
    path: "/forums",
    tags: ["Forums"],
    summary: "Create forum",
```

```
    security: [["bearerAuth" => []]],
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["nama"],
        properties: [
          new OAT\Property(property: "nama", type: "string"),
          new OAT\Property(property: "deskripsi", type: "string"),
             new OAT\Property(property: "jenis_forum", type: "string", enum:
["akademik","proyek","komunitas","lainnya"]),
              new OAT\Property(property: "is_public", type: "integer", enum:
[0,1])
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request"),
      new OAT\Response(response: 401, description: "Unauthorized")
    ]
  )]
  public function store()
  {
    $rules = config('Validation')->forumStore;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $data    = $this->request->getJSON(true) ?? $this->request->getPost();
    $current = $this->currentUser();
    $model   = new ForumModel();

    $kode = $this->generateUniqueKode();
    $forumId = $model->insert([
      'admin_id'     => $current->user_id,
      'nama'         => $data['nama'],
      'deskripsi'    => $data['deskripsi'] ?? null,
      'jenis_forum'  => $data['jenis_forum'] ?? 'akademik',
      'is_public'    => (int) ($data['is_public'] ?? 0),
      'kode_undangan'=> $kode,
    ], true);
    $forum = $model->find($forumId);

    // auto-join admin
    (new AnggotaForumModel())->insert([
      'forum_id'       => $forumId,
      'user_id'        => $current->user_id,
      'allowed_upload' => 1,
    ]);

    return $this->success($forum, 'Created', null, 201);
  }

  #[OAT\Get(
    path: "/forums",
    tags: ["Forums"],
    summary: "List forums",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "scope", in: "query", required: false, schema:
new OAT\Schema(type: "string", enum: ["mine","public","all"])),
       new OAT\Parameter(name: "q", in: "query", required: false, schema: new
OAT\Schema(type: "string")),
       new OAT\Parameter(name: "sort", in: "query", required: false, schema:
new OAT\Schema(type: "string", enum: ["created_at","nama"])),
```

```
        new OAT\Parameter(name: "page", in: "query", required: false, schema:
new OAT\Schema(type: "integer")),
        new OAT\Parameter(name: "per_page", in: "query", required: false,
schema: new OAT\Schema(type: "integer"))
    ],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index()
  {
    $current    = $this->currentUser();
    $scope      = $this->request->getGet('scope') ?? 'all';
    $q          = trim((string) ($this->request->getGet('q') ?? ''));
    $sort       = $this->request->getGet('sort') ?? 'created_at';
    $page       = max(1, (int) ($this->request->getGet('page') ?? 1));
     $perPage     = min(100, max(1, (int) ($this->request->getGet('per_page')
?? 10)));

    $builder = (new ForumModel())->builder();
    if ($scope === 'mine') {
        $builder->join('anggota_forum af', 'af.forum_id = forums.forum_id',
'inner')
                ->where('af.user_id', $current->user_id);
    } elseif ($scope === 'public') {
      $builder->where('is_public', 1);
    }
    if ($q !== '') {
      $builder->groupStart()
        ->like('nama', $q)
        ->orLike('deskripsi', $q)
      ->groupEnd();
    }
    $allowedSort = ['created_at', 'nama'];
    if (! in_array($sort, $allowedSort, true)) {
      $sort = 'created_at';
    }
    $builder->orderBy($sort, 'DESC');

    // Pagination
    $total   = (clone $builder)->countAllResults(false);
    $results = $builder->get(($page - 1) * $perPage, $perPage)->getResult();

    $meta = service('paginationSvc')->buildMeta($page, $perPage, $total);
    return $this->success($results, null, $meta);
  }

  #[OAT\Get(
    path: "/forums/recommended",
    tags: ["Forums"],
    summary: "Recommended public forums",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "limit", in: "query", required: false, schema:
new OAT\Schema(type: "integer"))
    ],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function recommended()
  {
      $limit     = min(50, max(1, (int) ($this->request->getGet('limit') ??
10)));
            $builder   =   (new   ForumModel())->builder()->where('is_public',
1)->orderBy('created_at', 'DESC');
    $data    = $builder->get($limit)->getResult();
    return $this->success($data);
```

```
  }

  #[OAT\Get(
    path: "/forums/{id}",
    tags: ["Forums"],
    summary: "Show forum",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer"))
    ],
    responses: [
      new OAT\Response(response: 200, description: "OK"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function show(int $forumId)
  {
    $forum = (new ForumModel())->find($forumId);
    if (! $forum) {
      return $this->fail('Forum not found', 404);
    }
           $membersCount   =   (new   AnggotaForumModel())->where('forum_id',
$forumId)->countAllResults();
            $tasksCount         =   (new   KanbanModel())->where('forum_id',
$forumId)->countAllResults();

    return $this->success([
      'forum'         => $forum,
      'counts'        => [
        'members' => $membersCount,
        'tasks'   => $tasksCount,
      ],
    ]);
  }

  #[OAT\Patch(
    path: "/forums/{id}",
    tags: ["Forums"],
    summary: "Update forum (admin)",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer"))
    ],
    requestBody: new OAT\RequestBody(
      required: false,
      content: new OAT\JsonContent(
        properties: [
          new OAT\Property(property: "nama", type: "string"),
          new OAT\Property(property: "deskripsi", type: "string"),
          new OAT\Property(property: "jenis_forum", type: "string"),
              new  OAT\Property(property: "is_public", type: "integer", enum:
[0,1]),
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 200, description: "Updated"),
      new OAT\Response(response: 400, description: "Bad Request"),
      new OAT\Response(response: 403, description: "Forbidden")
    ]
  )]
  public function update(int $forumId)
```

```
  {
    $rules = config('Validation')->forumUpdate;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $data  = $this->request->getJSON(true) ?? $this->request->getRawInput();
       $patch  =  array_intersect_key($data,  array_flip(['nama',  'deskripsi',
'jenis_forum', 'is_public']));
    $model = new ForumModel();
    $model->update($forumId, $patch);
    $forum = $model->find($forumId);
    return $this->success($forum, 'Updated');
  }

  #[OAT\Delete(
    path: "/forums/{id}",
    tags: ["Forums"],
    summary: "Delete forum (admin)",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer"))
    ],
    responses: [new OAT\Response(response: 200, description: "Deleted")]
  )]
  public function destroy(int $forumId)
  {
    $model = new ForumModel();
    $model->delete($forumId);
    return $this->success(['ok' => true], 'Deleted');
  }

  private function generateUniqueKode(): string
  {
    $model = new ForumModel();
    for ($i = 0; $i < 5; $i++) {
      $len  = random_int(6, 8);
         $code  =  substr(str_shuffle('ABCDEFGHJKLMNPQRSTUVWXYZ23456789'),  0,
$len);
      if (! $model->where('kode_undangan', $code)->first()) {
        return $code;
      }
    }
    return bin2hex(random_bytes(4));
  }
}
```

ForumMemberController.php

```
<?php

namespace App\Controllers\API;

use App\Models\ForumModel;
use App\Models\AnggotaForumModel;
use App\Models\UserModel;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class ForumMemberController extends BaseAPIController
{
  #[OAT\Post(
```

```
    path: "/forums/{id}/join",
    tags: ["Forums"],
    summary: "Join forum by kode_undangan",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer"))
    ],
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["kode_undangan"],
             properties: [new OAT\Property(property: "kode_undangan", type:
"string")]
      )
    ),
    responses: [
      new OAT\Response(response: 200, description: "Joined"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
  public function join(int $forumId)
  {
    $rules = config('Validation')->forumJoin;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $data  = $this->request->getJSON(true) ?? $this->request->getPost();
    $kode  = $data['kode_undangan'];
    $forum = (new ForumModel())->find($forumId);
    if (! $forum) {
      return $this->fail('Forum not found', 404);
    }
    if ($forum->kode_undangan !== $kode) {
      return $this->fail('Invalid invitation code', 400);
    }
    $user   = $this->currentUser();
    $model  = new AnggotaForumModel();
         $exists  =  $model->where(['forum_id'  =>  $forumId,  'user_id'  =>
$user->user_id])->first();
    if (! $exists) {
      $model->insert([
        'forum_id'       => $forumId,
        'user_id'        => $user->user_id,
        'allowed_upload' => 0,
      ]);
    }
    return $this->success(['ok' => true], 'Joined');
  }

  #[OAT\Post(
    path: "/forums/{id}/leave",
    tags: ["Forums"],
    summary: "Leave forum",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "Left"),
      new OAT\Response(response: 403, description: "Forbidden")
    ]
  )]
  public function leave(int $forumId)
  {
```

```
    $current = $this->currentUser();
    $forum   = (new ForumModel())->find($forumId);
    if (! $forum) {
      return $this->fail('Forum not found', 404);
    }
    if ((int) $forum->admin_id === (int) $current->user_id) {
      return $this->fail('Admin cannot leave the forum', 403);
    }
    $model = new AnggotaForumModel();
                $model->where(['forum_id'    =>    $forumId,    'user_id'    =>
$current->user_id])->delete();
    return $this->success(['ok' => true], 'Left forum');
  }

  #[OAT\Get(
    path: "/forums/{id}/members",
    tags: ["Forums"],
    summary: "List forum members",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function members(int $forumId)
  {
    $builder = (new AnggotaForumModel())->builder()
      ->select('u.user_id, u.nama, u.email, af.allowed_upload, af.joined_at')
      ->from('anggota_forum af')
      ->join('users u', 'u.user_id = af.user_id', 'inner')
      ->where('af.forum_id', $forumId)
      ->orderBy('u.nama', 'ASC');
    $rows = $builder->get()->getResultArray();
    return $this->success($rows);
  }

  #[OAT\Patch(
    path: "/forums/{id}/members/{userId}",
    tags: ["Forums"],
    summary: "Update member allowed_upload (admin)",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer")),
       new OAT\Parameter(name: "userId", in: "path", required: true, schema:
new OAT\Schema(type: "integer")),
    ],
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["allowed_upload"],
            properties: [new OAT\Property(property: "allowed_upload", type:
"integer", enum: [0,1])]
      )
    ),
    responses: [new OAT\Response(response: 200, description: "Updated")]
  )]
  public function update(int $forumId, int $userId)
  {
    $rules = config('Validation')->memberUpdate;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $data = $this->request->getJSON(true) ?? $this->request->getRawInput();
```

```
      (new AnggotaForumModel())->where(['forum_id' => $forumId, 'user_id' =>
$userId])
      ->set(['allowed_upload' => (int) $data['allowed_upload']])
      ->update();

    return $this->success(['ok' => true], 'Updated');
  }
}
```

MediaController.php

```
<?php

namespace App\Controllers\API;

use App\Models\MediaModel;
use App\Models\ForumModel;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class MediaController extends BaseAPIController
{
  #[OAT\Post(
    path: "/media",
    tags: ["Media"],
    summary: "Upload media",
    security: [["bearerAuth" => []]],
    requestBody: new OAT\RequestBody(
      required: true,
      content: [
        new OAT\MediaType(
          mediaType: "multipart/form-data",
          schema: new OAT\Schema(
            type: "object",
            required: ["forum_id"],
            properties: [
              new OAT\Property(property: "forum_id", type: "integer"),
              new OAT\Property(property: "note_id", type: "integer"),
              new OAT\Property(property: "ref_id", type: "integer"),
                  new OAT\Property(property: "file", type: "string", format:
"binary")
            ]
          )
        ),
        new OAT\MediaType(
          mediaType: "application/json",
          schema: new OAT\Schema(
            type: "object",
            required: ["forum_id","file_url"],
            properties: [
              new OAT\Property(property: "forum_id", type: "integer"),
               new OAT\Property(property: "file_url", type: "string", format:
"uri"),
              new OAT\Property(property: "note_id", type: "integer"),
              new OAT\Property(property: "ref_id", type: "integer")
            ]
          )
        )
      ]
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
```

```
  public function store()
  {
    // Validate via PHP side to allow optional note/ref
            $forumId   =   (int)   ($this->request->getPost('forum_id')   ??
$this->request->getJSON(true)['forum_id'] ?? 0);
    if ($forumId <= 0) {
      return $this->fail('forum_id is required', 400);
    }
    $current = $this->currentUser();
    $file    = $this->request->getFile('file');
    $fileUrl = null;
    if ($file && $file->isValid()) {
      $fileUrl = $this->moveUploadedFile($file, $forumId);
    } else {
      $body    = $this->request->getJSON(true) ?? $this->request->getPost();
      $fileUrl = $body['file_url'] ?? null;
      if (! $fileUrl) {
        return $this->fail('No file or file_url provided', 400);
      }
    }

             $noteId   =   (int)   ($this->request->getPost('note_id')   ??
$this->request->getJSON(true)['note_id'] ?? 0);
            $refId      =   (int)   ($this->request->getPost('ref_id')   ??
$this->request->getJSON(true)['ref_id'] ?? 0);

    $id = (new MediaModel())->insert([
      'user_id'  => $current->user_id,
      'forum_id' => $forumId,
      'note_id'  => $noteId ?: null,
      'ref_id'   => $refId ?: null,
      'file_url' => $fileUrl,
    ], true);

      return $this->success((new MediaModel())->find($id), 'Created', null,
201);
  }

  #[OAT\Get(
    path: "/forums/{id}/media",
    tags: ["Media"],
    summary: "List forum media",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer")),
         new OAT\Parameter(name: "note_id", in: "query", required: false,
schema: new OAT\Schema(type: "integer")),
      new OAT\Parameter(name: "ref_id", in: "query", required: false, schema:
new OAT\Schema(type: "integer"))
    ],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index(int $forumId)
  {
    $noteId = $this->request->getGet('note_id');
    $refId  = $this->request->getGet('ref_id');
           $builder   =   (new   MediaModel())->builder()->where('forum_id',
$forumId)->orderBy('created_at', 'DESC');
    if ($noteId) {
      $builder->where('note_id', (int) $noteId);
    }
    if ($refId) {
      $builder->where('ref_id', (int) $refId);
```

```
    }
    $data = $builder->get()->getResult();
    return $this->success($data);
  }

  #[OAT\Get(
    path: "/media/{id}",
    tags: ["Media"],
    summary: "Show media",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "OK"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function show(int $mediaId)
  {
    $media = (new MediaModel())->find($mediaId);
    if (! $media) {
      return $this->fail('Not found', 404);
    }
    return $this->success($media);
  }

  #[OAT\Delete(
    path: "/media/{id}",
    tags: ["Media"],
    summary: "Delete media",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "Deleted"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function destroy(int $mediaId)
  {
    $model = new MediaModel();
    $media = $model->find($mediaId);
    if (! $media) {
      return $this->fail('Not found', 404);
    }
    $current = $this->currentUser();
    $forum   = (new ForumModel())->find($media->forum_id);
    $isOwner = (int) $media->user_id === (int) $current->user_id;
    $isAdmin = $forum && (int) $forum->admin_id === (int) $current->user_id;
    if (! $isOwner && ! $isAdmin) {
      return $this->fail('Forbidden', 403);
    }
    $model->delete($mediaId);
    return $this->success(['ok' => true], 'Deleted');
  }

    private  function  moveUploadedFile(\CodeIgniter\HTTP\Files\UploadedFile
$file, int $forumId): string
  {
    $sanitized = $this->sanitizeFilename($file->getClientName());
    $subdir = 'uploads/forums/' . $forumId . '/' . gmdate('Y/m');
    $targetDir = FCPATH . $subdir;
    if (! is_dir($targetDir)) {
      mkdir($targetDir, 0775, true);
```

```
    }
    $newName = uniqid('', true) . '_' . $sanitized;
    $file->move($targetDir, $newName, true);
    return base_url($subdir . '/' . $newName);
  }

  private function sanitizeFilename(string $name): string
  {
    $name = preg_replace('/[^A-Za-z0-9._-]+/', '_', $name);
    return trim($name, '_');
  }
}
```

NoteController.php

```
<?php

namespace App\Controllers\API;

use App\Models\NoteModel;
use App\Models\ForumModel;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class NoteController extends BaseAPIController
{
  #[OAT\Post(
    path: "/forums/{id}/notes",
    tags: ["Notes"],
    summary: "Create note",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["judul"],
        properties: [
          new OAT\Property(property: "judul", type: "string"),
          new OAT\Property(property: "kategori", type: "string"),
          new OAT\Property(property: "mata_kuliah", type: "string"),
          new OAT\Property(property: "deskripsi", type: "string")
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
  public function store(int $forumId)
  {
    $rules = config('Validation')->noteStore;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $data    = $this->request->getJSON(true) ?? $this->request->getPost();
        $current = $this->currentUser();
    $model   = new NoteModel();
    $id      = $model->insert([
```

```
      'forum_id'    => $forumId,
      'user_id'     => $current->user_id,
      'judul'       => $data['judul'],
      'kategori'    => $data['kategori'] ?? null,
      'mata_kuliah' => $data['mata_kuliah'] ?? null,
      'deskripsi'   => $data['deskripsi'] ?? null,
    ], true);
    return $this->success($model->find($id), 'Created', null, 201);
  }

  #[OAT\Get(
    path: "/forums/{id}/notes",
    tags: ["Notes"],
    summary: "List notes",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer")),
        new OAT\Parameter(name: "kategori", in: "query", required: false,
schema: new OAT\Schema(type: "string")),
       new OAT\Parameter(name: "mata_kuliah", in: "query", required: false,
schema: new OAT\Schema(type: "string")),
      new OAT\Parameter(name: "q", in: "query", required: false, schema: new
OAT\Schema(type: "string")),
       new OAT\Parameter(name: "page", in: "query", required: false, schema:
new OAT\Schema(type: "integer")),
        new OAT\Parameter(name: "per_page", in: "query", required: false,
schema: new OAT\Schema(type: "integer")),
    ],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index(int $forumId)
  {
    $q           = trim((string) ($this->request->getGet('q') ?? ''));
    $kategori    = $this->request->getGet('kategori');
    $mataKuliah  = $this->request->getGet('mata_kuliah');
    $page        = max(1, (int) ($this->request->getGet('page') ?? 1));
    $perPage     = min(100, max(1, (int) ($this->request->getGet('per_page')
?? 10)));

    $builder = (new NoteModel())->builder()->where('forum_id', $forumId);
    if ($kategori) {
      $builder->where('kategori', $kategori);
    }
    if ($mataKuliah) {
      $builder->where('mata_kuliah', $mataKuliah);
    }
    if ($q !== '') {
      $builder->groupStart()
        ->like('judul', $q)
        ->orLike('deskripsi', $q)
      ->groupEnd();
    }
    $builder->orderBy('created_at', 'DESC');
    $total   = (clone $builder)->countAllResults(false);
    $data    = $builder->get(($page - 1) * $perPage, $perPage)->getResult();
    $meta    = service('paginationSvc')->buildMeta($page, $perPage, $total);
    return $this->success($data, null, $meta);
  }

  #[OAT\Get(
    path: "/notes/{id}",
    tags: ["Notes"],
    summary: "Show note",
```

```
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "OK"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function show(int $noteId)
  {
    $note = (new NoteModel())->find($noteId);
    if (! $note) {
      return $this->fail('Not found', 404);
    }
    return $this->success($note);
  }

  #[OAT\Patch(
    path: "/notes/{id}",
    tags: ["Notes"],
    summary: "Update note",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: false,
      content: new OAT\JsonContent(
        properties: [
          new OAT\Property(property: "judul", type: "string"),
          new OAT\Property(property: "kategori", type: "string"),
          new OAT\Property(property: "mata_kuliah", type: "string"),
          new OAT\Property(property: "deskripsi", type: "string")
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 200, description: "Updated"),
      new OAT\Response(response: 403, description: "Forbidden")
    ]
  )]
  public function update(int $noteId)
  {
    $model = new NoteModel();
    $note  = $model->find($noteId);
    if (! $note) {
      return $this->fail('Not found', 404);
    }
    if (! $this->canManage($note->forum_id, $note->user_id)) {
      return $this->fail('Forbidden', 403);
    }
    $data  = $this->request->getJSON(true) ?? $this->request->getRawInput();
      $patch = array_intersect_key($data, array_flip(['judul', 'kategori',
'mata_kuliah', 'deskripsi']));
    $model->update($noteId, $patch);
    return $this->success($model->find($noteId), 'Updated');
  }

  #[OAT\Delete(
    path: "/notes/{id}",
    tags: ["Notes"],
    summary: "Delete note",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
```

```
    responses: [
      new OAT\Response(response: 200, description: "Deleted"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function destroy(int $noteId)
  {
    $model = new NoteModel();
    $note  = $model->find($noteId);
    if (! $note) {
      return $this->fail('Not found', 404);
    }
    if (! $this->canManage($note->forum_id, $note->user_id)) {
      return $this->fail('Forbidden', 403);
    }
    $model->delete($noteId);
    return $this->success(['ok' => true], 'Deleted');
  }

  private function canManage(int $forumId, int $ownerId): bool
  {
    $current = $this->currentUser();
    if (! $current) {
     return false;
    }
    if ($ownerId === (int) $current->user_id) {
      return true;
    }
    $forum = (new ForumModel())->find($forumId);
    return $forum && (int) $forum->admin_id === (int) $current->user_id;
  }
}
```

NotificationController.php

```
<?php

namespace App\Controllers\API;

use App\Models\AnggotaForumModel;
use App\Models\KanbanModel;
use App\Models\DiscussionModel;
use CodeIgniter\Database\BaseConnection;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class NotificationController extends BaseAPIController
{
  #[OAT\Get(
    path: "/notifications",
    tags: ["Notifications"],
    summary: "Get forum notifications summary",
    security: [["bearerAuth" => []]],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index()
  {
    $db      = db_connect();
    $current = $this->currentUser();

    $forums = (new AnggotaForumModel())->builder()
      ->select('forum_id')
      ->where('user_id', $current->user_id)
      ->get()->getResultArray();
```

```
    $forumIds = array_map(static fn($r) => (int) $r['forum_id'], $forums);
    $summary  = [];
    foreach ($forumIds as $fid) {
      $lastSeen = $this->getLastSeen($db, $current->user_id, $fid);
      $newTasks = (new KanbanModel())->where('forum_id', $fid)
        ->where('created_at >', $lastSeen)->countAllResults();
      $newDiscussions = (new DiscussionModel())->where('forum_id', $fid)
        ->where('created_at >', $lastSeen)->countAllResults();
      $summary[] = [
        'forum_id'        => $fid,
        'new_tasks'       => $newTasks,
        'new_discussions' => $newDiscussions,
      ];
    }
    return $this->success(['forums' => $summary]);
  }

    private function getLastSeen(BaseConnection $db, int $userId, int
$forumId): string
  {
    $row = $db->table('user_forum_seen')
      ->where(['user_id' => $userId, 'forum_id' => $forumId])
      ->get()->getRowArray();
    return $row['last_seen_at'] ?? '1970-01-01 00:00:00';
  }
}
```

ReminderController.php

```
<?php

namespace App\Controllers\API;

use App\Models\ReminderModel;
use App\Models\KanbanModel;
use App\Models\ForumModel;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class ReminderController extends BaseAPIController
{
  #[OAT\Post(
    path: "/tasks/{id}/reminder",
    tags: ["Reminders"],
    summary: "Create reminder for task",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["title","waktu"],
        properties: [
          new OAT\Property(property: "title", type: "string"),
               new OAT\Property(property: "waktu", type: "string", format:
"date-time")
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request"),
      new OAT\Response(response: 409, description: "Conflict")
    ]
  )]
```

```
  public function store(int $taskId)
  {
    $rules = config('Validation')->reminderStore;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $task = (new KanbanModel())->find($taskId);
    if (! $task) {
      return $this->fail('Task not found', 404);
    }
    $model = new ReminderModel();
    $existing = $model->where('kanban_id', $taskId)->first();
    if ($existing) {
      return $this->fail('Reminder already exists for this task', 409);
    }
    $data    = $this->request->getJSON(true) ?? $this->request->getPost();
    $current = $this->currentUser();
    $reminderId = $model->insert([
      'kanban_id' => $taskId,
      'user_id'   => $current->user_id,
      'title'     => $data['title'],
      'waktu'     => $data['waktu'],
    ], true);
    return $this->success($model->find($reminderId), 'Created', null, 201);
  }

  #[OAT\Get(
    path: "/reminders",
    tags: ["Reminders"],
    summary: "List my reminders",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "upcoming", in: "query", required:
false, schema: new OAT\Schema(type: "boolean"))],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index()
  {
    $current  = $this->currentUser();
       $upcoming  =  filter_var($this->request->getGet('upcoming')  ??  'true',
FILTER_VALIDATE_BOOLEAN);
         $builder     =   (new   ReminderModel())->builder()->where('user_id',
$current->user_id);
    if ($upcoming) {
      $builder->where('waktu >=', gmdate('Y-m-d H:i:s'));
    }
    $builder->orderBy('waktu', 'ASC');
    $data = $builder->get()->getResult();
    return $this->success($data);
  }

  #[OAT\Delete(
    path: "/reminders/{id}",
    tags: ["Reminders"],
    summary: "Delete reminder",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "Deleted"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function destroy(int $reminderId)
  {
```

```
    $model    = new ReminderModel();
    $reminder = $model->find($reminderId);
    if (! $reminder) {
      return $this->fail('Reminder not found', 404);
    }
    $task = (new KanbanModel())->find($reminder->kanban_id);
    if (! $task) {
      return $this->fail('Task not found', 404);
    }
            if    (!    $this->canManageTask((int)    $task->forum_id,    (int)
$task->created_by)      &&      (int)      $reminder->user_id      !==      (int)
$this->currentUser()->user_id) {
      return $this->fail('Forbidden', 403);
    }
    $model->delete($reminderId);
    return $this->success(['ok' => true], 'Deleted');
  }

  private function canManageTask(int $forumId, int $createdBy): bool
  {
    $current = $this->currentUser();
    if (! $current) {
      return false;
    }
    if ($createdBy === (int) $current->user_id) {
      return true;
    }
    $forum = (new ForumModel())->find($forumId);
    return $forum && (int) $forum->admin_id === (int) $current->user_id;
  }
}
```

SearchController.php

```
<?php

namespace App\Controllers\API;

use App\Models\ForumModel;
use App\Models\KanbanModel;
use App\Models\NoteModel;
use App\Models\DiscussionModel;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class SearchController extends BaseAPIController
{
  #[OAT\Get(
    path: "/search",
    tags: ["Search"],
    summary: "Search across entities",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "scope", in: "query", required: false, schema:
new               OAT\Schema(type:               "string",               enum:
["forums","tasks","notes","discussions","all"])),
       new OAT\Parameter(name: "q", in: "query", required: true, schema: new
OAT\Schema(type: "string"))
    ],
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index()
  {
    $scope = $this->request->getGet('scope') ?? 'all';
    $q     = trim((string) ($this->request->getGet('q') ?? ''));
```

```
    if ($q === '') {
      return $this->success([]);
    }
    $results = [];
    if ($scope === 'forums' || $scope === 'all') {
      $rows = (new ForumModel())->builder()
                    ->groupStart()->like('nama',   $q)->orLike('deskripsi',
$q)->groupEnd()
        ->limit(20)->get()->getResultArray();
      foreach ($rows as $r) {
        $r['type'] = 'forum';
        $results[] = $r;
      }
    }
    if ($scope === 'tasks' || $scope === 'all') {
      $rows = (new KanbanModel())->builder()
                    ->groupStart()->like('judul',   $q)->orLike('deskripsi',
$q)->groupEnd()
        ->limit(20)->get()->getResultArray();
      foreach ($rows as $r) {
        $r['type'] = 'task';
        $results[] = $r;
      }
    }
    if ($scope === 'notes' || $scope === 'all') {
      $rows = (new NoteModel())->builder()
                    ->groupStart()->like('judul',   $q)->orLike('deskripsi',
$q)->groupEnd()
        ->limit(20)->get()->getResultArray();
      foreach ($rows as $r) {
        $r['type'] = 'note';
        $results[] = $r;
      }
    }
    if ($scope === 'discussions' || $scope === 'all') {
      $rows = (new DiscussionModel())->builder()
        ->like('isi', $q)
        ->limit(20)->get()->getResultArray();
      foreach ($rows as $r) {
        $r['type'] = 'discussion';
        $results[] = $r;
      }
    }
    return $this->success($results);
  }
}
```

TaskController.php

```
<?php

namespace App\Controllers\API;

use App\Models\KanbanModel;
use App\Models\ForumModel;
use App\Models\MediaModel;
use CodeIgniter\Files\File;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class TaskController extends BaseAPIController
{
  #[OAT\Post(
    path: "/forums/{id}/tasks",
    tags: ["Tasks"],
```

```
    summary: "Create task in forum",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: true,
      content: new OAT\JsonContent(
        required: ["judul"],
        properties: [
          new OAT\Property(property: "judul", type: "string"),
          new OAT\Property(property: "deskripsi", type: "string"),
          new OAT\Property(property: "tenggat_waktu", type: "string", format:
"date-time"),
             new OAT\Property(property: "file_url", type: "string", format:
"uri")
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
  public function store(int $forumId)
  {
    $rules = config('Validation')->taskStore;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $data    = $this->request->getJSON(true) ?? $this->request->getPost();
    $current = $this->currentUser();
    $model   = new KanbanModel();
    $taskId  = $model->insert([
      'forum_id'      => $forumId,
      'judul'         => $data['judul'],
      'deskripsi'     => $data['deskripsi'] ?? null,
      'tenggat_waktu' => $data['tenggat_waktu'] ?? null,
      'file_url'      => $data['file_url'] ?? null,
      'status'        => 'todo',
      'created_by'    => $current->user_id,
    ], true);
    $task = $model->find($taskId);
    return $this->success($task, 'Created', null, 201);
  }

  #[OAT\Get(
    path: "/forums/{id}/tasks",
    tags: ["Tasks"],
    summary: "List tasks in forum",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer")),
      new OAT\Parameter(name: "status", in: "query", required: false, schema:
new OAT\Schema(type: "string", enum: ["todo","doing","done"])),
       new OAT\Parameter(name: "q", in: "query", required: false, schema: new
OAT\Schema(type: "string")),
       new OAT\Parameter(name: "sort", in: "query", required: false, schema:
new OAT\Schema(type: "string", enum: ["deadline","created_at"])),
       new OAT\Parameter(name: "page", in: "query", required: false, schema:
new OAT\Schema(type: "integer")),
         new OAT\Parameter(name: "per_page", in: "query", required: false,
schema: new OAT\Schema(type: "integer"))
    ],
```

```
    responses: [new OAT\Response(response: 200, description: "OK")]
  )]
  public function index(int $forumId)
  {
    $status    = $this->request->getGet('status');
    $q         = trim((string) ($this->request->getGet('q') ?? ''));
    $sort      = $this->request->getGet('sort') ?? 'created_at';
    $page      = max(1, (int) ($this->request->getGet('page') ?? 1));
    $perPage   = min(100, max(1, (int) ($this->request->getGet('per_page') ??
10)));

    $builder = (new KanbanModel())->builder()->where('forum_id', $forumId);
    if (in_array($status, ['todo', 'doing', 'done'], true)) {
      $builder->where('status', $status);
    }
    if ($q !== '') {
      $builder->groupStart()
        ->like('judul', $q)
        ->orLike('deskripsi', $q)
      ->groupEnd();
    }
    $sortMap = ['deadline' => 'tenggat_waktu', 'created_at' => 'created_at'];
    $orderBy = $sortMap[$sort] ?? 'created_at';
    $builder->orderBy($orderBy, 'DESC');

    $total   = (clone $builder)->countAllResults(false);
    $data    = $builder->get(($page - 1) * $perPage, $perPage)->getResult();
    $meta    = service('paginationSvc')->buildMeta($page, $perPage, $total);
    return $this->success($data, null, $meta);
  }

  #[OAT\Get(
    path: "/tasks/{id}",
    tags: ["Tasks"],
    summary: "Show task",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "OK"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function show(int $taskId)
  {
    $task = (new KanbanModel())->find($taskId);
    if (! $task) {
      return $this->fail('Task not found', 404);
    }
    return $this->success($task);
  }

  #[OAT\Patch(
    path: "/tasks/{id}",
    tags: ["Tasks"],
    summary: "Update task",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: false,
      content: new OAT\JsonContent(
        properties: [
          new OAT\Property(property: "judul", type: "string"),
```

```
          new OAT\Property(property: "deskripsi", type: "string"),
          new OAT\Property(property: "tenggat_waktu", type: "string", format:
"date-time"),
                new OAT\Property(property: "status", type: "string", enum:
["todo","doing","done"]),
              new OAT\Property(property: "file_url", type: "string", format:
"uri")
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 200, description: "Updated"),
      new OAT\Response(response: 400, description: "Bad Request"),
      new OAT\Response(response: 403, description: "Forbidden")
    ]
  )]
  public function update(int $taskId)
  {
    $rules = config('Validation')->taskUpdate;
    if (! $this->validate($rules)) {
      return $this->fail(implode('; ', $this->validator->getErrors()), 400);
    }
    $model = new KanbanModel();
    $task  = $model->find($taskId);
    if (! $task) {
      return $this->fail('Task not found', 404);
    }
    if (! $this->canManageTask($task->forum_id, $task->created_by)) {
      return $this->fail('Forbidden', 403);
    }

    $data  = $this->request->getJSON(true) ?? $this->request->getRawInput();
      $patch = array_intersect_key($data, array_flip(['judul', 'deskripsi',
'tenggat_waktu', 'status', 'file_url']));
    $model->update($taskId, $patch);
    return $this->success($model->find($taskId), 'Updated');
  }

  #[OAT\Delete(
    path: "/tasks/{id}",
    tags: ["Tasks"],
    summary: "Delete task",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    responses: [
      new OAT\Response(response: 200, description: "Deleted"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
  public function destroy(int $taskId)
  {
    $model = new KanbanModel();
    $task  = $model->find($taskId);
    if (! $task) {
      return $this->fail('Task not found', 404);
    }
    if (! $this->canManageTask($task->forum_id, $task->created_by)) {
      return $this->fail('Forbidden', 403);
    }
    $model->delete($taskId);
    return $this->success(['ok' => true], 'Deleted');
  }
```

```
  #[OAT\Post(
    path: "/tasks/{id}/attachments",
    tags: ["Tasks"],
    summary: "Attach file or link to task",
    security: [["bearerAuth" => []]],
     parameters: [new OAT\Parameter(name: "id", in: "path", required: true,
schema: new OAT\Schema(type: "integer"))],
    requestBody: new OAT\RequestBody(
      required: true,
      content: [
        new OAT\MediaType(
          mediaType: "multipart/form-data",
          schema: new OAT\Schema(
            type: "object",
            properties: [
                  new OAT\Property(property: "file", type: "string", format:
"binary"),
                new OAT\Property(property: "file_url", type: "string", format:
"uri")
            ]
          )
        ),
        new OAT\MediaType(
          mediaType: "application/json",
          schema: new OAT\Schema(
            type: "object",
                  properties: [new OAT\Property(property: "file_url", type:
"string", format: "uri")]
          )
        )
      ]
    ),
    responses: [
      new OAT\Response(response: 201, description: "Created"),
      new OAT\Response(response: 400, description: "Bad Request")
    ]
  )]
  public function attach(int $taskId)
  {
    $task = (new KanbanModel())->find($taskId);
    if (! $task) {
      return $this->fail('Task not found', 404);
    }
    $current = $this->currentUser();
    $mediaModel = new MediaModel();

    $file = $this->request->getFile('file');
    $fileUrl = null;
    if ($file && $file->isValid()) {
      $fileUrl = $this->moveUploadedFile($file, (int) $task->forum_id);
    } else {
      $body = $this->request->getJSON(true) ?? $this->request->getPost();
      $fileUrl = $body['file_url'] ?? null;
      if (! $fileUrl) {
        return $this->fail('No file or file_url provided', 400);
      }
    }

    $mediaId = $mediaModel->insert([
      'user_id'  => $current->user_id,
      'forum_id' => $task->forum_id,
      'ref_id'   => $taskId,
      'file_url' => $fileUrl,
    ], true);
```

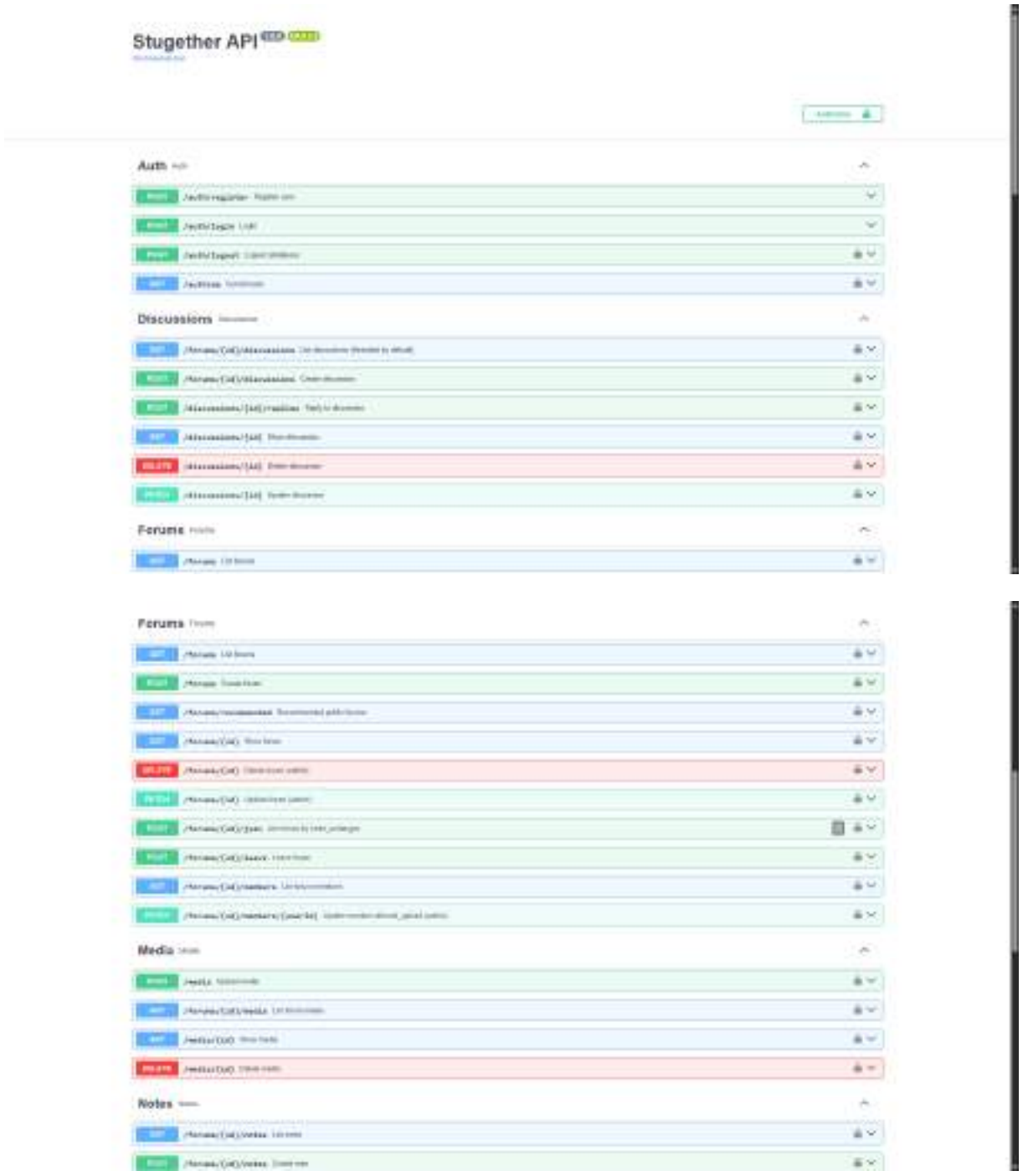
```
      return $this->success($mediaModel->find($mediaId), 'Attached', null,
201);
  }

  private function canManageTask(int $forumId, int $createdBy): bool
  {
    $current = $this->currentUser();
    if (! $current) {
      return false;
    }
    if ((int) $createdBy === (int) $current->user_id) {
      return true;
    }
    $forum = (new ForumModel())->find($forumId);
    return $forum && (int) $forum->admin_id === (int) $current->user_id;
  }

    private function moveUploadedFile(\CodeIgniter\HTTP\Files\UploadedFile
$file, int $forumId): string
  {
    $sanitized = $this->sanitizeFilename($file->getClientName());
    $subdir = 'uploads/forums/' . $forumId . '/' . gmdate('Y/m');
    $targetDir = FCPATH . $subdir;
    if (! is_dir($targetDir)) {
      mkdir($targetDir, 0775, true);
    }
    $newName = uniqid('', true) . '_' . $sanitized;
    $file->move($targetDir, $newName, true);
    return base_url($subdir . '/' . $newName);
  }

  private function sanitizeFilename(string $name): string
  {
    $name = preg_replace('/[^A-Za-z0-9._-]+/', '_', $name);
    return trim($name, '_');
  }
}
```

UserController.php

```
<?php

namespace App\Controllers\API;

use App\Models\UserModel;
use OpenApi\Annotations as OA;
use OpenApi\Attributes as OAT;

class UserController extends BaseAPIController
{
  #[OAT\Get(
    path: "/users/{id}",
    tags: ["Users"],
    summary: "Show user",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer"))
    ],
    responses: [
      new OAT\Response(response: 200, description: "User"),
      new OAT\Response(response: 404, description: "Not found")
    ]
  )]
```

```
  public function show(int $id)
  {
    $user = (new UserModel())->find($id);
    if (! $user) {
      return $this->fail('User not found', 404);
    }
    return $this->success($user);
  }

  #[OAT\Put(
    path: "/users/{id}",
    tags: ["Users"],
    summary: "Update user (self only)",
    security: [["bearerAuth" => []]],
    parameters: [
       new OAT\Parameter(name: "id", in: "path", required: true, schema: new
OAT\Schema(type: "integer"))
    ],
    requestBody: new OAT\RequestBody(
      required: false,
      content: new OAT\JsonContent(
        properties: [
          new OAT\Property(property: "nim", type: "string"),
          new OAT\Property(property: "nama", type: "string"),
          new OAT\Property(property: "kelas", type: "string"),
          new OAT\Property(property: "semester", type: "integer"),
              new OAT\Property(property: "password", type: "string", format:
"password"),
        ]
      )
    ),
    responses: [
      new OAT\Response(response: 200, description: "Updated"),
      new OAT\Response(response: 400, description: "Bad Request"),
      new OAT\Response(response: 403, description: "Forbidden")
    ]
  )]
  public function update(int $id)
  {
    $current = $this->currentUser();
    if (! $current || (int) $current->user_id !== (int) $id) {
      return $this->fail('You can only update your own profile', 403);
    }
    $data  = $this->request->getJSON(true) ?? $this->request->getRawInput();
     $patch = array_intersect_key($data, array_flip(['nim', 'nama', 'kelas',
'semester']));
    if (isset($data['password']) && $data['password']) {
      $patch['password'] = password_hash($data['password'], PASSWORD_BCRYPT);
    }
    $model = new UserModel();
    $model->update($id, $patch);
    $user = $model->find($id);
    return $this->success($user, 'Updated');
  }
}
```

c. Screenshot Hasil



d. Kendala dan Solusi

Kendala	Solusi	Hasil
Swagger belum native di CI4	Integrasi library pihak ketiga	Dokumentasi bisa dibuat
Kode undangan harus unique	Menggunakan hash + upper random	Berjalan stabil

Struktur endpoint awal masih besar	Menyusun template prompt di Cursor	Endpoint lebih konsisten

BAB IV: PENGUJIAN

4.1 Test Case yang Dilakukan

a. Authentications Tests

No	Yang Diuji	Input	Output Diharapkan	Status
1	Registrasi user baru	`POST /auth/register` `{nama: "Alice", email: "alice@example.com", password: "password123"}`	Status 201, response berisi token	✅
2	Login user	`POST /auth/login` `{email: "alice@example.com", password: "password123"}`	Status 200, response berisi token	✅
3	Get user info (me)	`GET /auth/me` Header: `Authorization: Bearer {token}`	Status 200, response berisi data user dengan email yang sesuai	✅

b. Form Tests

No	Yang Diuji	Input	Output Diharapkan	Status
4	Membuat forum baru	`POST /forums` `{nama: "Private Forum", deskripsi: "Test", jenis_forum: "akademik", is_public: 0}`	Status 201, forum berhasil dibuat	✅
5	List forum dengan scope "mine"	`GET /forums?scope=mine`	Status 200, total forum >= 1	✅
6	List forum dengan scope "public"	`GET /forums?scope=public`	Status 200, total forum = 0 (karena forum private)	✅

7	Join forum via kode undangan	`POST /forums/{id}/join` `{kode_undangan: "{kode}"}`	Status 200, user berhasil join forum	✅
8	Non-admin tidak bisa update forum	`PATCH /forums/{id}` `{nama: "New Name"}` User: non-admin	Status 403, Forbidden	✅

c. Task (Kanban) Tests

No	Yang Diuji	Input	Output Diharapkan	Status
9	Membuat task baru	`POST /forums/{id}/tasks` `{judul: "First Task", deskripsi: "Do something"}`	Status 201, task berhasil dibuat	✅
10	Update status task	`PATCH /tasks/{id}` `{status: "doing"}`	Status 200, status task berhasil diupdate	✅
11	Membuat reminder untuk task	`POST /tasks/{id}/reminder` `{title: "Ping", waktu: "{datetime}"}`	Status 201, reminder berhasil dibuat	✅
12	Membuat reminder kedua untuk task yang sama	`POST /tasks/{id}/reminder` `{title: "Ping again", waktu: "{datetime}"}`	Status 409, Conflict (task sudah punya reminder)	✅
13	Non-member tidak bisa membuat task di forum private	`POST /forums/{id}/tasks` `{judul: "X"}` User: non-member	Status 403, Forbidden	✅

d. Discussion Tests

No	Yang Diuji	Input	Output Diharapkan	Status
14	Membuat diskusi baru	`POST /forums/{id}/discussions` `{isi: "Hello"}`	Status 201, diskusi berhasil dibuat	✅
15	Membalas diskusi	`POST /discussions/{id}/replies` `{isi: "Reply"}`	Status 201, reply berhasil dibuat	✅
16	List diskusi dengan threaded view	`GET /forums/{id}/discussions?threaded=true`	Status 200, response berisi diskusi dengan children (replies)	✅

e. Notes Tests

No	Yang Diuji	Input	Output Diharapkan	Status
17	Membuat catatan baru	`POST /forums/{id}/notes` `{judul: "Lecture 1", kategori: "math", mata_kuliah: "algebra"}`	Status 201, catatan berhasil dibuat	✅
18	Filter catatan berdasarkan kategori	`GET /forums/{id}/notes?kategori=math`	Status 200, total catatan >= 1	✅
19	Update catatan	`PATCH /notes/{id}` `{judul: "Lecture 1 updated"}`	Status 200, catatan berhasil diupdate	✅
20	Hapus catatan	`DELETE /notes/{id}`	Status 200, catatan berhasil dihapus	✅

f. Media Tests

No	Yang Diuji	Input	Output Diharapkan	Status
21	Upload media dengan file_url	`POST /media` `{forum_id: {id}, file_url: "https://example.com/file.pdf"}`	Status 201, media berhasil diupload	✅
22	User lain tidak bisa hapus media milik admin	`DELETE /media/{id}` User: bukan pemilik media	Status 403, Forbidden	✅

4.2 Bug dan Perbaikan

- Bug: Migration gagal karena ENUM tidak dikenali
- Perbaikan: Mengganti ENUM dengan VARCHAR + validation

BAB V: KESIMPULAN

5.1 Kesimpulan

Pada proyek Stugether, saya berperan sebagai backend developer dan database designer. Saya merancang struktur database, menyiapkan migration, menyusun endpoint, hingga menentukan rencana dokumentasi API. Hasil yang dicapai berupa fondasi sistem yang siap dikembangkan ke fitur-fitur lanjutan.

5.2 Saran

- Menambahkan tabel ForumMembers
- Menambahkan fitur tugas & file sharing
- Membangun UI frontend dan mobile app

DAFTAR PUSTAKA

Dokumentasi CodeIgniter 4

Dokumentasi MySQL

OpenAPI / Swagger Docs

Chat dan diskusi perancangan pada proyek

ChatGPT

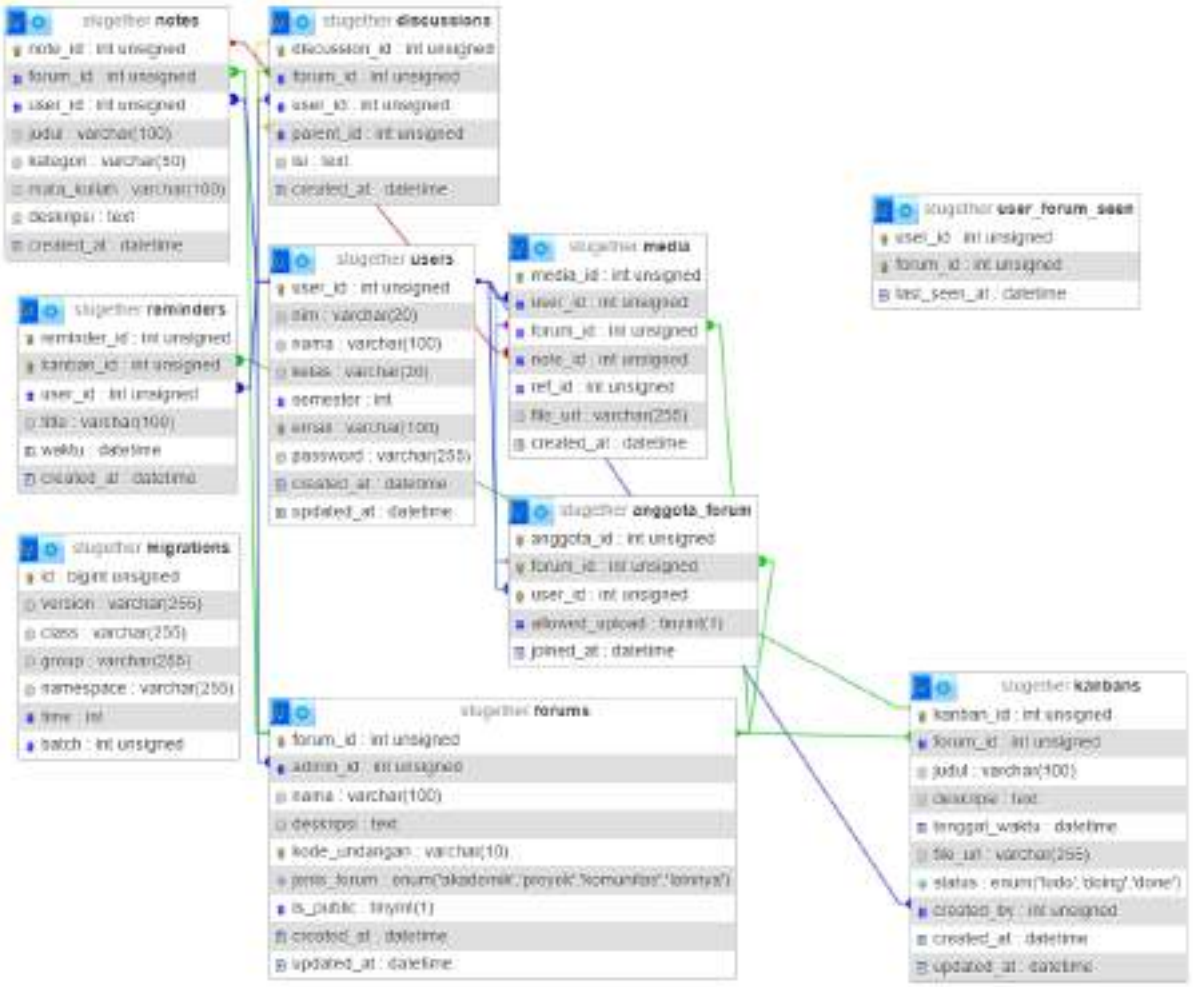
Cursor AI

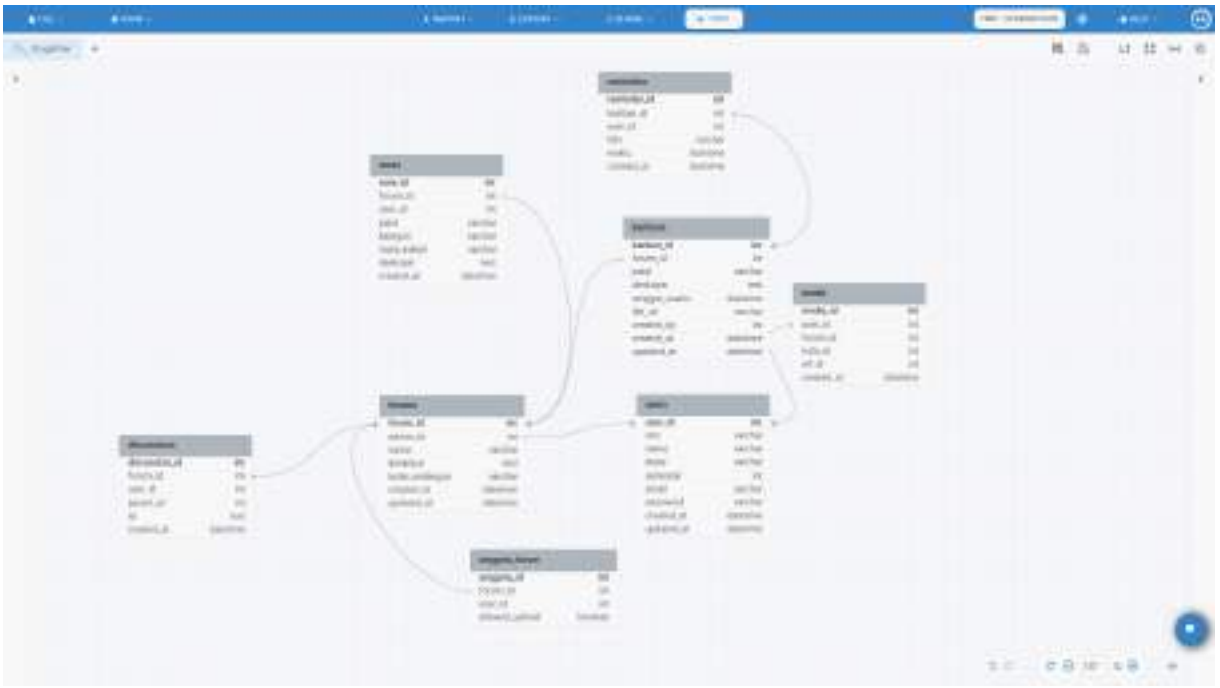
LAMPIRAN

Lampiran A: Source Code

https://github.com/faridreaming/stugether

Lampiran B: Dokumentasi Database





Lampiran C: Screenshot Tambahan