**Penjelasan Diagram Use Case:**

**Administrator:**

* **Kelola Kursus**: Administrator memiliki kemampuan untuk **menambah**, **mengedit**, dan **menghapus** kursus.
* **Kelola Tugas**: Administrator dapat menambah, mengedit, dan menghapus tugas yang harus diselesaikan oleh siswa.
* **Kelola Galeri**: Administrator bisa melihat semua konten yang diunggah ke galeri harian dan dapat menghapus konten yang tidak sesuai.

**Siswa:**

* **Lihat Kursus**: Siswa bisa melihat daftar kursus yang tersedia. Mereka juga bisa melihat detail kursus untuk mengetahui lebih lanjut tentang kursus tersebut.
* **Lihat Tugas**: Siswa dapat melihat daftar tugas yang harus mereka kerjakan.
* **Kirim Tugas**: Siswa bisa mengumpulkan tugas mereka, yang bisa berupa file (misalnya, PDF, Word, gambar).
* **Kirim Galeri**: Siswa dapat mengunggah gambar, video, atau teks ke galeri harian mereka sebagai bagian dari dokumentasi atau kegiatan sekolah.
* **Lihat Galeri**: Siswa juga bisa melihat galeri harian yang berisi konten dari teman-temannya.

**Sistem:**

* **Menyimpan Data Kursus**: Sistem akan menyimpan data kursus yang dimasukkan oleh administrator.
* **Menyimpan Data Tugas**: Sistem akan menyimpan tugas yang dikirim oleh administrator dan tugas yang dikumpulkan oleh siswa.
* **Menyimpan Galeri**: Sistem akan menyimpan file gambar, video, dan teks yang diunggah oleh siswa ke galeri harian.

**DATA BASE**

const sqlite3 = require('sqlite3').verbose();

const bcrypt = require('bcryptjs');

const db = new sqlite3.Database('./registration.db');

db.serialize(() => {

    // Hapus dan buat ulang tabel utama

    db.run(`DROP TABLE IF EXISTS courses`);

    db.run(`DROP TABLE IF EXISTS tugas`);

    db.run(`DROP TABLE IF EXISTS jawaban`);

    // Tabel kursus

    db.run(`

        CREATE TABLE IF NOT EXISTS courses (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            name TEXT NOT NULL,

            description TEXT NOT NULL,

            instructor TEXT NOT NULL,

            files TEXT

        );

    `);

    // Tabel pengguna

    db.run(`

        CREATE TABLE IF NOT EXISTS users (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            name TEXT NOT NULL,

            class TEXT NOT NULL,

            student\_id TEXT NOT NULL UNIQUE,

            email TEXT NOT NULL UNIQUE,

            password TEXT NOT NULL,

            role TEXT NOT NULL

        );

    `);

    // Tabel tugas

    db.run(`

        CREATE TABLE IF NOT EXISTS tugas (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            title TEXT NOT NULL,

            description TEXT NOT NULL,

            deadline TEXT,

            file TEXT

        );

    `);

    // Tabel jawaban siswa

    db.run(`

            CREATE TABLE IF NOT EXISTS jawaban (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            task\_id INTEGER,

            student\_id TEXT,

            text TEXT,

            file TEXT,

            created\_at TEXT,

            nilai TEXT,

            feedback TEXT

        );

    `);

// tabel galeri

    db.run(`

        CREATE TABLE IF NOT EXISTS galeri (

          id INTEGER PRIMARY KEY AUTOINCREMENT,

          student\_id TEXT NOT NULL,

          text TEXT,

          file TEXT,

          created\_at TEXT NOT NULL

        );

      `);

    // Tambahkan user admin default

    const insertUser = (name, studentClass, studentId, email, password, role) => {

        db.get(`SELECT \* FROM users WHERE email = ? OR student\_id = ?`, [email, studentId], (err, row) => {

            if (row) {

                console.log(`User ${name} already exists.`);

                return;

            }

            bcrypt.hash(password, 10, (err, hash) => {

                if (err) return console.error(err);

                db.run(

                    `INSERT INTO users (name, class, student\_id, email, password, role) VALUES (?, ?, ?, ?, ?, ?)`,

                    [name, studentClass, studentId, email, hash, role],

                    (err) => {

                        if (err) console.error(err.message);

                        else console.log(`✅ User ${name} added.`);

                    }

                );

            });

        });

    };

    insertUser('Admin User', 'Admin', '0001', 'admin@example.com', 'admin123', 'admin');

});

db.close();

**PROGRAM APLIKASI**

const express = require('express');

const sqlite3 = require('sqlite3').verbose();

const bcrypt = require('bcryptjs');

const cors = require('cors');

const path = require('path');

const multer = require('multer');

const upload = require('./config/multer');

const app = express();

const db = new sqlite3.Database('./registration.db');

const port = 3000;

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/uploads', express.static(path.join(\_\_dirname, 'public/uploads')));

// ------------------ Upload Config ------------------

const tugasStorage = multer.diskStorage({

    destination: (req, file, cb) => cb(null, 'public/uploads/tugas'),

    filename: (req, file, cb) => cb(null, 'tugas-' + Date.now() + path.extname(file.originalname))

});

const uploadTugas = multer({ storage: tugasStorage, limits: { fileSize: 10 \* 1024 \* 1024 }, fileFilter: upload.fileFilter });

const uploadJawaban = multer({

    storage: multer.diskStorage({

        destination: (req, file, cb) => cb(null, 'public/uploads/jawaban'),

        filename: (req, file, cb) => cb(null, 'jawaban-' + Date.now() + '-' + file.originalname)

    }),

    limits: { fileSize: 10 \* 1024 \* 1024 }

});

const galeriStorage = multer.diskStorage({

    destination: (req, file, cb) => cb(null, 'public/uploads/galeri'),

    filename: (req, file, cb) => cb(null, 'galeri-' + Date.now() + path.extname(file.originalname))

  });

  const uploadGaleri = multer({

    storage: galeriStorage,

    limits: { fileSize: 20 \* 1024 \* 1024 },

    fileFilter: (req, file, cb) => {

      const allowedTypes = ['image/jpeg', 'image/png', 'image/gif', 'video/mp4', 'video/quicktime'];

      if (allowedTypes.includes(file.mimetype)) {

        cb(null, true);

      } else {

        cb(new Error('Format file tidak didukung'), false);

      }

    }

  });

// ------------------ Autentikasi ------------------

app.post('/register', (req, res) => {

    const { name, class: className, studentId, email, password, role } = req.body;

    if (!name || !className || !studentId || !email || !password || !role) {

        return res.status(400).json({ error: 'Semua field harus diisi.' });

    }

    db.get('SELECT \* FROM users WHERE student\_id = ? OR email = ?', [studentId, email], (err, row) => {

        if (row) return res.status(400).json({ error: 'Email atau Student ID sudah digunakan.' });

        bcrypt.hash(password, 10, (err, hash) => {

            if (err) return res.status(500).json({ error: 'Gagal hash password' });

            db.run(`INSERT INTO users (name, class, student\_id, email, password, role) VALUES (?, ?, ?, ?, ?, ?)`,

                [name, className, studentId, email, hash, role],

                function (err) {

                    if (err) return res.status(400).json({ error: err.message });

                    res.json({ message: 'Register berhasil', userId: this.lastID });

                });

        });

    });

});

app.post('/login', (req, res) => {

    const { email, password } = req.body;

    db.get(`SELECT \* FROM users WHERE email = ?`, [email], (err, user) => {

        if (!user) return res.status(404).json({ error: 'User tidak ditemukan' });

        bcrypt.compare(password, user.password, (err, result) => {

            if (!result) return res.status(401).json({ error: 'Password salah' });

            res.json({ message: 'Login berhasil', user: { id: user.id, name: user.name, role: user.role, student\_id: user.student\_id } });

        });

    });

});

// ------------------ Manajemen Pengguna ------------------

app.get('/users', (req, res) => {

    db.all("SELECT id, name, class, student\_id, email, role FROM users", [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

// Ambil detail pengguna

app.get('/users/:id', (req, res) => {

    const id = req.params.id;

    db.get("SELECT id, name, class, student\_id, email, role FROM users WHERE id = ?", [id], (err, row) => {

        if (err) return res.status(500).json({ error: err.message });

        if (!row) return res.status(404).json({ error: 'Pengguna tidak ditemukan' });

        res.json(row);

    });

});

// Edit pengguna

app.put('/users/:id', (req, res) => {

    const { name, class: className, studentId, email, role } = req.body;

    const id = req.params.id;

    db.run(`UPDATE users SET name = ?, class = ?, student\_id = ?, email = ?, role = ? WHERE id = ?`,

        [name, className, studentId, email, role, id],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Pengguna berhasil diperbarui' });

        });

});

// Hapus pengguna

app.delete('/users/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM users WHERE id = ?`, [id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Pengguna berhasil dihapus' });

    });

});

// ------------------ Kursus ------------------

app.get('/courses', (req, res) => {

    db.all("SELECT \* FROM courses", [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

app.post('/courses', upload.array('files', 5), (req, res) => {

    const { name, description, instructor } = req.body;

    const files = req.files.map(file => file.filename);

    db.run(`INSERT INTO courses (name, description, instructor, files) VALUES (?, ?, ?, ?)`,

        [name, description, instructor, JSON.stringify(files)],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Kursus ditambahkan', courseId: this.lastID });

        });

});

app.put('/courses/:id', upload.array('files', 5), (req, res) => {

    const { name, description, instructor } = req.body;

    const files = req.files.map(file => file.filename);

    const id = req.params.id;

    db.run(`UPDATE courses SET name = ?, description = ?, instructor = ?, files = ? WHERE id = ?`,

        [name, description, instructor, JSON.stringify(files), id],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Kursus diperbarui' });

        });

});

app.delete('/courses/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM courses WHERE id = ?`, [id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Kursus berhasil dihapus' });

    });

});

// ------------------ Tugas ------------------

app.get('/tugas', (req, res) => {

    db.all("SELECT \* FROM tugas", [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

app.get('/tugas/:id', (req, res) => {

    const id = req.params.id;

    db.get("SELECT \* FROM tugas WHERE id = ?", [id], (err, row) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(row || {});

    });

});

app.post('/tugas', uploadTugas.single('file'), (req, res) => {

    const { title, description, deadline } = req.body;

    const file = req.file?.filename || null;

    db.run(`INSERT INTO tugas (title, description, deadline, file) VALUES (?, ?, ?, ?)`,

        [title, description, deadline, file],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Tugas ditambahkan', id: this.lastID });

        });

});

app.put('/tugas/:id', uploadTugas.single('file'), (req, res) => {

    const id = req.params.id;

    const { title, description, deadline } = req.body;

    const file = req.file?.filename;

    const updateQuery = file

        ? `UPDATE tugas SET title = ?, description = ?, deadline = ?, file = ? WHERE id = ?`

        : `UPDATE tugas SET title = ?, description = ?, deadline = ? WHERE id = ?`;

    const params = file

        ? [title, description, deadline, file, id]

        : [title, description, deadline, id];

    db.run(updateQuery, params, function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Tugas diperbarui' });

    });

});

app.delete('/tugas/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM tugas WHERE id = ?`, [id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Tugas berhasil dihapus' });

    });

});

// ------------------ Jawaban ------------------

// Cek deadline sebelum insert jawaban

app.post('/jawaban', uploadJawaban.single('file'), (req, res) => {

    const { task\_id, student\_id, text } = req.body;

    const file = req.file?.filename || null;

    const created\_at = new Date().toISOString();

    db.get(`SELECT \* FROM tugas WHERE id = ?`, [task\_id], (err, tugas) => {

        if (err) return res.status(500).json({ error: err.message });

        if (tugas?.deadline && new Date(tugas.deadline) < new Date()) {

            return res.status(400).json({ error: 'Tugas sudah melewati deadline.' });

        }

        db.get(`SELECT \* FROM jawaban WHERE task\_id = ? AND student\_id = ?`, [task\_id, student\_id], (err, existing) => {

            if (existing) return res.status(400).json({ error: 'Jawaban sudah pernah dikirim.' });

            db.run(`INSERT INTO jawaban (task\_id, student\_id, text, file, created\_at) VALUES (?, ?, ?, ?, ?)`,

                [task\_id, student\_id, text, file, created\_at],

                function (err) {

                    if (err) return res.status(500).json({ error: err.message });

                    res.json({ message: 'Jawaban dikirim', id: this.lastID });

                });

        });

    });

});

// Edit jawaban (jika belum dinilai dan belum lewat deadline)

app.put('/jawaban/:id', uploadJawaban.single('file'), (req, res) => {

    const { text } = req.body;

    const file = req.file?.filename || null;

    const id = req.params.id;

    db.get(`SELECT \* FROM jawaban WHERE id = ?`, [id], (err, jawaban) => {

        if (!jawaban) return res.status(404).json({ error: 'Jawaban tidak ditemukan' });

        if (jawaban.nilai) return res.status(400).json({ error: 'Jawaban sudah dinilai, tidak bisa diedit' });

        // Cek deadline tugas

        db.get(`SELECT \* FROM tugas WHERE id = ?`, [jawaban.task\_id], (err, tugas) => {

            if (tugas?.deadline && new Date(tugas.deadline) < new Date()) {

                return res.status(400).json({ error: 'Tugas sudah melewati deadline, tidak bisa diedit.' });

            }

            const sql = file

                ? `UPDATE jawaban SET text = ?, file = ?, created\_at = ? WHERE id = ?`

                : `UPDATE jawaban SET text = ?, created\_at = ? WHERE id = ?`;

            const params = file

                ? [text, file, new Date().toISOString(), id]

                : [text, new Date().toISOString(), id];

            db.run(sql, params, function (err) {

                if (err) return res.status(500).json({ error: err.message });

                res.json({ message: 'Jawaban diperbarui' });

            });

        });

    });

});

app.get('/jawaban/cek/:task\_id/:student\_id', (req, res) => {

    const { task\_id, student\_id } = req.params;

    db.get(`SELECT \* FROM jawaban WHERE task\_id = ? AND student\_id = ?`, [task\_id, student\_id], (err, row) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(row || null);

    });

});

app.get('/jawaban', (req, res) => {

    db.all(`SELECT jawaban.\*, tugas.title AS task\_title FROM jawaban LEFT JOIN tugas ON jawaban.task\_id = tugas.id ORDER BY jawaban.created\_at DESC`, [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

app.put('/jawaban/:id/nilai', (req, res) => {

    const { nilai, feedback } = req.body;

    const id = req.params.id;

    db.run(`UPDATE jawaban SET nilai = ?, feedback = ? WHERE id = ?`, [nilai, feedback, id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Nilai berhasil disimpan' });

    });

});

app.get('/laporan-nilai', (req, res) => {

    const sql = `

      SELECT

        users.name AS student\_name,

        users.student\_id,

        tugas.title AS task\_title,

        jawaban.nilai,

        jawaban.feedback,

        jawaban.created\_at

      FROM users

      CROSS JOIN tugas

      LEFT JOIN jawaban

        ON jawaban.task\_id = tugas.id

        AND CAST(jawaban.student\_id AS TEXT) = users.student\_id

      WHERE users.role = 'student'

      ORDER BY tugas.title, users.name;

    `;

    db.all(sql, [], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

// ------------------ Galeri Harian ------------------

app.post('/galeri', uploadGaleri.single('file'), (req, res) => {

    const { student\_id, text } = req.body;

    const file = req.file?.filename || null;

    const created\_at = new Date().toISOString();

    if (!student\_id) return res.status(400).json({ error: 'student\_id wajib diisi.' });

    db.run(`INSERT INTO galeri (student\_id, text, file, created\_at) VALUES (?, ?, ?, ?)`,

      [student\_id, text || '', file, created\_at],

      function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Konten galeri berhasil dikirim', id: this.lastID });

      });

  });

  app.get('/galeri', (req, res) => {

    db.all(`SELECT \* FROM galeri ORDER BY created\_at DESC`, [], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

  app.delete('/galeri/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM galeri WHERE id = ?`, [id], function (err) {

      if (err) return res.status(500).json({ error: err.message });

      res.json({ message: 'Konten galeri dihapus' });

    });

  });

// ------------------ Jalankan Server ------------------

app.listen(port, () => {

    console.log(`🚀 Server berjalan di http://localhost:${port}`);

});

Update

const express = require('express');

const sqlite3 = require('sqlite3').verbose();

const bcrypt = require('bcryptjs');

const cors = require('cors');

const path = require('path');

const multer = require('multer');

const upload = require('./config/multer');

const app = express();

const db = new sqlite3.Database('./registration.db');

const port = 3000;

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/uploads', express.static(path.join(\_\_dirname, 'public/uploads')));

// ------------------ Upload Config ------------------

const tugasStorage = multer.diskStorage({

    destination: (req, file, cb) => cb(null, 'public/uploads/tugas'),

    filename: (req, file, cb) => cb(null, 'tugas-' + Date.now() + path.extname(file.originalname))

});

const uploadTugas = multer({ storage: tugasStorage, limits: { fileSize: 10 \* 1024 \* 1024 }, fileFilter: upload.fileFilter });

const uploadJawaban = multer({

    storage: multer.diskStorage({

        destination: (req, file, cb) => cb(null, 'public/uploads/jawaban'),

        filename: (req, file, cb) => cb(null, 'jawaban-' + Date.now() + '-' + file.originalname)

    }),

    limits: { fileSize: 10 \* 1024 \* 1024 }

});

const galeriStorage = multer.diskStorage({

    destination: (req, file, cb) => cb(null, 'public/uploads/galeri'),

    filename: (req, file, cb) => cb(null, 'galeri-' + Date.now() + path.extname(file.originalname))

  });

  const uploadGaleri = multer({

    storage: galeriStorage,

    limits: { fileSize: 20 \* 1024 \* 1024 },

    fileFilter: (req, file, cb) => {

      const allowedTypes = ['image/jpeg', 'image/png', 'image/gif', 'video/mp4', 'video/quicktime'];

      if (allowedTypes.includes(file.mimetype)) {

        cb(null, true);

      } else {

        cb(new Error('Format file tidak didukung'), false);

      }

    }

  });

// ------------------ Autentikasi ------------------

app.post('/register', (req, res) => {

    const { name, class: className, studentId, email, password, role } = req.body;

    if (!name || !className || !studentId || !email || !password || !role) {

        return res.status(400).json({ error: 'Semua field harus diisi.' });

    }

    db.get('SELECT \* FROM users WHERE student\_id = ? OR email = ?', [studentId, email], (err, row) => {

        if (row) return res.status(400).json({ error: 'Email atau Student ID sudah digunakan.' });

        bcrypt.hash(password, 10, (err, hash) => {

            if (err) return res.status(500).json({ error: 'Gagal hash password' });

            db.run(`INSERT INTO users (name, class, student\_id, email, password, role) VALUES (?, ?, ?, ?, ?, ?)`,

                [name, className, studentId, email, hash, role],

                function (err) {

                    if (err) return res.status(400).json({ error: err.message });

                    res.json({ message: 'Register berhasil', userId: this.lastID });

                });

        });

    });

});

app.post('/login', (req, res) => {

    const { email, password } = req.body;

    db.get(`SELECT \* FROM users WHERE email = ?`, [email], (err, user) => {

        if (!user) return res.status(404).json({ error: 'User tidak ditemukan' });

        bcrypt.compare(password, user.password, (err, result) => {

            if (!result) return res.status(401).json({ error: 'Password salah' });

            res.json({ message: 'Login berhasil', user: { id: user.id, name: user.name, role: user.role, student\_id: user.student\_id } });

        });

    });

});

// ------------------ Manajemen Pengguna ------------------

app.get('/users', (req, res) => {

    db.all("SELECT id, name, class, student\_id, email, role FROM users", [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

// Ambil detail pengguna

app.get('/users/:id', (req, res) => {

    const id = req.params.id;

    db.get("SELECT id, name, class, student\_id, email, role FROM users WHERE id = ?", [id], (err, row) => {

        if (err) return res.status(500).json({ error: err.message });

        if (!row) return res.status(404).json({ error: 'Pengguna tidak ditemukan' });

        res.json(row);

    });

});

// Edit pengguna

app.put('/users/:id', (req, res) => {

    const { name, class: className, studentId, email, role } = req.body;

    const id = req.params.id;

    db.run(`UPDATE users SET name = ?, class = ?, student\_id = ?, email = ?, role = ? WHERE id = ?`,

        [name, className, studentId, email, role, id],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Pengguna berhasil diperbarui' });

        });

});

// Hapus pengguna

app.delete('/users/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM users WHERE id = ?`, [id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Pengguna berhasil dihapus' });

    });

});

// ------------------ Kursus ------------------

app.get('/courses', (req, res) => {

    db.all("SELECT \* FROM courses", [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

app.post('/courses', upload.array('files', 5), (req, res) => {

    const { name, description, instructor } = req.body;

    const files = req.files.map(file => file.filename);

    db.run(`INSERT INTO courses (name, description, instructor, files) VALUES (?, ?, ?, ?)`,

        [name, description, instructor, JSON.stringify(files)],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Kursus ditambahkan', courseId: this.lastID });

        });

});

app.put('/courses/:id', upload.array('files', 5), (req, res) => {

    const { name, description, instructor } = req.body;

    const files = req.files.map(file => file.filename);

    const id = req.params.id;

    db.run(`UPDATE courses SET name = ?, description = ?, instructor = ?, files = ? WHERE id = ?`,

        [name, description, instructor, JSON.stringify(files), id],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Kursus diperbarui' });

        });

});

app.delete('/courses/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM courses WHERE id = ?`, [id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Kursus berhasil dihapus' });

    });

});

//komentar kursus

app.post('/courses/:id/komentar', (req, res) => {

    const course\_id = req.params.id;

    const { student\_id, text } = req.body;

    const created\_at = new Date().toISOString();

    db.run(`INSERT INTO komentar\_kursus (course\_id, student\_id, text, created\_at) VALUES (?, ?, ?, ?)`,

      [course\_id, student\_id, text, created\_at],

      function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Komentar kursus berhasil ditambahkan', id: this.lastID });

      });

  });

  app.get('/courses/:id/komentar', (req, res) => {

    const course\_id = req.params.id;

    const sql = `

      SELECT komentar\_kursus.id, komentar\_kursus.text, komentar\_kursus.created\_at, users.name AS student\_name

      FROM komentar\_kursus

      JOIN users ON users.student\_id = komentar\_kursus.student\_id

      WHERE komentar\_kursus.course\_id = ?

      ORDER BY komentar\_kursus.created\_at ASC

    `;

    db.all(sql, [course\_id], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

  // hapus komentar

  app.delete('/courses/komentar/:id', (req, res) => {

    const komentarId = req.params.id;

    const { user\_role, student\_id } = req.query;

    const sql = `SELECT \* FROM komentar\_kursus WHERE id = ?`;

    db.get(sql, [komentarId], (err, komentar) => {

      if (err) return res.status(500).json({ error: err.message });

      if (!komentar) return res.status(404).json({ error: 'Komentar tidak ditemukan.' });

      if (user\_role !== 'admin' && komentar.student\_id !== student\_id) {

        return res.status(403).json({ error: 'Tidak diizinkan menghapus komentar ini.' });

      }

      db.run(`DELETE FROM komentar\_kursus WHERE id = ?`, [komentarId], (err) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Komentar berhasil dihapus.' });

      });

    });

  });

  // view kursus

  app.post('/courses/:id/view', (req, res) => {

    const course\_id = req.params.id;

    const { student\_id } = req.body;

    const viewed\_at = new Date().toISOString();

    if (!student\_id) return res.status(400).json({ error: 'student\_id wajib diisi.' });

    db.run(`INSERT OR IGNORE INTO course\_views (course\_id, student\_id, viewed\_at) VALUES (?, ?, ?)`,

      [course\_id, student\_id, viewed\_at],

      function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'View kursus tercatat' });

      });

  });

  app.get('/courses/:id/views', (req, res) => {

    const course\_id = req.params.id;

    db.all(`

      SELECT course\_views.\*, users.name

      FROM course\_views

      JOIN users ON users.student\_id = course\_views.student\_id

      WHERE course\_views.course\_id = ?

      ORDER BY course\_views.viewed\_at ASC

    `, [course\_id], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

// ------------------ Tugas ------------------

app.get('/tugas', (req, res) => {

    db.all("SELECT \* FROM tugas", [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

app.get('/tugas/:id', (req, res) => {

    const id = req.params.id;

    db.get("SELECT \* FROM tugas WHERE id = ?", [id], (err, row) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(row || {});

    });

});

app.post('/tugas', uploadTugas.single('file'), (req, res) => {

    const { title, description, deadline } = req.body;

    const file = req.file?.filename || null;

    db.run(`INSERT INTO tugas (title, description, deadline, file) VALUES (?, ?, ?, ?)`,

        [title, description, deadline, file],

        function (err) {

            if (err) return res.status(500).json({ error: err.message });

            res.json({ message: 'Tugas ditambahkan', id: this.lastID });

        });

});

app.put('/tugas/:id', uploadTugas.single('file'), (req, res) => {

    const id = req.params.id;

    const { title, description, deadline } = req.body;

    const file = req.file?.filename;

    const updateQuery = file

        ? `UPDATE tugas SET title = ?, description = ?, deadline = ?, file = ? WHERE id = ?`

        : `UPDATE tugas SET title = ?, description = ?, deadline = ? WHERE id = ?`;

    const params = file

        ? [title, description, deadline, file, id]

        : [title, description, deadline, id];

    db.run(updateQuery, params, function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Tugas diperbarui' });

    });

});

app.delete('/tugas/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM tugas WHERE id = ?`, [id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Tugas berhasil dihapus' });

    });

});

// komentar tugas

app.post('/tugas/:id/komentar', (req, res) => {

    const task\_id = req.params.id;

    const { student\_id, text } = req.body;

    const created\_at = new Date().toISOString();

    db.run(`INSERT INTO komentar\_tugas (task\_id, student\_id, text, created\_at) VALUES (?, ?, ?, ?)`,

      [task\_id, student\_id, text, created\_at],

      function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Komentar tugas berhasil ditambahkan', id: this.lastID });

      });

  });

  app.get('/tugas/:id/komentar', (req, res) => {

    const task\_id = req.params.id;

    const sql = `

      SELECT komentar\_tugas.id, komentar\_tugas.text, komentar\_tugas.created\_at, users.name AS student\_name

      FROM komentar\_tugas

      JOIN users ON users.student\_id = komentar\_tugas.student\_id

      WHERE komentar\_tugas.task\_id = ?

      ORDER BY komentar\_tugas.created\_at ASC

    `;

    db.all(sql, [task\_id], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

  // hapus komentar

  app.delete('/tugas/komentar/:id', (req, res) => {

    const komentarId = req.params.id;

    const { user\_role, student\_id } = req.query;

    const sql = `SELECT \* FROM komentar\_tugas WHERE id = ?`;

    db.get(sql, [komentarId], (err, komentar) => {

      if (err) return res.status(500).json({ error: err.message });

      if (!komentar) return res.status(404).json({ error: 'Komentar tidak ditemukan.' });

      if (user\_role !== 'admin' && komentar.student\_id !== student\_id) {

        return res.status(403).json({ error: 'Tidak diizinkan menghapus komentar ini.' });

      }

      db.run(`DELETE FROM komentar\_tugas WHERE id = ?`, [komentarId], (err) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Komentar berhasil dihapus.' });

      });

    });

  });

  // view tugas

  app.post('/tugas/:id/view', (req, res) => {

    const task\_id = req.params.id;

    const { student\_id } = req.body;

    const viewed\_at = new Date().toISOString();

    if (!student\_id) return res.status(400).json({ error: 'student\_id wajib diisi.' });

    db.run(`

        INSERT OR IGNORE INTO tugas\_views (task\_id, student\_id, viewed\_at)

        VALUES (?, ?, ?)

    `, [task\_id, student\_id, viewed\_at], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'View tercatat' });

    });

});

app.get('/tugas/:id/views', (req, res) => {

  const task\_id = req.params.id;

  db.all(`

      SELECT tugas\_views.\*, users.name

      FROM tugas\_views

      JOIN users ON users.student\_id = tugas\_views.student\_id

      WHERE tugas\_views.task\_id = ?

      ORDER BY tugas\_views.viewed\_at ASC

  `, [task\_id], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

  });

});

// ------------------ Jawaban ------------------

// Cek deadline sebelum insert jawaban

app.post('/jawaban', uploadJawaban.single('file'), (req, res) => {

    const { task\_id, student\_id, text } = req.body;

    const file = req.file?.filename || null;

    const created\_at = new Date().toISOString();

    db.get(`SELECT \* FROM tugas WHERE id = ?`, [task\_id], (err, tugas) => {

        if (err) return res.status(500).json({ error: err.message });

        if (tugas?.deadline && new Date(tugas.deadline) < new Date()) {

            return res.status(400).json({ error: 'Tugas sudah melewati deadline.' });

        }

        db.get(`SELECT \* FROM jawaban WHERE task\_id = ? AND student\_id = ?`, [task\_id, student\_id], (err, existing) => {

            if (existing) return res.status(400).json({ error: 'Jawaban sudah pernah dikirim.' });

            db.run(`INSERT INTO jawaban (task\_id, student\_id, text, file, created\_at) VALUES (?, ?, ?, ?, ?)`,

                [task\_id, student\_id, text, file, created\_at],

                function (err) {

                    if (err) return res.status(500).json({ error: err.message });

                    res.json({ message: 'Jawaban dikirim', id: this.lastID });

                });

        });

    });

});

// Edit jawaban (jika belum dinilai dan belum lewat deadline)

app.put('/jawaban/:id', uploadJawaban.single('file'), (req, res) => {

    const { text } = req.body;

    const file = req.file?.filename || null;

    const id = req.params.id;

    db.get(`SELECT \* FROM jawaban WHERE id = ?`, [id], (err, jawaban) => {

        if (!jawaban) return res.status(404).json({ error: 'Jawaban tidak ditemukan' });

        if (jawaban.nilai) return res.status(400).json({ error: 'Jawaban sudah dinilai, tidak bisa diedit' });

        // Cek deadline tugas

        db.get(`SELECT \* FROM tugas WHERE id = ?`, [jawaban.task\_id], (err, tugas) => {

            if (tugas?.deadline && new Date(tugas.deadline) < new Date()) {

                return res.status(400).json({ error: 'Tugas sudah melewati deadline, tidak bisa diedit.' });

            }

            const sql = file

                ? `UPDATE jawaban SET text = ?, file = ?, created\_at = ? WHERE id = ?`

                : `UPDATE jawaban SET text = ?, created\_at = ? WHERE id = ?`;

            const params = file

                ? [text, file, new Date().toISOString(), id]

                : [text, new Date().toISOString(), id];

            db.run(sql, params, function (err) {

                if (err) return res.status(500).json({ error: err.message });

                res.json({ message: 'Jawaban diperbarui' });

            });

        });

    });

});

app.get('/jawaban/cek/:task\_id/:student\_id', (req, res) => {

    const { task\_id, student\_id } = req.params;

    db.get(`SELECT \* FROM jawaban WHERE task\_id = ? AND student\_id = ?`, [task\_id, student\_id], (err, row) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(row || null);

    });

});

app.get('/jawaban', (req, res) => {

    db.all(`SELECT jawaban.\*, tugas.title AS task\_title FROM jawaban LEFT JOIN tugas ON jawaban.task\_id = tugas.id ORDER BY jawaban.created\_at DESC`, [], (err, rows) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json(rows);

    });

});

app.put('/jawaban/:id/nilai', (req, res) => {

    const { nilai, feedback } = req.body;

    const id = req.params.id;

    db.run(`UPDATE jawaban SET nilai = ?, feedback = ? WHERE id = ?`, [nilai, feedback, id], function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Nilai berhasil disimpan' });

    });

});

app.get('/laporan-nilai', (req, res) => {

    const sql = `

      SELECT

        users.name AS student\_name,

        users.student\_id,

        tugas.title AS task\_title,

        jawaban.nilai,

        jawaban.feedback,

        jawaban.created\_at

      FROM users

      CROSS JOIN tugas

      LEFT JOIN jawaban

        ON jawaban.task\_id = tugas.id

        AND CAST(jawaban.student\_id AS TEXT) = users.student\_id

      WHERE users.role = 'student'

      ORDER BY tugas.title, users.name;

    `;

    db.all(sql, [], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

// ------------------ Galeri Harian ------------------

app.post('/galeri', uploadGaleri.single('file'), (req, res) => {

    const { student\_id, text } = req.body;

    const file = req.file?.filename || null;

    const created\_at = new Date().toISOString();

    if (!student\_id) return res.status(400).json({ error: 'student\_id wajib diisi.' });

    db.run(`INSERT INTO galeri (student\_id, text, file, created\_at) VALUES (?, ?, ?, ?)`,

      [student\_id, text || '', file, created\_at],

      function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Konten galeri berhasil dikirim', id: this.lastID });

      });

  });

  app.get('/galeri', (req, res) => {

    db.all(`SELECT \* FROM galeri ORDER BY created\_at DESC`, [], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

  app.delete('/galeri/:id', (req, res) => {

    const id = req.params.id;

    db.run(`DELETE FROM galeri WHERE id = ?`, [id], function (err) {

      if (err) return res.status(500).json({ error: err.message });

      res.json({ message: 'Konten galeri dihapus' });

    });

  });

  // edit konten galeri //

  app.put('/galeri/:id', (req, res) => {

    const id = req.params.id;

    const { text } = req.body;

    db.run(`UPDATE galeri SET text = ? WHERE id = ?`, [text, id], function (err) {

      if (err) return res.status(500).json({ error: err.message });

      res.json({ message: 'Konten galeri berhasil diperbarui' });

    });

  });

  // komentar galeri //

  app.post('/galeri/:id/komentar', (req, res) => {

    const galeri\_id = req.params.id;

    const { student\_id, text } = req.body;

    const created\_at = new Date().toISOString();

    db.run(`INSERT INTO komentar (galeri\_id, student\_id, text, created\_at) VALUES (?, ?, ?, ?)`,

      [galeri\_id, student\_id, text, created\_at],

      function (err) {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Komentar berhasil ditambahkan', id: this.lastID });

      });

  });

  // hapus komentar

  app.delete('/galeri/komentar/:id', (req, res) => {

    const komentarId = req.params.id;

    const { user\_role, student\_id } = req.query;

    const sql = `SELECT \* FROM komentar WHERE id = ?`;

    db.get(sql, [komentarId], (err, komentar) => {

      if (err) return res.status(500).json({ error: err.message });

      if (!komentar) return res.status(404).json({ error: 'Komentar tidak ditemukan.' });

      // Cek izin

      if (user\_role !== 'admin' && komentar.student\_id !== student\_id) {

        return res.status(403).json({ error: 'Tidak diizinkan menghapus komentar ini.' });

      }

      db.run(`DELETE FROM komentar WHERE id = ?`, [komentarId], (err) => {

        if (err) return res.status(500).json({ error: err.message });

        res.json({ message: 'Komentar berhasil dihapus.' });

      });

    });

  });

// lihat komentar

app.get('/galeri/:id/komentar', (req, res) => {

    const galeri\_id = req.params.id;

    const sql = `

      SELECT komentar.id, komentar.text, komentar.created\_at, users.name AS student\_name

      FROM komentar

      JOIN users ON users.student\_id = komentar.student\_id

      WHERE komentar.galeri\_id = ?

      ORDER BY komentar.created\_at ASC

    `;

    db.all(sql, [galeri\_id], (err, rows) => {

      if (err) return res.status(500).json({ error: err.message });

      res.json(rows);

    });

  });

// ------------------ Jalankan Server ------------------

app.listen(port, () => {

    console.log(`🚀 Server berjalan di http://localhost:${port}`);

});

UPDATE DATA BASE

const sqlite3 = require('sqlite3').verbose();

const bcrypt = require('bcryptjs');

const db = new sqlite3.Database('./registration.db');

db.serialize(() => {

    // Hapus dan buat ulang tabel utama

    db.run(`DROP TABLE IF EXISTS courses`);

    db.run(`DROP TABLE IF EXISTS tugas`);

    db.run(`DROP TABLE IF EXISTS jawaban`);

    // Tabel kursus

    db.run(`

        CREATE TABLE IF NOT EXISTS courses (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            name TEXT NOT NULL,

            description TEXT NOT NULL,

            instructor TEXT NOT NULL,

            files TEXT

        );

    `);

    // Tabel pengguna

    db.run(`

        CREATE TABLE IF NOT EXISTS users (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            name TEXT NOT NULL,

            class TEXT NOT NULL,

            student\_id TEXT NOT NULL UNIQUE,

            email TEXT NOT NULL UNIQUE,

            password TEXT NOT NULL,

            role TEXT NOT NULL

        );

    `);

    // Tabel tugas

    db.run(`

        CREATE TABLE IF NOT EXISTS tugas (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            title TEXT NOT NULL,

            description TEXT NOT NULL,

            deadline TEXT,

            file TEXT

        );

    `);

    // Tabel jawaban siswa

    db.run(`

            CREATE TABLE IF NOT EXISTS jawaban (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            task\_id INTEGER,

            student\_id TEXT,

            text TEXT,

            file TEXT,

            created\_at TEXT,

            nilai TEXT,

            feedback TEXT

        );

    `);

    // tabel galeri

    db.run(`

        CREATE TABLE IF NOT EXISTS galeri (

          id INTEGER PRIMARY KEY AUTOINCREMENT,

          student\_id TEXT NOT NULL,

          text TEXT,

          file TEXT,

          created\_at TEXT NOT NULL

        );

      `);

      // table komentar

      db.run(`

        CREATE TABLE IF NOT EXISTS komentar (

          id INTEGER PRIMARY KEY AUTOINCREMENT,

          galeri\_id INTEGER NOT NULL,

          student\_id TEXT NOT NULL,

          text TEXT NOT NULL,

          created\_at TEXT NOT NULL,

          FOREIGN KEY (galeri\_id) REFERENCES galeri(id)

        );

      `);

      // komen kursus

      db.run(`

        CREATE TABLE IF NOT EXISTS komentar\_kursus (

          id INTEGER PRIMARY KEY AUTOINCREMENT,

          course\_id INTEGER NOT NULL,

          student\_id TEXT NOT NULL,

          text TEXT NOT NULL,

          created\_at TEXT NOT NULL,

          FOREIGN KEY (course\_id) REFERENCES courses(id)

        );

      `);

      // komen tugas

      db.run(`

        CREATE TABLE IF NOT EXISTS komentar\_tugas (

          id INTEGER PRIMARY KEY AUTOINCREMENT,

          task\_id INTEGER NOT NULL,

          student\_id TEXT NOT NULL,

          text TEXT NOT NULL,

          created\_at TEXT NOT NULL,

          FOREIGN KEY (task\_id) REFERENCES tugas(id)

        );

      `);

// taabel view tugas

      db.run(`

        CREATE TABLE IF NOT EXISTS tugas\_views (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        task\_id INTEGER NOT NULL,

        student\_id TEXT NOT NULL,

        viewed\_at TEXT NOT NULL,

        UNIQUE (task\_id, student\_id)

      );

      `);

    // tabel view kursus

    db.run(`CREATE TABLE IF NOT EXISTS course\_views (

      id INTEGER PRIMARY KEY AUTOINCREMENT,

      course\_id INTEGER NOT NULL,

      student\_id TEXT NOT NULL,

      viewed\_at TEXT NOT NULL,

      UNIQUE (course\_id, student\_id)

    );

    `);

    // Tambahkan user admin default

    const insertUser = (name, studentClass, studentId, email, password, role) => {

        db.get(`SELECT \* FROM users WHERE email = ? OR student\_id = ?`, [email, studentId], (err, row) => {

            if (row) {

                console.log(`User ${name} already exists.`);

                return;

            }

            bcrypt.hash(password, 10, (err, hash) => {

                if (err) return console.error(err);

                db.run(

                    `INSERT INTO users (name, class, student\_id, email, password, role) VALUES (?, ?, ?, ?, ?, ?)`,

                    [name, studentClass, studentId, email, hash, role],

                    (err) => {

                        if (err) console.error(err.message);

                        else console.log(`✅ User ${name} added.`);

                    }

                );

            });

        });

    };

    insertUser('Admin User', 'Admin', '0001', 'admin@example.com', 'admin123', 'admin');

});

db.close();