BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama menjalani masa magang di PT Braincode Digital Teknologi, posisi yang ditempati berada pada peran *Software Engineer*. Dalam pelaksanaannya, koordinasi dilakukan secara langsung dengan Mas Ramdani yang menjabat sebagai *Technical Lead* sekaligus menjadi *supervisor* selama kegiatan magang berlangsung. Posisi *Software Engineer* tersebut termasuk dalam divisi *Backend*.

a) **Nama:** Muhammad Tristan Ajibrilyan Nandipinto
Asal Kampus: Universitas Multimedia Nusantara
Jurusan: Informatika
Angkatan: 2022

b) **Nama:** Reinhard Javera Maheswara
Asal Kampus: Universitas Multimedia Nusantara
Jurusan: Informatika
Angkatan: 2022

c) **Nama:** Muhammad Ukasah Hayata, S.Si.
Asal Kampus: Universitas Pendidikan Indonesia
Jurusan: Sistem Informasi
Angkatan: 2020

d) **Nama:** Farhan Syarif Hidayatulloh
Asal Sekolah: SMK Wikrama Bogor
Jurusan: *Rekayasa Perangkat Lunak (RPL)*
Angkatan: 2023

Braincode LMS (Learning Management System) merupakan proyek internal dari PT Braincode Digital Teknologi yang dirancang untuk mendukung perkembangan pembelajaran karyawan perusahaan. Aplikasi ini dikembangkan guna meningkatkan pengetahuan dan wawasan karyawan, baik dalam bidang yang sedang mereka jalani maupun bidang yang diminati, sekaligus melakukan pelacakan terhadap perkembangan individu, kehadiran, serta tingkat kelayakan untuk promosi jabatan.

Tim pengembang dalam proyek ini terdiri dari Rizki Altino sebagai *Project Director*, Ramdani sebagai *Technical Leader* sekaligus *Supervisor*, Singgih Budi Hartono dan Reinhard Javera Maheswara sebagai *Frontend Developer*, serta Arya Zafri dan Muhammad Tristan Ajibrilyan Nandipinto sebagai *Backend Developer*. Singgih Budi Hartono dan Arya Zafri merupakan karyawan tetap PT Braincode Digital Teknologi, sementara Reinhard Javera Maheswara dan Muhammad Tristan Ajibrilyan Nandipinto merupakan peserta magang.

Proyek ini dikembangkan menggunakan metodologi *Agile* dengan pendekatan *Sprint* mingguan, yang dievaluasi setiap hari Jumat untuk meninjau kemajuan serta perencanaan tugas selanjutnya. Gitlab digunakan untuk membantu proses perkembangan proyek dan me-*review* perkembangan *Sprint* setiap minggunya, serta sebagai sarana untuk memantau *daily task* yang diberikan pada setiap *Sprint* di proyek LMS.

3.2 Tugas yang Dilakukan

Selama menjalani kegiatan magang di PT Braincode Digital Teknologi, beberapa tugas utama yang dilaksanakan meliputi aspek perancangan basis data, pengembangan *API*, hingga proses *deployment* layanan. Adapun uraian tugas tersebut adalah sebagai berikut:

a) Merancang struktur tabel dan membangun relasi antar tabel pada basis data *PostgreSQL*[2] dengan memanfaatkan antarmuka *DBeaver*.

b) Mengembangkan *service* dan *endpoint REST API (Representational State Transfer Application Programming Interface)*[3] menggunakan *framework Express.js*[4], bahasa pemrograman *TypeScript*, serta *Sequelize* sebagai *ORM (Object-Relational Mapping)*[5] untuk pengelolaan basis data *PostgreSQL*.

c) Membuat *script* Python untuk melakukan *feeding* data GitLab ke *database PostgreSQL* LMS, dan meng-*import* data absensi bulanan ke *database PostgreSQL* LMS.

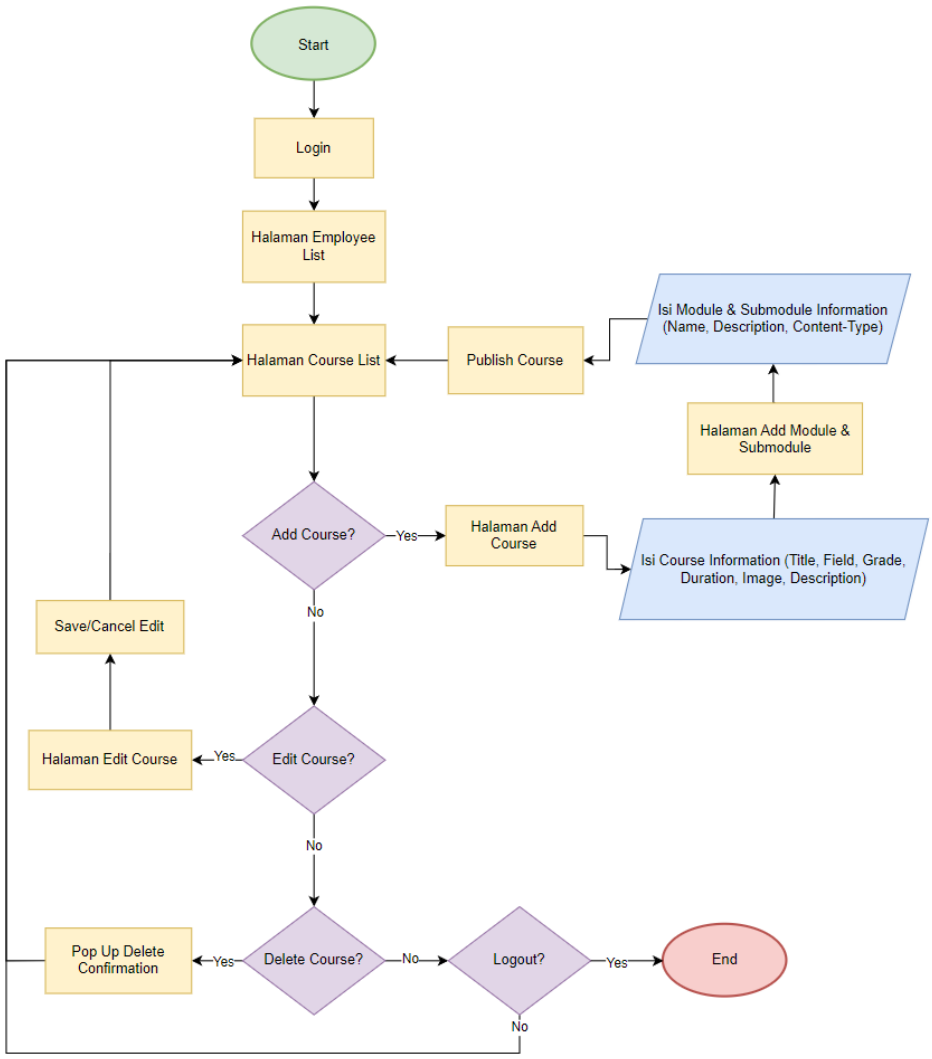
d) Melakukan pengoperasian *terminal Linux* untuk keperluan *deployment backend service* dan *API endpoint* agar dapat diakses serta dimanfaatkan oleh tim *frontend*.

Proyek utama yang dikerjakan selama kegiatan magang di PT Braincode Digital Teknologi sebagai *Backend Software Developer* adalah pembangunan sistem *Learning Management System* (LMS) berbasis web, yang diberi nama *Braincode LMS*. Sistem ini ditujukan untuk digunakan oleh seluruh karyawan internal PT Braincode Digital Teknologi sebagai sarana manajemen pembelajaran dan pemantauan kinerja.

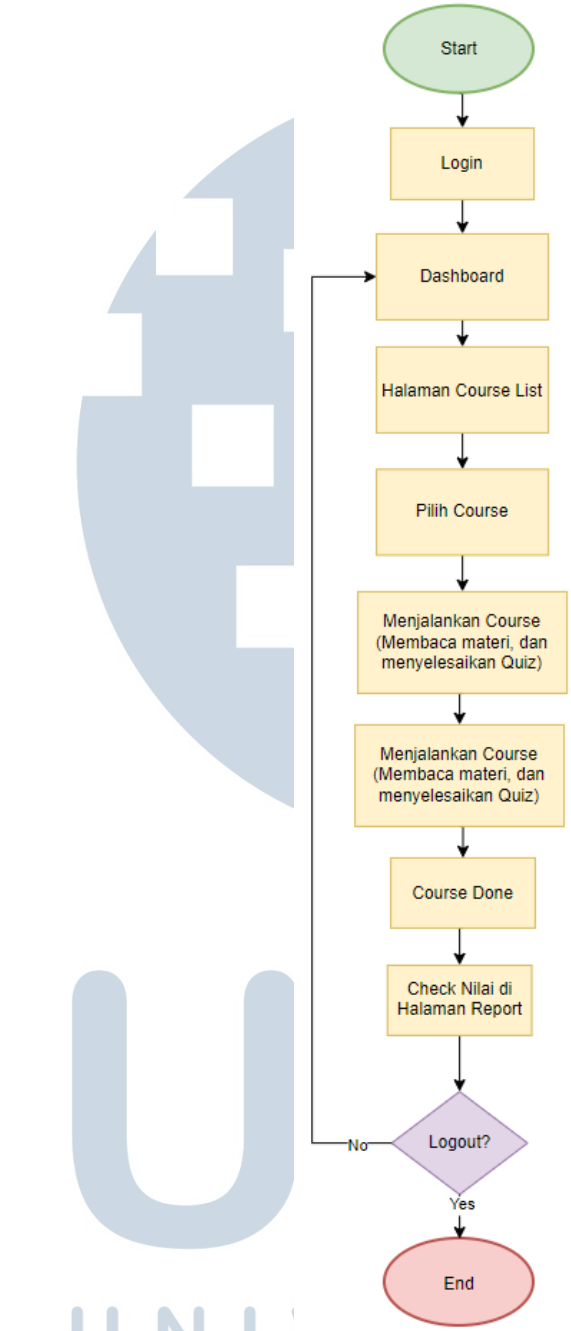
Terdapat dua jenis peran (*role*) pengguna dalam sistem ini, yaitu *admin* dan *employee*.

a) ***Admin***: Role ini ditujukan bagi pengguna dengan jabatan manajerial seperti *Project Manager*, *Project Leader*, dan *Technical Lead*. Fungsinya untuk memantau perkembangan pembelajaran serta performa kerja dari para karyawan. Fitur utama yang disediakan bagi admin meliputi manajemen data karyawan secara menyeluruh (*CRUD user*), pemantauan statistik pembelajaran baik secara global maupun individual, serta manajemen konten *course* seperti modul, submodul, kuis, pertanyaan, dan *attachment* berupa berkas PDF, teks, atau video.

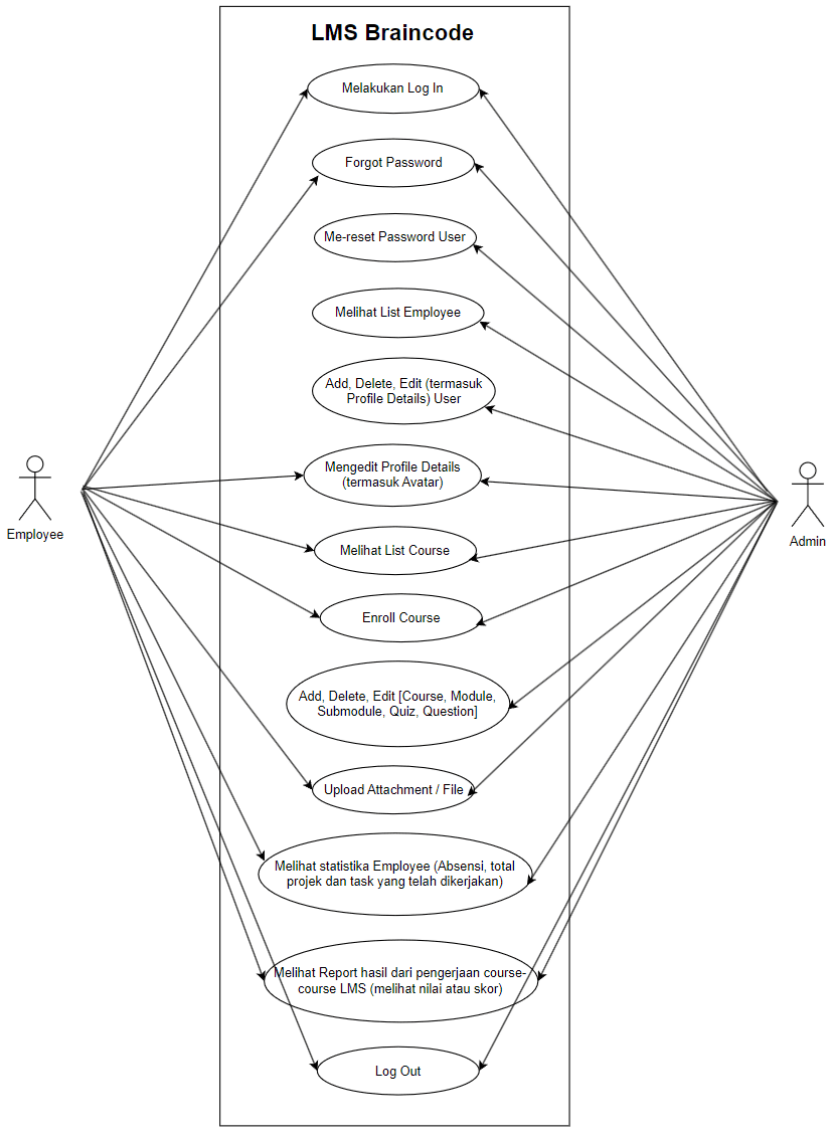
b) ***Employee***: Role ini ditujukan bagi karyawan biasa. Fitur yang tersedia mencakup pendaftaran dan penyelesaian *course*, akses terhadap materi pembelajaran, serta pengerjaan kuis sebagai syarat kelulusan. Statistik pembelajaran dan performa juga ditampilkan untuk setiap karyawan. Selain itu, sistem tidak membatasi akses karyawan hanya pada bidang yang sedang dijalani. Seorang karyawan dapat mengambil *course* dari bidang lain sesuai minatnya. Sebagai contoh, seorang *Frontend Developer* dapat mengikuti *course* mengenai pengembangan *backend*, begitu pula sebaliknya.



Gambar 3.1. *Flow Admin Course Management* Braincode LMS



Gambar 3.2. *Flow Course Enroll* Braincode LMS



Gambar 3.3. *Use Case* Braincode LMS

Untuk mendukung pengembangan aplikasi ini, perusahaan menyediakan dua buah *server*:

1. **Private Server (Internal)**: Digunakan untuk keperluan *development* dan *testing* yang hanya dapat diakses melalui jaringan lokal kantor.
2. **Virtual Private Server (VPS)**: Digunakan sebagai *proxy server*[6] yang menghubungkan server internal dengan domain publik `braincodetech.id`. Server ini memungkinkan aplikasi diakses dari luar jaringan perusahaan untuk keperluan demonstrasi atau uji akses, namun tidak digunakan sebagai server *production*.

Berikut adalah daftar *API* yang dirancang dan dikembangkan sesuai kebutuhan sistem LMS Braincode:

1. **API User:** membuat pengguna, menghapus pengguna, memperbarui pengguna, menampilkan detail pengguna, serta manajemen foto profil.
2. **API Autentikasi:** login berdasarkan *role*, *logout*, *refresh token*, *reset password* (khusus admin), dan *forgot password* (khusus employee).
3. **API Course:** penambahan *course*, penghapusan *course*, pengeditan *course*, serta daftar *course*.
4. **API Module:** penambahan *module*, penghapusan *module*, pengeditan *module*, serta daftar *module*.
5. **API Submodule:** penambahan *submodule*, penghapusan *submodule*, pengeditan *submodule*, serta daftar *submodulesubmodule*.
6. **API Quiz:** penambahan *quiz*, penghapusan *quiz*, pengeditan *quiz*, serta daftar *quiz*.
7. **API Question:** manajemen pertanyaan pada kuis (tambah, hapus, ubah, tampilkan).
8. **API Attachment:** unggah berkas secara bertahap (*chunked*), dan ambil lampiran.
9. **API Statistics:** penyediaan ringkasan laporan pengguna, persentase dan total kehadiran absen (seperti telat dan *ontime*, statistik proyek dan tugas, serta integrasi data dari GitLab.

3.3 Uraian Pelaksanaan Magang

Selama pelaksanaan magang, adapun tugas-tugas yang telah dikerjakan seperti yang dapat dilihat pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Mempelajari ClickHouse DB (Query, Tipe Data, dll.)

Minggu Ke -	Pekerjaan yang dilakukan
2	a) Mempelajari dan mencoba *Rust programming language*. b) Mempelajari *PostgreSQL database*. c) Mempelajari perangkat lunak *DBeaver* sebagai platform untuk *database management*.
3	a) Mempelajari dan mencoba membuat API dengan Rust menggunakan data di PostgreSQL. b) Mempelajari cara deploy service (API) agar dapat diakses oleh front-end.
4	a) Memasuki proyek internal (LMS Braincode). b) Membuat tabel *user* di database LMS menggunakan PostgreSQL. c) Setup project menggunakan Node.js TypeScript dengan Express.js. d) Membuat API login.
5	a) Menerapkan enkripsi data user menggunakan Blowfish, bcrypt untuk password. b) Membuat API GET *Employee_details*. c) Membuat tabel *field, course, module, submodule, tracking, user_course, quiz, report*. d) Membuat berbagai API service (*GET/POST*) untuk course.

Minggu Ke -	Pekerjaan yang dilakukan
6	a) Membuat API terkait module dan submodule course. b) Mengisi data dummy dan memperbaiki beberapa request body. c) Memperbaiki API yang berkaitan dengan *user_course*.
7	a) Memperbaiki API course dan field, hapus tabel join *course_field*. b) Menambahkan kolom *contract_start, contract_end*. c) Membuat API Reset Password, Edit Profile, dan Avatar.
8	a) Menyesuaikan struktur tabel *user* dengan kolom id baru. b) Menyesuaikan Sequelize model dan seluruh API terkait. c) Membuat API *users/is_unique*, dan memperbaiki bug di Edit Profile dan image error.
9	a) Membuat tabel *resetpw_history*, API */forgot_password*, */refresh_token*, dan */reencrypt*. b) Membuat tabel *encrypt_history*. c) Memperbaiki timezone bug pada forgot password.

Minggu Ke -	Pekerjaan yang dilakukan
10	a) Membuat tabel dan API terkait *attachment_meta, attachment_chunk*. b) Membuat API untuk CMS: *add, delete course, metadata*. c) Membuat tabel *question, answer*.
11	a) Menyesuaikan struktur tabel course dan membuat relasi many-to-many. b) Membuat API *list, detail, edit* untuk CMS course/module/submodule/quiz/question. c) Menambahkan kolom *created_time, last_updated*.
12	a) Membuat API */course/enroll*. b) Penyesuaian kolom *order_number* di beberapa tabel. c) Membuat API reordering dan answer quiz.
13	a) Fix perhitungan *total_page* di */cms/list*. b) Sinkronisasi *last_updated*. c) Membuat tabel *git_raw_commits, git_raw_issues*.
14	a) Import data commit dan issue dari file zip dan json. b) Buat script Python untuk otomasi import ke PostgreSQL. c) Eksekusi feeding data ke database.

Minggu Ke -	Pekerjaan yang dilakukan
15	a) Modifikasi script feeding-issues.py dengan filter waktu. b) Buat API statistik user dan enrollment. c) Membuat tabel *attendance*, dan script impor data excel ke PostgreSQL.
16	a) Membuat endpoint GET */attendance/:start_date/:end_date*. b) Modifikasi script csv importer untuk memilih file terbaru. c) Jalankan otomatisasi dengan *crontab*. d) Membuat endpoint GET */task_delivered/:start_date/:end_date*.

Minggu Ke -	Pekerjaan yang dilakukan
17	a) Membuat endpoint GET `/project_enrolled`. b) Membuat API statistik profil dengan endpoint `/statistics/profile_dashboard`. c) Membuat API statistik course enrolled dengan endpoint `/statistics/course_enrolled`. d) Menyesuaikan API `/task_delivered/:start_date/:end_date` untuk role Admin. e) Mengubah endpoint menjadi `/task_delivered/{granularity}/{date}/{date}` dan menangani error jika format granularity dan date tidak sesuai. f) Mendesain skema filter data GitLab dari tabel `git_raw_issues` dan `git_raw_commits`.

Minggu Ke -	Pekerjaan yang dilakukan
18	a) Membuat tabel `git_task` dan `git_project`. b) Memperbaiki regex bug pada endpoint `task_delivered`. c) Menambahkan kolom `iid_task` di tabel `git_task`, memperbarui model TypeScript, dan menyesuaikan API Git sync. d) Menambahkan skrip Python ke dalam `crontab` untuk otomatisasi impor data dari GitLab. e) Menambahkan kolom `status` pada tabel `attendance` dan menyesuaikan API terkait. f) Menambahkan parameter granularity (`daily, weekly, monthly`) pada API attendance. g) Menyesuaikan endpoint `GET /attendance/{date}/{date}` dan `/project_enrolled` untuk Admin. h) Membuat tabel `score_list` untuk mendukung statistik nilai di dashboard.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Selama masa menjalani kerja magang di PT Braincode Digital Teknologi ini, kurang-lebih tugas yang dikerjakan adalah manajemen database, pembuatan API dan *testing*, serta deployment *service* ke *server*.

3.3.1 Infrastruktur Backend Database

Dalam pembangunan sistem *Learning Management System* (LMS) berbasis web yang dikembangkan di PT Braincode Digital Teknologi, basis data memegang peranan sentral sebagai fondasi penyimpanan dan pengelolaan data. Sistem ini mengandalkan **PostgreSQL** sebagai *Relational Database Management System* (RDBMS)[7] utama yang digunakan dalam seluruh proses pengembangan backend.

Pemilihan PostgreSQL didasarkan pada sejumlah keunggulan yang relevan dengan kebutuhan sistem, antara lain:

a) **Kepatuhan terhadap standar SQL dan dukungan relasional yang kuat:** PostgreSQL secara konsisten mendukung fitur relasional tingkat lanjut, seperti *foreign key*, *join* antar tabel, *view*, dan *transaction*, yang sangat penting dalam menjaga integritas data antar entitas seperti course, modul, kuis, hingga hasil statistik pengguna.

b) **Dukungan skema (*schema*) sebagai *namespace*:** PostgreSQL memungkinkan penggunaan skema untuk mengelompokkan tabel berdasarkan fungsi atau lingkup tertentu dalam satu basis data. Fitur ini meningkatkan keteraturan sistem serta memudahkan pemisahan antara data utama, data pengujian, maupun data log atau audit.

c) **Open-source dan komunitas yang aktif:** PostgreSQL bersifat gratis, bebas digunakan, dan memiliki ekosistem komunitas yang besar sehingga mudah dalam hal *troubleshooting* dan pengembangan lebih lanjut.

d) **Kemampuan ekspansi dan kustomisasi:** PostgreSQL mendukung ekstensi seperti `pgcrypto`, `uuid-ossp`, serta dapat diintegrasikan dengan berbagai ORM seperti Sequelize, yang digunakan dalam proyek ini untuk interaksi basis data menggunakan TypeScript.

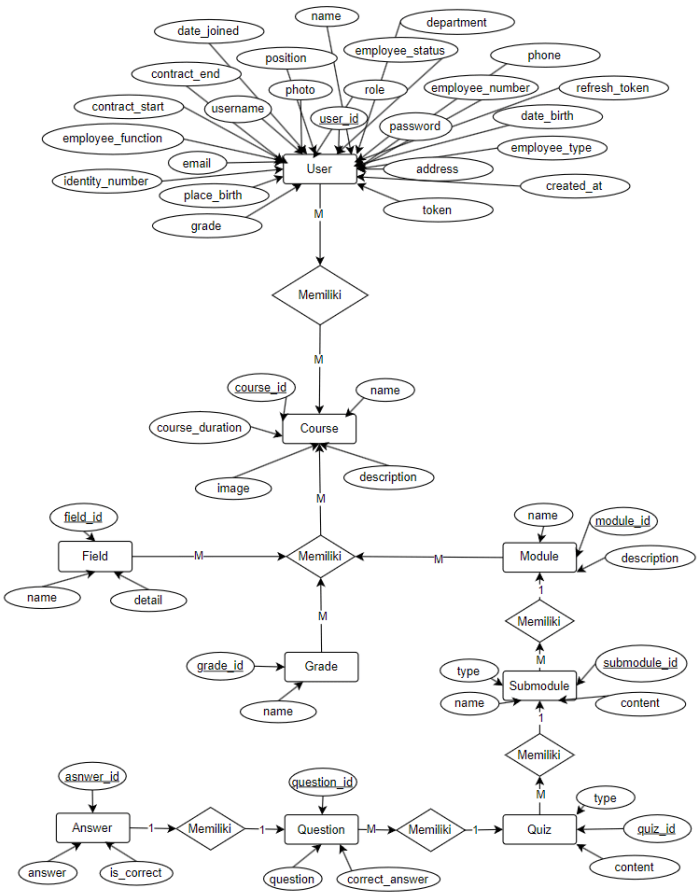
e) **Stabilitas dan performa tinggi:** PostgreSQL dikenal sebagai salah satu RDBMS paling stabil dan dapat menangani transaksi dalam skala besar, yang sangat penting untuk sistem LMS dengan potensi data pengguna yang bertumbuh.

Struktur dan Relasi Database LMS

Struktur basis data dalam sistem LMS dirancang secara modular dan relasional untuk mendukung skenario penggunaan yang kompleks, mulai dari pembelajaran mandiri hingga pemantauan kinerja oleh admin. Tabel-tabel utama yang digunakan dalam basis data PostgreSQL LMS antara lain:

Tabel 3.2. Struktur Tabel Utama dalam Database LMS

No	Nama Tabel	Fungsi
1	`user`	Menyimpan data pengguna, termasuk peran (*role*), identitas, dan foto.
2	`field`	Menyimpan bidang keahlian atau departemen yang tersedia.
3	`grade`	Menyimpan tingkatan karyawan, seperti junior, middle, atau senior.
4	`course`	Menyimpan informasi course utama, termasuk relasi ke `field` dan `grade`.
5	`module`	Menyimpan modul dalam suatu course, terkait dengan `course`, `field`, dan `grade`.
6	`submodule`	Menyimpan submodul sebagai bagian dari suatu `module`.
7	`quiz`	Menyimpan informasi kuis yang terkait dengan suatu `module`.
8	`question`	Menyimpan pertanyaan dalam kuis.
9	`answer`	Menyimpan jawaban untuk masing-masing pertanyaan dan data pengguna terkait.



Gambar 3.4. ERD LMS Braincode

Setiap tabel di atas memiliki *foreign key* yang menghubungkan data satu sama lain untuk menjaga integritas dan konsistensi. Misalnya:

a) Tabel `course` memiliki `field_id` dan `grade_id` sebagai relasi ke tabel `field` dan `grade`.

b) Tabel `module` merujuk pada `course_id`, `field_id`, dan `grade_id`.

c) Tabel `submodule` memiliki `module_id` sebagai penghubung ke modul induknya.

d) Tabel `quiz` memiliki `module_id`, dan `question` memiliki `quiz_id`.

e) Tabel `answer` memiliki `question_id` serta `user_id` untuk mencatat jawaban pengguna.

Seluruh skema dan tabel tersebut dikembangkan menggunakan antarmuka grafis **DBeaver**, yang mempermudah visualisasi relasi dan penyesuaian struktur saat proses pengembangan berlangsung. Integrasi dengan Sequelize sebagai ORM memastikan bahwa setiap perubahan pada model backend dapat disinkronkan secara otomatis ke dalam struktur basis data PostgreSQL.

Dengan memanfaatkan PostgreSQL dan desain basis data yang relasional serta modular, sistem LMS Braincode mampu menyajikan data secara efisien, fleksibel, dan konsisten untuk memenuhi kebutuhan baik pengguna karyawan maupun admin.

3.3.2 Infrastruktur Backend API

Perancangan *backend API* pada website *LMS* (*Learning Management System*) ini menggunakan *tech stack Node.js*, *Express.js*, *TypeScript*, *Sequelize*, dan *PostgreSQL*. Untuk *backend service* digunakan *framework Express.js* yang berjalan di atas *Node.js*, dengan *TypeScript* sebagai bahasa pemrograman utama. *Sequelize* digunakan sebagai *ORM (Object-Relational Mapping)* untuk mengelola interaksi antara aplikasi dan basis data *PostgreSQL*.

Berikut merupakan spesifikasi dari infrastruktur maupun *framework* yang digunakan untuk merancang *backend* website *LMS*:

a) *Node.js* versi v22.11.0

b) *Express.js* versi 4.21.2

c) *TypeScript* versi 5.7.3

d) *Sequelize* versi 6.37.5

e) *PostgreSQL* versi 17.2

f) *OS Ubuntu Linux* versi 22.04 64-bit (atau dapat berjalan di *Windows 10/11*)

Pada *backend Express.js*, proses dimulai ketika pengguna mengirimkan permintaan (*request*) ke *API*. Permintaan ini diterima oleh sistem *routing Express.js*, yang menentukan *controller* mana yang harus menangani permintaan tersebut berdasarkan *URL* dan metode *HTTP* yang digunakan. *Controller* kemudian memproses logika bisnis yang diperlukan, seperti validasi data, autentikasi dan otorisasi menggunakan *JWT*, serta interaksi dengan basis data melalui *Sequelize ORM* untuk mengambil, menyimpan, atau memodifikasi data.

Setelah semua logika diproses, *controller* akan mengembalikan respons yang sesuai, seperti data dalam format *JSON* kepada pengguna (*client*). Dengan arsitektur ini, *backend API* dapat dengan mudah diintegrasikan dengan *frontend* berbasis web maupun aplikasi *mobile*.

3.3.3 Pembuatan Model dan API User

Pada tahap ini, dilakukan perancangan model *User* menggunakan *Sequelize* sebagai *ORM* yang terintegrasi dengan basis data *PostgreSQL*. Model *User* mencakup atribut-atribut seperti `user_id`, `username`, `email`, `password`, `role`, `photo`, `name`, `employee_number`, `employee_status`, `position`, `phone`, dan atribut personal lainnya. Model ini juga dihubungkan dengan model lain seperti *Role*, *Position*, *Department*, dan *Grade* melalui relasi *foreign key*.

Setelah model selesai dibuat, dilakukan pengembangan berbagai *endpoint* (*API*) untuk kebutuhan manajemen pengguna. Berikut adalah daftar *API* yang diimplementasikan beserta penjelasan singkat dan jenis pengujian yang digunakan:

Tabel 3.3. Daftar API *User*

Method	Endpoint	Deskripsi Singkat	Jenis Pengujian
POST	`/add`	Menambah *user* baru	*Whitebox* & *Blackbox*
GET	`/get_users`	Mengambil seluruh data *user*	*Blackbox*
GET	`/get_user/:id`	Mengambil detail *user* berdasarkan *ID*	*Blackbox*
DELETE	`/delete_user/:id`	Menghapus *user* berdasarkan *ID*	*Whitebox* & *Blackbox*
POST	`/get_user/:id`	Mengedit data atau profil *user* berdasarkan *ID*	*Whitebox* & *Blackbox*
GET	`/profile`	Mengambil profil *user* yang sedang login	*Blackbox*
POST	`/profile`	Mengedit profil *user* yang sedang login	*Whitebox* & *Blackbox*
GET	`/avatar`	Mengambil *avatar user* yang sedang login	*Blackbox*
GET	`/avatar/:user_id`	Mengambil *avatar user* berdasarkan *user_id*	*Blackbox*
PATCH	`/avatar`	Mengubah *avatar user* yang sedang login	*Whitebox* & *Blackbox*
PATCH	`/avatar/:user_id`	Mengubah *avatar user* berdasarkan *user_id*	*Whitebox* & *Blackbox*
GET	`/is_unique`	Mengecek keunikan data *user* (*username*, *email*, dll)	*Blackbox*

Penjelasan Jenis Pengujian

Whitebox Testing

Pengujian dilakukan dengan mengetahui struktur internal kode, logika, dan alur program. Pada *API* seperti `/add`, `/get_user/:id` (`POST`), `/delete_user/:id`, `/profile` (`POST`), `/avatar` (`PATCH`), dan `/avatar/:user_id` (`PATCH`), pengujian *whitebox* dilakukan untuk memastikan validasi *input*, proses enkripsi/dekripsi, serta integrasi dengan basis data berjalan sesuai logika yang diharapkan.

Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan *input* dan *output* yang dihasilkan. *API* seperti `/get_users`, `/get_user/:id` (`GET`), `/profile` (`GET`), `/avatar` (`GET`), `/avatar/:user_id` (`GET`), dan `/is_unique` diuji dengan memberikan berbagai skenario *input* dan memeriksa apakah *output*/*response* sesuai ekspektasi.

API `POST /add`

Endpoint `POST /add` digunakan untuk menambahkan *user* baru ke dalam sistem. Pada endpoint ini, *controller* yang digunakan adalah `createUser` yang terdapat pada berkas `userController.ts`. *Controller* ini bertanggung jawab untuk menerima data *user* baru dari *request body*, melakukan validasi terhadap seluruh *field* yang dikirimkan (seperti `username`, `email`, `password`, `role`, dan atribut lain yang diperlukan), serta memastikan bahwa tidak ada data yang duplikat di basis data, seperti `username`, `email`, nomor induk karyawan, nomor identitas, dan nomor telepon.

```
const {
  username, password, email, role, photo, name,
  employee_number, employee_status, position, department,
  employee_function, phone, grade, identity_number,
  address, date_birth, employee_type, place_birth,
  contract_start, contract_end, date_joined
} = req.body;

if (!username || !password || !email || !role) {
  return res.status(400).json({ error: "Missing required fields" });
}
```

Gambar 3.5. *req body* untuk API Add User

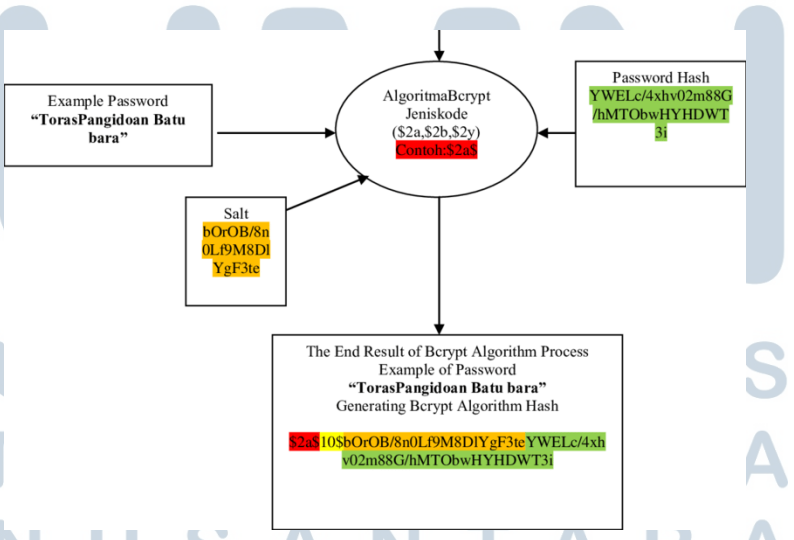
```
// 🔐 Encrypt function
const encrypt = (text: string): string => {
  return Buffer.from(bf.encode(text)).toString("base64");
};

// 🔓 Decrypt function
const decrypt = (encryptedText: string): string => {
  return bf.decode(Buffer.from(encryptedText, "base64"), Blowfish.TYPE.STRING).replace(/\0/g, "");
};
```

Gambar 3.6. Fungsi enkripsi dan dekripsi

```
if (existingUser) {
  if (decrypt(existingUser.username).toLowerCase() === username.toLowerCase()) {
    errors.push("Username already exists");
  }
  if (decrypt(existingUser.email).toLowerCase() === email.toLowerCase()) {
    errors.push("Email already exists");
  }
  if (decrypt(existingUser.employee_number).toLowerCase() === employee_number.toLowerCase()) {
    errors.push("Employee Number already exists");
  }
  if (decrypt(existingUser.identity_number).toLowerCase() === identity_number.toLowerCase()) {
    errors.push("Identity Number already exists");
  }
  if (decrypt(existingUser.phone).toLowerCase() === phone.toLowerCase()) {
    errors.push("Phone number already exists");
  }
}

if (errors.length > 0) {
  return res.status(400).json({ error: "Duplicate data found", details: errors });
}
```

Gambar 3.7. Penerapan *existingUsers* untuk Cek Keunikan Data

Setelah validasi berhasil, *controller* akan melakukan enkripsi pada data sensitif menggunakan Blowfish[8] dan melakukan *hashing*[9] pada `password` dengan `bcrypt`[10] dari library *bcryptjs* sebelum data disimpan ke dalam basis data.



Gambar 3.8. Algoritma *hashing* menggunakan *bcrypt*

```
const hashedPassword = await bcrypt.hash(password, 10);

const encryptedUser = {
  username: encrypt(username),
  password: hashedPassword,
  email: encrypt(email),
  role_id: roleData.role_id,
  photo: encrypt(photo),
  name: encrypt(name),
  employee_number: encrypt(employee_number),
  employee_status: encrypt(employee_status),
  position_id: positionData?.position_id || null,
  department_id: departmentData?.department_id || null,
  employee_function_id: employeeFunctionData?.employee_function_id || null,
  phone: encrypt(phone),
  grade_id: gradeData?.grade_id || null,
  identity_number: encrypt(identity_number),
  date_birth: encrypt(date_birth),
  place_birth: encrypt(place_birth),
  employee_type_id: employeeTypeData?.employee_type_id || null,
  address: encrypt(address),
  contract_start: encrypt(contract_start),
  contract_end: encrypt(contract_end),
  date_joined: encrypt(date_joined),
  created_at: new Date(),
};
```

Gambar 3.9. Enkripsi Data User Baru menggunakan Enkripsi *Blowfish* dan *Hashing password* menggunakan *bcrypt*

Selain itu, *controller* juga menangani pembuatan atau pengambilan data relasi seperti `role`, `department`, `position`, `employee function`, `employee type`, dan `grade` menggunakan fungsi bantu `getOrCreateInsensitive`.

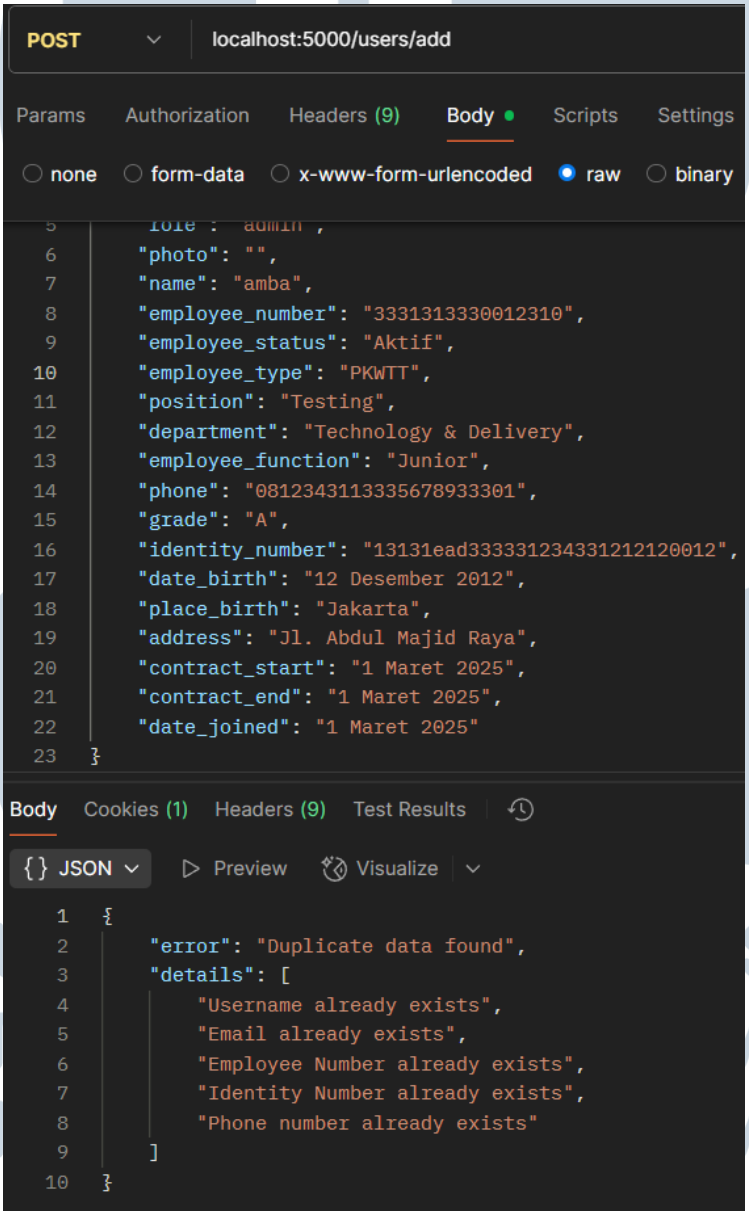
```
export const getOrCreateInsensitive = async (model: any, column: string,
  value: string, additionalDefaults: Record<string, any> = {}) => {
  const allEntries = await model.findAll();

  for (let entry of allEntries) {
    const decrypted = decrypt(entry[column]);
    if (decrypted.toLowerCase() === value.toLowerCase()) {
      return [entry, false]; // ditemukan
    }
  }

  const encryptedValue = encrypt(value);
      You, 2 months ago • update bug createuser (department_id null di
  const [newEntry] = await model.findOrCreate({
    where: { [column]: encryptedValue },
    defaults: {
      [column]: encryptedValue,
      ...additionalDefaults,
    },
  });

  return [newEntry, true];
};
```

Gambar 3.10. Fungsi *getOrCreateInsensitive*

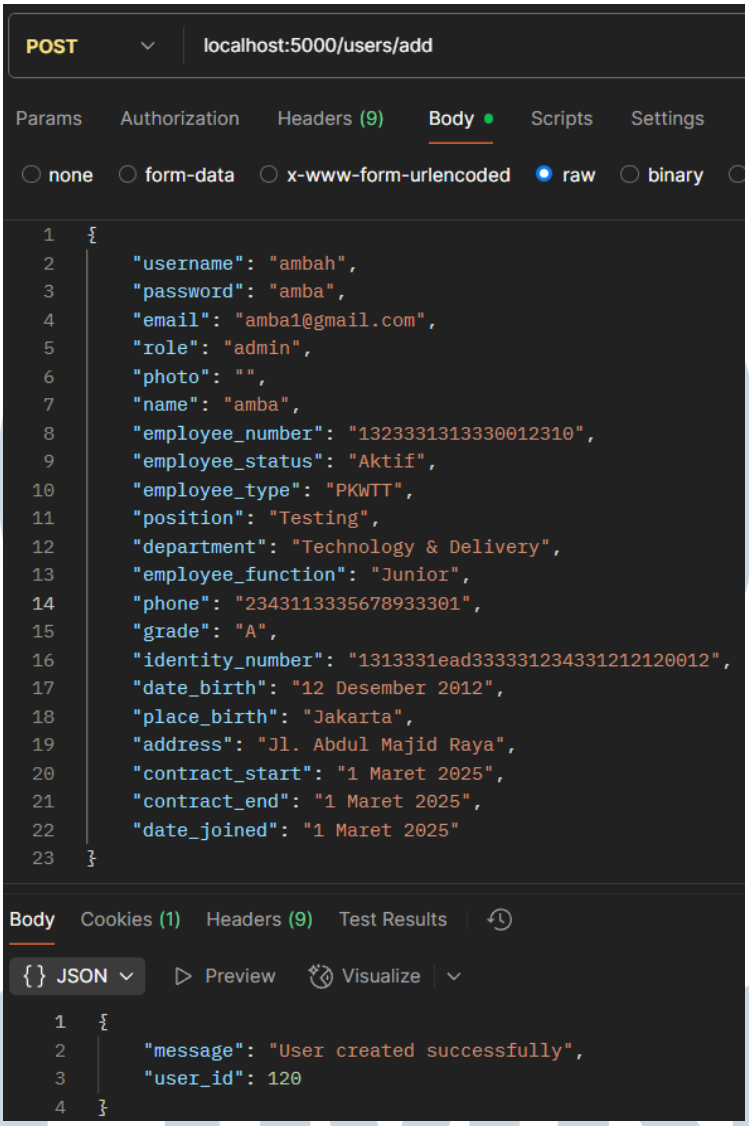
```
const [roleData] = await getOrCreateInsensitive(Role, "role_name", role);
const [departmentData] = department ? await getOrCreateInsensitive(Department, "department_name", department) : [null];
const [positionData] = position ? await getOrCreateInsensitive(Position, "position_name", position,
  departmentData ? { department_id: departmentData.department_id } : {}) : [null];
if (positionData && !positionData.department_id && departmentData) {
  await positionData.update({ department_id: departmentData.department_id });
}
const [employeeFunctionData] = employee_function ? await getOrCreateInsensitive(Employee_function, "employee_function_level", employee_function) : [null];
const [employeeTypeData] = employee_type ? await getOrCreateInsensitive(Employee_type, "employee_type", employee_type) : [null];
const [gradeData] = grade ? await getOrCreateInsensitive(Grade, "grade", grade) : [null];
```

Gambar 3.11. Penerapan Fungsi *getOrCreateInsensitive*

Jika seluruh proses berhasil, maka data *user* baru akan disimpan melalui model Sequelize, dan *response* sukses akan dikirimkan ke *client*. Apabila terjadi kesalahan, seperti data duplikat atau *field* wajib tidak diisi, maka *controller* akan mengembalikan *response error* beserta pesan yang sesuai. Dengan demikian, endpoint ini memastikan bahwa proses pendaftaran *user* baru berjalan secara aman, terstruktur, dan terintegrasi dengan baik ke seluruh sistem *backend*.



Gambar 3.12. Error Duplikat *Add User*



Gambar 3.13. Add User Berhasil

API `GET /get_users`

Endpoint `GET /get_users` digunakan untuk mengambil seluruh data *user* yang terdaftar di dalam sistem. Endpoint ini biasanya diakses oleh *admin* atau *user* dengan hak akses tertentu untuk keperluan manajemen data *user* seperti menampilkan daftar *employee*. Pada endpoint ini, *controller* yang digunakan adalah `getUsers` yang terdapat pada berkas `userController.ts`.

Controller `getUsers` bertugas untuk mengambil data semua *user* dari basis data menggunakan model Sequelize. Data yang diambil biasanya mencakup informasi penting seperti `user_id`, `username`, `email`, `nama`, `role`, `status`, dan

atribut lain yang relevan. *Controller* juga dapat melakukan *join* dengan tabel lain seperti `Role`, `Department`, atau `Position` untuk menampilkan informasi yang lebih lengkap pada setiap *user*.

Setelah data berhasil diambil, *controller* akan mengembalikan *response* berupa array data *user* dalam format JSON kepada *client*. Jika terjadi kesalahan saat pengambilan data, seperti masalah koneksi basis data atau *error query*, maka *controller* akan mengembalikan *response error* beserta pesan yang sesuai. Endpoint ini sangat penting untuk kebutuhan *monitoring*, administrasi, dan pengelolaan *user* secara menyeluruh di sistem *backend*.


```
{
    "error": "Forbidden: Admins only"
}
```


Gambar 3.14. Pesan Error *Admins Only*


```
[
    {
        "user_id": 95,
        "employee_name": "Ade Esarrendy Intris",
        "email": "ade.esarrendy@braincodesolution.com",
        "nik": "3301222408030002",
        "work_type": "PKWT",
        "job_position": "Software Developer",
        "date_joined": "26 Mei 2023",
        "status": "Aktif"
    },
    {
        "user_id": 1,
        "employee_name": "Admin Testing",
        "email": "admin123@gmail.com",
        "nik": "3312340909990009",
        "work_type": "PKWT",
        "job_position": "Software Developer",
        "date_joined": "1 Maret 2025",
        "status": "Aktif"
    },
    {
        "user_id": 86,
        "employee_name": "Affan Abdillah",
```

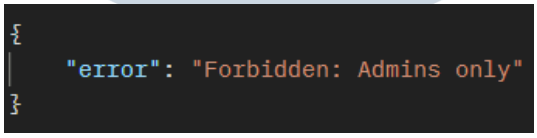

Gambar 3.15. API getUsers

API `GET /get_user/:id`

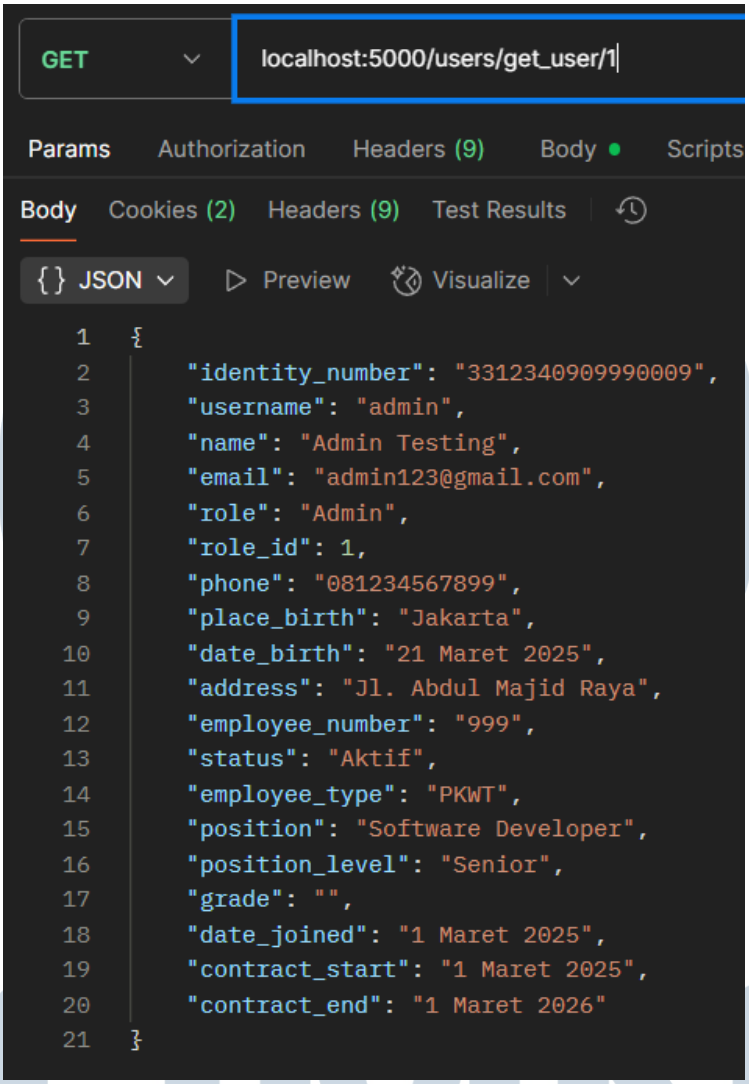
Endpoint `GET /get_user/:id` digunakan untuk mengambil detail data *user* berdasarkan *user ID* tertentu. Endpoint ini biasanya digunakan ketika *admin* atau *user* ingin melihat informasi lengkap dari satu *user* secara spesifik. Pada endpoint ini, *controller* yang digunakan adalah `getUserById` yang terdapat pada berkas `userController.ts`.

Controller `getUserById` akan menerima parameter `id` dari URL, kemudian melakukan pencarian data *user* di basis data menggunakan model Sequelize berdasarkan `user_id` tersebut. Data yang diambil meliputi informasi lengkap *user* seperti `username`, `email`, `nama`, `role`, `status`, serta atribut lain yang relevan.

Jika *user* dengan `ID` tersebut ditemukan, *controller* akan mengembalikan *response* berupa data *user* dalam format JSON. Namun, jika *user* tidak ditemukan atau terjadi kesalahan saat pengambilan data, *controller* akan mengembalikan *response error* beserta pesan yang sesuai. Endpoint ini sangat berguna untuk menampilkan detail profil *user* pada halaman *admin* atau saat proses verifikasi data *user* di sistem.



Gambar 3.16. Pesan Error *Admins Only*



Gambar 3.17. API getUserById

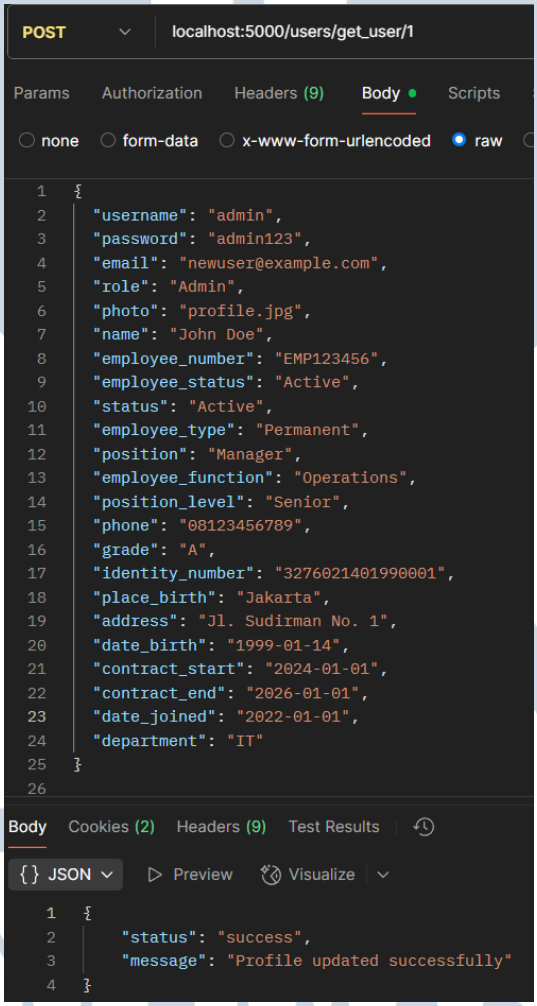
API `POST /get_user/:id`

Endpoint `POST /get_user/:id` digunakan untuk mengedit atau memperbarui data *user* berdasarkan *user ID* tertentu. Endpoint ini biasanya diakses oleh *admin* atau *user* dengan hak akses khusus untuk melakukan perubahan data pada *user* lain. Pada endpoint ini, *controller* yang digunakan adalah `updateUserById` yang terdapat pada berkas `userController.ts`.

Controller `updateUserById` akan menerima parameter `id` dari URL dan data baru yang akan diperbarui melalui *request body*. *Controller* ini akan melakukan validasi terhadap data yang dikirim, seperti memastikan format `email` yang benar, `username` yang unik, serta *field* lain yang sesuai dengan aturan sistem. Setelah

validasi berhasil, *controller* akan memperbarui data *user* di basis data menggunakan model Sequelize.

Jika proses *update* berhasil, *controller* akan mengembalikan *response sukses* beserta data *user* yang telah diperbarui. Namun, jika terjadi kesalahan seperti data tidak valid, *user* tidak ditemukan, atau terjadi konflik data (misalnya: `email` sudah digunakan *user* lain), maka *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini sangat penting untuk kebutuhan administrasi dan pengelolaan data *user* secara dinamis di dalam sistem.



Gambar 3.18. API editProfileById

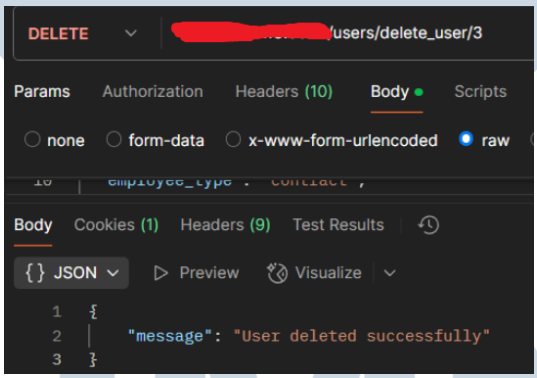
API `DELETE /delete_user/:id`

Endpoint `DELETE /delete_user/:id` digunakan untuk menghapus data *user* berdasarkan *user ID* tertentu dari sistem. Endpoint ini umumnya hanya

dapat diakses oleh *admin* atau *user* dengan hak akses khusus untuk menjaga keamanan dan integritas data. Pada endpoint ini, *controller* yang digunakan adalah `deleteUserById` yang terdapat pada berkas `userController.ts`.

Controller `deleteUserById` akan menerima parameter `id` dari URL, kemudian melakukan pencarian *user* di basis data menggunakan model Sequelize. Jika *user* dengan ID tersebut ditemukan, *controller* akan menghapus data *user* dari basis data. Selain itu, *controller* juga dapat menangani penghapusan data terkait lain yang berelasi dengan *user* tersebut, jika diperlukan (misalnya, penghapusan *cascading* pada data absensi, tugas, atau relasi lainnya).

Setelah proses penghapusan berhasil, *controller* akan mengembalikan *response sukses* kepada *client*. Namun, jika *user* tidak ditemukan atau terjadi kesalahan saat proses penghapusan, *controller* akan mengembalikan *response error* beserta pesan yang sesuai. Endpoint ini sangat penting untuk menjaga kebersihan dan keamanan data *user* di dalam sistem.



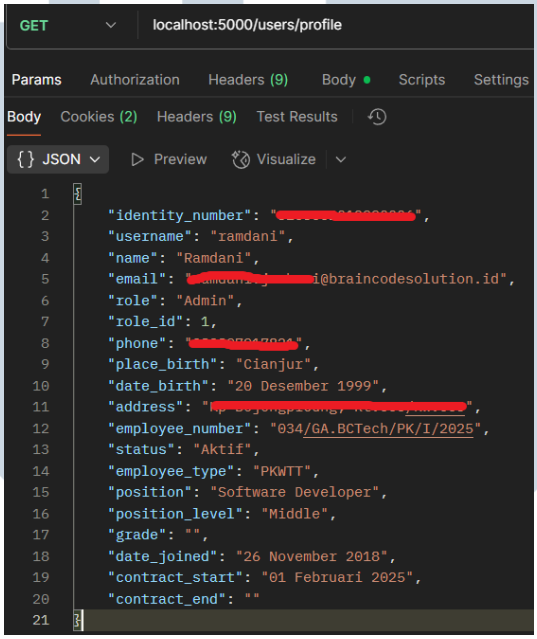
Gambar 3.19. API deleteUserById

API `GET /profile`

Endpoint `GET /profile` digunakan untuk mengambil data profil *user* yang sedang login ke dalam sistem. Endpoint ini biasanya diakses oleh *user* setelah berhasil melakukan autentikasi untuk melihat informasi pribadi mereka sendiri. Pada endpoint ini, *controller* yang digunakan adalah `getProfile` yang terdapat pada berkas `userController.ts`.

Controller `getProfile` akan mengambil informasi *user* yang sedang login berdasarkan data autentikasi (biasanya dari *token JWT* yang dikirimkan pada *request*). *Controller* kemudian mencari data *user* di basis data menggunakan model Sequelize, dan mengambil informasi lengkap seperti *username*, *email*, *nama*, *role*, *status*, serta atribut personal lainnya.

Jika data *user* ditemukan, *controller* akan mengembalikan *response* berupa data profil *user* dalam format JSON. Jika terjadi kesalahan, seperti *token* tidak valid atau *user* tidak ditemukan, *controller* akan mengembalikan *response error* beserta pesan yang sesuai. Endpoint ini sangat penting untuk memberikan transparansi dan kemudahan bagi *user* dalam mengelola serta memeriksa data pribadinya di sistem.



Gambar 3.20. API getProfile

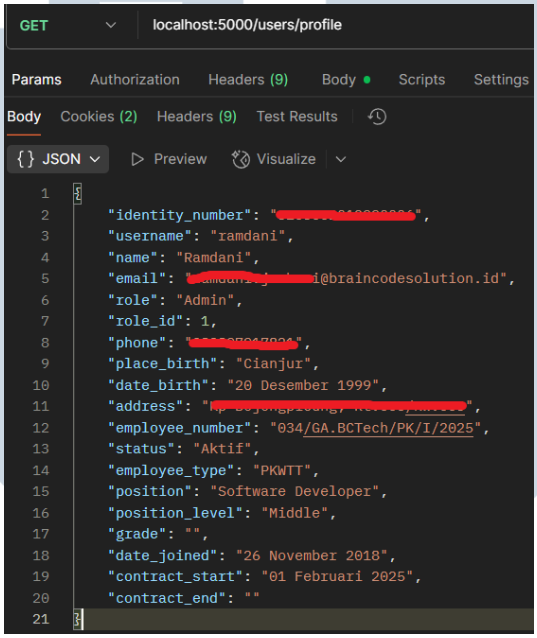
API `POST /profile`

Endpoint `POST /profile` digunakan untuk memperbarui atau mengedit data profil *user* yang sedang login. Endpoint ini biasanya diakses oleh *user* untuk mengubah informasi pribadi mereka sendiri, seperti *nama*, *email*, *nomor telepon*, atau data lain yang diizinkan untuk diubah. Pada endpoint ini, *controller* yang digunakan adalah `updateProfile` yang terdapat pada berkas `userController.ts`.

Controller `updateProfile` akan menerima data baru yang akan diperbarui melalui *request body*. *Controller* ini akan melakukan validasi terhadap data yang dikirim, seperti memastikan format *email* yang benar, *nomor telepon* valid, serta memastikan bahwa data yang diubah tidak menyebabkan duplikasi (misalnya *email* sudah digunakan *user* lain).

Setelah validasi berhasil, *controller* akan memperbarui data *user* di basis data menggunakan model Sequelize. Jika proses *update* berhasil, *controller* akan mengembalikan *response sukses* beserta data profil *user* yang telah diperbarui.

Namun, jika terjadi kesalahan seperti data tidak valid, *user* tidak ditemukan, atau terjadi konflik data, maka *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini sangat penting untuk memberikan fleksibilitas kepada *user* dalam mengelola dan memperbarui data pribadinya secara mandiri di dalam sistem.



Gambar 3.21. API updateProfile

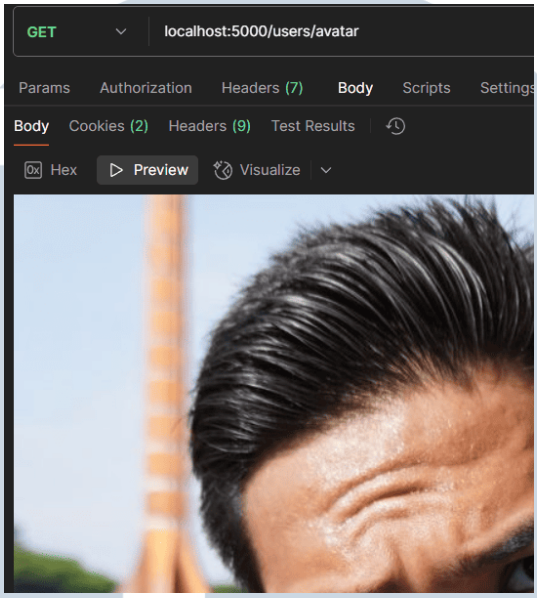
API `GET /avatar`

Endpoint `GET /avatar` digunakan untuk mengambil data avatar (foto profil) dari *user* yang sedang login ke dalam sistem. Endpoint ini biasanya diakses oleh *user* setelah login untuk menampilkan foto profil mereka pada aplikasi atau website. Pada endpoint ini, *controller* yang digunakan adalah `getAvatar` yang terdapat pada berkas `userController.ts`.

Controller `getAvatar` akan mengambil informasi *user* yang sedang login berdasarkan data autentikasi (biasanya dari token JWT). Setelah itu, *controller* mencari data avatar *user* di basis data menggunakan model Sequelize. Jika *user* memiliki avatar, *controller* akan mengembalikan *response* berupa file gambar, URL, atau data avatar dalam format yang sesuai (misal: `base64` atau *link*). Jika *user* belum mengunggah avatar, *controller* dapat mengembalikan avatar default atau *response* kosong.

Jika terjadi kesalahan, seperti token tidak valid atau *user* tidak ditemukan,

controller akan mengembalikan *response error* beserta pesan yang sesuai. Endpoint ini penting untuk mendukung fitur personalisasi tampilan profil *user* di aplikasi.



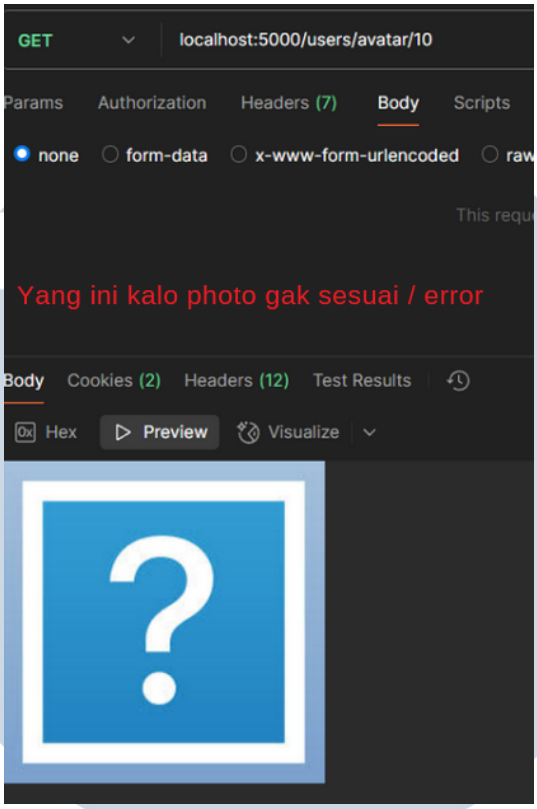
Gambar 3.22. API getAvatar

API `GET /avatar/:user_id`

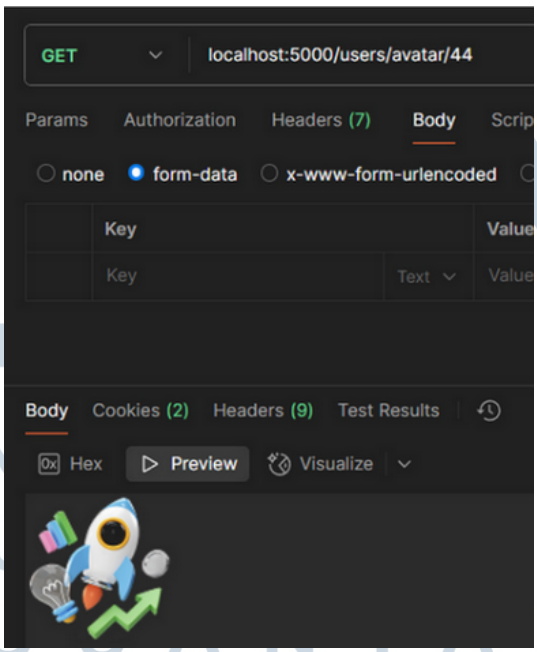
Endpoint `GET /avatar/:user_id` digunakan untuk mengambil data avatar (foto profil) dari *user* lain berdasarkan `user_id` tertentu. Endpoint ini biasanya diakses oleh *admin* atau *user* lain yang ingin menampilkan foto profil *user* tertentu, misalnya pada halaman daftar *user* atau detail *user*. Pada endpoint ini, *controller* yang digunakan adalah `getAvatarById` yang terdapat pada berkas `userController.ts`.

Controller `getAvatarById` akan menerima parameter `user_id` dari URL, kemudian mencari data avatar *user* tersebut di basis data menggunakan model Sequelize. Jika *user* dengan ID tersebut memiliki avatar, *controller* akan mengembalikan *response* berupa file gambar, URL, atau data avatar dalam format yang sesuai. Jika *user* belum mengunggah avatar, *controller* dapat mengembalikan avatar default atau *response* kosong.

Jika *user* tidak ditemukan atau terjadi kesalahan saat pengambilan data, *controller* akan mengembalikan *response error* beserta pesan yang sesuai. Endpoint ini sangat berguna untuk menampilkan foto profil *user* lain pada berbagai fitur di aplikasi, seperti daftar anggota tim atau detail *user*.



Gambar 3.23. Avatar Error atau Not Found



Gambar 3.24. API getAvatarById

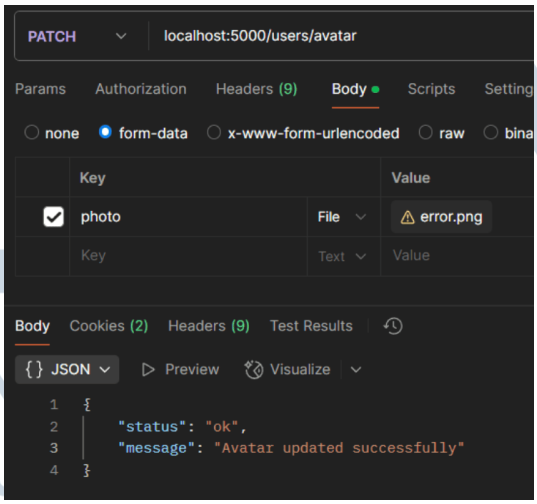
API `PATCH /avatar`

Endpoint `PATCH /avatar` digunakan untuk mengubah atau memperbarui avatar (foto profil) *user* yang sedang login. Endpoint ini biasanya diakses oleh *user* ketika ingin mengganti foto profil mereka sendiri melalui aplikasi atau website. Pada endpoint ini, *controller* yang digunakan adalah `updateAvatar` yang terdapat pada berkas `userController.ts`.

Controller `updateAvatar` akan menerima file gambar baru yang diunggah oleh *user* melalui *request* (biasanya menggunakan `form-data` atau `multipart upload`). *Controller* kemudian melakukan validasi terhadap file yang diunggah, seperti memastikan tipe file adalah gambar (JPEG, PNG, dan sebagainya) serta ukuran file tidak melebihi batas yang ditentukan.

Jika validasi berhasil, *controller* akan menyimpan file gambar ke *storage* (lokal, server, atau cloud) dan memperbarui data avatar *user* di basis data menggunakan model Sequelize. Jika proses *update* berhasil, *controller* akan mengembalikan *response sukses* beserta informasi avatar yang baru.

Namun, jika terjadi kesalahan seperti file tidak valid, *user* tidak ditemukan, atau error saat penyimpanan file, maka *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini sangat penting untuk memberikan fleksibilitas kepada *user* dalam memperbarui tampilan profil mereka di sistem.



Gambar 3.25. API updateAvatar

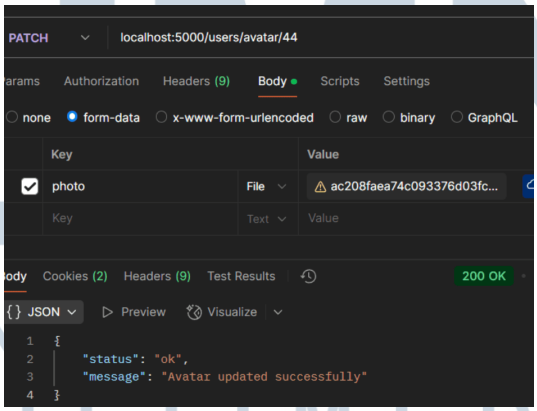
API `PATCH /avatar/:user_id`

Endpoint `PATCH /avatar/:user_id` digunakan untuk mengubah atau memperbarui avatar (foto profil) *user* lain berdasarkan `user_id` tertentu. Endpoint ini biasanya hanya dapat diakses oleh *admin* atau *user* dengan hak akses khusus, misalnya untuk membantu *user* yang mengalami kendala dalam mengganti foto profilnya. Pada endpoint ini, *controller* yang digunakan adalah `updateAvatarById` yang terdapat pada berkas `userController.ts`.

Controller `updateAvatarById` akan menerima parameter `user_id` dari URL dan file gambar baru yang diunggah melalui *request* (biasanya menggunakan `form-data` atau `multipart upload`). *Controller* melakukan validasi terhadap file yang diunggah, seperti memastikan tipe file adalah gambar dan ukuran file sesuai ketentuan.

Setelah validasi berhasil, *controller* akan menyimpan file gambar ke *storage* dan memperbarui data avatar *user* yang bersangkutan di basis data menggunakan model Sequelize. Jika proses *update* berhasil, *controller* akan mengembalikan *response sukses* beserta informasi avatar yang baru.

Namun, jika terjadi kesalahan seperti file tidak valid, *user* tidak ditemukan, atau error saat penyimpanan file, maka *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini sangat penting untuk mendukung kebutuhan administrasi dan membantu *user* dalam pengelolaan profil di sistem.



Gambar 3.26. API updateAvatarById

API `GET /is_unique`

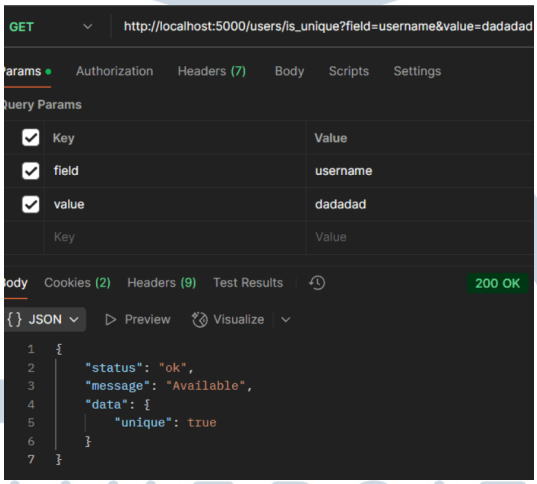
Endpoint `GET /is_unique` digunakan untuk memeriksa keunikan data tertentu pada *user*, seperti `username`, `email`, `nomor induk karyawan`, atau atribut

lain yang harus unik di dalam sistem. Endpoint ini biasanya digunakan pada proses pendaftaran atau pengeditan data *user* untuk memastikan bahwa data yang dimasukkan tidak duplikat dengan data *user* lain di basis data. Pada endpoint ini, *controller* yang digunakan adalah `checkIsUnique` yang terdapat pada berkas `userController.ts`.

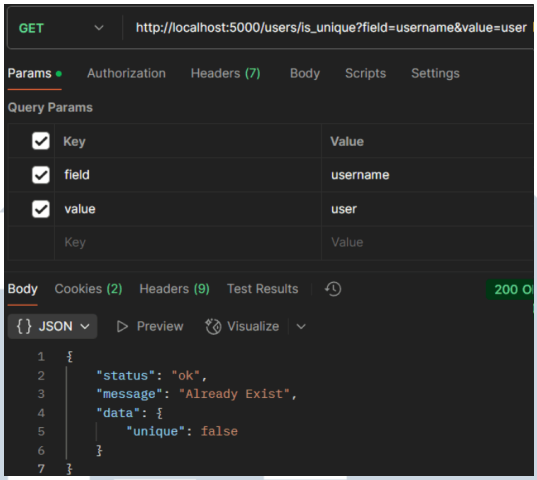
Controller `checkIsUnique` akan menerima parameter pencarian (misalnya `username` atau `email`) melalui *query string* pada URL. *Controller* kemudian melakukan pencarian di basis data menggunakan model Sequelize untuk memastikan apakah data tersebut sudah ada atau belum.

Jika data yang dicek belum ada di basis data, *controller* akan mengembalikan *response* yang menyatakan data tersebut unik (`unique: true`). Sebaliknya, jika data sudah ada, *controller* akan mengembalikan *response* bahwa data tersebut tidak unik (`unique: false`).

Jika terjadi kesalahan, seperti parameter tidak lengkap atau *error* saat *query* ke basis data, *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini sangat penting untuk menjaga integritas data dan mencegah terjadinya duplikasi pada data *user* di sistem.



Gambar 3.27. API is_unique *True*



Gambar 3.28. API is_unique *False*

Tabel 3.4. Tabel Pengujian API *User* - *Whitebox*

No	Endpoint	Skenario Pengujian	Status
1	`POST /add`	Validasi field wajib, format email/password, cek duplikasi, enkripsi, *foreign key*	Lulus
4	`POST /get_user/:id`	Validasi field update, format data, update DB, tidak ubah *primary key*	Lulus
5	`DELETE /delete_user/:id`	Cek *user* ada, hapus dari DB, cek *cascading data*	Lulus
7	`POST /profile`	Validasi field update, update DB	Lulus
10	`PATCH /avatar`	Validasi file upload, simpan file, update *avatar*	Lulus
11	`PATCH /avatar/:user_id`	Validasi file, update *avatar user* lain (admin)	Lulus

Tabel 3.5. Tabel Pengujian API *User* - *Blackbox*

No	Endpoint	Skenario Pengujian	Status
1	`POST /add`	Input valid, email tidak valid, username/email duplikat, field wajib kosong	Lulus
2	`GET /get_users`	Ambil semua *user*, database kosong, cek format response	Lulus
3	`GET /get_user/:id`	Ambil *user* ID valid, ID tidak ada, cek format response	Lulus
4	`POST /get_user/:id`	Update valid, email/username duplikat, field tidak valid, *user* tidak ada	Lulus
5	`DELETE /delete_user/:id`	Hapus *user* ID valid, *user* tidak ada	Lulus
6	`GET /profile`	Ambil profil *user* login, token tidak valid/expired	Lulus
7	`POST /profile`	Update valid, data tidak valid, tanpa login	Lulus
8	`GET /avatar`	Ambil *avatar user* login, *user* tidak punya *avatar*	Lulus
9	`GET /avatar/:user_id`	Ambil *avatar user* lain ID valid, *user* tidak ada	Lulus
10	`PATCH /avatar`	Upload valid, file bukan gambar, tanpa login	Lulus
11	`PATCH /avatar/:user_id`	Admin update *avatar user* lain, *user* biasa update *avatar user* lain	Lulus
12	`GET /is_unique`	Cek username/email belum ada, sudah ada, input kosong	Lulus

3.3.4 Pembuatan Model dan API Autentikasi

Pada tahap ini, dilakukan perancangan model dan pengembangan berbagai endpoint (API) yang berkaitan dengan proses autentikasi dan keamanan user di sistem. Model autentikasi umumnya berhubungan dengan model *User*, serta pengelolaan token autentikasi (seperti JWT) dan proses enkripsi/dekripsi data sensitif. Seluruh proses autentikasi diimplementasikan menggunakan *Express.js*, *TypeScript*, dan *Sequelize*, sehingga terintegrasi dengan baik ke dalam sistem *backend*.

Setelah model dan *middleware* autentikasi selesai dibuat, dilakukan pengembangan berbagai endpoint untuk kebutuhan login, logout, *refresh token*, *reset password*, serta proses lupa password. Berikut adalah daftar API autentikasi yang diimplementasikan beserta penjelasan singkat dan jenis pengujian yang digunakan:

Tabel 3.6. Daftar API Autentikasi

Method	Endpoint	Deskripsi Singkat	Jenis Pengujian
POST	`/login`	Login *user*, *generate token*	*Whitebox* & *Blackbox*
POST	`/refresh_token`	Memperbarui *JWT token*	*Whitebox* & *Blackbox*
POST	`/logout`	*Logout user*, hapus token	*Whitebox* & *Blackbox*
POST	`/reset_password`	Ganti *password* user yang login	*Whitebox* & *Blackbox*
POST	`/forgot_password`	Request *reset password* (kirim kode/email)	*Whitebox* & *Blackbox*
POST	`/forgot_password/reset`	*Reset password* dengan kode	*Whitebox* & *Blackbox*
POST	`/forgot_password/check`	Verifikasi kode *reset password*	*Whitebox* & *Blackbox*

Penjelasan Jenis Pengujian

Whitebox Testing

Pengujian dilakukan dengan mengetahui struktur internal kode, logika autentikasi, proses enkripsi/dekripsi, validasi token, serta integrasi dengan database. Pada API seperti `/login`, `/refresh_token`, `/logout`, `/reset_password`, `/forgot_password`, `/forgot_password/reset`, `/forgot_password/check`, dan `/reencrypt`, pengujian whitebox dilakukan untuk memastikan seluruh proses autentikasi, validasi, dan keamanan berjalan sesuai logika yang diharapkan.

Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan. Pada seluruh endpoint autentikasi, dilakukan pengujian dengan berbagai skenario input (data valid, data tidak valid, token kedaluwarsa, kode salah, dan sebagainya) dan memeriksa apakah *response* yang dihasilkan sesuai ekspektasi.

API `POST /login`

Endpoint `POST /login` digunakan untuk melakukan proses autentikasi *user* ke dalam sistem. Endpoint ini menerima data login berupa `username` atau `email` dan `password` melalui *request body*.

Pada endpoint ini, *controller* yang digunakan adalah `loginUser` yang terdapat pada berkas `authController.ts`. *Controller* akan melakukan validasi terhadap data yang dikirimkan, kemudian mencari *user* yang sesuai di basis data menggunakan model Sequelize.

Jika *user* ditemukan, *controller* akan memverifikasi kecocokan `password` dengan membandingkan *hash password* yang tersimpan di basis data menggunakan pustaka `bcrypt`. Jika proses autentikasi berhasil, sistem akan menghasilkan *token*

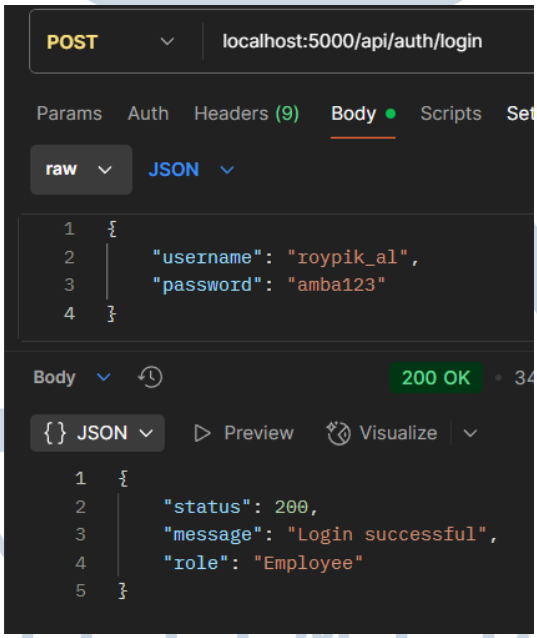
autentikasi (JWT) yang digunakan untuk mengakses endpoint-endpoint lain yang membutuhkan otorisasi.

Token ini biasanya dikirimkan kembali ke klien melalui *response*, baik dalam bentuk *header*, *cookie*, maupun *body response*. Jika proses login gagal—misalnya karena `password` salah, *user* tidak ditemukan, atau akun nonaktif—maka *controller* akan mengembalikan *response error* beserta pesan yang sesuai.

Endpoint ini merupakan pintu gerbang utama bagi *user* untuk dapat mengakses seluruh fitur yang ada di dalam sistem.

```
res.cookie("access_token", token, {
  httpOnly: false,
  secure: process.env.NODE_ENV === "production",
  sameSite: "strict",
  maxAge: 60 * 60 * 1000, // 1 jam
});

res.cookie("refresh_token", refreshToken, {
  httpOnly: false,
  secure: process.env.NODE_ENV === "production",
  maxAge: 7 * 24 * 60 * 60 * 1000, // 7 hari
});
```

Gambar 3.29. Durasi token JWT di *cookies*



Gambar 3.30. Berhasil Login

API `POST /refresh_token`

Endpoint `POST /refresh_token` digunakan untuk memperbarui atau mendapatkan *token autentikasi* (JWT) yang baru ketika *token* sebelumnya sudah

hampir kedaluwarsa atau telah kedaluwarsa. Endpoint ini biasanya menerima *refresh token* yang masih valid melalui *request body* atau *cookie*.

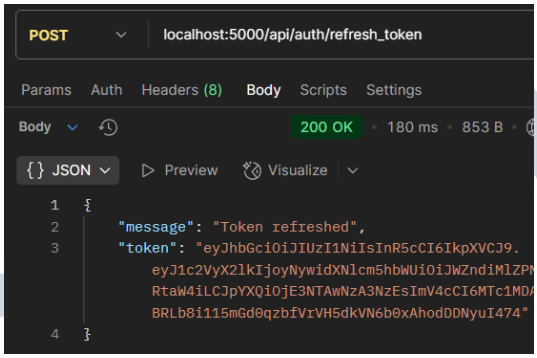
Pada endpoint ini, *controller* yang digunakan adalah `refreshToken` yang terdapat pada berkas `authController.ts`. *Controller* akan memverifikasi *refresh token* yang dikirimkan, memastikan bahwa token tersebut masih berlaku dan terdaftar di basis data.

Jika validasi berhasil, sistem akan menghasilkan dan mengirimkan *token JWT* yang baru kepada klien, sehingga *user* dapat tetap mengakses fitur-fitur yang memerlukan autentikasi tanpa harus melakukan login ulang. Jika *refresh token* tidak valid, sudah kedaluwarsa, atau tidak ditemukan di basis data, maka *controller* akan mengembalikan *response error* beserta pesan yang sesuai.

Endpoint ini sangat penting untuk menjaga keamanan dan kenyamanan *user* dalam sesi autentikasi yang berkelanjutan di dalam aplikasi.

```
// Set ulang access_token di cookie
res.cookie("access_token", newAccessToken, {
  httpOnly: false,
  secure: process.env.NODE_ENV === "production",
  sameSite: "strict",
  maxAge: 60 * 60 * 1000, // 1 jam
});
```

Gambar 3.31. Durasi API *refreshToken*



Gambar 3.32. Berhasil *Refresh* Token

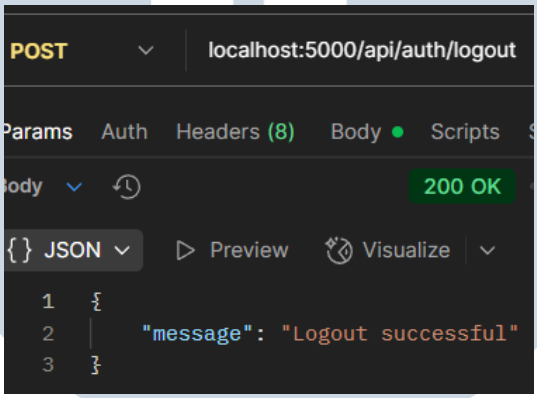
API `POST /logout`

Endpoint `POST /logout` digunakan untuk mengakhiri sesi autentikasi *user* pada sistem. Endpoint ini biasanya menerima *token autentikasi* (JWT) atau *refresh token* dari klien, baik melalui *header*, *cookie*, atau *body request*.

Pada endpoint ini, *controller* yang digunakan adalah `logoutUser` yang terdapat pada berkas `authController.ts`. *Controller* akan menghapus atau

menonaktifkan token yang terkait dengan *user* dari basis data atau *session store*, sehingga token tersebut tidak dapat digunakan lagi untuk mengakses endpoint yang membutuhkan autentikasi.

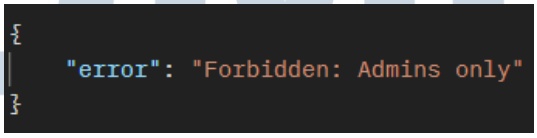
Jika proses *logout* berhasil, sistem akan mengembalikan *response sukses* kepada klien. Jika terjadi kesalahan, seperti token tidak valid atau sudah tidak aktif, *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini penting untuk menjaga keamanan akun *user*, terutama saat keluar dari aplikasi pada perangkat publik atau bersama.



Gambar 3.33. Berhasil *Logout*

API `POST /reset_password`

Endpoint `POST /reset_password` digunakan untuk mengganti *password user* yang ada di *database*. Endpoint ini menerima *password* baru melalui *request body*, dan biasanya hanya dapat diakses oleh *user admin*.

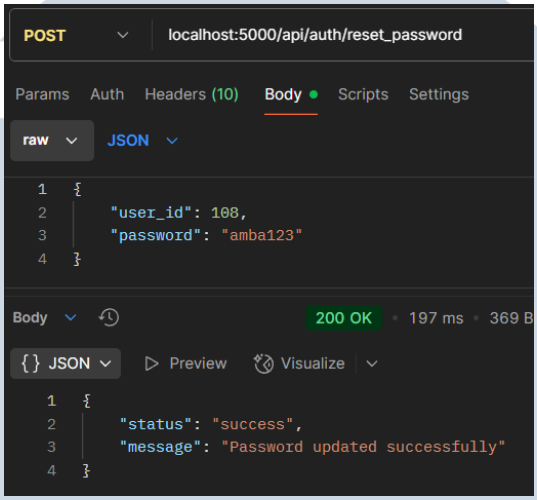


Gambar 3.34. Forbidden *Admins Only*

Pada endpoint ini, *controller* yang digunakan adalah `resetPassword` yang terdapat pada berkas `authController.ts`. *Controller* akan melakukan validasi terhadap *password* baru, memastikan *password* memenuhi syarat keamanan (seperti panjang minimal dan kombinasi karakter). Setelah validasi berhasil, *password* baru akan di-*hash* menggunakan `bcrypt` dan disimpan ke basis data.

Jika proses *reset password* berhasil, sistem akan mengembalikan *response sukses*. Jika terjadi kesalahan, seperti *password* tidak valid atau *user* tidak ditemukan,

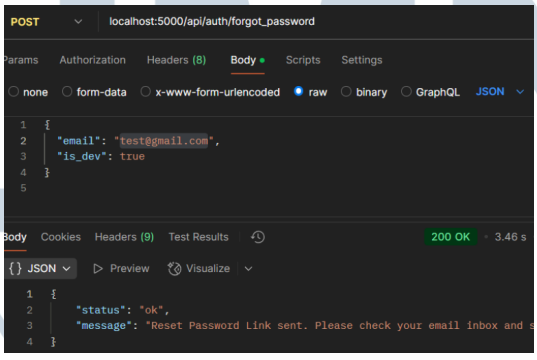
controller akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini penting untuk menjaga keamanan akun *user* dan memungkinkan *user* mengganti *password* secara mandiri.



Gambar 3.35. Berhasil *Reset Password*

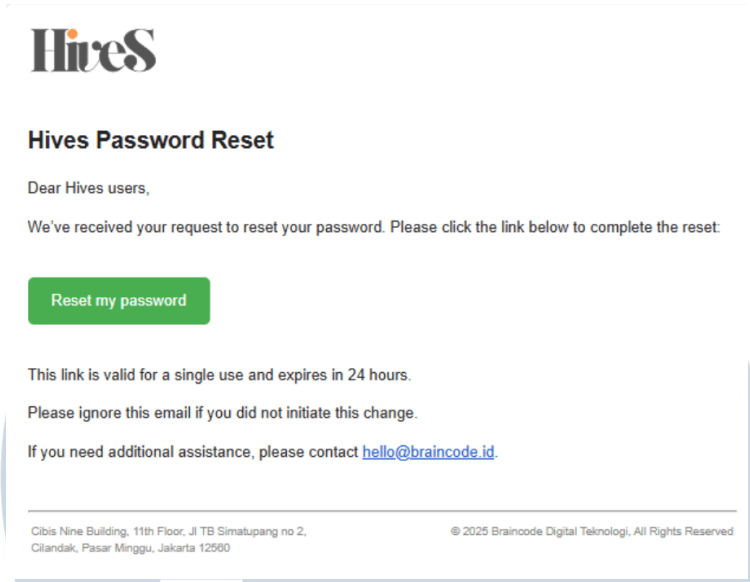
API `POST /forgot_password`

Endpoint `POST /forgot_password` digunakan ketika *user* lupa *password* dan ingin melakukan proses *reset password*. Endpoint ini menerima *email user* melalui *request body*.



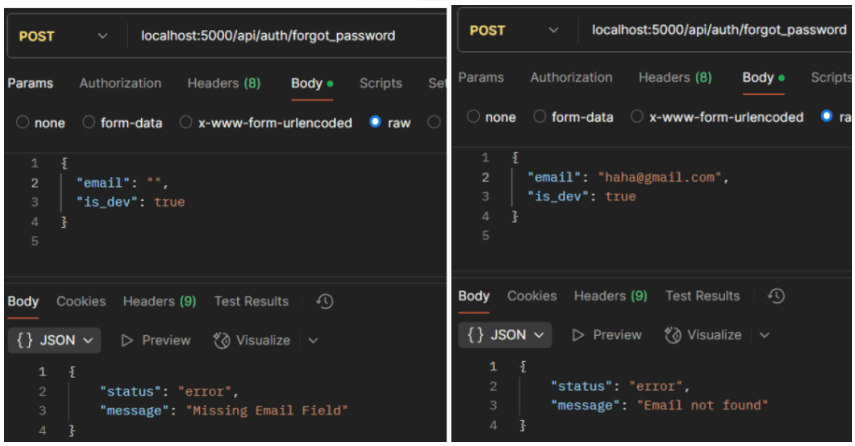
Gambar 3.36. API *Forgot Password*

Pada endpoint ini, *controller* yang digunakan adalah `forgotPassword` yang terdapat pada berkas `authController.ts`. *Controller* akan memeriksa apakah *email* yang dikirimkan terdaftar di basis data. Jika *email* ditemukan, sistem akan menghasilkan kode atau *token reset password* dan mengirimkannya ke *email user* menggunakan library *nodemailer*.



Gambar 3.37. *Email Forgot Password*

Jika *email* tidak ditemukan atau terjadi kesalahan, *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini sangat penting untuk memberikan kemudahan bagi *user* dalam memulihkan akses ke akun mereka secara mandiri.



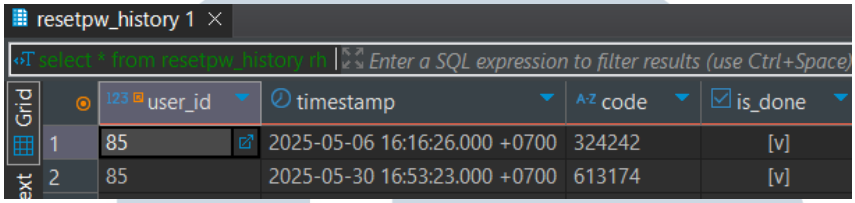
Gambar 3.38. Error *Forgot Password*

API `POST /forgot_password/check`

Endpoint `POST /forgot_password/check` digunakan untuk memverifikasi validitas kode atau *token reset password* yang telah dikirimkan ke *email user*. Endpoint ini menerima kode *reset* melalui *request body*.

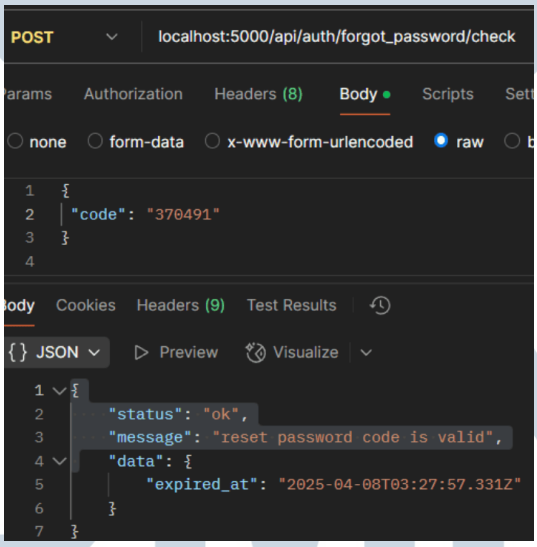
Pada endpoint ini, *controller* yang digunakan adalah `checkForgotPasswordCode` yang terdapat pada berkas `authController.ts`.

Controller akan memeriksa apakah kode *reset* yang dikirimkan masih berlaku dan sesuai dengan *user* yang bersangkutan dengan mengecek status kode di tabel *resetpw_history* di kolom *is_done*.



Gambar 3.39. Tabel *resetpw_history*

Jika kode valid, sistem akan mengembalikan *response sukses*. Jika kode salah atau sudah kedaluwarsa, *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini penting untuk memastikan keamanan proses *reset password* sebelum *password* benar-benar diubah.



Gambar 3.40. Cek kevalidan kode *Forgot Password*

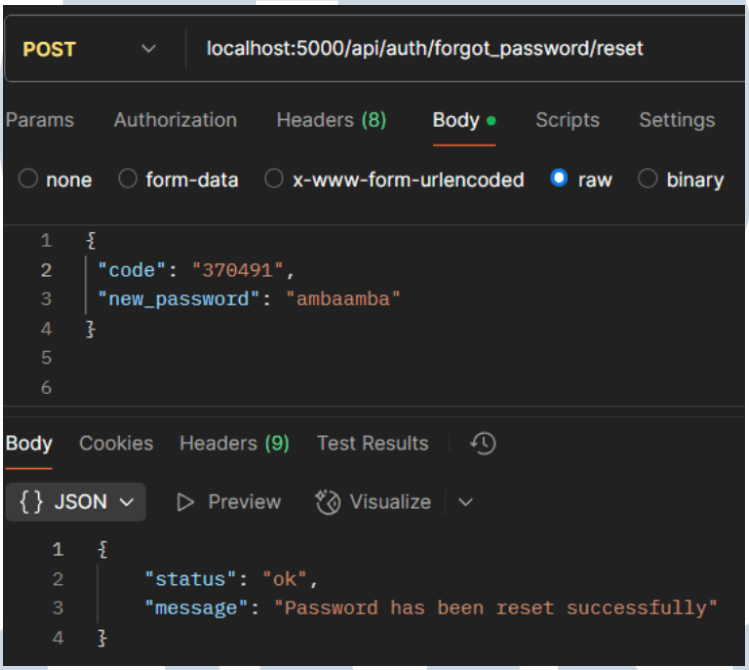
API `POST /forgot_password/reset`

Endpoint `POST /forgot_password/reset` digunakan untuk mengatur ulang *password user* menggunakan kode atau *token reset* yang telah dikirimkan ke *email user*. Endpoint ini menerima kode *reset* dan *password* baru melalui *request body*.

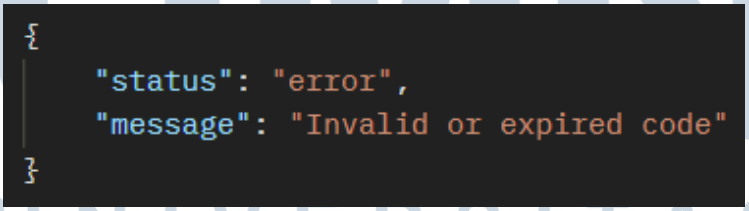
Pada endpoint ini, *controller* yang digunakan adalah `resetForgotPassword` yang terdapat pada berkas `authController.ts`. *Controller* akan memverifikasi

kode *reset* yang dikirimkan, memastikan kode masih berlaku dan sesuai dengan *user* yang bersangkutan.

Jika validasi berhasil, *password* baru akan di-*hash* dan disimpan ke basis data. Jika proses *reset* berhasil, sistem akan mengembalikan *response sukses*. Jika kode salah, sudah kedaluwarsa, atau *password* baru tidak valid, *controller* akan mengembalikan *response error* dengan pesan yang sesuai. Endpoint ini sangat penting untuk proses pemulihan akun secara aman.



Gambar 3.41. Finalisasi *Forgot Password*



Gambar 3.42. Error Kode *Forgot Password*

Tabel 3.7. Tabel Pengujian API Autentikasi - *Whitebox*

No	Endpoint	Skenario Pengujian	Status
1	`POST /login`	Validasi field, cek user, cek password hash, generate token, simpan token	Lulus
2	`POST /refresh_token`	Validasi refresh token, cek token di DB, generate token baru	Lulus
3	`POST /logout`	Hapus/invalidasi token dari DB/session	Lulus
4	`POST /reset_password`	Validasi password baru, hash password, update DB	Lulus
5	`POST /forgot_password`	Validasi email, cek email di DB, generate kode, kirim email	Lulus
6	`POST /forgot_password/reset`	Validasi kode reset, validasi password baru, update DB	Lulus
7	`POST /forgot_password/check`	Validasi kode/token reset	Lulus

Tabel 3.8. Tabel Pengujian API Autentikasi - *Blackbox*

No	Endpoint	Skenario Pengujian	Status
1	`POST /login`	Login valid, password salah, user tidak terdaftar, akun nonaktif	Lulus
2	`POST /refresh_token`	Refresh token valid, refresh token expired/tidak valid	Lulus
3	`POST /logout`	Logout dengan token valid, logout tanpa token/invalid	Lulus
4	`POST /reset_password`	Reset password valid, password tidak valid, tanpa login	Lulus
5	`POST /forgot_password`	Request reset dengan email terdaftar/tidak terdaftar	Lulus
6	`POST /forgot_password/reset`	Reset dengan kode valid/password valid, kode salah/expired, password tidak valid	Lulus
7	`POST /forgot_password/check`	Cek kode valid, cek kode salah/expired	Lulus

3.3.5 Pembuatan Model dan API Course

Pada tahap ini, dilakukan perancangan model dan pengembangan berbagai endpoint (API) yang berkaitan dengan fitur *course* (pembelajaran daring) di sistem. Model *course* dirancang menggunakan Sequelize dan PostgreSQL, mencakup entitas utama seperti *Course*, *Module*, *Submodule*, *Quiz*, *Question*, dan relasi-relasi antar entitas tersebut. Pengembangan API dilakukan menggunakan *Express.js* dan *TypeScript*, sehingga seluruh proses manajemen *course*, baik dari sisi pengguna maupun admin, dapat berjalan secara terstruktur dan terintegrasi.

Setelah model selesai dibuat, dilakukan pengembangan berbagai endpoint untuk kebutuhan pengguna (public route), seperti melihat daftar *course*, detail *course*, *module*, *submodule*, *quiz*, serta proses *enroll* dan pengumpulan jawaban *quiz*. Selain itu, terdapat juga endpoint khusus admin (CMS route) untuk menambah, mengedit,

menghapus, dan mengatur urutan *course*, *module*, *submodule*, *quiz*, dan soal. Berikut adalah daftar API *course* yang diimplementasikan beserta penjelasan singkat dan jenis pengujian yang digunakan:

Tabel 3.9. Daftar API Course (Bagian 1)

No	Endpoint	Method	Deskripsi Singkat	Jenis Pengujian
1	`/field`	GET	Ambil daftar bidang *course*	*Blackbox*
2	`/detail/:course_id`	GET	Ambil detail course tertentu	*Blackbox*
3	`/module/submodule/:course_id`	GET	Ambil modul dan submodul course	*Blackbox*
4	`/module/submodule_content`	POST	Submit konten submodul	*Whitebox & Blackbox*
5	`/module/quiz/answer_question`	POST	Submit jawaban quiz	*Whitebox & Blackbox*
6	`/module/quiz/question`	GET	Ambil soal quiz	*Blackbox*
7	`/module/quiz`	GET	Ambil daftar quiz	*Blackbox*
8	`/module`	GET	Ambil daftar modul	*Blackbox*
9	`/cms/list`	GET	Ambil daftar course (CMS)	*Blackbox*
10	`/cms/:course_id`	GET	Ambil detail course (CMS)	*Blackbox*
11	`/list`	POST	Tambah course ke daftar user	*Whitebox & Blackbox*
12	`/enroll`	POST	Enroll ke course	*Whitebox & Blackbox*
13	`/cms/add`	POST	Tambah course baru (admin)	*Whitebox & Blackbox*

Tabel 3.10. Daftar API Course (Bagian 2)

No	Endpoint	Method	Deskripsi Singkat	Jenis Pengujian
14	`/cms/edit`	POST	Edit course (admin)	*Whitebox & Blackbox*
15	`/cms/module/edit`	POST	Edit modul (admin)	*Whitebox & Blackbox*
16	`/cms/submodule/edit`	POST	Edit submodul (admin)	*Whitebox & Blackbox*
17	`/cms/quiz/edit`	POST	Edit quiz (admin)	*Whitebox & Blackbox*
18	`/cms/question/edit`	POST	Edit soal quiz (admin)	*Whitebox & Blackbox*
19	`/cms/reorder_module`	PATCH	Ubah urutan modul (admin)	*Whitebox & Blackbox*
20	`/cms/reorder_submodule`	PATCH	Ubah urutan submodul (admin)	*Whitebox & Blackbox*
21	`/cms/reorder_quiz`	PATCH	Ubah urutan quiz (admin)	*Whitebox & Blackbox*
22	`/cms/reorder_question`	PATCH	Ubah urutan soal (admin)	*Whitebox & Blackbox*
23	`/cms/delete`	DELETE	Hapus course (admin)	*Whitebox & Blackbox*
24	`/cms/module/delete`	DELETE	Hapus modul (admin)	*Whitebox & Blackbox*
25	`/cms/submodule/delete`	DELETE	Hapus submodul (admin)	*Whitebox & Blackbox*
26	`/cms/quiz/delete`	DELETE	Hapus quiz (admin)	*Whitebox & Blackbox*
27	`/cms/question/delete`	DELETE	Hapus soal quiz (admin)	*Whitebox & Blackbox*

Penjelasan Jenis Pengujian

Whitebox Testing

Pengujian dilakukan dengan memahami struktur internal kode, termasuk proses validasi, penyimpanan, serta penghapusan data. Biasanya diterapkan pada endpoint yang mengubah data (seperti POST, PATCH, DELETE).

Blackbox Testing

Pengujian dilakukan berdasarkan input dan output tanpa melihat struktur internal kode. Umumnya digunakan pada endpoint pengambilan data (seperti GET) dengan berbagai skenario input.

API `GET /field`

Endpoint `GET /field` digunakan untuk mengambil daftar bidang atau kategori *course* yang tersedia di sistem. Endpoint ini biasanya digunakan pada halaman pencarian atau *filter course*. *Controller* yang digunakan adalah `getCourseFields` yang terdapat pada berkas `courseController.ts`. *Controller* akan mengambil data bidang dari basis data dan mengembalikannya dalam format array. Jika tidak ada bidang, *response* berupa array kosong.

Gambar 3.43. API getCourseFields

API `GET /detail/:course_id`

Endpoint `GET /detail/:course_id` digunakan untuk mengambil detail lengkap dari sebuah *course* berdasarkan `course_id` tertentu. *Controller* yang digunakan adalah `getCourseDetail` pada berkas `courseController.ts`. *Controller* akan mencari *course* berdasarkan ID di basis data dan mengembalikan data lengkap *course*, termasuk deskripsi, modul, dan informasi lain yang relevan.


```
{
    "course_id": 3,
    "course_name": "Introduction to Frontend Stack",
    "fields": [
        {
            "field_id": 2,
            "field_name": "Front End",
            "color": "rgba(103, 186, 247, 0.8)"
        },
        {
            "field_id": 4,
            "field_name": "Project Management",
            "color": "rgba(255, 136, 0, 0.8)"
        }
    ],
    "course_description": "RichText:{\"ops\":[{\"attributes\":
        {\"background\":\"#ffffff\",\"color\":\"#b26b00\",
        \"underline\":true},\"insert\":\"Semantic HTML\"},{\"attributes\":
        {\"align\":\"center\",\"header\":2},\"insert\":\"\\n\"},
        {\"attributes\":{\"background\":\"#ffffff\",
        \"color\":\"#202124\"},\"insert\":\"With over \"},{\"attributes\":
        {\"background\":\"#ffffff\",\"color\":\"#008a00\",\"bold\":true},
        \"insert\":\"100\"},{\"attributes\":{\"background\":\"#ffffff\",
        \"color\":\"#202124\"},\"insert\":\" HTML elements, and the
        ability to create \"},{\"attributes\":{\"background\":\"#ffffff\",
        \"color\":\"#202124\",\"bold\":true},\"insert\":\"custom
        elements\"},{\"attributes\":{\"background\":\"#ffffff\",
        \"color\":\"#202124\"},\"insert\":\", there are infinite ways to
        mark up your content; but some ways—notably \"},{\"attributes\":
        {\"background\":\"#ffffff\",\"color\":\"#202124\",
        \"italic\":true},\"insert\":\"semantically\"},{\"attributes\":
        {\"background\":\"#ffffff\",\"color\":\"#202124\"},
        \"insert\":\"—are better than\"},{\"attributes\":
        {\"background\":\"#ffffff\",\"color\":\"#202124\",
        \"strike\":true},\"insert\":\" others\"},{\"attributes\":
        {\"background\":\"#ffffff\",\"color\":\"#202124\"},\"insert\":\".
        \"},{\"insert\":\"\\n\"}]}",
    "has_enrolled": true,
    "course_elapsed_minutes": null,
    "course_duration_minute": 0,
    "course_duration_format": "Course has no time limit"
}
```


Gambar 3.44. API getCourseDetail

API `GET /module/submodule/:course_id`

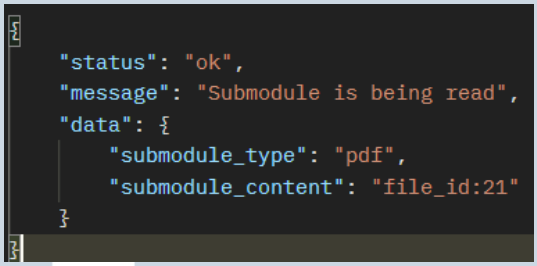
Endpoint `GET /module/submodule/:course_id` digunakan untuk mengambil daftar modul dan submodul dari *course* tertentu. *Controller* yang digunakan adalah `getModulesAndSubmodules` pada berkas `courseController.ts`. *Controller* akan mencari semua modul dan submodul yang terkait dengan `course_id` yang diberikan, lalu mengembalikannya dalam format terstruktur.


```
"message": "Submodule info received",
"data": [
    {
        "module_id": 3,
        "module_name": "Introduction HTML",
        "order_number": 3,
        "status": "module_read",
        "submodules": [
            {
                "submodule_id": 1,
                "submodule_name": "Learn HTML Tags",
                "submodule_type": "free_text",
                "order_number": 1,
                "status": "submodule_read"
            },
            {
                "submodule_id": 2,
                "submodule_name": "Learn HTML Attributes",
                "submodule_type": "video",
                "order_number": 2,
                "status": "submodule_read"
            },
            {
                "submodule_id": 3,
                "submodule_name": "Learn HTML Semantic",
                "submodule_type": "pdf",
                "order_number": 3,
                "status": "submodule_read"
            }
        ],
        "quizzes": [
            {
                "quiz_id": 1,
                "quiz_type": true,
                "quiz_name": "HTML Tags Knowledge Test",
                "order_number": 1
            },
            {
                "quiz_id": 2,
                "quiz_type": true,
                "quiz_name": "HTML Attributes Knowledge Test",
                "order_number": 2
            }
```


Gambar 3.45. API getModulesAndSubmodules

API `POST /module/submodule_content`

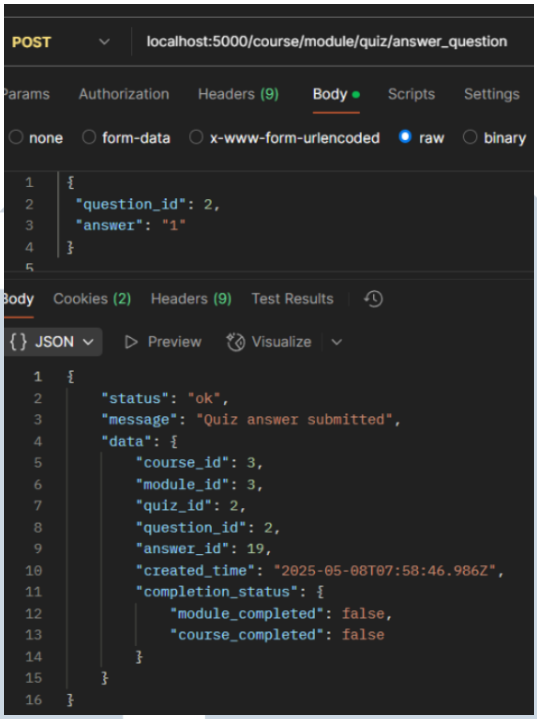
Endpoint `POST /module/submodule_content` digunakan untuk menyimpan atau memperbarui konten pada submodul tertentu. *Controller* yang digunakan adalah `submitSubmoduleContent` pada berkas `courseController.ts`. *Controller* akan menerima data konten dari *request body*, melakukan validasi, lalu menyimpan atau memperbarui konten submodul di basis data.



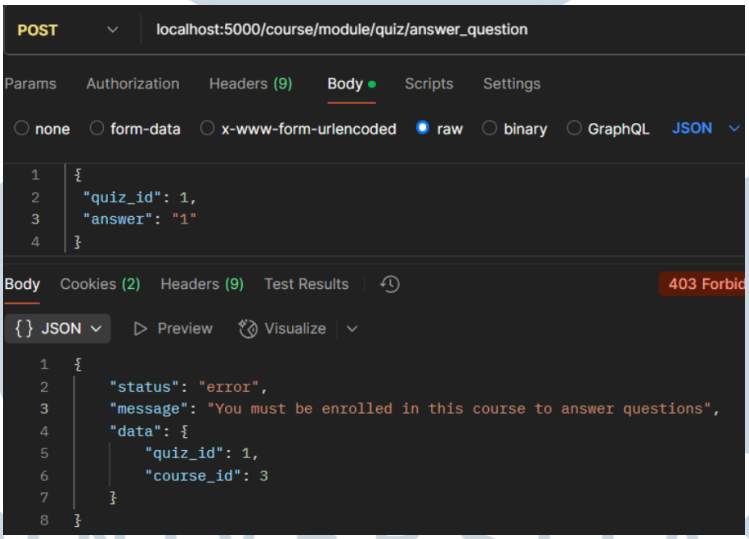
Gambar 3.46. API submitSubmoduleContent

API `POST /module/quiz/answer_question`

Endpoint `POST /module/quiz/answer_question` digunakan untuk mengirimkan jawaban *user* pada soal kuis di suatu modul. *Controller* yang digunakan adalah `answerQuizQuestion` pada berkas `courseController.ts`. *Controller* akan menerima jawaban dari *user*, melakukan validasi, memeriksa kebenaran jawaban, dan memperbarui skor *user* di basis data.



Gambar 3.47. API untuk menjawab sebuah *question* di suatu quiz

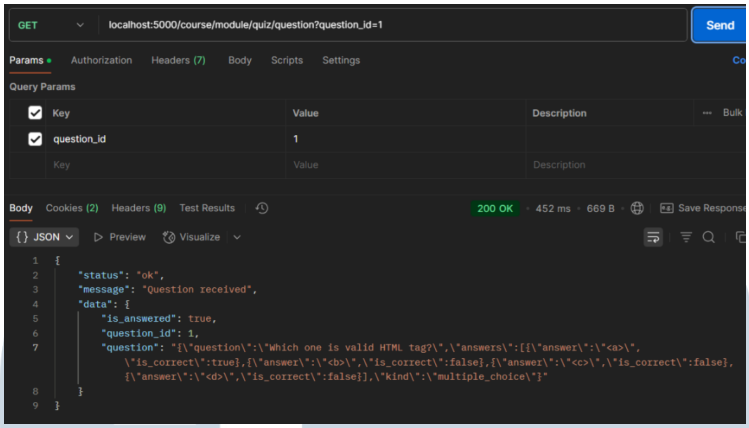


Gambar 3.48. Error *forbidden*: harus *enroll* di course yang bersangkutan terlebih dahulu

API `GET /module/quiz/question`

Endpoint `GET /module/quiz/question` digunakan untuk mengambil daftar soal kuis pada modul tertentu. *Controller* yang digunakan adalah `getQuizQuestions` pada berkas `courseController.ts`. *Controller* akan mengambil soal-soal kuis dari basis data dan mengembalikannya dalam format

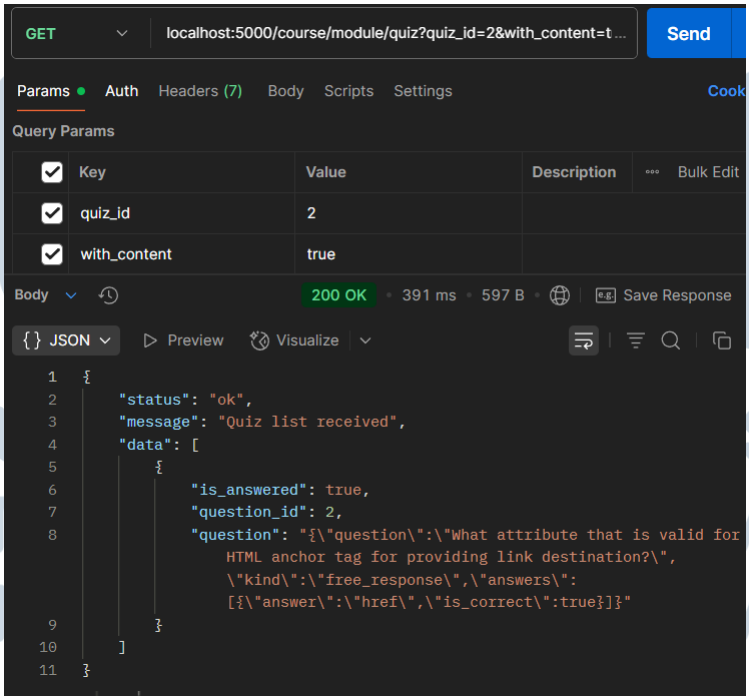
array.



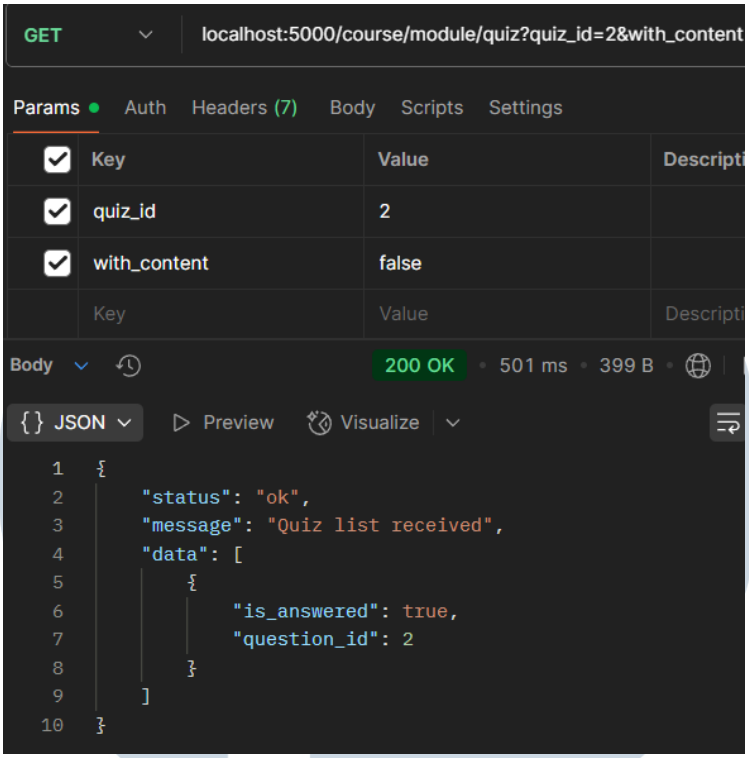
Gambar 3.49. API untuk mengambil daftar *question* di dalam suatu *quiz*

API `GET /module/quiz`

Endpoint `GET /module/quiz` digunakan untuk mengambil daftar kuis yang tersedia pada suatu modul atau *course*. *Controller* yang digunakan adalah `getQuizzes` pada berkas `courseController.ts`. *Controller* akan mengambil data kuis dari basis data dan mengembalikannya dalam format array.



Gambar 3.50. Mengambil Quiz dengan *content*



Gambar 3.51. Mengambil Quiz tidak dengan *content*

API `GET /module`

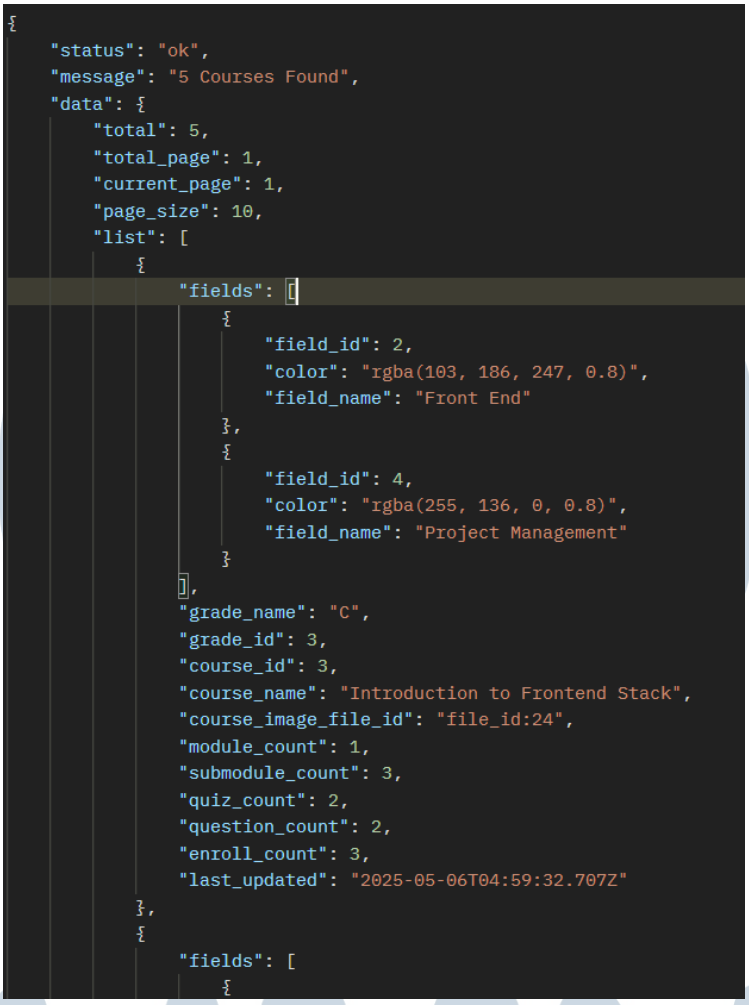
Endpoint `GET /module` digunakan untuk mengambil daftar modul yang tersedia pada suatu *course*. *Controller* yang digunakan adalah `getModules` pada berkas `courseController.ts`. *Controller* akan mengambil data modul dari basis data dan mengembalikannya dalam format array.

```
[
    {
        "module_id": 4,
        "module_name": "Introduction HTML",
        "module_description": "RichText:Will learn HTML test",
        "video_count": 2,
        "reading_count": 0,
        "quiz_count": 2
    },
    {
        "module_id": 3,
        "module_name": "Introduction HTML",
        "module_description": "RichText:Will learn HTML 1",
        "video_count": 1,
        "reading_count": 0,
        "quiz_count": 2
    },
    {
        "module_id": 28,
        "module_name": "dad",
        "module_description": "dad",
        "video_count": 0,
        "reading_count": 0,
        "quiz_count": 1
    },
    {
        "module_id": 5,
        "module_name": "Introduction HTML",
        "module_description": "RichText:Will learn HTML",
        "video_count": 1,
        "reading_count": 0,
        "quiz_count": 2
    },
```


Gambar 3.52. Mengambil daftar modul

API `GET /cms/list`

Endpoint `GET /cms/list` digunakan untuk mengambil daftar *course* untuk keperluan manajemen (CMS). *Controller* yang digunakan adalah `getCmsCourseList` pada berkas `courseController.ts`. *Controller* akan mengambil data *course* dari basis data dan mengembalikannya dalam format array, biasanya dengan informasi lebih detail untuk *admin*.

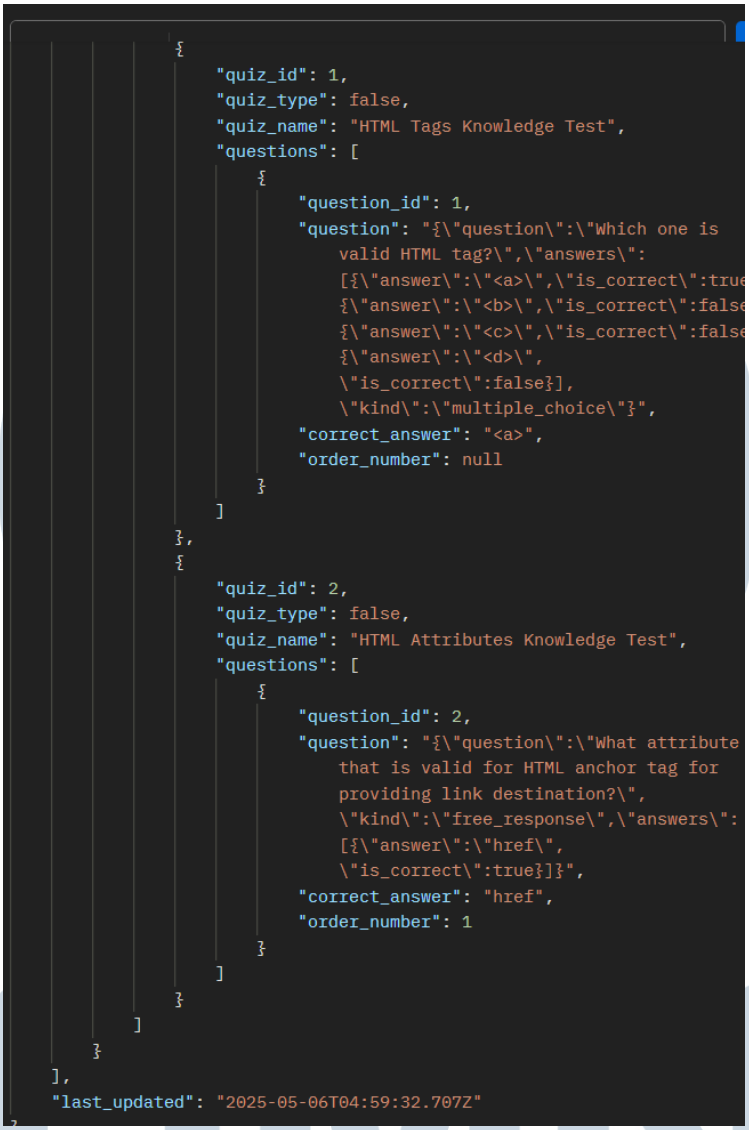


Gambar 3.53. Mengambil daftar *Course* dengan *detail* lebih lengkap

API `GET /cms/:course_id`

Endpoint `GET /cms/:course_id` digunakan untuk mengambil detail *course* tertentu untuk keperluan manajemen (CMS). *Controller* yang digunakan adalah `getCmsCourseDetail` pada berkas `courseController.ts`. *Controller* akan mengambil data *course* berdasarkan ID dari basis data dan mengembalikannya dalam format detail.

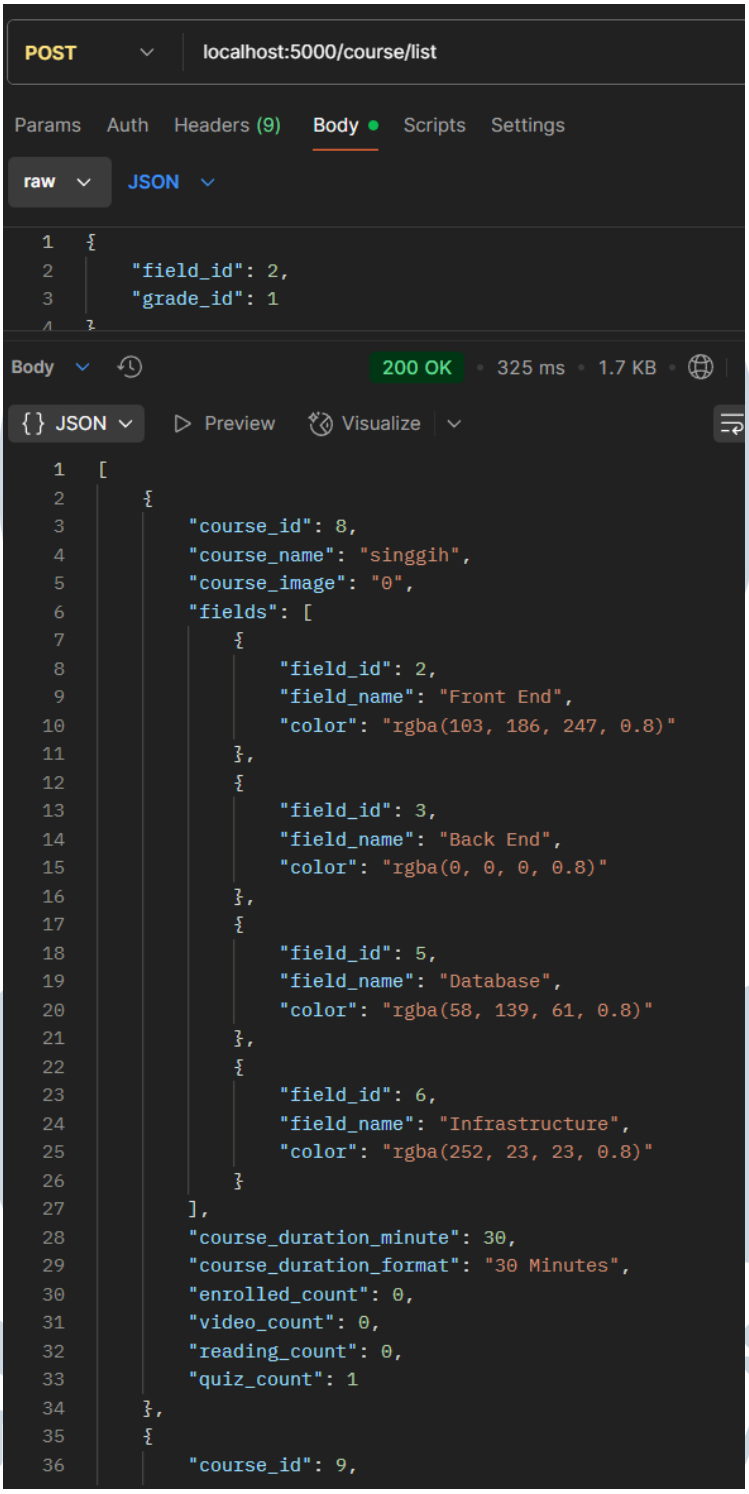
Gambar 3.54. (Bagian 1) Mengambil *detail course*, tetapi lebih lengkap dan rinci daripada API *GET /detail/:course id*



Gambar 3.55. (Bagian 2) Mengambil *detail course*, tetapi lebih lengkap dan rinci daripada API *GET /detail/:course id*

API `POST /list`

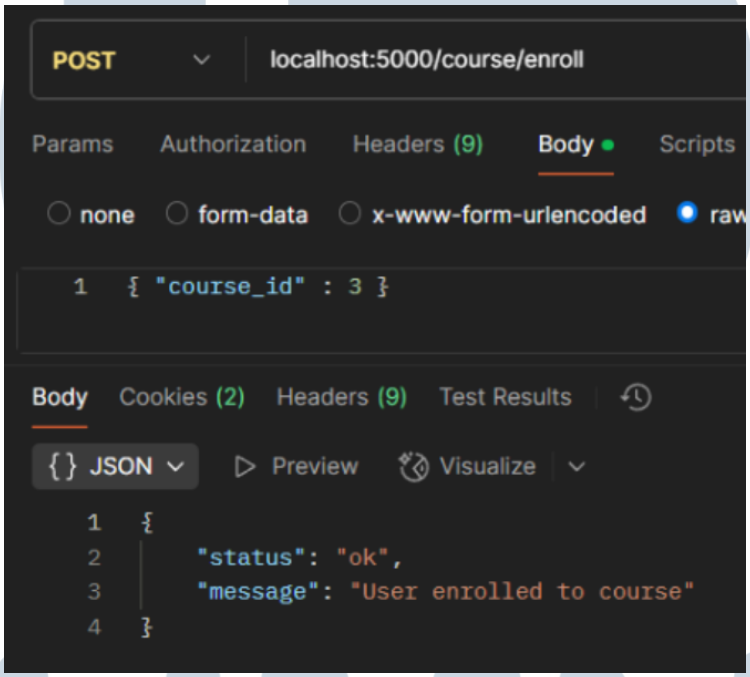
Endpoint `POST /list` digunakan untuk menambahkan *course* ke daftar *course user*. *Controller* yang digunakan adalah `addCourseToList` pada berkas `courseController.ts`. *Controller* akan menerima data *course* dari *request body*, melakukan validasi, dan menambahkannya ke daftar *user* di basis data.



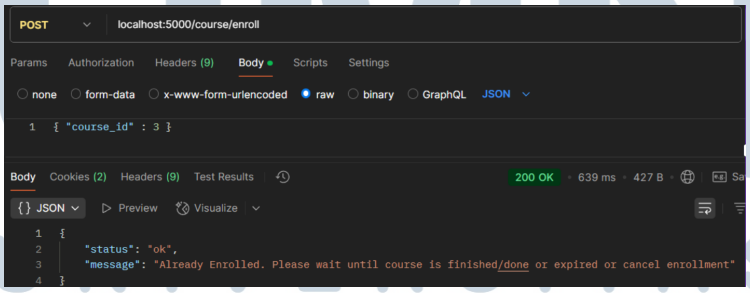
Gambar 3.56. Mengambil *course* berdasarkan *filter by field* dan *grade*

API `POST /enroll`

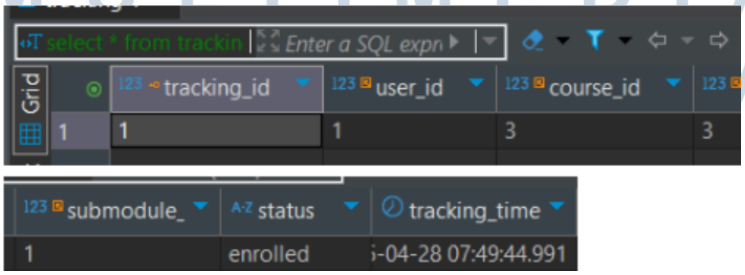
Endpoint `POST /enroll` digunakan untuk mendaftarkan *user* ke dalam *course* tertentu. *Controller* yang digunakan adalah `enrollCourse` pada berkas `courseController.ts`. *Controller* akan menerima `course_id` dari *request body*, melakukan validasi, dan menambahkan *user* ke daftar peserta *course* di basis data di table *tracking*.



Gambar 3.57. Berhasil *enroll*



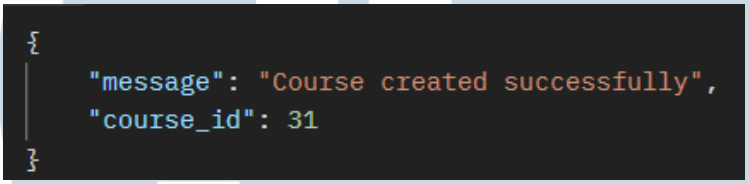
Gambar 3.58. Pesan error sudah pernah atau sedang *enroll*



Gambar 3.59. Isi tabel *tracking* setelah ada *user* yang *enroll*

API `POST /cms/add`

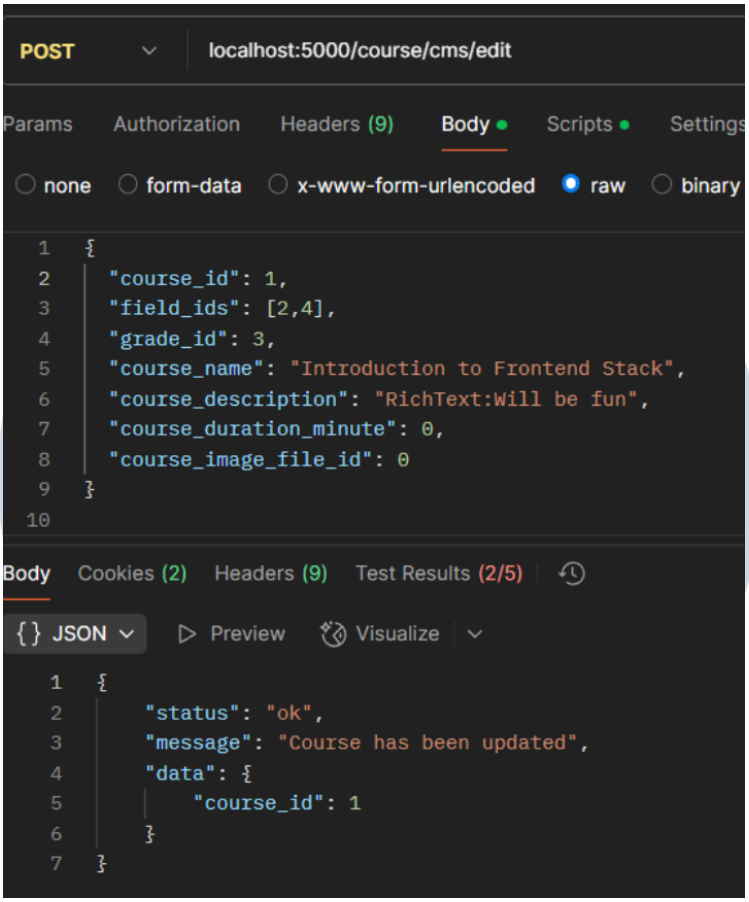
Endpoint `POST /cms/add` digunakan untuk menambah *course* baru ke sistem (hanya untuk *admin*). *Controller* yang digunakan adalah `addCourse` pada berkas `courseController.ts`. *Controller* akan menerima data *course* baru dari *request body*, melakukan validasi, dan menyimpan *course* ke basis data.



Gambar 3.60. CMS *add* berhasil

API `POST /cms/edit`

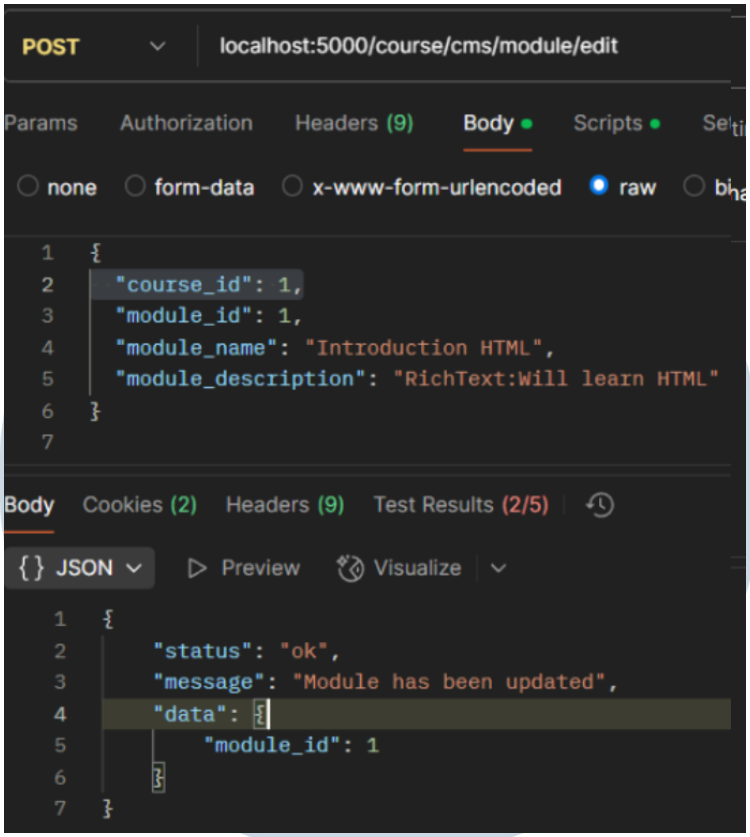
Endpoint `POST /cms/edit` digunakan untuk mengedit data *course* yang sudah ada (hanya untuk *admin*). *Controller* yang digunakan adalah `editCourse` pada berkas `courseController.ts`. *Controller* akan menerima data perubahan dari *request body*, melakukan validasi, dan memperbarui data *course* di basis data.



Gambar 3.61. CMS *edit* berhasil

API `POST /cms/module/edit`

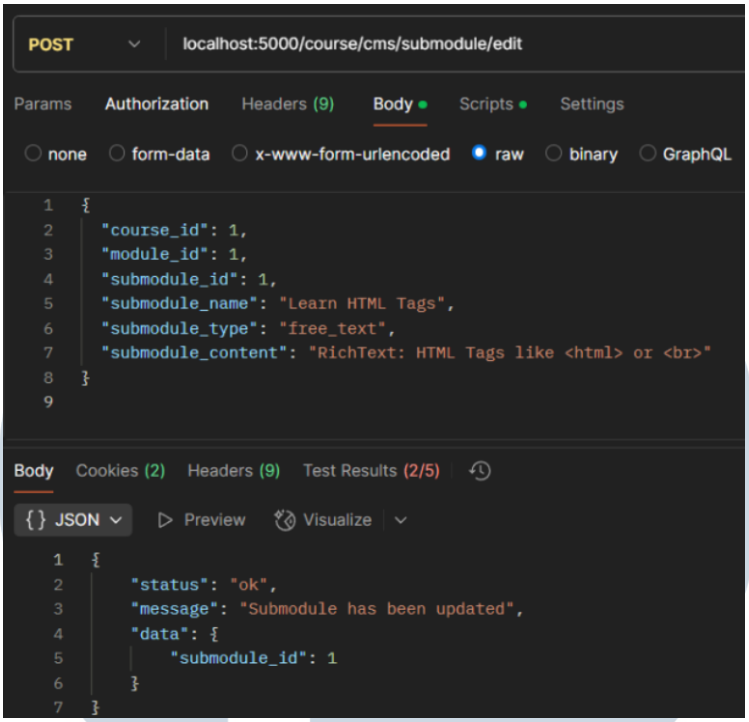
Endpoint `POST /cms/module/edit` digunakan untuk mengedit data *modul* pada *course* (hanya untuk *admin*). *Controller* yang digunakan adalah `editModule` pada berkas `courseController.ts`. *Controller* akan menerima data perubahan *modul* dari *request body*, melakukan validasi, dan memperbarui data *modul* di basis data.



Gambar 3.62. *Module edit* berhasil

API `POST /cms/submodule/edit`

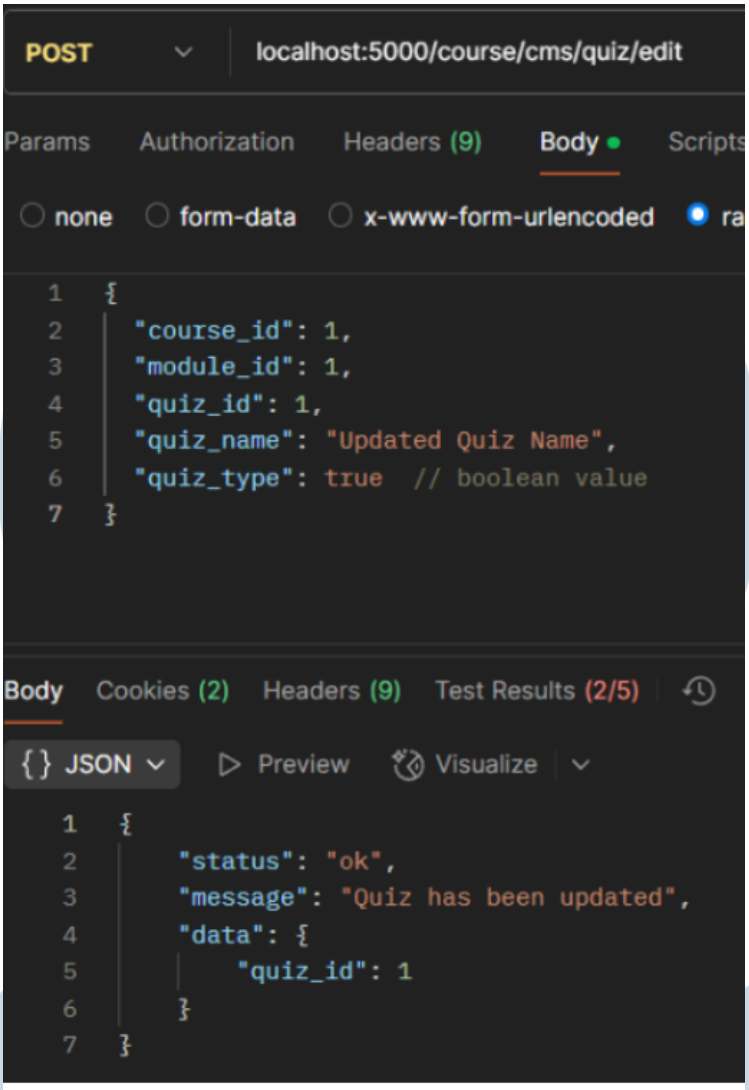
Endpoint `POST /cms/submodule/edit` digunakan untuk mengedit data *submodul* pada *modul* (hanya untuk *admin*). *Controller* yang digunakan adalah `editSubmodule` pada berkas `courseController.ts`. *Controller* akan menerima data perubahan *submodul* dari *request body*, melakukan validasi, dan memperbarui data *submodul* di basis data.



Gambar 3.63. *Submodule edit* berhasil

API `POST /cms/quiz/edit`

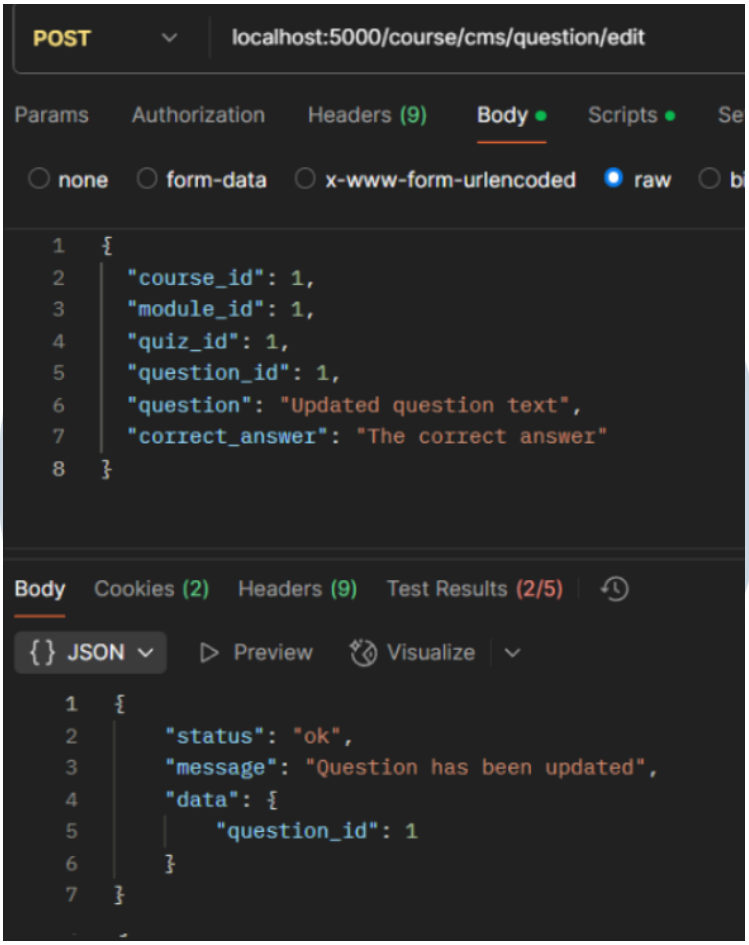
Endpoint `POST /cms/quiz/edit` digunakan untuk mengedit data *quiz* pada *modul* (hanya untuk *admin*). *Controller* yang digunakan adalah `editQuiz` pada berkas `courseController.ts`. *Controller* akan menerima data perubahan *quiz* dari *request body*, melakukan validasi, dan memperbarui data *quiz* di basis data.



Gambar 3.64. *Quiz edit* berhasil

API `POST /cms/question/edit`

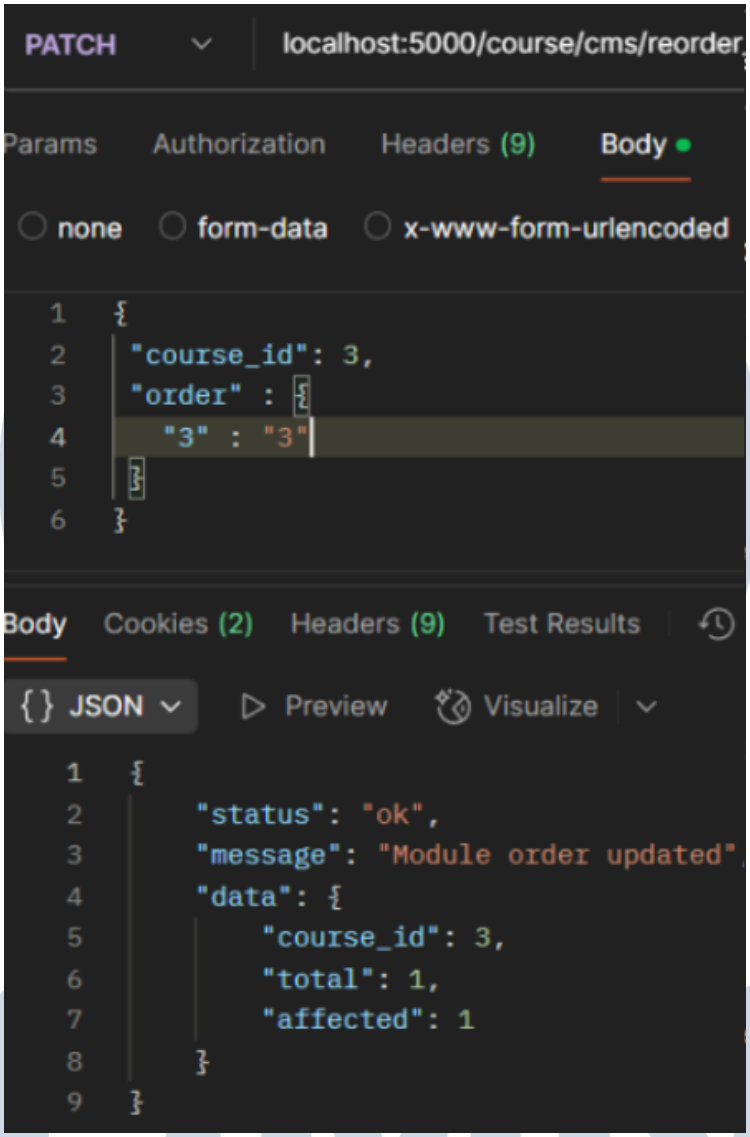
Endpoint `POST /cms/question/edit` digunakan untuk mengedit data soal pada *quiz* (hanya untuk *admin*). *Controller* yang digunakan adalah `editQuestion` pada berkas `courseController.ts`. *Controller* akan menerima data perubahan soal dari *request body*, melakukan validasi, dan memperbarui data soal di basis data.



Gambar 3.65. *Question edit* berhasil

API `PATCH /cms/reorder_module`

Endpoint `PATCH /cms/reorder_module` digunakan untuk mengubah urutan *modul* dalam *course* (hanya untuk *admin*). *Controller* yang digunakan adalah `reorderModule` pada berkas `courseController.ts`. *Controller* akan menerima urutan baru dari *request body* dan memperbarui urutan *modul* di basis data.



Gambar 3.66. *Module reorder* berhasil

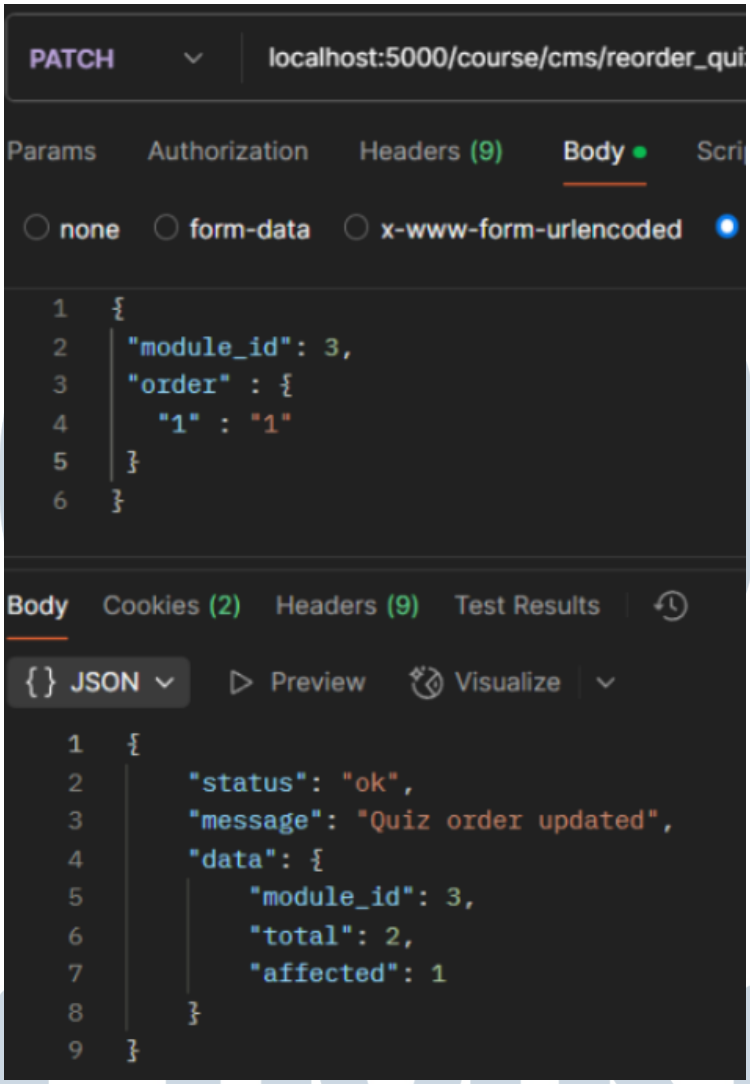
API `PATCH /cms/reorder_submodule`

Endpoint `PATCH /cms/reorder_submodule` digunakan untuk mengubah urutan *submodul* dalam *modul* (hanya untuk *admin*). *Controller* yang digunakan adalah `reorderSubmodule` pada berkas `courseController.ts`. *Controller* akan menerima urutan baru dari *request body* dan memperbarui urutan *submodul* di basis data.

Gambar 3.67. *Submodule reorder* berhasil

API `PATCH /cms/reorder_quiz`

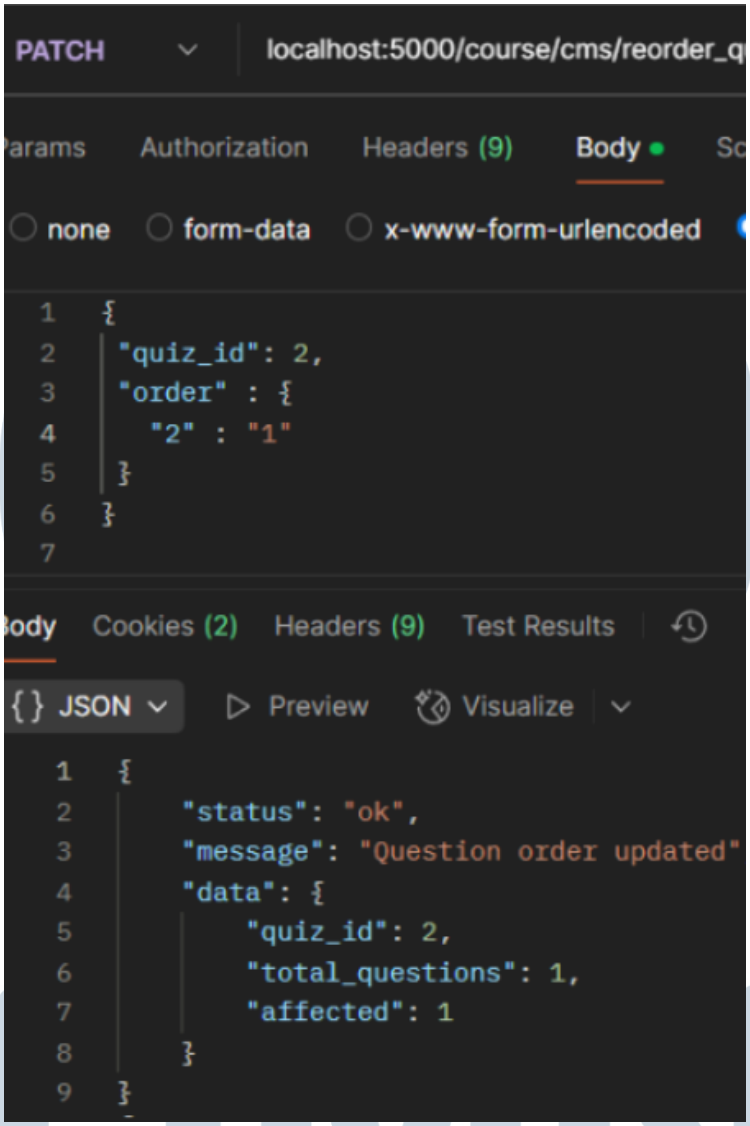
Endpoint `PATCH /cms/reorder_quiz` digunakan untuk mengubah urutan *quiz* dalam *modul* (hanya untuk *admin*). *Controller* yang digunakan adalah `reorderQuiz` pada berkas `courseController.ts`. *Controller* akan menerima urutan baru dari *request body* dan memperbarui urutan *quiz* di basis data.



Gambar 3.68. *Quiz reorder* berhasil

API `PATCH /cms/reorder_question`

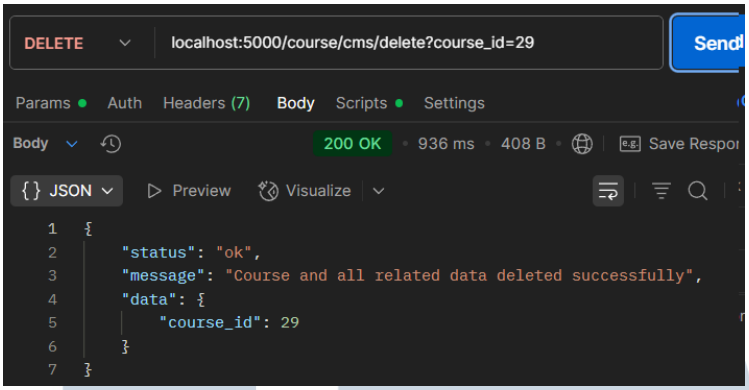
Endpoint `PATCH /cms/reorder_question` digunakan untuk mengubah urutan *soal* dalam *quiz* (hanya untuk *admin*). *Controller* yang digunakan adalah `reorderQuestion` pada berkas `courseController.ts`. *Controller* akan menerima urutan baru dari *request body* dan memperbarui urutan *soal* di basis data.



Gambar 3.69. *Question reorder* berhasil

API `DELETE /cms/delete`

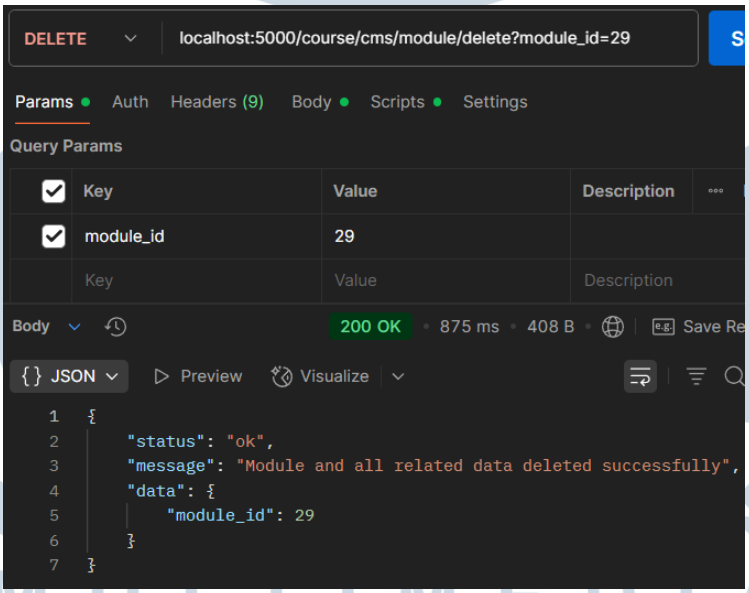
Endpoint `DELETE /cms/delete` digunakan untuk menghapus *course* dari sistem (hanya untuk *admin*). *Controller* yang digunakan adalah `deleteCourse` pada berkas `courseController.ts`. *Controller* akan menerima `course_id` dari *request body* atau parameter, melakukan validasi, dan menghapus *course* dari basis data.



Gambar 3.70. *Course delete* berhasil

API `DELETE /cms/module/delete`

Endpoint `DELETE /cms/module/delete` digunakan untuk menghapus *modul* dari *course* (hanya untuk *admin*). *Controller* yang digunakan adalah `deleteModule` pada berkas `courseController.ts`. *Controller* akan menerima `module_id` dari *request body* atau parameter, melakukan validasi, dan menghapus *modul* dari basis data.

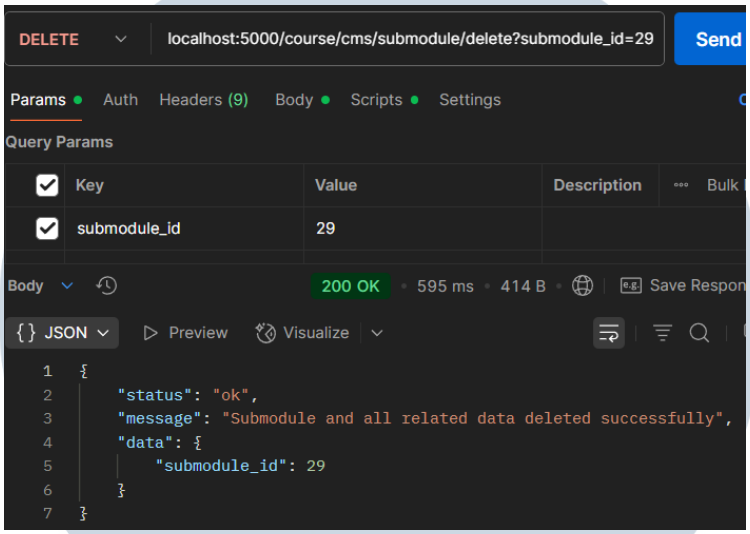


Gambar 3.71. *Module delete* berhasil

API `DELETE /cms/submodule/delete`

Endpoint `DELETE /cms/submodule/delete` digunakan untuk menghapus *submodul* dari *modul* (hanya untuk *admin*). *Controller* yang digunakan adalah

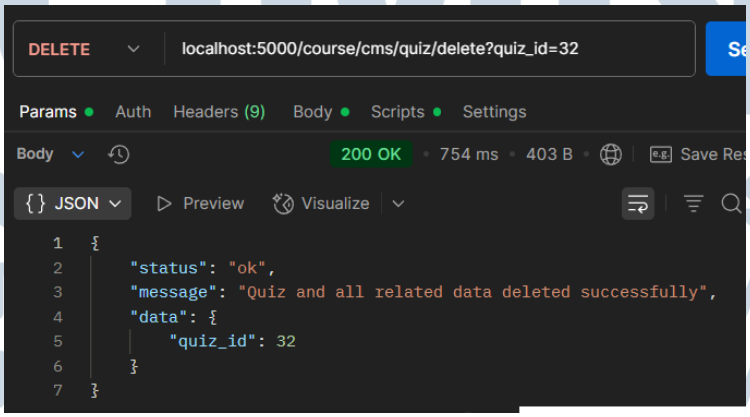
`deleteSubmodule` pada berkas `courseController.ts`. *Controller* akan menerima `submodule_id` dari *request body* atau parameter, melakukan validasi, dan menghapus *submodul* dari basis data.



Gambar 3.72. *Submodule delete* berhasil

API `DELETE /cms/quiz/delete`

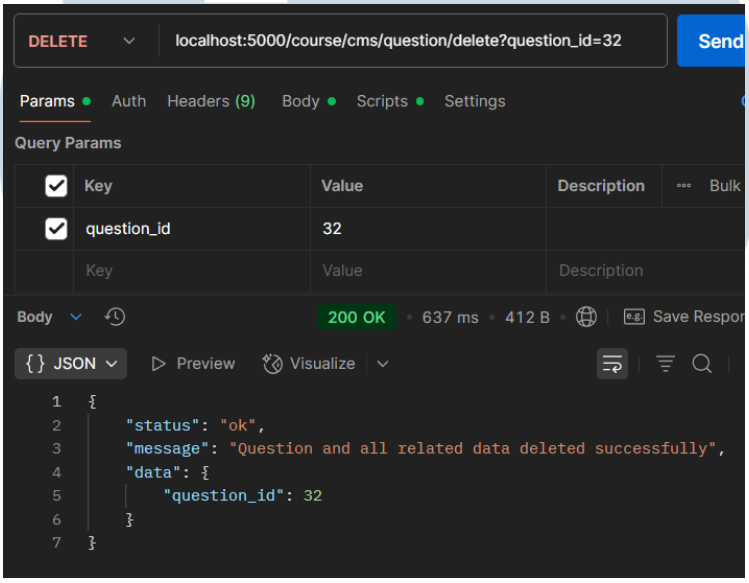
Endpoint `DELETE /cms/quiz/delete` digunakan untuk menghapus *quiz* dari *modul* (hanya untuk *admin*). *Controller* yang digunakan adalah `deleteQuiz` pada berkas `courseController.ts`. *Controller* akan menerima `quiz_id` dari *request body* atau parameter, melakukan validasi, dan menghapus *quiz* dari basis data.



Gambar 3.73. *Quiz delete* berhasil

API `DELETE /cms/question/delete`

Endpoint `DELETE /cms/question/delete` digunakan untuk menghapus *soal* dari *quiz* (hanya untuk *admin*). *Controller* yang digunakan adalah `deleteQuestion` pada berkas `courseController.ts`. *Controller* akan menerima `question_id` dari *request body* atau parameter, melakukan validasi, dan menghapus *soal* dari basis data.



Gambar 3.74. *Question delete* berhasil

Tabel 3.11. Tabel Pengujian API Course *(Whitebox)*

No	Endpoint	Skenario Pengujian (Whitebox)	Status
1	`POST /module/submodule_content`	Validasi input konten, simpan/update konten	Lulus
2	`POST /module/quiz/answer_question`	Validasi jawaban, cek user, update skor	Lulus
3	`POST /list`	Validasi input, tambah course ke daftar user	Lulus
4	`POST /enroll`	Validasi input, enroll user ke course	Lulus
5	`POST /cms/add`	Validasi data course baru, simpan ke database	Lulus
6	`POST /cms/edit`	Validasi data edit, update course di database	Lulus
7	`POST /cms/module/edit`	Validasi data edit, update modul di database	Lulus
8	`POST /cms/submodule/edit`	Validasi data edit, update submodul di database	Lulus
9	`POST /cms/quiz/edit`	Validasi data edit, update quiz di database	Lulus
10	`POST /cms/question/edit`	Validasi data edit, update soal di database	Lulus
11	`PATCH /cms/reorder_module`	Validasi urutan baru, update urutan modul	Lulus
12	`PATCH /cms/reorder_submodule`	Validasi urutan baru, update urutan submodul	Lulus
13	`PATCH /cms/reorder_quiz`	Validasi urutan baru, update urutan quiz	Lulus
14	`PATCH /cms/reorder_question`	Validasi urutan baru, update urutan soal	Lulus
15	`DELETE /cms/delete`	Validasi ID, hapus course dari database	Lulus
16	`DELETE /cms/module/delete`	Validasi ID, hapus modul dari database	Lulus
17	`DELETE /cms/submodule/delete`	Validasi ID, hapus submodul dari database	Lulus
18	`DELETE /cms/quiz/delete`	Validasi ID, hapus quiz dari database	Lulus
19	`DELETE /cms/question/delete`	Validasi ID, hapus soal dari database	Lulus

Tabel 3.12. Tabel Pengujian API Course *(Blackbox)*

No	Endpoint	Skenario Pengujian *(Blackbox)*	Status
1	`GET /field`	Ambil semua bidang course, tidak ada bidang di database	Lulus
2	`GET /detail/:course_id`	Ambil detail course ID valid, course ID tidak ada	Lulus
3	`GET /module/submodule/:course_id`	Ambil modul & submodul course ID valid, course ID tidak ada	Lulus
4	`POST /module/submodule_content`	Submit konten valid, submit konten tidak valid	Lulus
5	`POST /module/quiz/answer_question`	Jawab quiz valid, jawab quiz tidak valid	Lulus
6	`GET /module/quiz/question`	Ambil soal quiz, quiz tidak ada	Lulus
7	`GET /module/quiz`	Ambil daftar quiz, tidak ada quiz	Lulus
8	`GET /module`	Ambil daftar modul, tidak ada modul	Lulus
9	`GET /cms/list`	Ambil daftar course (CMS), tidak ada course	Lulus
10	`GET /cms/:course_id`	Ambil detail course (CMS) ID valid, course ID tidak ada	Lulus
11	`POST /list`	Tambah course valid, tambah course duplikat	Lulus
12	`POST /enroll`	Enroll course valid, enroll course sudah pernah	Lulus
13	`POST /cms/add`	Tambah course valid, tambah course duplikat	Lulus
14	`POST /cms/edit`	Edit course valid, edit course tidak valid	Lulus
15	`POST /cms/module/edit`	Edit modul valid, edit modul tidak valid	Lulus
16	`POST /cms/submodule/edit`	Edit submodul valid, edit submodul tidak valid	Lulus
17	`POST /cms/quiz/edit`	Edit quiz valid, edit quiz tidak valid	Lulus
18	`POST /cms/question/edit`	Edit soal valid, edit soal tidak valid	Lulus
19	`PATCH /cms/reorder_module`	Ubah urutan modul valid, ubah urutan tidak valid	Lulus
20	`PATCH /cms/reorder_submodule`	Ubah urutan submodul valid, ubah urutan tidak valid	Lulus
21	`PATCH /cms/reorder_quiz`	Ubah urutan quiz valid, ubah urutan tidak valid	Lulus
22	`PATCH /cms/reorder_question`	Ubah urutan soal valid, ubah urutan tidak valid	Lulus
23	`DELETE /cms/delete`	Hapus course valid, hapus course tidak ada	Lulus
24	`DELETE /cms/module/delete`	Hapus modul valid, hapus modul tidak ada	Lulus
25	`DELETE /cms/submodule/delete`	Hapus submodul valid, hapus submodul tidak ada	Lulus
26	`DELETE /cms/quiz/delete`	Hapus quiz valid, hapus quiz tidak ada	Lulus
27	`DELETE /cms/question/delete`	Hapus soal valid, hapus soal tidak ada	Lulus

3.3.6 Pembuatan Model dan API Attachment

Pada tahap ini, dilakukan perancangan model dan pengembangan berbagai *endpoint* (API) yang berkaitan dengan fitur *attachment* (unggah dan manajemen *file*) di sistem. Model *attachment* dirancang menggunakan *Sequelize* dan *PostgreSQL*, mencakup entitas utama seperti `AttachmentMeta` dan `AttachmentChunk` untuk mendukung penyimpanan *metadata file* dan data *file* secara terpisah (chunked upload). Pengembangan API dilakukan dengan *Express.js* dan *TypeScript*, sehingga seluruh proses *upload, download*, dan pengelolaan *file* dapat berjalan secara terstruktur dan terintegrasi dengan baik ke dalam sistem *backend*.

Setelah model selesai dibuat, dilakukan pengembangan berbagai *endpoint*

untuk kebutuhan pengguna dalam mengelola *file*, seperti menginisialisasi *metadata file*, mengunggah data *file* secara bertahap (*chunk*), menyelesaikan proses *upload*, mengambil *metadata file*, mengunduh *file*, serta menghapus *file*. Berikut adalah daftar API *attachment* yang diimplementasikan beserta penjelasan singkat dan jenis pengujian yang digunakan:

Tabel 3.13. Daftar API Attachment

No	Endpoint	Method	Deskripsi Singkat	Jenis Pengujian
1	`/metadata`	GET	Mengambil metadata semua *file*	*Blackbox*
2	`/:file_id`	GET	Mengunduh *file* berdasarkan `file_id`	*Blackbox*
3	`/upload_chunk_init`	POST	Inisialisasi metadata *file* sebelum *upload*	*Whitebox* & *Blackbox*
4	`/upload_chunk_data`	POST	Mengunggah data *file* per *chunk*	*Whitebox* & *Blackbox*
5	`/upload_chunk_finalize`	POST	Menyelesaikan proses *upload file*	*Whitebox* & *Blackbox*
6	`/:file_id`	DELETE	Menghapus *file* dan *metadata* berdasarkan `file_id`	*Whitebox* & *Blackbox*

Penjelasan Jenis Pengujian

Whitebox Testing

Pengujian dilakukan dengan mengetahui struktur internal kode, logika validasi, proses penyimpanan dan update data, serta integrasi antar model. Pada endpoint yang melakukan perubahan data (POST, DELETE), pengujian whitebox dilakukan untuk memastikan seluruh proses validasi, update, dan penghapusan data berjalan sesuai logika yang diharapkan, termasuk pengecekan urutan chunk, validasi ukuran file, dan konsistensi metadata.

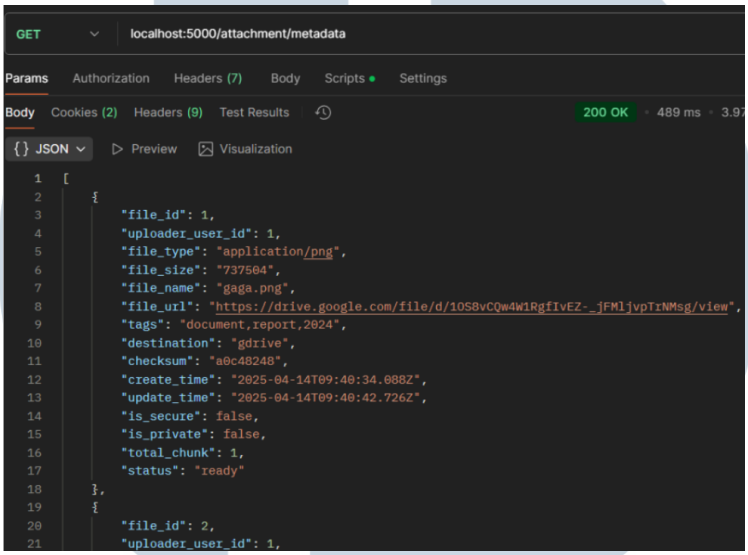
Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan. Pada endpoint yang hanya mengambil data (GET), pengujian blackbox dilakukan dengan berbagai skenario input dan memeriksa apakah response yang dihasilkan sesuai ekspektasi, seperti pengambilan metadata file, pengunduhan file, dan penanganan file yang tidak ditemukan.

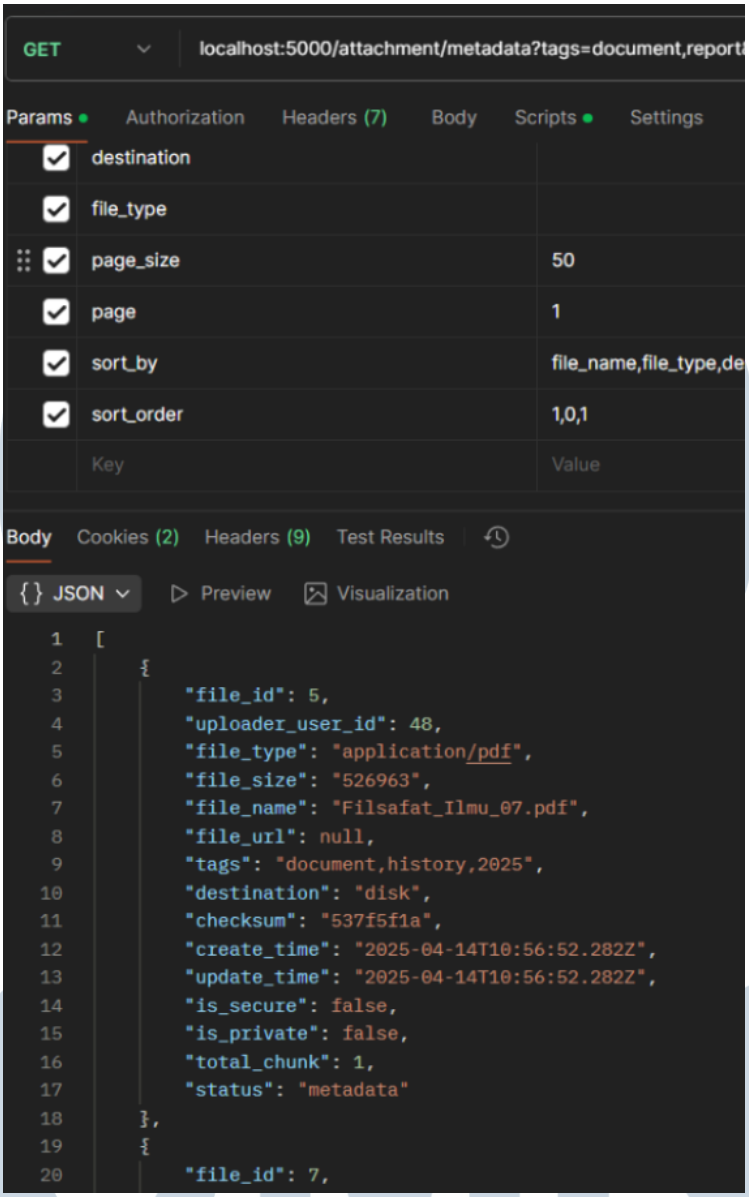
API `GET /metadata`

Endpoint `GET /metadata` digunakan untuk mengambil *metadata* dari seluruh *file attachment* yang telah diunggah ke sistem. Endpoint ini dapat menerima parameter pencarian seperti `tags`, `destination`, `file_type`, `file_name`, serta mendukung *pagination* dan *sorting*. *Controller* yang digunakan adalah

`getAttachmentMetadata` pada berkas `attachmentController.ts`. *Controller* ini akan mengambil data metadata file dari basis data `AttachmentMeta`, melakukan filter sesuai *query*, dan mengembalikannya dalam format array JSON. Endpoint ini penting untuk menampilkan daftar file yang tersedia beserta informasi detailnya.



Gambar 3.75. Mengambil daftar *metadata* yang ada di database

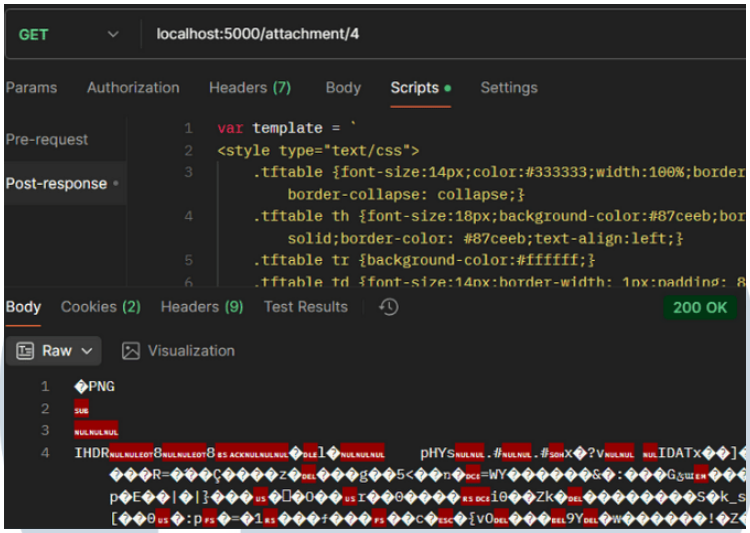


Gambar 3.76. Mengambil *metadata* dengan filter

API `GET /:file_id`

Endpoint `GET /:file_id` digunakan untuk mengunduh *file attachment* berdasarkan `file_id` tertentu. *Controller* yang digunakan adalah `getAttachment` pada berkas `attachmentController.ts`. *Controller* ini akan mencari metadata file berdasarkan `file_id`, memeriksa status file (apakah sudah siap diunduh), serta memverifikasi hak akses jika file bersifat *private* atau *secure*. Setelah validasi, file akan dikirimkan ke *client* dari basis data, disk, atau Google Drive sesuai lokasi penyimpanan. Jika file tidak ditemukan atau belum siap, maka akan dikembalikan

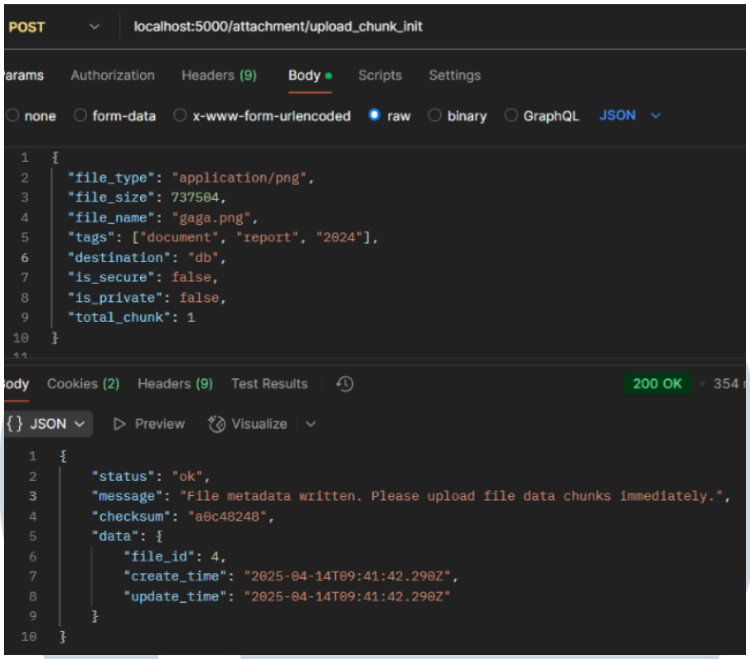
respons *error*.



Gambar 3.77. Mengambil *metadata* berdasarkan file id

API `POST /upload_chunk_init`

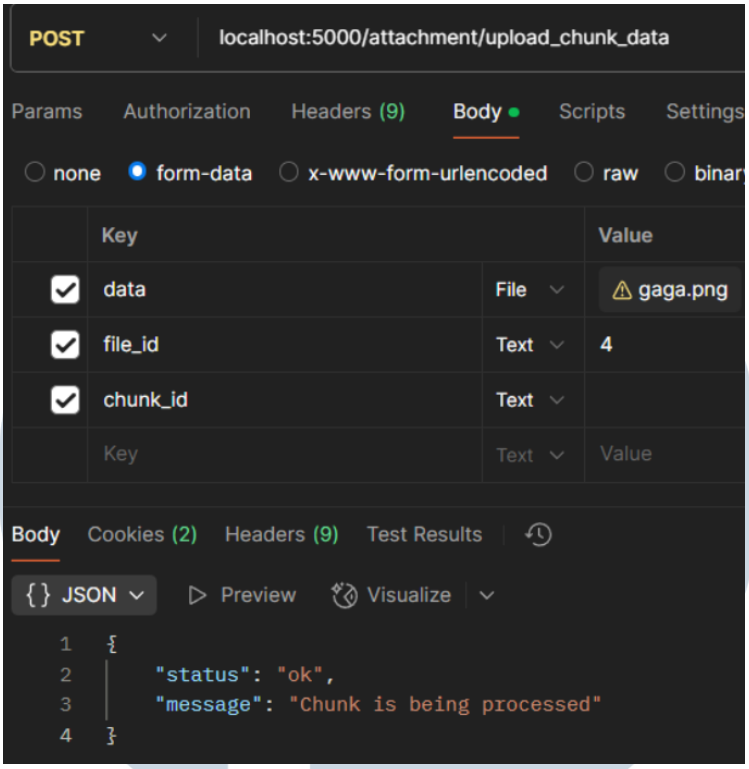
Endpoint `POST /upload_chunk_init` digunakan untuk menginisialisasi *metadata file* sebelum proses *chunked upload*. *Controller* yang digunakan adalah `uploadChunkInit` pada berkas `attachmentController.ts`. *Controller* ini menerima metadata file seperti nama, tipe, ukuran, tujuan penyimpanan, dan jumlah *chunk*, lalu menyimpan metadata tersebut ke basis data `AttachmentMeta` dan mengembalikan `file_id` untuk proses upload selanjutnya.



Gambar 3.78. Inisialisasi *upload chunk data*

API `POST /upload_chunk_data`

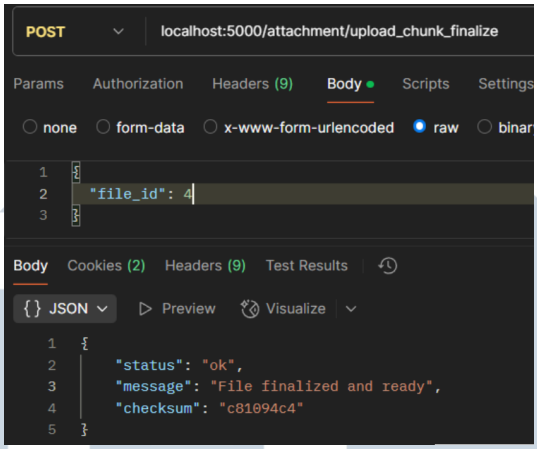
Endpoint `POST /upload_chunk_data` digunakan untuk mengunggah bagian file (*chunk*) berdasarkan `file_id` dan `chunk_id`. *Controller* yang digunakan adalah `uploadChunkData` pada berkas `attachmentController.ts`. *Controller* ini menerima data *chunk* melalui *form-data*, melakukan validasi urutan chunk, dan menyimpan setiap chunk ke basis data `AttachmentChunk` atau *storage* lain sesuai konfigurasi.



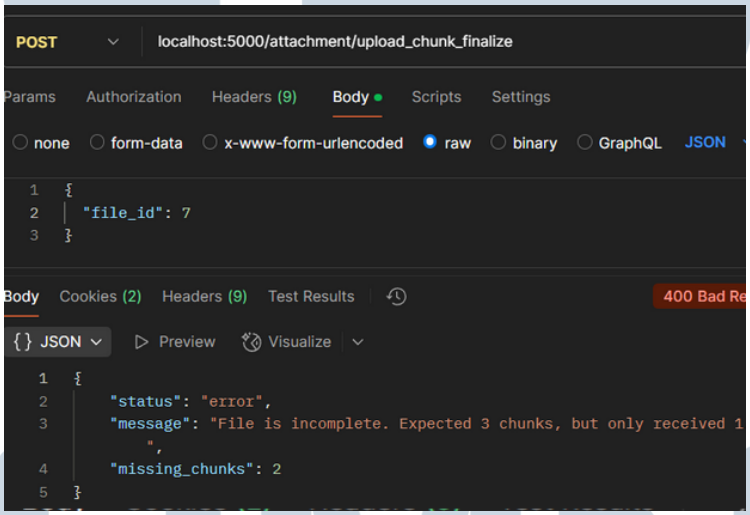
Gambar 3.79. Memulai proses *upload chunk data*

API `POST /upload_chunk_finalize`

Endpoint `POST /upload_chunk_finalize` digunakan untuk menyelesaikan proses upload setelah seluruh chunk berhasil diunggah. *Controller* yang digunakan adalah `uploadChunkFinalize` pada berkas `attachmentController.ts`. *Controller* ini akan memverifikasi bahwa semua chunk telah diterima, menggabungkannya menjadi satu file utuh, menghitung checksum untuk memastikan integritas file, memperbarui status file menjadi `ready`, dan menghapus data chunk sementara jika diperlukan.



Gambar 3.80. Menyelesaikan proses *upload chunk data*

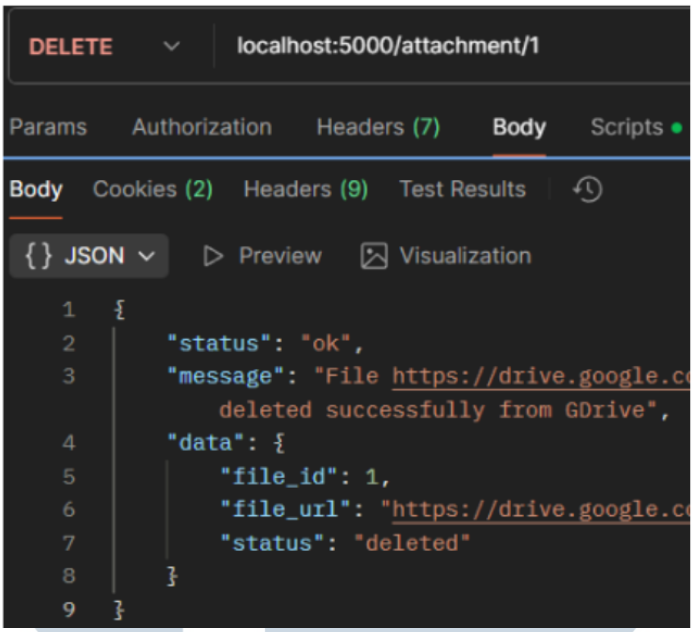


Gambar 3.81. Error dalam menyelesaikan proses *upload chunk data* karena ada chunk yang belum selesai ter-*upload*

API `DELETE /:file_id`

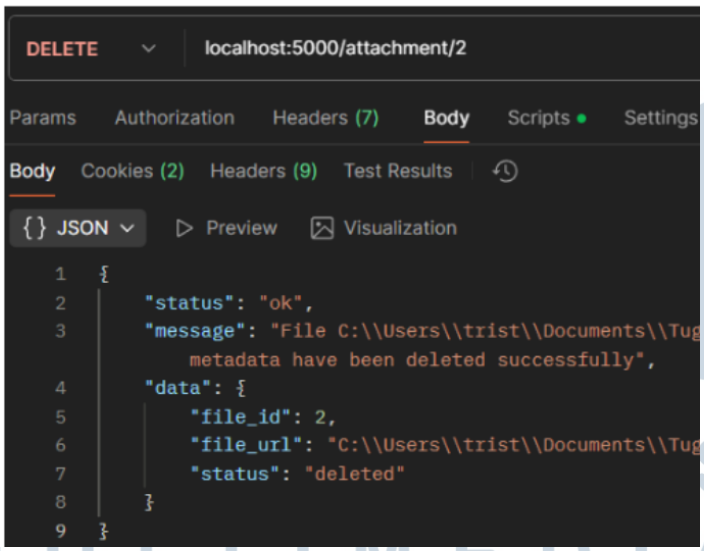
Endpoint `DELETE /:file_id` digunakan untuk menghapus file attachment beserta metadata dan seluruh chunk berdasarkan `file_id` tertentu. *Controller* yang digunakan adalah `deleteAttachment` pada berkas `attachmentController.ts`. *Controller* ini akan mencari metadata file, menghapus file dari *storage* (disk, basis data, atau Google Drive), serta menghapus metadata dan data chunk dari basis data. Endpoint ini penting untuk menjaga kebersihan dan keamanan data file di sistem.

Gdrive



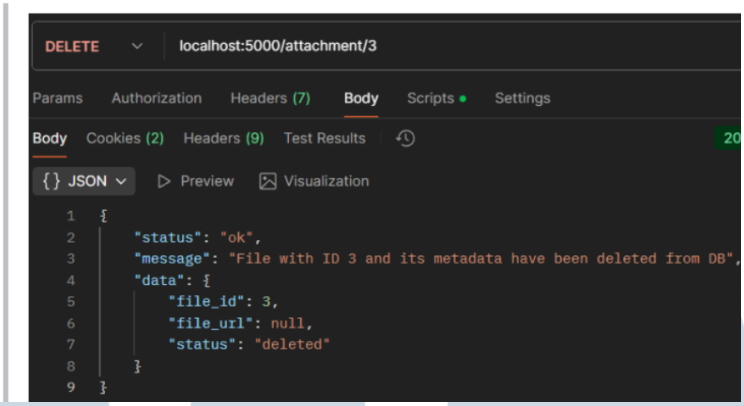
Gambar 3.82. *Delete file* dengan tipe *gdrive* berhasil

Disk



Gambar 3.83. *Delete file* dengan tipe *disk* berhasil

DB



Gambar 3.84. *Delete file* dengan tipe *db* berhasil

Tabel 3.14. Hasil Pengujian API Attachment *(Whitebox)*

No	Endpoint	Skenario Pengujian	Status
1	`POST /upload_chunk_init`	Validasi metadata file, simpan metadata, cek duplikasi file	Lulus
2	`POST /upload_chunk_data`	Validasi urutan chunk, simpan chunk, cek ukuran chunk	Lulus
3	`POST /upload_chunk_finalize`	Validasi semua chunk, gabungkan chunk, update status, validasi checksum	Lulus
4	`DELETE /:file_id`	Validasi file_id, hapus metadata, hapus semua chunk	Lulus

Tabel 3.15. Hasil Pengujian API Attachment *(Blackbox)*

No	Endpoint	Skenario Pengujian	Status
1	`GET /metadata`	Ambil metadata semua file, tidak ada file di database, cek format response	Lulus
2	`GET /:file_id`	Download file_id valid, file_id tidak ada, file sudah dihapus	Lulus
3	`POST /upload_chunk_init`	Inisialisasi upload valid, metadata tidak valid	Lulus
4	`POST /upload_chunk_data`	Upload chunk valid, urutan salah, data corrupt	Lulus
5	`POST /upload_chunk_finalize`	Finalisasi upload lengkap, chunk kurang, sudah difinalisasi	Lulus
6	`DELETE /:file_id`	Hapus file_id valid, file_id tidak ada, file sudah dihapus	Lulus

3.3.7 Pembuatan Model dan API Attendance

Pada tahap ini, dilakukan perancangan model dan pengembangan *endpoint* (API) yang berkaitan dengan fitur *attendance* (unggah dan manajemen *file*) di sistem. Model *attendance* dirancang menggunakan *Sequelize* dan *PostgreSQL*, mencakup entitas utama seperti `Attendance`. Pengembangan API dilakukan dengan *Express.js* dan *TypeScript*, sehingga proses pengambilan data absensi dapat berjalan secara terstruktur dan terintegrasi dengan baik ke dalam sistem *backend*.

Setelah model selesai dibuat, dilakukan pengembangan *endpoint* untuk kebutuhan mengambil data absensi dalam waktu rentang yang ditentukan di URL endpoint. Berikut adalah daftar API *attendance* yang diimplementasikan beserta penjelasan singkat dan jenis pengujian yang digunakan:

Tabel 3.16. Daftar API Attendance

No	Endpoint	Method	Deskripsi Singkat	Jenis Pengujian
1	`/:start_date/:end_date`	GET	Mengambil data absensi dalam rentang *start_date* ke *end_date*	*Blackbox*

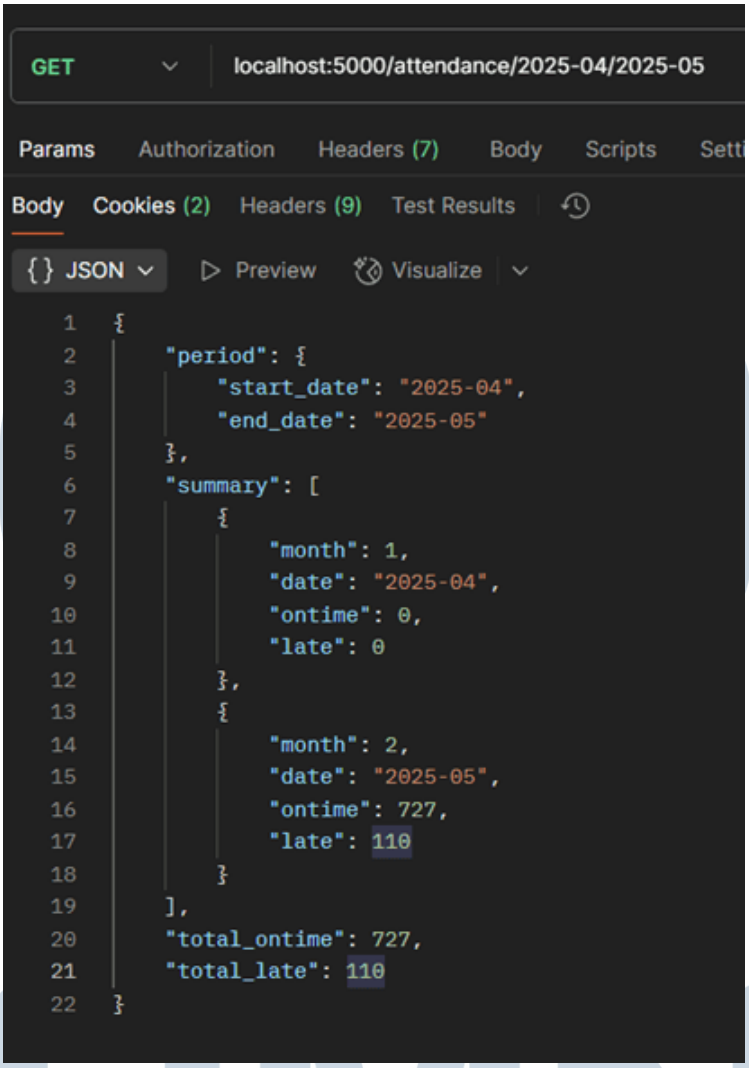
Penjelasan Jenis Pengujian

Blackbox Testing

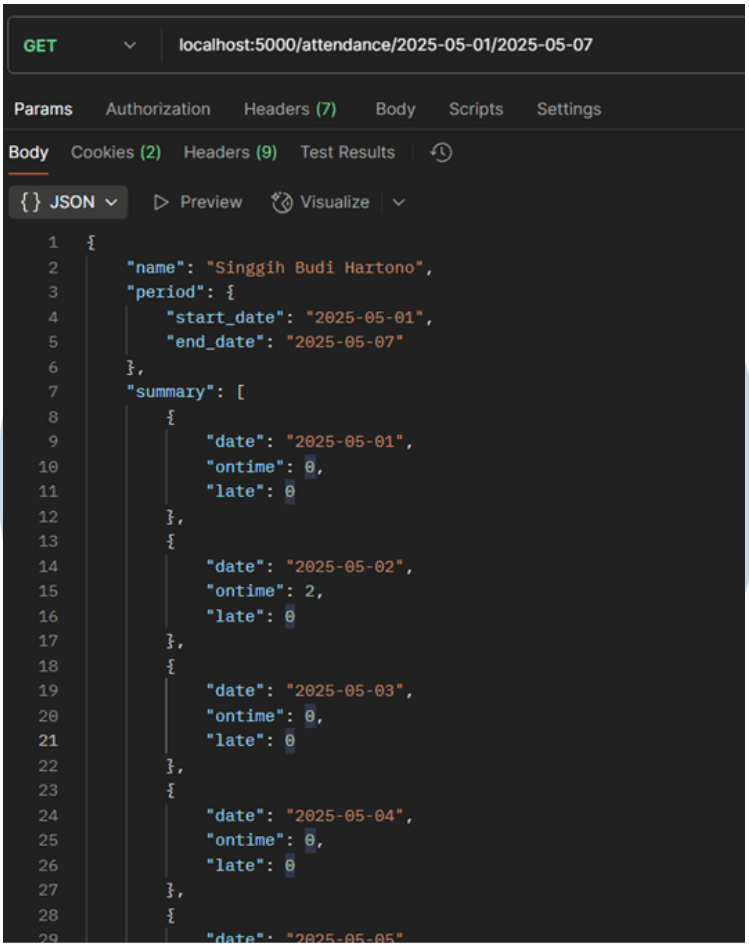
Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan *input* dan *output* yang dihasilkan. Pada endpoint yang hanya mengambil data (GET), pengujian blackbox dilakukan dengan berbagai skenario input dan memeriksa apakah response yang dihasilkan sesuai ekspektasi, seperti pengambilan absensi selama rentang waktu yang telah ditentukan di URL *endpoint*.

API `GET /:start_date/:end_date`

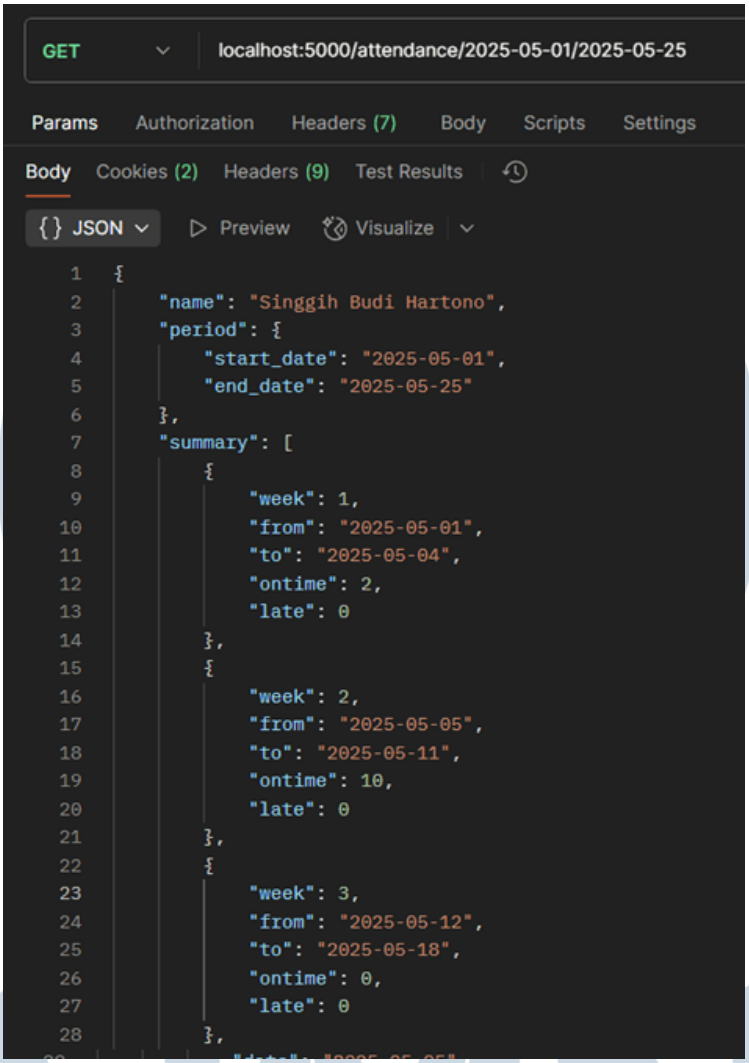
Endpoint `GET /:start_date/:end_date` digunakan untuk mengambil ringkasan data kehadiran (*attendance summary*) dari seorang pengguna atau seluruh pengguna (jika pengguna tersebut adalah *admin*) dalam periode waktu tertentu. Endpoint ini memerlukan *autentikasi JWT* dan pemeriksaan konsistensi *token*, lalu mengecek apakah pengguna yang mengakses adalah *admin* atau bukan. Format `start_date` dan `end_date` bisa berupa *YYYY-MM-DD* untuk mode harian atau mingguan, dan *YYYY-MM* untuk mode bulanan. Berdasarkan format dan rentang tanggal yang diberikan, sistem akan menentukan mode perhitungan: harian jika durasi di bawah 7 hari, mingguan jika lebih dari 7 hari, atau bulanan jika dalam format *YYYY-MM*. Untuk setiap periode, sistem akan menghitung jumlah kehadiran *ontime* (masuk sebelum jam 09:15) dan *late* (masuk setelah jam 09:15). Jika pengguna bukan *admin*, data hanya ditampilkan untuk dirinya sendiri. Hasil akhirnya adalah objek *JSON* yang berisi nama pengguna (jika bukan *admin*), periode, rekap kehadiran berdasarkan periode (*daily/weekly/monthly*), dan total kehadiran *ontime* serta *late*.



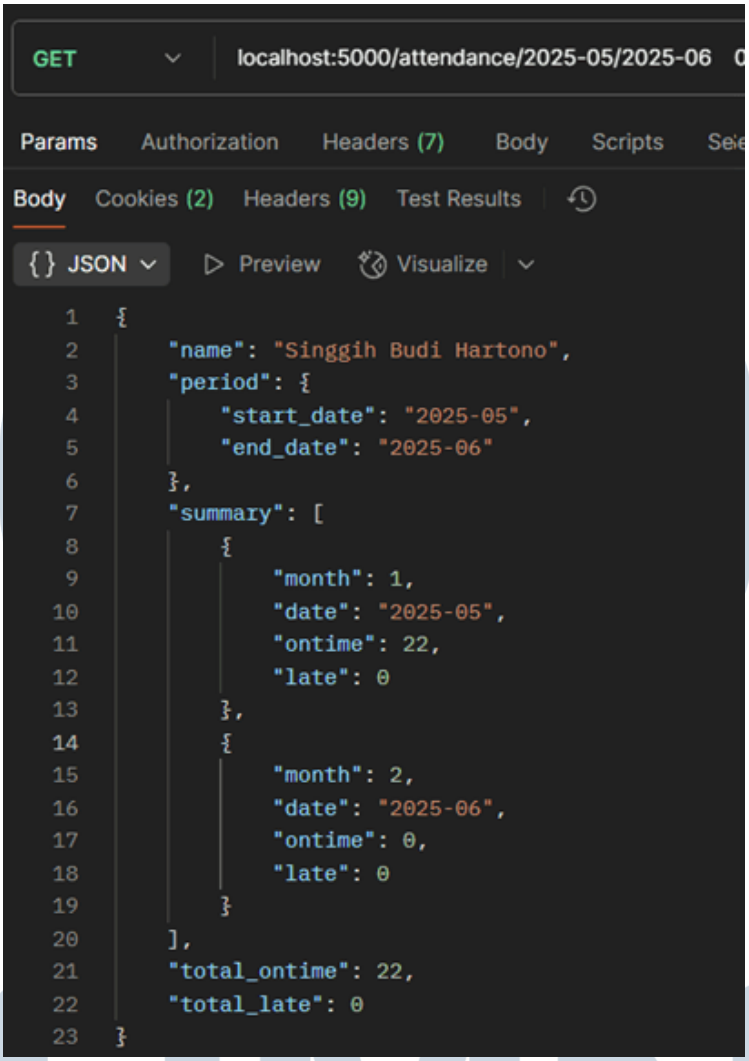
Gambar 3.85. Pengambilan total data absensi untuk *admin*



Gambar 3.86. Penampilan data absensi dari 1 Mei 2025 sampai 7 Mei 2025 (*daily*) untuk *employee*



Gambar 3.87. Penampilan data absensi dari selama 3 minggu dari 1 Mei 2025 sampai 25 Mei 2025 (*weekly*) untuk *employee*



Gambar 3.88. Penampilan data absensi dari selama 1 bulan dari 1 Mei 2025 sampai 1 Juni 2025 (*monthly*) untuk *employee*

Tabel 3.17. Hasil Pengujian API Attendance

No	Endpoint	Jenis Pengujian	Skenario Pengujian	Status
1	`GET /:start_date/:end_date`	*Blackbox*	Mengambil data absensi suatu karyawan dalam rentang waktu yang ditentukan di URL *endpoint*	Lulus

3.3.8 Pembuatan Model dan API Statistics

Pada tahap ini, dilakukan perancangan model dan pengembangan berbagai *endpoint* (API) yang berkaitan dengan fitur statistik dalam sistem, yang bertujuan untuk menyajikan data ringkasan dan laporan berdasarkan aktivitas pengguna, partisipasi dalam kursus, serta performa tugas yang dikirimkan. Model statistik

terintegrasi dengan database utama dan dikembangkan menggunakan *Express.js* serta *TypeScript* untuk menangani logika dan pemrosesan data secara efisien.

API yang dikembangkan mencakup berbagai jenis laporan dan metrik, seperti ringkasan aktivitas pengguna, data kursus yang diikuti, jumlah proyek yang diikuti, tampilan dashboard profil, hingga data pengumpulan tugas dalam rentang waktu tertentu. Seluruh endpoint ini dirancang agar dapat memberikan data yang relevan dan siap digunakan untuk keperluan visualisasi atau analisis oleh frontend. Berikut adalah daftar API *statistics* yang diimplementasikan beserta penjelasan singkat dan jenis pengujian yang digunakan:

Tabel 3.18. Daftar API Statistics

No	Endpoint	Method	Deskripsi Singkat	Jenis Pengujian
1	`/report_user_summary`	GET	Menyediakan ringkasan umum aktivitas pengguna dalam sistem	Blackbox
2	`/enrolled_course`	GET	Mengambil daftar kursus yang telah diikuti pengguna	Blackbox
3	`/project_enrolled`	GET	Menyajikan data proyek yang diikuti oleh pengguna	Blackbox
4	`/profile_dashboard`	GET	Memberikan data ringkasan yang ditampilkan pada halaman dashboard pengguna	Blackbox
5	`/course_enrolled`	GET	Mengambil detail statistik kursus yang telah diikuti	Blackbox
6	`/task_delivered/:start_date/:end_date`	GET	Menyajikan data tugas yang telah dikirimkan dalam rentang waktu tertentu	Whitebox & Blackbox

Penjelasan Jenis Pengujian

Blackbox Testing

Pengujian dilakukan dengan fokus pada input dan output dari endpoint, tanpa mengetahui implementasi logika internalnya. Digunakan pada endpoint seperti `/report_user_summary`, `/enrolled_course`, `/project_enrolled`, `/profile_dashboard`, dan `/course_enrolled`, yang hanya melakukan pengambilan data dari database dan menyajikannya dalam format JSON.

Whitebox & Blackbox Testing

Endpoint `/task_delivered/:start_date/:end_date` memerlukan logika tambahan seperti filter berdasarkan tanggal dan pengolahan data tugas yang dikirim. Oleh karena itu, dilakukan pengujian whitebox untuk memastikan proses filtering dan agregasi data berjalan sesuai, serta pengujian blackbox untuk memverifikasi bahwa hasil yang dikembalikan sesuai dengan input yang diberikan oleh pengguna.

Feeding Data Gitlab dan Attendance

Sebelum membuat API yang dibutuhkan untuk statistika performa *employee*, dibutuhkan data asli dari GitLab yang mencatat aktivitas karyawan selama masa kerjanya. Salah satu informasi penting yang diambil dari GitLab adalah data *git_raw_issues*, yaitu daftar tugas atau *issues* yang dikerjakan oleh setiap karyawan selama masa jabatannya di Braincode.

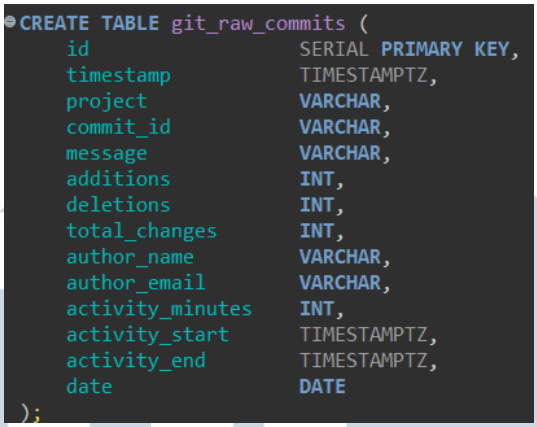
Untuk mendapatkan data ini secara efisien dan rutin, dibutuhkan proses otomasi pemasukan data ke dalam database PostgreSQL. Solusi yang digunakan adalah membuat dua buah *script* Python yang bekerja untuk proses *data feeding*, yakni:

- **feeding_issues.py**: berfungsi untuk mengambil data *issues* dari GitLab API dan langsung menyimpannya ke tabel *git_raw_issues* di database PostgreSQL.
- **import_to_postgre.py**: berfungsi untuk mengimpor data *issues* maupun *commits* dari file lokal (berformat `.json`, `.csv`, atau `.zip`) ke dalam tabel *git_raw_issues* dan *git_raw_commits*.

feeding_issues.py bekerja dengan mengambil daftar `project` dari tabel `git_raw_commits`, lalu mengambil semua `issues` dari GitLab API berdasarkan masing-masing project tersebut.

```
CREATE TABLE git_raw_issues (
    id                  BIGINT PRIMARY KEY,
    iid                 INT,
    project_id          BIGINT,
    title               TEXT,
    description         TEXT,
    state               VARCHAR,
    created_at          TIMESTAMPTZ,
    updated_at          TIMESTAMPTZ,
    closed_at           TIMESTAMPTZ,
    labels              TEXT[], -- array
    assignee_username   VARCHAR,
    assignee_name       VARCHAR,
    author_username     VARCHAR,
    author_name         VARCHAR,
    web_url             TEXT,
    raw_json            JSONB -- menyimpan
);
```

Gambar 3.89. Struktur Tabel *git_raw_issues*



```
CREATE TABLE git_raw_commits (
    id                  SERIAL PRIMARY KEY,
    timestamp           TIMESTAMPTZ,
    project             VARCHAR,
    commit_id           VARCHAR,
    message             VARCHAR,
    additions           INT,
    deletions           INT,
    total_changes       INT,
    author_name         VARCHAR,
    author_email        VARCHAR,
    activity_minutes    INT,
    activity_start      TIMESTAMPTZ,
    activity_end        TIMESTAMPTZ,
    date                DATE
);
```

Gambar 3.90. Struktur Tabel *git_raw_commits*

Data `issues` yang berhasil diambil kemudian dimasukkan ke dalam tabel `git_raw_issues` dengan query `INSERT` yang dilengkapi pengecekan duplikat menggunakan `ON CONFLICT DO NOTHING`. Script ini mendukung parameter tambahan berupa waktu tertentu agar hanya mengambil `issues` yang diupdate setelah waktu tersebut.

```
def import_json(filepath):
    with open(filepath, 'r', encoding='utf-8') as f:
        data = json.load(f)
        items = data if isinstance(data, list) else [data]

        for issue in items:
            assignee = issue.get("assignee") or {}
            author = issue.get("author") or {}

            cur.execute("""
                INSERT INTO git_raw_issues (
                    id, iid, project_id, title, description, state,
                    created_at, updated_at, closed_at, labels,
                    assignee_username, assignee_name,
                    author_username, author_name,
                    web_url, raw_json
                ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                ON CONFLICT (id) DO NOTHING
            """, (
                issue["id"],
                issue["iid"],
                issue["project_id"],
                issue["title"],
                issue["description"],
                issue["state"],
                issue["created_at"],
                issue["updated_at"],
                issue.get("closed_at"),
                issue.get("labels", []),
                assignee.get("username"),
                assignee.get("name"),
                author.get("username"),
                author.get("name"),
                issue["web_url"],
                json.dumps(issue)
            ))
```

Gambar 3.91. *Query Insert* ke PostgreSQL dengan pengecekan duplikat

Eksekusi `feeding_issues.py` dapat dijalankan secara berkala menggunakan `crontab` di Linux server agar proses *data feeding* berlangsung secara otomatis dan terjadwal.

import_to_postgre.py digunakan untuk proses *data feeding* manual dari file lokal. Script ini mendukung tiga jenis file:

1. File `.csv` untuk `git_raw_commits` yang berasal dari GitLab API.
2. File `.json` untuk `git_raw_issues` yang diambil dari GitLab API.
3. File `.zip` yang berisi gabungan file JSON dan CSV.

Script ini akan secara otomatis mendeteksi jenis file, melakukan ekstraksi jika berbentuk ZIP, lalu menjalankan proses import ke tabel yang sesuai. Struktur data yang dimasukkan ke dalam tabel sudah sesuai dengan skema kolom yang dimiliki masing-masing tabel. Script ini juga dilengkapi dengan validasi jumlah kolom agar tidak terjadi error selama proses insert data.

Selain data aktivitas GitLab, sistem juga membutuhkan data kehadiran karyawan sebagai bagian dari indikator performa. Data ini diperoleh melalui proses **feeding attendance** menggunakan script Python `attendance_import.py`. Script ini mengambil file Excel berisi data kehadiran dari folder lokal, memproses file tersebut menggunakan pustaka `openpyxl`, dan menyimpannya ke tabel PostgreSQL yang telah disiapkan, misalnya tabel `attendance_records`.

Skrip attendance_import.py merupakan program Python yang berfungsi untuk mengolah dan mengimpor data kehadiran dari *file CSV* ke dalam sistem *database* PostgreSQL.

```
create table attendance (
    id bigint primary key,
    name text,
    time_in TIMESTAMPTZ,
    time_out TIMESTAMPTZ,
    status boolean
);
```

Gambar 3.92. Tabel *Attendance*

File CSV yang digunakan harus berisi kolom *name, date,* dan *time*, di mana *name* merupakan *username* singkat yang akan dipetakan ke nama lengkap karyawan

melalui dictionary *USERNAME_TO_FULLNAME* yang telah ditentukan di dalam skrip. Dengan demikian, data absensi yang awalnya hanya mencantumkan *username* dapat dikonversi menjadi data kehadiran berdasarkan nama lengkap yang sesuai dengan data sistem.

```
# --- MAPPING USERNAME KE FULL NAME ---
USERNAME_TO_FULLNAME = {
    "rendy": "Ade Esarrendy Intris",
    "affan": "Affan Abdillah",
    "alan": "Alan Setiawan",
    "andi": "Andi Candra",
    "alvina": "Alvina Tahta Indal Karim",
    "arya": "Arya Zafri",
    "bagus": "Bagus Triwibowo",
    "daffin": "Dhafin Zhilal Syaikoh Adz-Dzaki",
    "rara": "Demitriyana Zachra Humaira",
    "dzul": "Dzulfikar Al Ansari",
    "fadhil": "Fadhil Budi Prasetya",
    "genta": "Genta Gilang Galuh",
    "indra": "Girindra Wijaya",
    "ihza": "Ihza Dzikri Fauzi",
    "ilham": "Ilham Nur Ramadhan",
    "intan": "Intan Puji Astuti",
    "maruf": "Maruf Salaffudin",
    "maya": "Ulfa Maya Syafira",
    "alfat": "Miftakh Alfath Nurullah",
    "zaki": "Muhammad Zaki",
    "altino": "Muhammad Rizki Altino",
    "nanda": "Nanda Aisyiah",
    "rakan": "Rakan Fawaz",
    "rafli": "Rafli Raihan Zahrandika",
    "ramdan": "Ramdani",
    "roypiq": "Roypik Al Imron",
    "salim": "Salim Lukman Daroni",
    "singgih": "Singgih Budi Hartono",
    "dede": "Dede Herman",
    "tri": "Rizki Triharyo Ibrahim",
    "aries": "Aries Budi Kurniawan",
}
```

Gambar 3.93. Pemetaaan Nama Karyawan dari *username*

Proses kerja skrip ini dimulai dengan mendeteksi *delimiter* CSV secara otomatis, kemudian membaca data menggunakan pustaka pandas. Setelah kolom divalidasi, data dikelompokkan berdasarkan kombinasi username dan tanggal. Untuk setiap grup, waktu kehadiran pertama *(time in)* dan terakhir *(time out)* akan dihitung. Skrip juga menentukan status kehadiran berdasarkan apakah pengguna datang sebelum atau setelah batas waktu yang ditentukan (yaitu pukul 09:15).

```python
# Tentukan status (true = on time, false = late)
batas_masuk = datetime.combine(time_in.date(), time(9, 15))
status = time_in <= batas_masuk

unique_id = generate_unique_id(full_name, time_in.date())

cur.execute("""
    INSERT INTO attendance (id, name, time_in, time_out, status)
    VALUES (%s, %s, %s, %s, %s)
    ON CONFLICT (id) DO UPDATE SET
        name = EXCLUDED.name,
        time_in = EXCLUDED.time_in,
        time_out = EXCLUDED.time_out,
        status = EXCLUDED.status
""", (unique_id, full_name, time_in, time_out, status))

inserted_count += 1
status_str = "✅ ON TIME" if status else "⚠️ TERLAMBAT"
print(f"{status_str} - {full_name}: {time_in} - {time_out}")
```

Gambar 3.94. *Insert Query Attendance*

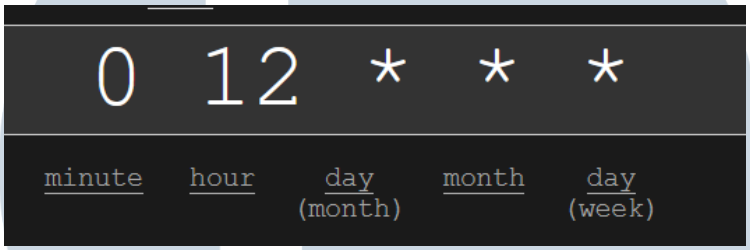
Data tersebut kemudian disimpan ke dalam tabel attendance di database PostgreSQL. Penanda unik (id) untuk setiap entri absensi dihasilkan menggunakan hash MD5 dari kombinasi nama lengkap dan tanggal. Penyimpanan dilakukan menggunakan query INSERT ... ON CONFLICT, yang memungkinkan pembaruan data jika entri sudah ada sebelumnya.

Skrip ini juga melakukan validasi tambahan di akhir proses untuk memastikan tidak ada nilai status yang bernilai NULL di database. Selain itu, skrip menangani berbagai kemungkinan error, seperti username yang tidak dikenal (data akan dilewati) dan error saat memproses atau menyimpan data (dicetak sebagai error di terminal). Saat dijalankan, skrip secara otomatis akan mencari file CSV terbaru di direktori tempat skrip berada dan memproses file tersebut. Dengan pendekatan ini, pengelolaan data kehadiran menjadi lebih efisien dan dapat diotomatisasi dalam sistem LMS perusahaan.

```python
# Tambahkan query untuk validasi status NULL
cur.execute("""
    UPDATE attendance
    SET status = time_in <= (date_trunc('day', time_in) + interval '9 hours 15 minutes')
    WHERE status IS NULL
""")

conn.commit()
cur.close()
conn.close()

print(f"\n🎉 Selesai!")
print(f"✅ Berhasil diimpor: {inserted_count} entri")
print(f"❌ Error: {error_count} entri")
print(f"⏭️  Dilewati: {skipped_count} entri")
```

Gambar 3.95. *Null Status Checker*

Agar proses *data feeding* berlangsung secara otomatis dan terjadwal, kedua script utama — yaitu `feeding_issues.py` dan `attendance_import.py` — dijalankan menggunakan `crontab` pada server Linux. Dengan pendekatan ini, sistem mampu terus memperbarui data dari GitLab (setiap jam 12 siang) dan absensi (setiap jam 16:24) secara berkala tanpa intervensi manual, menjadikan proses pemutakhiran data lebih andal dan efisien.



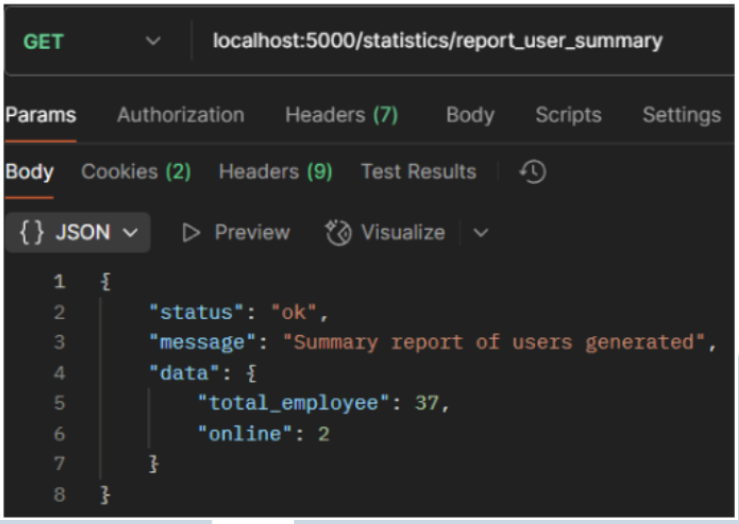
Gambar 3.96. Crontab Format

```
0 12 * * * /usr/bin/python3 /home/app/lms/scripts/import_commit/import_to_postgre.py
>> /home/app/lms/scripts/import_commit/import_to_postgre.log 2>&1
0 12 * * * /usr/bin/python3 /home/app/lms/scripts/feeding_issues/feeding_issues.py >>
 /home/app/lms/scripts/feeding_issues/feeding_issues.log 2>&1
24 16 * * * /usr/bin/python3 /home/app/lms/scripts/attendance/attendance_import.py >>
 /home/app/lms/scripts/attendance/attendance_import.log 2>&1
```

Gambar 3.97. Crontab Feeding

`GET /report_user_summary`

Endpoint ini digunakan untuk mengambil data ringkasan aktivitas pengguna dalam sistem. Endpoint ini menyajikan metrik seperti jumlah kursus yang diikuti, jumlah proyek yang diikuti, dan jumlah tugas yang telah dikumpulkan. Data yang dihasilkan digunakan untuk menampilkan informasi umum pada halaman pengguna, dan tidak memerlukan parameter tambahan selain autentikasi pengguna.



Gambar 3.98. API Report User Summary

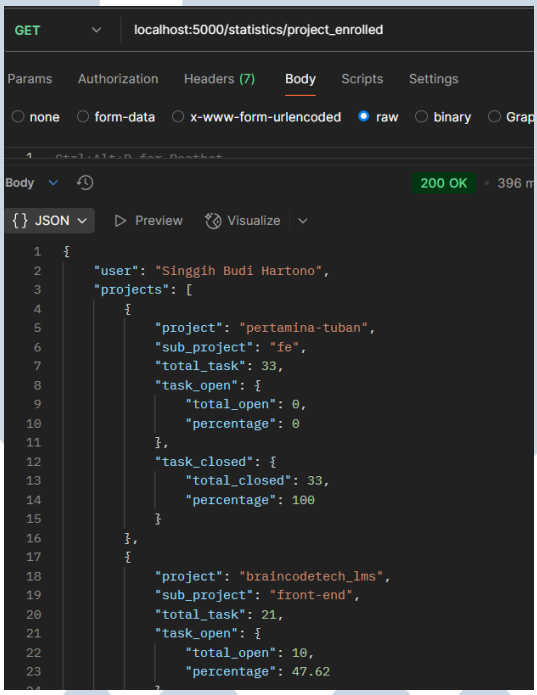
GET /enrolled_course

Endpoint ini berfungsi untuk menampilkan daftar kursus yang telah diikuti oleh pengguna. Setiap item dalam daftar memuat informasi dasar seperti nama kursus, instruktur, serta tanggal bergabung. Endpoint ini digunakan pada bagian manajemen kursus pengguna dan berguna untuk melacak partisipasi mereka dalam proses pembelajaran.

Gambar 3.99. API Enrolled Course

GET /project_enrolled

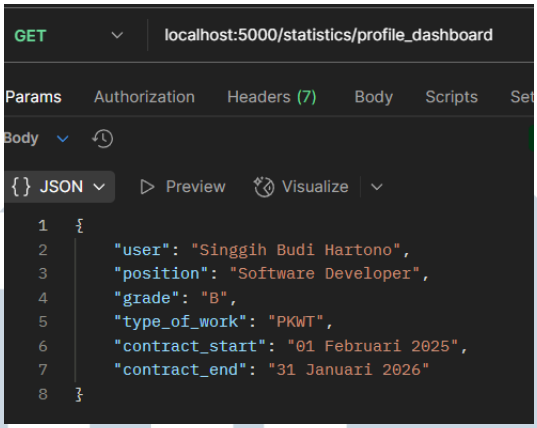
Endpoint ini menyajikan informasi tentang proyek-proyek yang telah diikuti oleh pengguna. Fungsinya mirip dengan `/enrolled_course`, namun fokus pada entitas proyek. Data ini ditampilkan dalam halaman proyek atau portofolio pengguna sebagai bentuk rekam jejak kontribusi mereka.



Gambar 3.100. Project Enrolled - Employee

GET /profile_dashboard

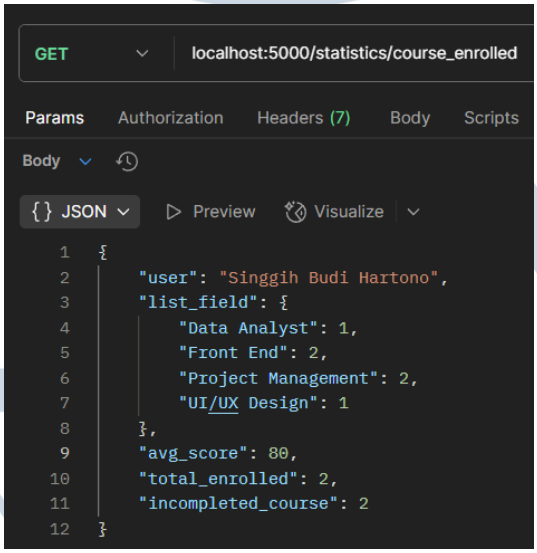
Endpoint ini merupakan endpoint agregatif, yang menyatukan berbagai data statistik untuk keperluan tampilan dashboard profil pengguna. Endpoint ini mengambil informasi dari berbagai sumber, seperti jumlah total kursus yang diikuti, tugas yang dikumpulkan, dan proyek yang dikerjakan, sehingga dapat memberikan tampilan ringkas mengenai aktivitas pengguna secara keseluruhan.



Gambar 3.101. User Dashboard

GET /course_enrolled

Endpoint ini digunakan untuk mengambil informasi detail dari kursus yang sedang atau telah diikuti oleh pengguna. Data ini dapat mencakup progress belajar, status penyelesaian, serta statistik lain yang relevan untuk mendukung fitur pelacakan pembelajaran.

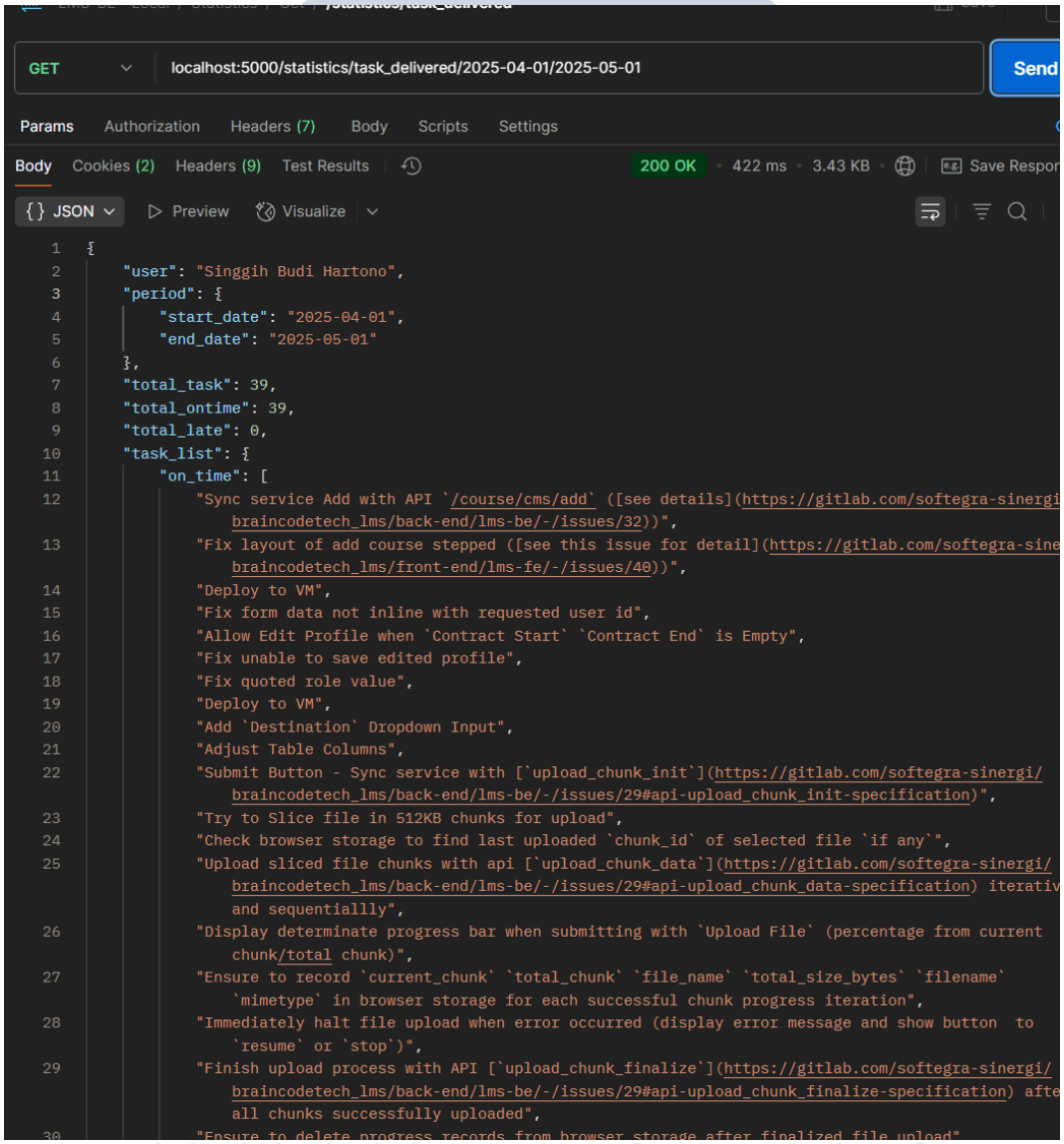


Gambar 3.102. Course Enrolled Employee

GET /task_delivered/:start_date/:end_date

Endpoint ini digunakan untuk mengambil daftar tugas yang telah dikirimkan oleh pengguna dalam rentang tanggal tertentu. Endpoint ini memiliki validasi

tambahan untuk memastikan bahwa parameter tanggal valid dan bahwa `start_date` tidak melebihi `end_date`. Endpoint ini sangat berguna untuk analisis produktivitas pengguna dalam periode waktu tertentu dan sering digunakan dalam laporan berkala.



Gambar 3.103. Task Delivered Employee

Tabel 3.19. Hasil Pengujian API Statistics *(Whitebox)*

No	Endpoint	Skenario Pengujian	Status
1	`GET /task_delivered/:start_date/:end_date`	Validasi format tanggal start dan end	Lulus
		Validasi bahwa start ¡ end, jika tidak maka error	Lulus
		Filter tanggal diterapkan secara benar di query database	Lulus

Tabel 3.20. Hasil Pengujian API Statistics *(Blackbox)*

No	Endpoint	Skenario Pengujian	Status
1	GET /report_user_summary	Akses endpoint tanpa autentikasi (jika dibutuhkan autentikasi)	Lulus
		Akses dengan pengguna valid dan autentikasi benar	Lulus
		Respons memuat ringkasan data pengguna secara lengkap	Lulus
		Pengguna baru tanpa data, respons tetap valid (kosong/nol)	Lulus
2	GET /enrolled_course	Pengguna belum mengikuti kursus, respons berupa array kosong	Lulus
		Pengguna memiliki kursus, data muncul lengkap	Lulus
		Respons memiliki atribut lengkap (judul, waktu, dll)	Lulus
		Token tidak valid menghasilkan 401 Unauthorized	Lulus
3	GET /project_enrolled	Tidak ada proyek yang diikuti, respons array kosong	Lulus
		Ada proyek yang diikuti, respons list proyek valid	Lulus
		Simulasi error database menghasilkan 500 Internal Server Error	Lulus
4	GET /profile_dashboard	Data ringkasan pengguna ditampilkan lengkap	Lulus
		Respons cepat dan ringan (waktu ¡ 500ms)	Lulus
		Struktur JSON sesuai skema frontend	Lulus
		Token tidak valid menghasilkan 401 Unauthorized	Lulus
5	GET /course_enrolled	Pengguna belum mengikuti kursus, respons kosong	Lulus
		Kursus memiliki progress, field progress muncul benar	Lulus
		Kursus tidak ditemukan, respons 404 Not Found	Lulus
6	GET /task_delivered/:start_date/:end_date	Tanggal valid, data tugas dikembalikan	Lulus
		Tidak ada data dalam rentang waktu, respons array kosong	Lulus
		Tanggal format aneh (ISO/local), tetap bisa diproses	Lulus
		Respons tetap optimal meskipun banyak data (stress test)	Lulus

3.4 Kendala dan Solusi yang Ditemukan

Kendala

Kendala yang ditemukan selama proses kerja magang di Braincode adalah sebagai berikut:

1. Dikarenakan LMS Braincode adalah proyek internal, yang berarti bukan prioritas penting kantor, storage space server testing yang disediakan oleh kantor untuk proyek LMS ini rentan terjadi overloading, sehingga untuk deploy service ke server menjadi sulit kadang-kadang.
2. Di sprint terakhir banyak task yang dituntut oleh PM untuk diselesaikan agar cepat selesai proyek LMS sebelum goal date.

Solusi

Beberapa solusi yang ditemukan untuk mengatasi beberapa kendala yang terjadi selama kerja magang tersebut adalah sebgai berikut:

1. Menggunakan localhost untuk testing kinerja API dengan data dummy yang dibuat di database lokal.
2. Lembur di kantor untuk mengurangi task yang berlebihan sehingga sprint dalam minggu tersebut dapat diselesaikan segera.