

# Simple Blog

Frontend dan Backend developer dapat melakukan :

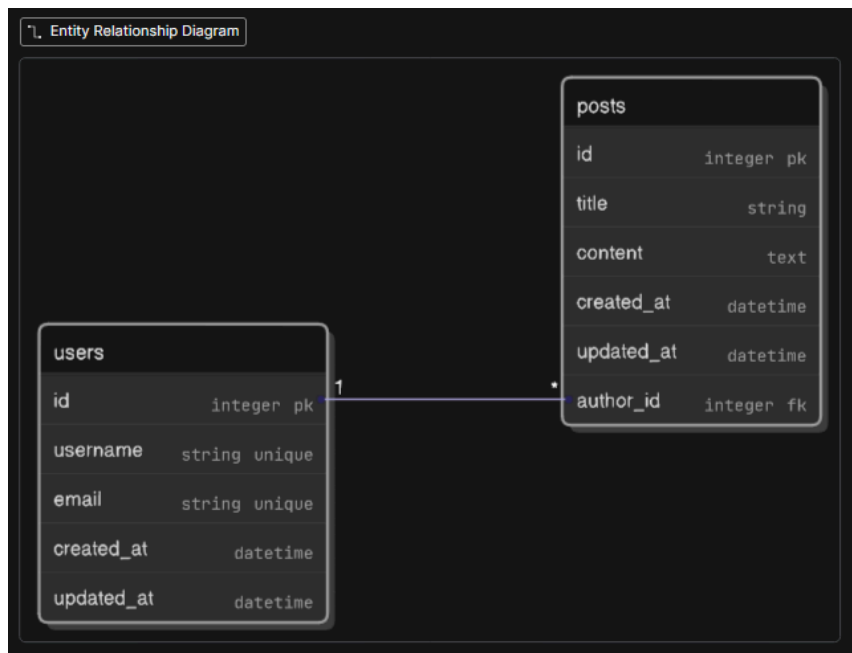
**Frontend:** Sebagai acuan untuk mengetahui endpoint mana yang harus diakses, data apa yang harus dikirim, dan format respons apa yang akan diterima.

**Backend:** Sebagai panduan untuk membangun server, struktur database, dan rute API sesuai dengan kontrak yang telah ditentukan.

## User Stories

- **Sebagai Pengunjung (Visitor):**
  - Saya ingin melihat daftar semua artikel agar saya bisa memilih artikel mana yang menarik untuk dibaca.
  - Saya ingin melihat siapa penulis dari setiap artikel agar saya bisa mengenali karya penulis favorit saya.
- **Sebagai Penulis (Admin):**
  - Saya ingin bisa membuat artikel baru (judul dan isi) agar dapat berbagi konten dengan para pengunjung.

## ERD



Note : Updated\_at tidak usah dimasukkan ke properties tiap entity ya

# API Contract

Users (Authors) — melakukan login (Sorry ga buat login API Contract-nya, klo kalian mau pake 3rd party Auth jg bole, yang penting paham gmn cara autentikasi)

## User object

```
{
  "id": 1,
  "username": "john_doe",
  "email": "john.doe@example.com",
  "created_at": "2025-09-22T10:00:00Z",
  "updated_at": "2025-09-22T10:00:00Z"
}
```

## GET /user/:id

Return user dengan id tertentu (untuk menampilkan data user aja)

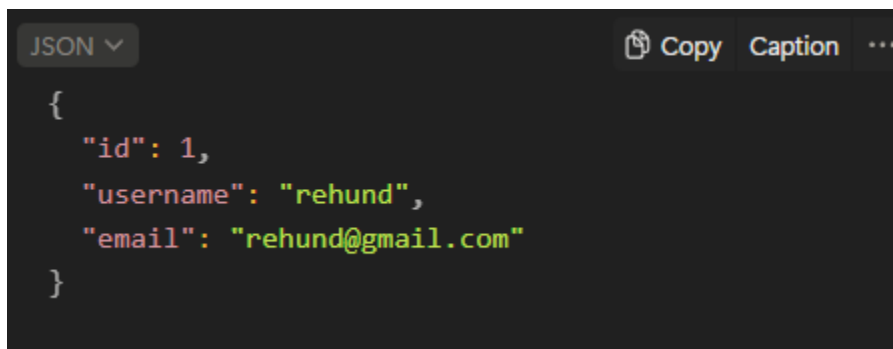
**URL Params (Path Variable)** : \*Required, id — (integer)

**Data Params (Request Body)** : None

**Headers** : Content-Type: application/json

**Success Response: Code:** 200

- **Content:**

A screenshot of a code editor showing a JSON object. The editor has a dark background. At the top left, there is a dropdown menu with 'JSON' selected. At the top right, there are buttons for 'Copy', 'Caption', and a three-dot menu. The JSON object is displayed in a light blue font and contains the following data: {"id": 1, "username": "rehund", "email": "rehund@gmail.com"}.

```
{
  "id": 1,
  "username": "rehund",
  "email": "rehund@gmail.com"
}
```

**Error Response: Code:** 404 (Not found)

- **Content:** { "error": "User doesn't exist" }

## Post object

```
{
  "id": 1,
```

```
"title": "Belajar Backend dari Nol",
"content": "Ini adalah isi lengkap dari artikel tentang bagaimana memulai belajar backend...",
"author_id": 1,
"created_at": "2025-09-22T11:30:00Z",
"updated_at": "2025-09-22T11:30:00Z"
}
```

## GET /posts

Return semua posts yang ada di system

**URL Params (Path Variable)** : None

**Data Params (Request Body)** : None

**Headers** : Content-Type: application/json

**Success Response: Code:** 200

- **Content:**

```
JSON ▼ Copy Caption ...
{
  "posts" : [
    {
      "id": 1,
      "title": "title 1",
      "content": "isi content 1",
      "author_id": 1,
      "created_at": "2025-09-22T11:30:00Z",
      "updated_at": "2025-09-22T11:30:00Z"
    },
    {
      "id": 2,
      "title": "title 1",
      "content": "isi content 1",
      "author_id": 1,
      "created_at": "2025-09-22T11:30:00Z",
      "updated_at": "2025-09-22T11:30:00Z"
    },
    ...
  ]
}
```

**Error Response: - Code: 404 (Not found)**

- **Content:** { "error": "No Posts" }

## **GET /posts/:id**

Return post dengan id spesifik


**URL Params (Path Variable) :** \*Required, id — (integer)

**Data Params (Request Body) :** None

**Headers :** Content-Type: application/json

**Success Response: Code: 200**

- **Content:**

A screenshot of a code editor showing a JSON object. The editor has a dark background with light-colored text. At the top left, there is a dropdown menu labeled 'JSON' with a downward arrow. At the top right, there are buttons for 'Copy', 'Caption', and a three-dot menu. The JSON object is displayed in the center, with syntax highlighting: strings are in green, numbers in blue, and property names in light gray. The object contains fields for id, title, content, author\_id, created\_at, and updated\_at, all with example values.

```
{
  "id": 1,
  "title": "title 1",
  "content": "isi content 1",
  "author_id": 1,
  "created_at": "2025-09-22T11:30:00Z",
  "updated_at": "2025-09-22T11:30:00Z"
}
```

**Error Response: Code: 404 (Not found)**

- **Content:** { "error": "Post doesn't exist" }

## **POST /posts**

Membuat postingan baru dan mengembalikan objek yang baru dibuat. HANYA BISA DIACCESS OLEH ADMIN (Buat guard clause ya)

**URL Params (Path Variable) :** None

**Data Params (Request Body) :**

```
JSON ▾ Copy Caption ...
{
  "title": "title 1",
  "content": "isi content 1",
  "author_id": 1,
}
```

Info dikit → disini id, created\_at itu otomatis dibuat ya, biasanya ORM itu nyediain hal ini jg  
Btw baru nyadar updated\_at itu ga perlu, karena ya kita ga ada update (PUT atau PATCH)

#### Headers :

- Content-Type: application/json
- Authorization: Bearer <Auth\_token\_Admin>

**Success Response: Code: 201 (Created)**

- **Content:**

```
JSON ▾ Copy Caption ...
{
  "id": 1,
  "title": "title 1",
  "content": "isi content 1",
  "author_id": 1,
  "created_at": "2025-09-22T11:30:00Z",
  "updated_at": "2025-09-22T11:30:00Z"
}
```

#### Error Response:

**Code: 401**

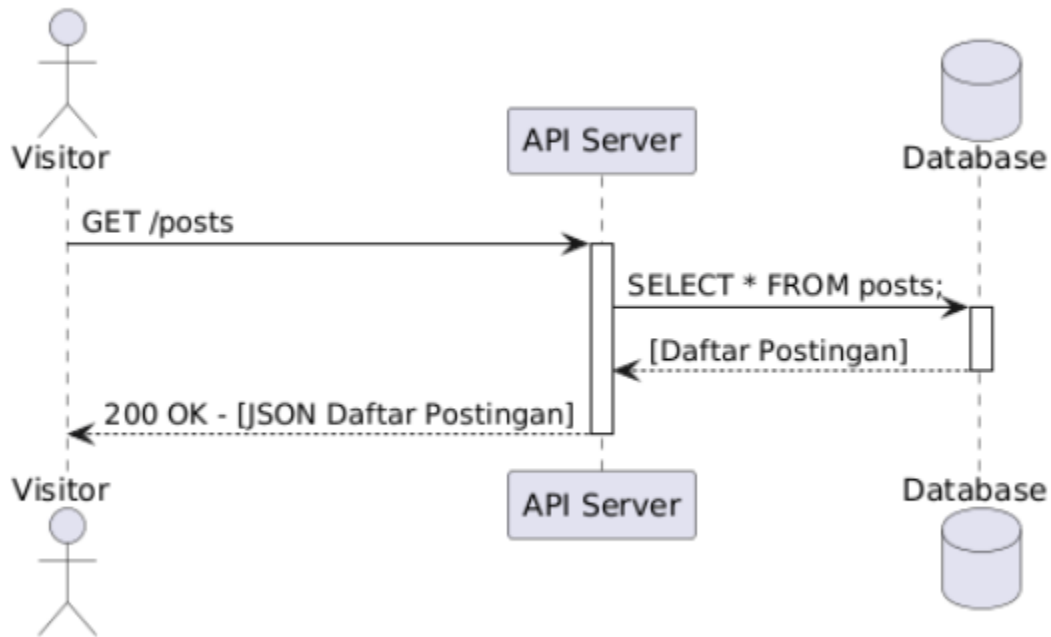
- **Content:** { "error": "Unauthorized" }

**Code: 400**

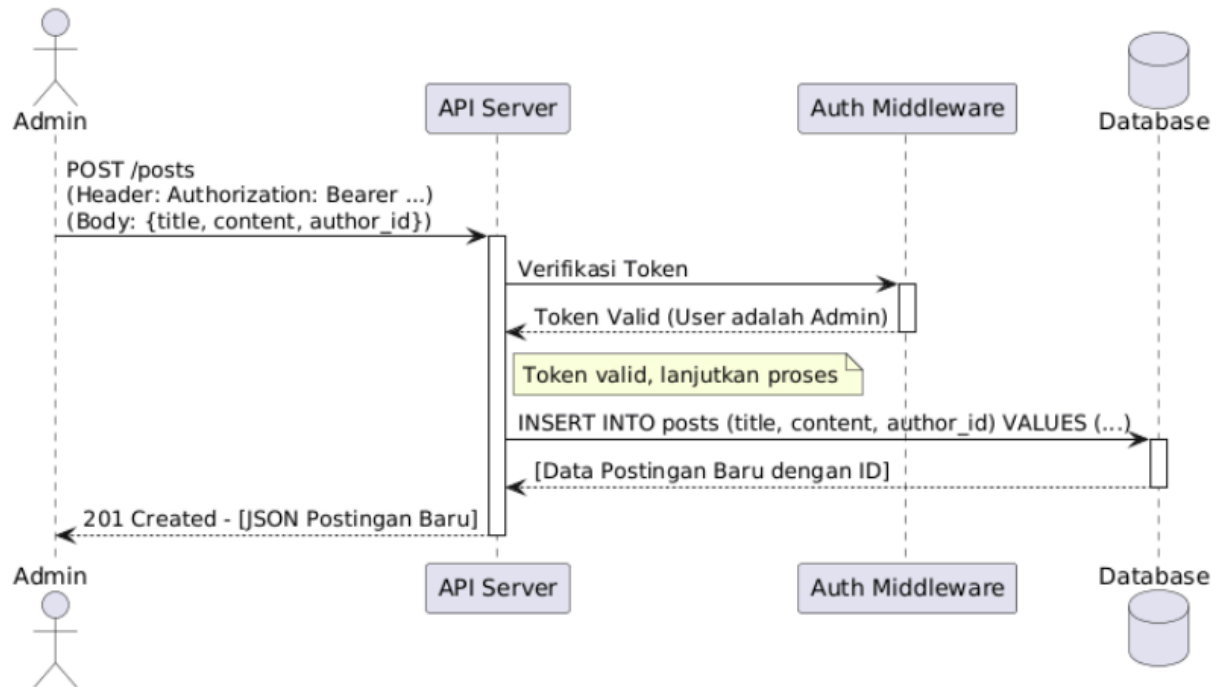
- **Content:** { "error": "Title and content cannot be empty" }

# Sequence Diagram

Skenario 1 : Visitor mengambil semua postingan (GET /posts)



## Skenario 2 : Admin Membuat Postingan Baru (POST /posts)



## Frontend (Ga usah bagus2 simple aja)

Ini gambaran kasar aja yang kurleb begini :

Login Page :

Login Page

Form Untuk Login  
Fetch (POST /login)

email

password

Submit

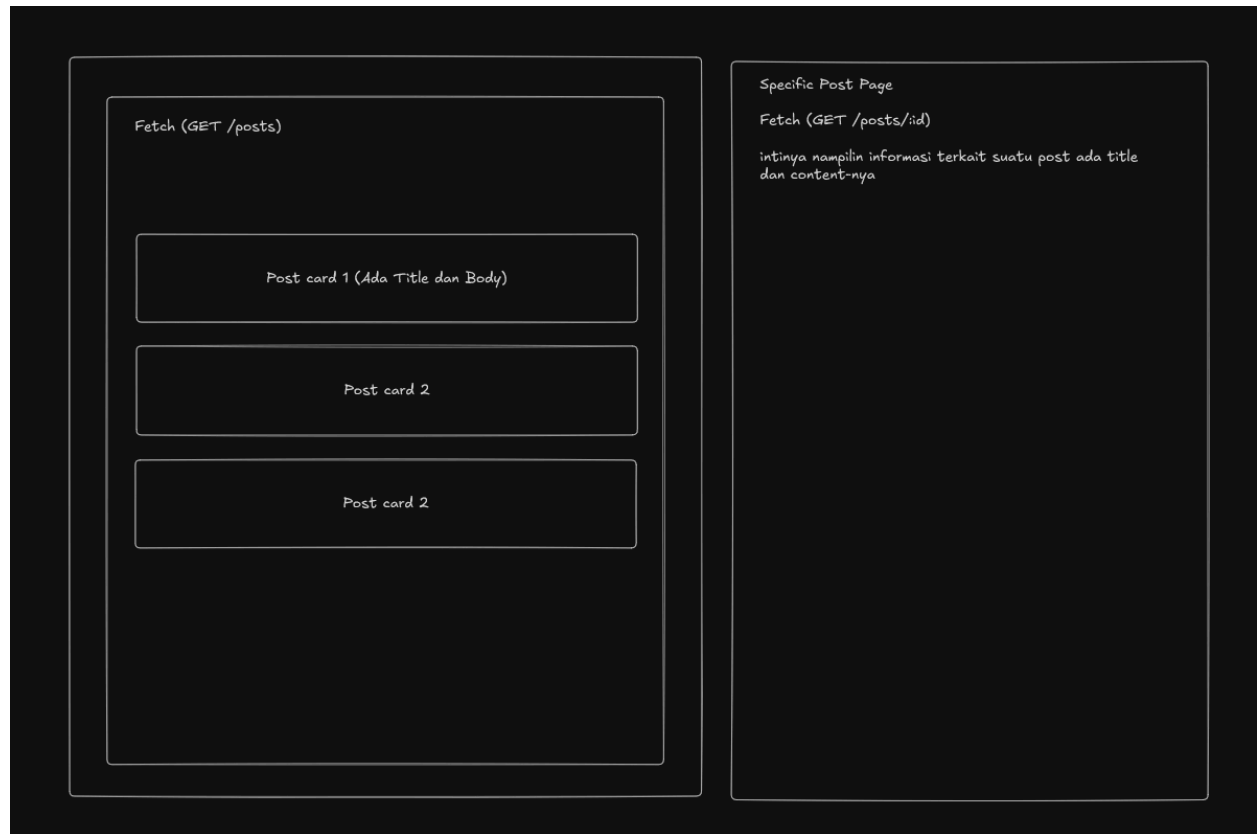
Kamu bukan admin? Boleh masuk aja langsung

Lihat Blog

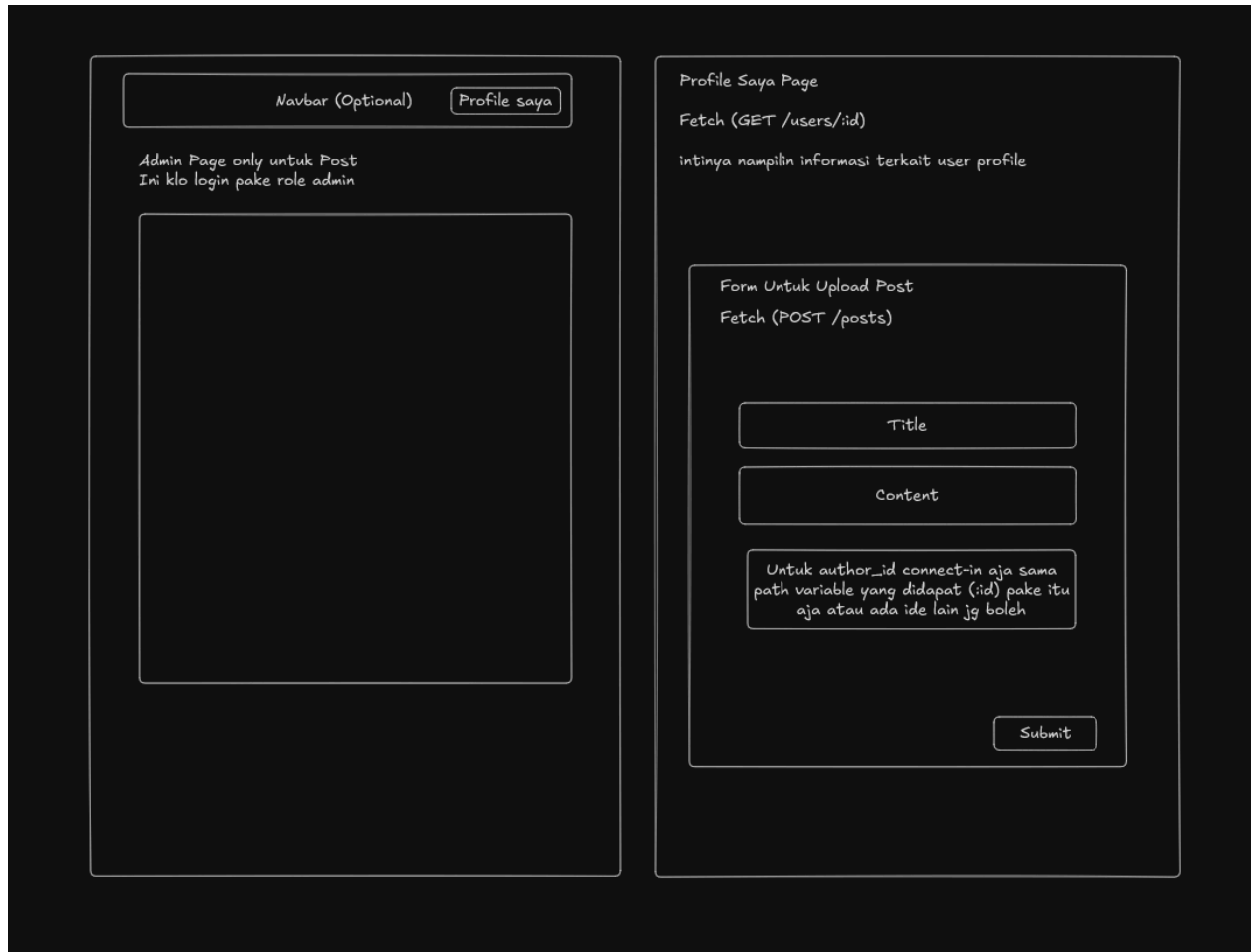
Dimana untuk pembuatan akun admin itu dilakukan langsung saja di database (Buat akun di database) hehe (Atau kamu ada ide lain boleh)

Buat user (non-admin) — tidak perlu login :





Buat (admin) — wajib login :



Disini pokoknya Ada “profile saya” button supaya bisa route ke profil saya page

Helper :

Klo yang supabase seharusnya lebih mudah ya untuk developnya, karena setauku dia BaaS kan, tinggal pake2 aja built-in functionnya

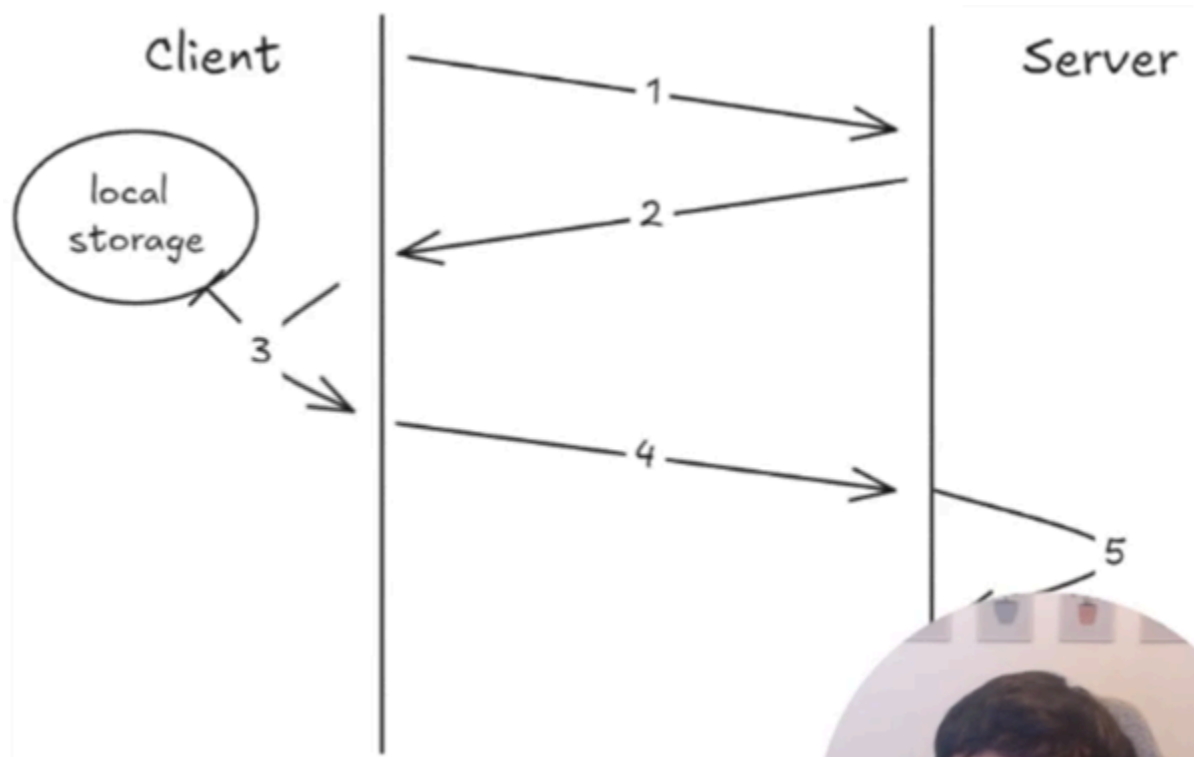
<https://www.youtube.com/watch?v=gTD8b5Yxuuo&t=59s>

<https://www.youtube.com/watch?v=RDM-nk5f4SE> (Klo yang mau pake typescript)

<https://www.youtube.com/watch?v=tpesx-g-7t4> (ini auth, cm implement OTP, agak ga relevan cm yawis)

Intinya keyword : JWT Authentication buat si admin

Klo dari Aku JWT tuh gini :



#### Langkah 1: User Login (Panah 1)

- **Diagram:** Panah nomor 1 menunjukkan Client mengirimkan informasi ke Server.
- **Penjelasan:** Ini adalah saat Anda memasukkan username dan password Anda di halaman login. Permintaan ini dikirim ke server untuk diverifikasi. Ini seperti Anda menunjukkan KTP Anda di gerbang utama konser.

#### Langkah 2: Server Memberikan Token (Panah 2)

- **Diagram:** Panah nomor 2 menunjukkan Server mengirimkan sesuatu kembali ke Client.
- **Penjelasan:** Setelah server berhasil memverifikasi username dan password Anda, server tidak menyimpan sesi login Anda. Sebaliknya, ia membuat sebuah "tiket" digital yang aman, yaitu **JWT**. Token ini berisi informasi tentang Anda (misalnya, user ID Anda) dan memiliki tanda tangan digital (signature) untuk membuktikan keasliannya. Server kemudian mengirimkan token ini kembali ke client (browser Anda).

#### Langkah 3: Client Menyimpan Token (Panah 3)

- **Diagram:** Panah nomor 3 menunjukkan token disimpan di local storage milik Client.
- **Penjelasan:** Browser Anda menerima token ini dan menyimpannya di tempat yang aman, biasanya di *Local Storage* atau *Session Storage*. Ini sama seperti Anda memakai gelang akses yang diberikan panitia di pergelangan tangan Anda.

#### Langkah 4: Menggunakan Token untuk Request Selanjutnya (Panah 4)

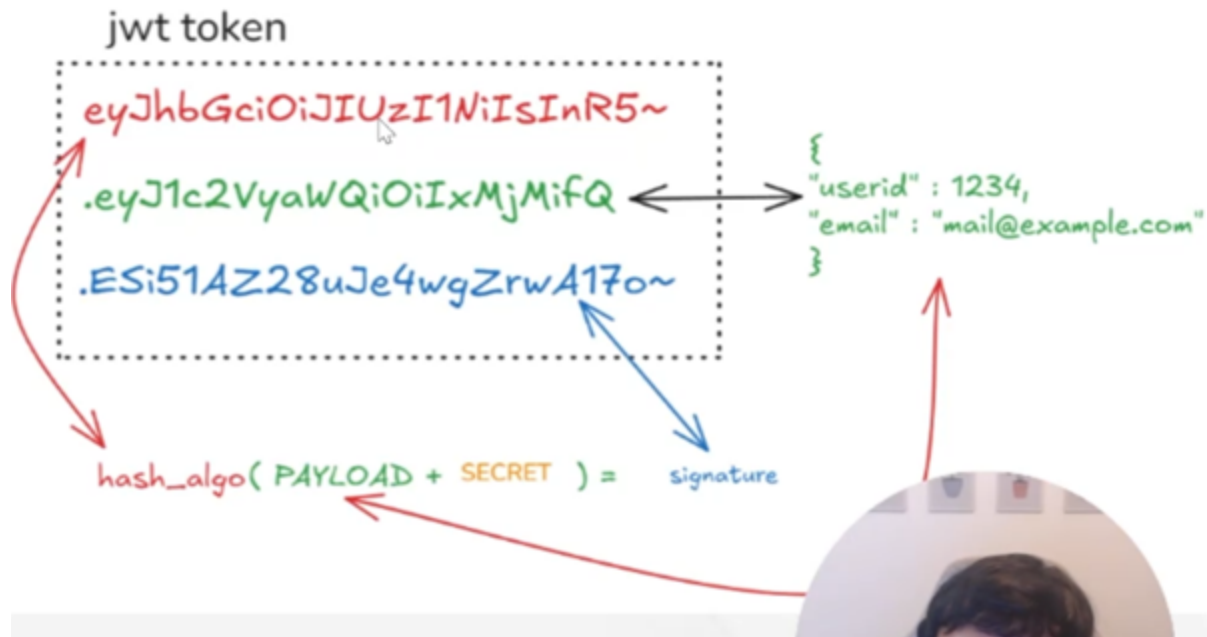
Ini adalah bagian yang membingungkan dalam teks Anda. Mari kita perjelas.

- **Maksud Sebenarnya: Untuk setiap permintaan selanjutnya** ke server (misalnya, saat Anda mencoba mengakses halaman profil, mengirim komentar, atau melihat data rahasia), **Client akan menyertakan token yang telah disimpan** di dalam *request header*-nya.
- **Diagram:** Panah nomor 4 menunjukkan Client mengirim permintaan baru ke Server. Bedanya, permintaan ini sekarang "membawa" token yang tadi disimpan.
- **Analogi:** Setiap kali Anda mau masuk ke area VIP di dalam konser, Anda tidak perlu lagi menunjukkan KTP. Anda cukup menunjukkan gelang akses (token) yang Anda kenakan kepada petugas keamanan.

#### Langkah 5: Server Validasi Token (Aksi 5 di Sisi Server)

- **Diagram:** Aksi nomor 5 terjadi di dalam Server setelah menerima permintaan dari panah 4.
- **Penjelasan:** Ketika server menerima permintaan yang berisi token, ia akan melakukan dua hal utama:
  1. **Memeriksa Tanda Tangan (Signature):** Server akan memverifikasi tanda tangan digital pada token untuk memastikan token itu asli (dikeluarkan oleh server ini) dan tidak diubah-ubah di tengah jalan.
  2. **Memeriksa Masa Berlaku:** Server juga akan memeriksa apakah token tersebut sudah kedaluwarsa atau belum.
- Jika token valid, server akan mempercayai informasi di dalamnya dan memproses permintaan Anda (misalnya, menampilkan halaman profil). Jika tidak valid, server akan menolak permintaan tersebut. Ini seperti petugas keamanan yang memeriksa apakah gelang Anda asli dan bukan sobekan kertas biasa.

## Struktur Token JWT



Terdapat 3 Bagian yaitu :

1. Header: Type and Algorithm
2. Payload : Data
3. Signature: Identitas

\*Note : SECRET di signature hanya disimpan di server