UJIAN AKHIR SEMESTER

KEAMANAN INFORMASI KJ003

Nama : Dhini Febrasari
NIM : 20210801231

Soal :

1. Buatlah analisisnya dengan kasus yang anda tentuka sendiri
2. Buatlah aplikasinya
3. Lakukan Vulnerability Assesment terhadap aplikasi yang dibikin
4. Lakukan Pengamanan data – datanya sesuai dengan kasus yang anda tentukan sendiri

Jawaban :

1. Proyek ini adalah **sistem manajemen proyek** yang bertujuan membantu tim mengelola tugas (tiket) dalam berbagai proyek. Sebuah proyek dapat mewakili klien atau inisiatif internal, dan setiap proyek akan memiliki beberapa tiket.

   Contoh Kasus Penggunaan:

   - Tim **Pengembangan Perangkat Lunak** menggunakan aplikasi ini untuk mengelola tugas-tugas di berbagai proyek. Tugas-tugas ini dapat mencakup pengodean, pengujian, atau penelusuran kesalahan, masing-masing dengan status yang berbeda.
   - Tim **Pemasaran** dapat menggunakan sistem ini untuk melacak berbagai kampanye pemasaran, menetapkan tiket untuk tugas-tugas seperti pembuatan konten, posting media sosial, dan pekerjaan desain.
   - **Manajer** akan dapat menugaskan tugas kepada anggota tim tertentu, melacak kemajuan setiap tugas, dan memastikan tenggat waktu terpenuhi.

   Analisis:

   - Sistem ini meningkatkan **kolaborasi** dan **visibilitas tugas** dalam tim.
   - Manajer proyek dapat dengan mudah mengawasi status tiket dan distribusi beban kerja, sehingga meningkatkan **produktivitas** dan **komunikasi** tim .

2. Link github : https://github.com/dinoyy22/sistem-manajemen-proyek-.git

Fitur Utama:

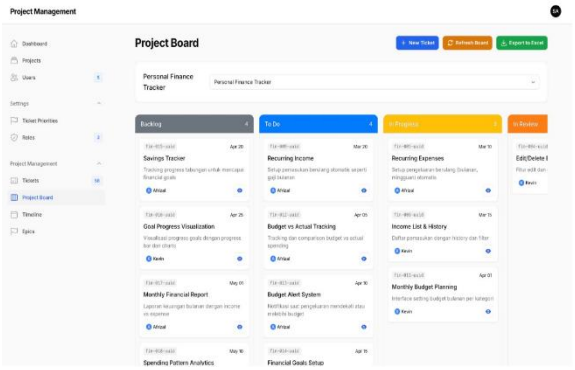
- **Manajemen Proyek** : Buat proyek baru, tetapkan ke anggota tim, dan lacak kemajuan proyek.
- **Pembuatan dan Pengelolaan Tiket** : Tiket (tugas) dibuat dalam proyek. Setiap tiket mencakup atribut seperti judul, deskripsi, tanggal jatuh tempo, prioritas, dan status terkini.
- **Manajemen Pengguna** : Pengguna dengan peran seperti Admin atau Manajer dapat membuat dan menetapkan tiket. Pengguna biasa hanya dapat melihat atau memperbarui tiket mereka sendiri.

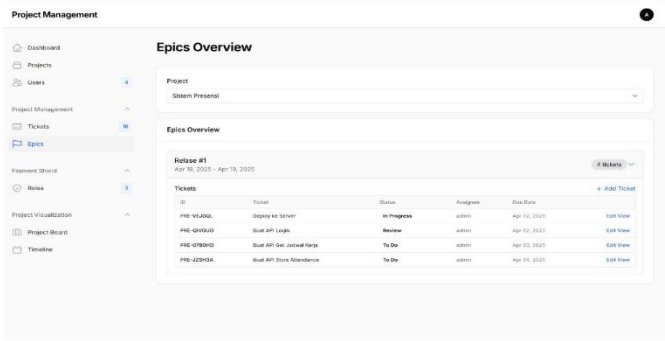
Implementasi Kode:

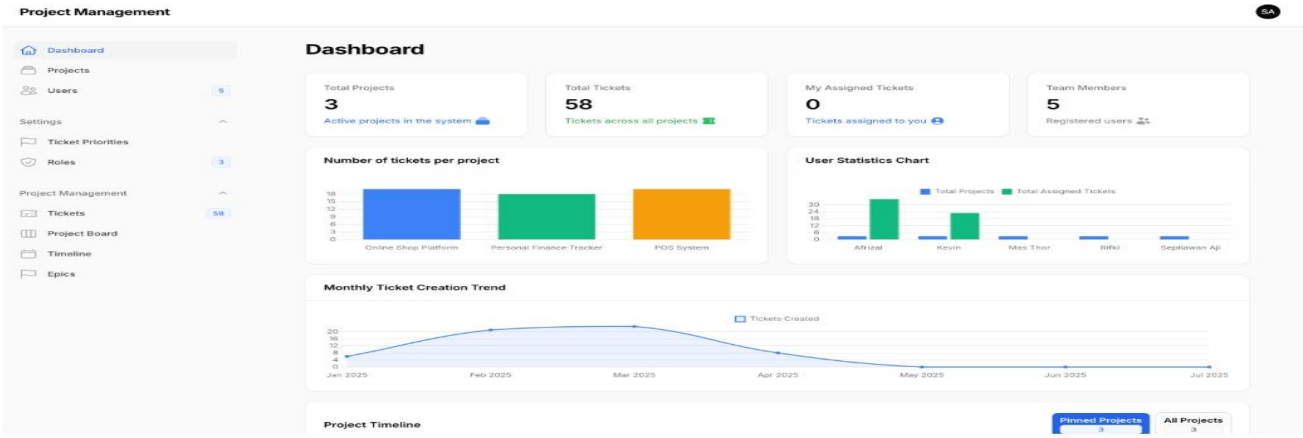
- **Model Tiket** :
  Modelnya Ticket mungkin terlihat seperti ini, mendefinisikan hubungan dengan User dan Project:

```php
46 references | 0 implementations
class Ticket extends Model    Undefined type 'Illuminate\Database\Eloquent\Model'.
{
    use HasFactory;    Undefined type 'Illuminate\Database\Eloquent\Factories\HasFactory'.

    0 references
    protected $fillable = [
        'project_id',
        'ticket_status_id',
        'priority_id',
        'name',
        'description',
        'due_date',
        'uuid',
        'epic_id',
        'created_by',
    ];

    0 references
    protected $casts = [
        'due_date' => 'date',
    ];
```

Panel Admin Filamen







3. Setelah website dilakukan pengujian analaisis assessment dengna menggunakan tools ptt website_scanner maka di dapatkan hasil pengujian dan analisis nya yaitu

A. Missing Security Headers

Content-Security-Policy (CSP)
Risiko: Tanpa **CSP**, aplikasi rentan terhadap serangan **Cross-Site Scripting (XSS)**, di mana penyerang dapat menyuntikkan skrip berbahaya yang dieksekusi di browser pengguna.

Rekomendasi: Tambahkan header **Content-Security-Policy** pada response server untuk membatasi sumber daya yang bisa dimuat oleh aplikasi. Contoh: Content-Security-Policy: default-src 'self'; script-src 'self' https://trusted.cdn.com; object-src 'none';

X-Content-Type-Options
Risiko: Tanpa header ini, **MIME sniffing** bisa terjadi, yang memungkinkan **content-type spoofing**, di mana file yang seharusnya bukan skrip bisa dieksekusi sebagai skrip.

Rekomendasi: Tambahkan header **X-Content-Type-Options: nosniff** untuk mencegah MIME sniffing.
Contoh: X-Content-Type-Options: nosniff

Strict-Transport-Security (HSTS)
Risiko: Tanpa **HSTS**, pengguna dapat terkoneksi melalui HTTP yang rentan terhadap serangan **man-in-the-middle (MITM)**, di mana data yang dikirim bisa disadap atau diubah.

Rekomendasi: Terapkan **Strict-Transport-Security** pada semua response HTTPS untuk memastikan koneksi hanya dilakukan menggunakan HTTPS. Contoh: Strict-Transport-Security: max-age=31536000; includeSubDomains

Referrer-Policy
Risiko: Tanpa **Referrer-Policy**, aplikasi mengirimkan informasi referer ke server lain yang bisa mengungkapkan informasi sensitif terkait halaman atau sumber asalnya.

Rekomendasi: Gunakan header **Referrer-Policy** untuk mengontrol informasi referer yang dikirim ke server lain. Contoh: Referrer-Policy: no-referrer

Server Software Disclosure
Risiko: Terbukanya informasi tentang framework, library, dan software yang digunakan (misalnya, Laravel, Filament, Livewire) dapat memberikan petunjuk bagi penyerang untuk mengeksploitasi kerentanannya.

Rekomendasi: Minimalkan eksposur informasi terkait software yang digunakan dalam header atau meta. Pastikan **APP_DEBUG** diset ke false di file `.env`.

robots.txt Terbuka
Risiko: File `robots.txt` yang terbuka dapat mengungkapkan direktori sensitif yang tidak seharusnya diketahui oleh penyerang atau crawler.

Rekomendasi: Review dan pastikan file `robots.txt` hanya mencantumkan direktori yang perlu diblokir untuk crawler dan hindari mencantumkan direktori sensitif.

Login Interface Ditemukan
Risiko: Halaman login yang dapat diakses tanpa perlindungan terhadap serangan **brute force** atau **credential stuffing** dapat menjadi target penyerang.

Rekomendasi: Gunakan **rate limiting**, **CAPTCHA**, dan audit kredensial secara berkala untuk halaman login.
Contoh implementasi rate limiting di Laravel:
Route::post('login', [LoginController::class, 'login'])->middleware('throttle:5,1');

Security.txt Tidak Ditemukan
Risiko: Tidak adanya file **security.txt** berarti tidak ada jalur kontak yang jelas untuk pelaporan kerentanannya. Ini adalah masalah best practice.

Rekomendasi: Tambahkan file **security.txt** di dalam folder `.well-known/` untuk memberikan informasi kontak yang jelas untuk pelaporan kerentanannya.
Contoh:
Contact: mailto:security@yourdomain.com

B. Kesimpulan

Tidak ditemukan kerentanan berisiko tinggi seperti:
• SQL Injection
• XSS aktif
• CSRF
• Auth Bypass
• Server Misconfiguration

Temuan-temuan ini bersifat informasi dan best practice yang disarankan untuk segera ditindaklanjuti demi memperkuat keamanan aplikasi. Perubahan-perubahan ini akan membantu mengurangi risiko dan memperkuat postur keamanan aplikasi secara keseluruhan.

4. Pada Bagian keamanan sistem informasi website yang dibuat, pengamanan data dilakukan dengan cara memberikan token pada saat user akan HIT request API data project. Pengamanan bekerja dengan cara menyelipkan api_token pada request API lalu ditaruh api_token tersebut pada authorization bearer token.

Pemberian API Token kepada User

Sebelum user dapat mengakses data dari server, mereka terlebih dahulu harus terverifikasi. Untuk itu, sistem mengharuskan API token yang berfungsi sebagai identitas autentikasi pengguna.

Langkah-langkah Pemberian API Token:

1. **User melakukan login**: Saat pertama kali login, user akan diminta untuk mengautentikasi dirinya menggunakan kredensial yang valid (misalnya username dan password).
2. **Server memverifikasi kredensial**: Server memeriksa apakah username dan password yang diberikan benar. Jika valid, server akan mengeluarkan **API token** yang akan diberikan kepada user sebagai bukti bahwa user sudah terautentikasi.
3. **Token diberikan kepada user**: Token yang diterima berupa string yang panjang, yang biasanya berformat **JWT (JSON Web Token)** atau token lainnya yang berisi informasi ter-enkripsi tentang user, termasuk user ID dan waktu kadaluarsa token.
4. Pengiriman API Token dalam Request API
   Setelah mendapatkan API token, user akan mengirimkan token tersebut dalam setiap request API yang dikirimkan untuk mengakses data. Token ini dimasukkan dalam header **Authorization** pada request API, sehingga server dapat memverifikasi apakah user tersebut berhak mengakses data yang diminta.

   Contoh Pengiriman API Token pada Header Request:
   ```http
   GET /api/project-data HTTP/1.1
   Host: example.com
   Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoxMjM0NTY3ODkwLCJleHBpcmF0aW9uX3RpbWUiOiJleHBpcnkgZG8gMzg5YWRhb2ppZXMifQ.Y1iTqkWf6u6F3k7uB85dF99O0BO5kA3qNY59m20cZvw
   ```

   - **Authorization Header**: Header ini berisi informasi **Bearer token**, yang diikuti oleh token yang didapatkan saat login. Token ini digunakan oleh server untuk memverifikasi apakah user yang mengirimkan request adalah pengguna yang sah dan apakah mereka memiliki hak akses yang sesuai.

   3. Verifikasi Token oleh Server

Setelah server menerima request API yang disertakan dengan Bearer token, server akan memverifikasi token tersebut untuk memastikan bahwa request tersebut berasal dari pengguna yang sah.

Langkah-langkah Verifikasi Token oleh Server:
1. **Dekode token**: Server akan mendekode token tersebut untuk memeriksa keaslian dan validitas token.
2. **Cek apakah token valid**: Server akan memverifikasi beberapa informasi dalam token, seperti:
- **User ID**: Memastikan bahwa token berhubungan dengan user yang sah.
- **Waktu kadaluarsa**: Memeriksa apakah token sudah kedaluwarsa atau masih valid untuk digunakan.
3. **Cek Hak Akses**: Server akan mengecek apakah user yang meminta data memiliki hak akses yang benar untuk mengakses data tertentu.
4. Pengiriman Data yang Telah Ter-enkripsi
Setelah server memverifikasi API token dan memastikan bahwa user yang membuat request sah, server kemudian akan mengirimkan data yang diminta dalam bentuk JSON yang telah terenkripsi.

Kenapa Data Dikirim dalam Bentuk Enkripsi?:
• **Kerahasiaan**: Mengirimkan data dalam bentuk terenkripsi memastikan bahwa data tersebut tidak dapat dibaca oleh pihak ketiga yang tidak berwenang, meskipun data tersebut disadap dalam perjalanan.
• **Keamanan**: Enkripsi data memberikan tingkat keamanan tambahan, terutama ketika aplikasi berkomunikasi melalui jaringan yang mungkin tidak sepenuhnya aman (misalnya, jaringan publik).
5. Dekripsi Data oleh User
Untuk mengakses informasi yang terkirim dalam bentuk terenkripsi, user harus menggunakan **key dekripsi** yang sama dengan **API token** yang mereka terima dari server.

Proses Dekripsi:
• **User menggunakan token sebagai key dekripsi**: Karena **API token** yang diterima saat login sama dengan kunci dekripsi yang digunakan oleh server untuk mengenkripsi data, maka user dapat menggunakan token tersebut untuk membuka data yang terenkripsi.
• **Dekripsi data**: Setelah data didekripsi, informasi yang terkandung dalam data JSON dapat diakses dalam bentuk **plain text**, sehingga user dapat melihat data yang diminta.
6. Kesimpulan
Secara keseluruhan, sistem pengamanan data pada aplikasi ini menggunakan **API token** untuk mengautentikasi user dan memastikan bahwa hanya user yang sah yang dapat mengakses data tertentu. Langkah-langkah pengamanan data tersebut adalah:

1. **User mengirimkan API token** sebagai bukti autentikasi.
2. **Server memverifikasi API token** dan memeriksa hak akses.
3. **Server mengirimkan data terenkripsi** dalam bentuk JSON untuk menjaga kerahasiaan data.
4. **User mendekripsi data menggunakan API token** sebagai key dekripsi

untuk mendapatkan data dalam bentuk plain text.

Dengan pendekatan ini, sistem memastikan bahwa **data tetap aman** selama proses transmisi karena hanya user yang memiliki token yang sah yang dapat mengakses data tersebut.