BAB 2
LANDASAN TEORI

Bagian ini menjelaskan landasan teori yang menjadi dasar penelitian dalam membangun alat bantu otomatisasi pengujian *endpoint* untuk deteksi kerentanan *Broken Access Control*. Teori yang disajikan mencakup konsep-konsep fundamental yang diterapkan dalam penelitian ini.

2.1 Back-End

Back-end merupakan bagian sistem perangkat lunak yang berfungsi untuk menangani logika aplikasi [11], berkomunikasi dan berinteraksi dengan *database* dan server [12]. *Back-end* tidak terlihat secara langsung oleh pengguna sebab bekerja di sisi server. Dalam penelitian ini, *back-end* bertanggung jawab untuk melayani permintaan dari *front-end* [13], seperti berinteraksi dengan *database* untuk mengambil, menyimpan, atau memperbarui data [14]. *Back-end* juga bertanggung jawab atas pengelolaan autentikasi dan otorisasi [15], memastikan bahwa hanya pengguna yang memiliki hak akses dan izin dapat melakukan tindakan tertentu dalam aplikasi. Selain itu, *back-end* saling terhubung dengan berbagai layanan dan aplikasi lain melalui *Application Programming Interface* (API), yang memungkinkan komunikasi antara perangkat yang berbeda (*website*, *mobile*, *Internet of Things*, dan *desktop*) [16].

2.2 Large Language Model

Large Language Model (LLM) merupakan sebuah model pembelajaran mendalam berbasis arsitektur *Transformer* yang dilatih pada korpus teks sangat besar [17] untuk membuat model distribusi bahasa yang memiliki cara kerja untuk memprediksi token berikutnya berdasarkan konteks sebelumnya. Inti *Transformer* adalah *self-attention*, mekanisme yang menimbang relevansi setiap token terhadap token lain sehingga model mampu menangkap ketergantungan jarak jauh (*long-range dependencies*) secara efisien [18]. Dalam penelitian ini, LLM digunakan untuk menganalisis *OpenAPI Specification* (OAS), matriks RBAC, dan *policy rules* guna menghasilkan rencana pengujian otomatis, mengidentifikasi *resource* ID untuk *test cases*, serta memberikan ringkasan evaluasi keamanan dari hasil pengujian.

2.3 LLM-Based Intelligent Test Planning

Pendekatan otomasi pengujian yang memanfaatkan penalaran LLM berbasis *Transformer* untuk menganalisis OAS dan matriks RBAC, lalu merencanakan uji yang terstruktur, prioritas, dan kontekstual dengan fokus pada *Broken Access Control* (IDOR/BOLA) [19]. Dibanding perencanaan manual, *random fuzzing*, dan pengujian berbasis aturan, pendekatan ini lebih *context-aware*, adaptif, dan *explainable* karena setiap prioritas disertai rasional berbasis risiko [20]. Alurnya mencakup perancangan *prompt* (OAS, RBAC, serta skema JSON), inferensi LLM dengan *self-attention* yang memetakan relasi *endpoint*–parameter–peran dan menghasilkan keluaran valid melalui *schema-enforced* JSON [21], diikuti *parsing* validasi (skema, *path* OAS, konsistensi RBAC) dan *ranking* temuan berbasis risiko yang hasilnya diukur melalui cakupan dan metrik turunan (*precision*, *recall*, F1-*score*, dan akurasi).

2.4 AI Agent

Artificial Intelligence Agent (AI *Agent*) merupakan sistem berbasis kecerdasan buatan yang dapat mengamati lingkungan, membuat keputusan, dan bertindak secara mandiri untuk mencapai tujuan tertentu [22]. Dalam arsitektur AI modern, *agent* dirancang untuk melakukan tugas spesifik seperti klasifikasi, pengambilan keputusan, atau interaksi dengan pengguna [23]. Tergantung pada kompleksitas tugas, *agent* dapat bersifat reaktif, *deliberative*, atau berbasis pembelajaran [24]. Pada penelitian ini, AI *Agent* berperan sebagai *orchestrator* yang menganalisis matriks RBAC, OAS, dan *policy rules* untuk menghasilkan *test plan*, mengeksekusi *HyperText Transfer Protocol* (HTTP) *request* multi-*role*, mengevaluasi hasil untuk mendeteksi kerentanan *Broken Access Control*, dan menghasilkan laporan dengan *confusion matrix* serta metrik evaluasi.

2.5 Orchestrator dalam Sistem AI Agent

Dalam sistem berbasis AI, *orchestrator* berfungsi sebagai komponen pengatur yang mengatur interaksi antar agen AI, mengalokasikan tugas, serta memastikan efisiensi dan konsistensi proses [25]. Dalam konteks pengujian keamanan aplikasi *back-end*, *orchestrator* bertugas mengelola banyak agen AI [26] yang mendeteksi kerentanan BAC, dengan memetakan peran pengguna dan mengalokasikan skenario uji yang sesuai. Sebagai contoh, terdapat penelitian

yang menjelaskan tentang *framework* orkestrasi berbasis AI-*aware* yang mampu mengelola sumber daya secara dinamis di lingkungan *cloud*, mendukung integrasi antar agen, serta memastikan kelancaran proses pengujian dan mitigasi kerentanan secara otomatis [27].

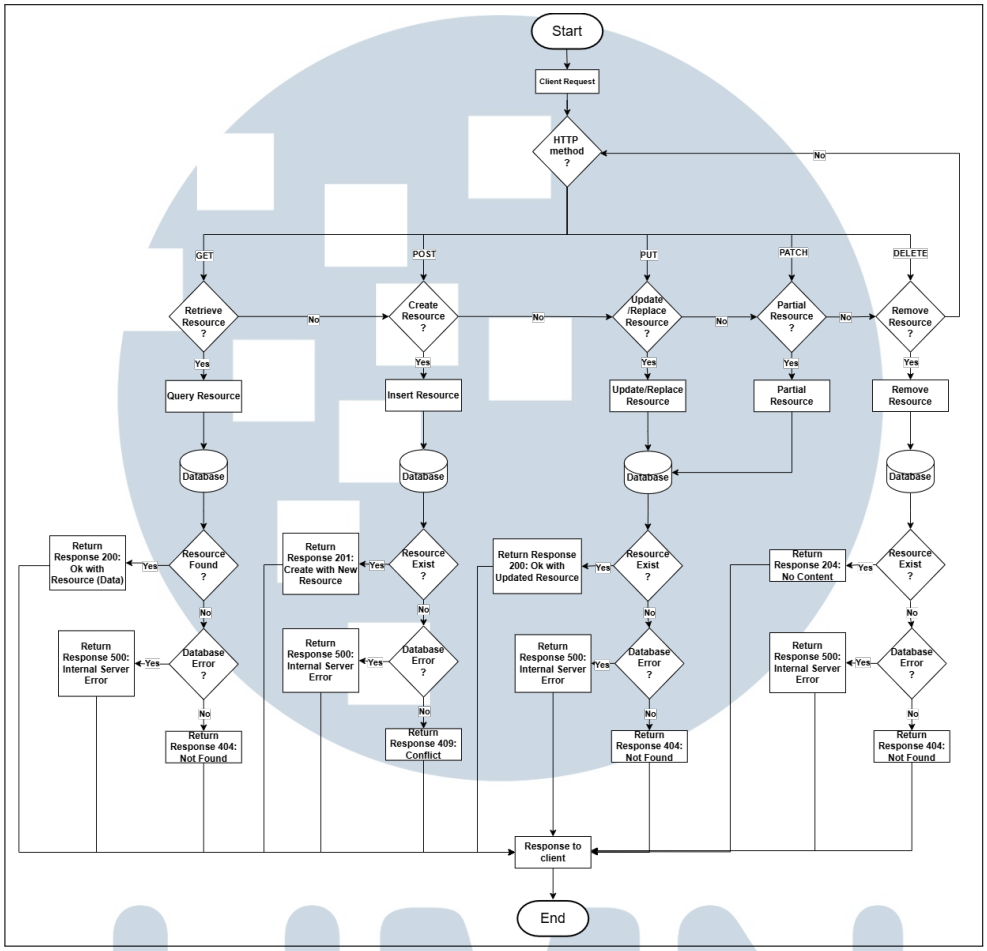
2.6 OpenAPI Specification

Dokumentasi *Representational State Transfer Application Programming Interface* (REST API) biasanya menggunakan *OpenAPI Specification* (OAS) atau standar yang menjelaskan *resource*, parameter, metode, dan contoh respons secara rinci [28]. OAS mempermudah *back-end* dan *front-end* dalam membangun, menguji, dan mengintegrasikan sistem, serta mendukung otomatisasi pengujian *endpoint* [29] sehingga tim *front-end* dapat melakukan pembangunan dan perancangan aplikasi tanpa menunggu *Uniform Resource Identifier* (URI) dari *back-end* [30]. Pada penelitian ini, OAS digunakan sebagai sumber *endpoint discovery* untuk mengekstraksi daftar *endpoint*, metode HTTP, dan parameter yang akan diprioritaskan berdasarkan tingkat risiko keamanan, kemudian digunakan oleh AI *Agent* untuk menghasilkan *test plan* otomatis dalam pengujian kerentanan *Broken Access Control*.

2.7 REST API

Representational State Transfer Application Programming Interface (REST API) merupakan arsitektur perangkat lunak yang banyak digunakan untuk membangun aplikasi web modern [31]. REST memiliki sifat *stateless* untuk memungkinkan *horizontal scale-out*, setiap *request* telah berisikan seluruh konteks berupa *JavaScript Object Notation* (JSON) atau *eXtensible Markup Language* (XML) sehingga beban terdistribusi merata di sisi server [32] dan menurunkan latensi atau beban server tanpa menambah logika khusus di aplikasi [33]. Selain itu, pemisahan (*decoupling*) antara klien dan server dipermudah dengan *uniform interface* dan *layered system*, sehingga perubahan skema data, *gateway*, dan *Web Application Firewall* (WAF) dapat dilakukan independen tanpa melakukan perubahan dari sisi klien [34] [35]. REST merancang domain dengan *resource* sebagai URI yang direpresentasikan melalui *content type*, *accept*, *application/json* [36]. Setiap *action* pada *resources* (*request*) dipetakan menjadi beberapa metode HTTP, di antaranya GET sebagai *read*, DELETE untuk *delete* [37], POST sebagai

create, dan metode PUT atau PATCH untuk *edit* [38], seperti Gambar 2.1.



Gambar 2.1. Flowchart representational state transfer application programming interface

2.8 Auto-Discovery Resource IDs

Algoritma *Auto-Discovery Resource IDs* merupakan ekstraksi heuristik berbasis *pattern matching* yang mengekstrak pengenal sumber daya (ID) dari respons API untuk mengisi *placeholder* pada pengujian *endpoint* [39]. Prosesnya meliputi identifikasi *endpoint* koleksi GET pada OAS, ekstraksi item per peran dari struktur respons, seleksi kolom ID melalui prioritas heuristik dengan validasi tipe, serta normalisasi alias untuk *test case*. Pendekatan ini mengurangi konfigurasi manual dan adaptif terhadap variasi skema, namun bergantung pada representativitas item pertama.

2.9 Rule-Based Pattern Algorithm

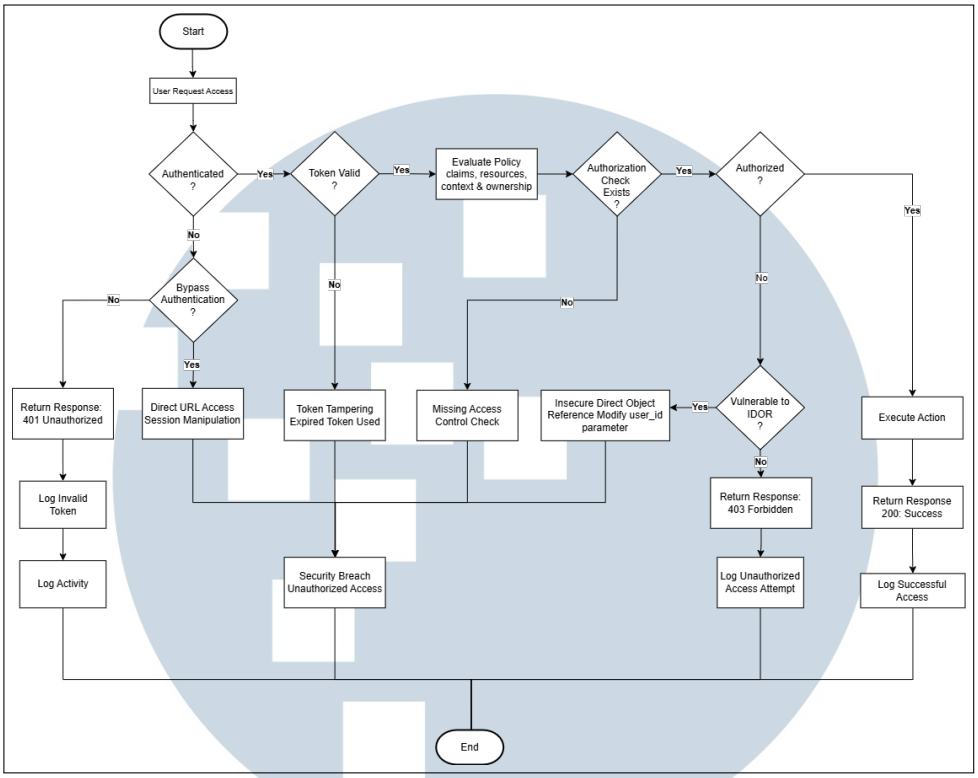
Rule-Based Pattern Algorithm merupakan aturan klasifikasi deterministik yang mengenali pola menggunakan logika eksplisit (*IF–THEN*) berdasarkan pengetahuan domain [40]. Alurnya mencakup praproses dan ekstraksi skema dari OAS serta kebijakan RBAC, pencocokan aturan terhadap respons aktual, resolusi konflik berbasis prioritas, lalu pelabelan kelas dan tingkat risiko. Dalam penelitian ini, aturan menandai indikasi *Broken Access Control* ketika *path* memuat identitas, operasi bersifat sensitif (PUT/DELETE), dan terjadi peran-izin tidak sesuai pada RBAC. Hasil berupa label dan prioritas *explainable* menjadi masukan bagi perencanaan uji berbasis LLM dan modul *auto-discovery* ID untuk fokus pada IDOR/BOLA. Pendekatan ini transparan dan hemat data, namun memerlukan pemeliharaan aturan agar tetap relevan.

2.10 Policy-Based Algorithm

Policy-Based Algorithm mengklasifikasikan keputusan otorisasi dengan merepresentasikan aturan RBAC sebagai aturan evaluatif [41]. Prosesnya mencakup perancangan secara cepat untuk pasangan *method–path* untuk pencarian cepat, pencocokan peran terhadap izin sah pada matriks RBAC, dan penetapan label keputusan (diizinkan/ditolak) berdasarkan hasil eksekusi. Dalam penelitian ini, algoritma diterapkan untuk mendeteksi BOLA dengan membandingkan kebijakan yang semestinya dengan hasil uji coba. Akses yang seharusnya ditolak namun berhasil misalnya *employee* mengakses *endpoint admin* dan menerima respons 200 bukan 401/403/404 diklasifikasikan sebagai *False Negative* dan ditandai sebagai indikasi kerentanan.

2.11 Broken Access Control

Broken Access Control merupakan salah satu ancaman keamanan logika bisnis [42] ketika sistem gagal mendeteksi siapa pengguna yang *login* (*authentication*) untuk mengakses atau melakukan aksi terhadap suatu data dan mengenali izin apa yang dimiliki pengguna (*authorization*) yang dapat mengakses sumber daya tertentu berdasarkan peran pengguna yang dimiliki [43][44] pada Gambar 2.2. Ancaman ini berbahaya karena dapat menyebabkan kebocoran data atau mendapatkan akses suatu *role* dengan mengubah URL atau mengganti ID pada suatu parameter (*bypass*) [45].

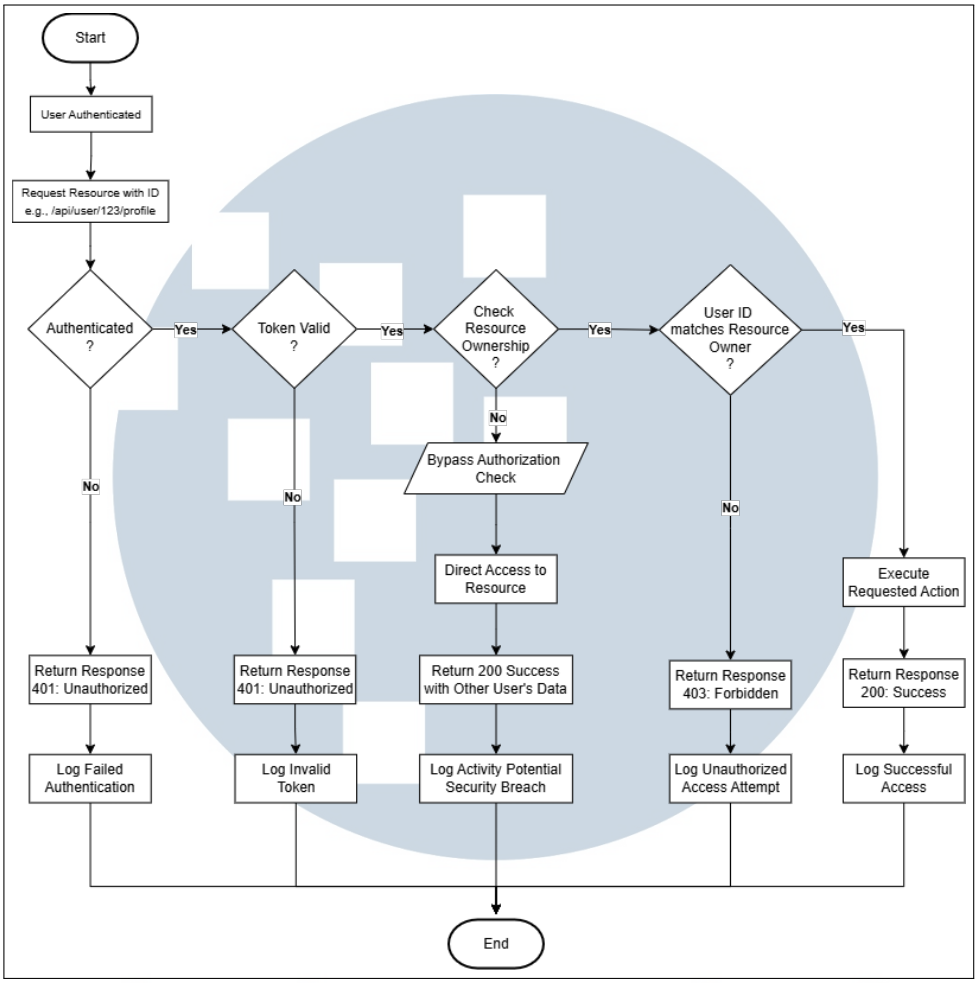


Gambar 2.2. Flowchart broken access control

Open Web Application Security Project (OWASP) membagikan 10 kategori ancaman tinggi pada aplikasi dengan urutan pertama ditempati oleh *Broken Access Control*. Hal ini meningkat dari urutan kelima menjadi pertama dan didukung temuan sebanyak 94% dari aplikasi yang diuji ditemukan BAC [46]. Ancaman ini dapat dilakukan oleh pihak internal dan eksternal, sehingga mengalahkan serangan keamanan lain seperti SQL *injection* dan *security misconfiguration* [47]. Kerentanannya dapat dibagi menjadi dua jenis utama:

- **Horizontal Broken Access Control**

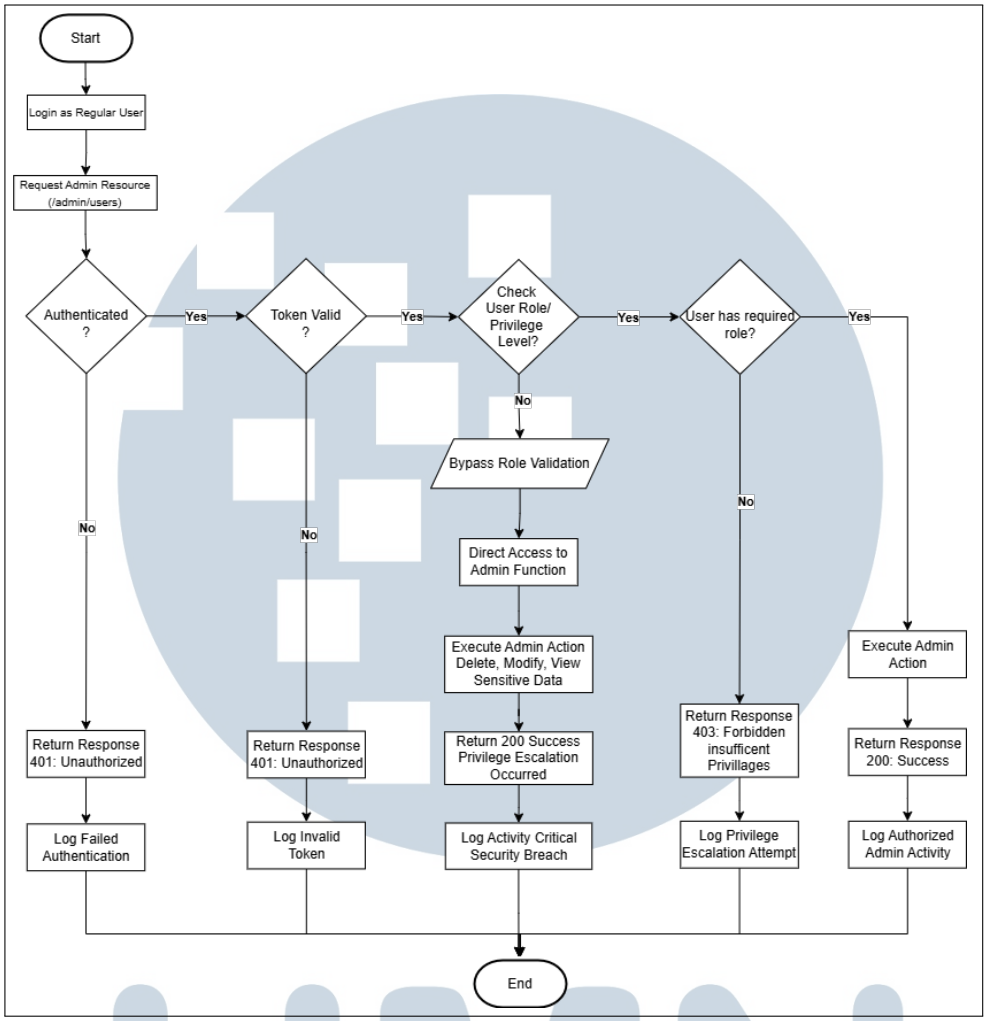
  Horizontal Access Control adalah mekanisme untuk menentukan pengguna yang berhak mengakses suatu *resource* [48]. Ketika *access control* ini rusak, pengguna dengan hak yang setara dapat mengakses, memodifikasi, dan bahkan menghapus data milik pengguna lain [49] dengan mengubah ID pada parameter seperti pada Gambar 2.3. Kerentanan ini disebut sebagai *Insecure Direct Object Reference* (IDOR) [50], di mana pengguna dapat mengakses atau melakukan *action* pada suatu *resource* tanpa sistem melakukan *authorization check*. Kerentanan ini rentan terjadi pada *endpoint* yang memiliki parameter ID.



Gambar 2.3. Flowchart horizontal broken access control

Vertical Broken Access Control

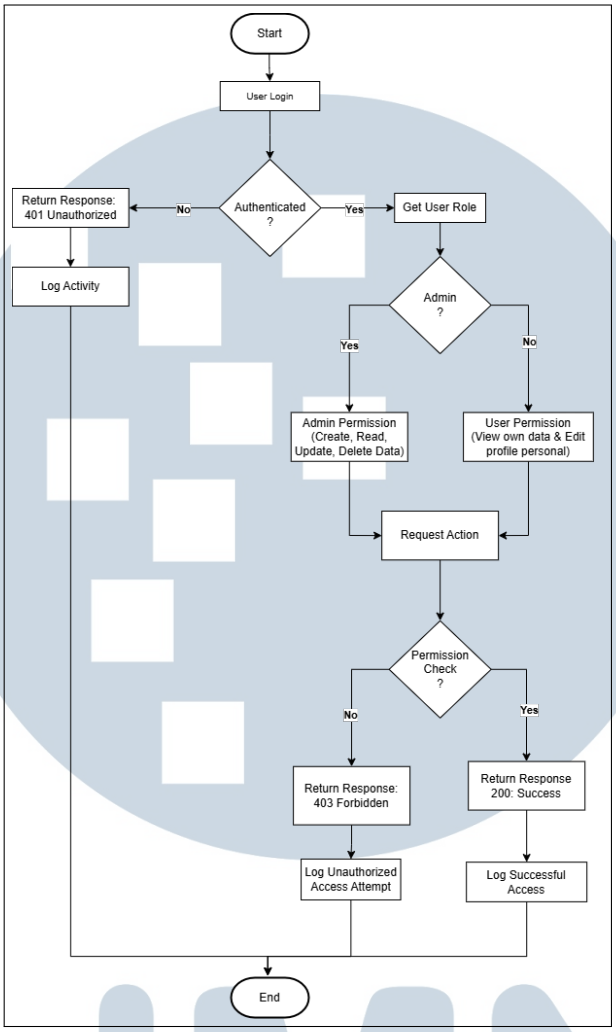
Vertical Access Control adalah mekanisme untuk membatasi akses pengguna terhadap suatu fitur berdasarkan tipe atau hak akses yang dimiliki [51]. Dapat dilihat pada Gambar 2.4 sistem melakukan pemeriksaan berdasarkan *permission* atau *role* yang dimiliki pengguna, sehingga pengguna tidak dapat mengakses fitur atau sumber daya admin yang membutuhkan *role* admin karena dibatasi oleh *Vertical Access Control*. Jika mekanisme *Vertical Access Control* rusak, seluruh pengguna dapat mengakses seluruh fitur tanpa adanya pembatasan atau pemeriksaan hak akses antar *role* [52].



Gambar 2.4. Flowchart vertical broken access control

2.12 Role-Based Access Control

Role-Based Access Control (RBAC) adalah model kontrol akses yang mengelola hak akses pengguna berdasarkan peran yang ditentukan dalam sistem [53]. Pada Gambar 2.5 menampilkan bahwa setiap peran memiliki akses ke sumber daya tertentu sesuai dengan *role* dan *permission*. RBAC membantu mengurangi risiko keamanan dengan memastikan bahwa pengguna hanya dapat mengakses data atau fungsionalitas yang sesuai dengan peran [54]. RBAC dinilai efektif untuk membatasi akses pengguna berdasarkan peran yang dimiliki dan dapat membantu mengurangi kerentanan keamanan yang berkaitan dengan *access control*.



Gambar 2.5. Flowchart role-based access control

2.13 Confusion Matrix

Confusion matrix adalah tabel yang bertujuan untuk mengevaluasi kinerja model klasifikasi dengan cara membandingkan hasil prediksi dan kondisi aktual [55]. Matriks ini membantu melihat seberapa sering model benar atau keliru ketika menyatakan suatu kasus sebagai positif (ada kerentanan) atau negatif (tidak ada kerentanan). Dalam konteks pengujian keamanan, *confusion matrix* digunakan untuk menilai seberapa baik sistem dalam mendeteksi kerentanan dengan mengklasifikasikan hasil uji ke dalam empat kategori utama:

- ***True Positive* (TP)**: *Request* yang seharusnya diizinkan berhasil mendapat respons sukses (200/201).

- ***False Positive* (FP)**: *Request* yang seharusnya diizinkan ditolak (403/401).
- ***False Negative* (FN)**: *Request* yang seharusnya ditolak berhasil mendapat respons sukses (200/201),
- ***True Negative* (TN)**: *Request* yang seharusnya ditolak berhasil ditolak (403/401/404).

Dari keempat kategori yang ada digunakan untuk mengukur kinerja model klasifikasi dengan hasil prediksi model yang sebenarnya, *metrix* yang dihitung berupa:

- *Accuracy*, dengan Rumus (2.1) *metrix* ini menggambarkan proporsi prediksi yang benar dari seluruh kasus.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} \tag{2.1}$$

- *Precision*, dengan Rumus (2.2) menunjukkan ketepatan saat model menyatakan "positif" (semakin sedikit *false positive*, semakin tinggi nilainya).

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2.2}$$

- *Recall* dengan Rumus (2.3) menunjukkan kemampuan menangkap semua kasus positif (semakin sedikit *false negative*, semakin tinggi nilainya).

$$\text{Recall} = \frac{TP}{TP+FN} \tag{2.3}$$

- *F1-score*, dengan Rumus (2.4) merupakan rata-rata harmonik yang menyeimbangkan *precision* dan *recall*.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Meskipun *accuracy* tinggi dianggap baik untuk model, metrik ini bisa menyesatkan jika data tidak seimbang. Oleh karena itu, pengukuran tidak terbatas pada *accuracy* saja, tetapi juga menggunakan metrik lain seperti *precision*, *recall*, dan *F1-score* untuk mengevaluasi kinerja model.

2.14 Coverage

Coverage mengukur seberapa banyak *endpoint* dan skenario yang diuji dibandingkan dengan jumlah total dalam sistem. *Coverage* yang lebih tinggi berarti pengujian lebih menyeluruh terhadap berbagai kombinasi *endpoint* dan peran dengan Rumus (2.5). Dalam pengujian kontrol akses, *coverage* penting untuk memastikan setiap kombinasi akses diuji guna menemukan celah keamanan.

$$\text{Coverage} = \frac{\text{Tested Endpoint} \times \text{Role}}{\text{Total Endpoint} \times \text{Role}} \times 100\% \tag{2.5}$$

Metrik *coverage* pada penelitian ini berperan sebagai indikator sejauh mana BYEBAC benar-benar menjelajahi seluruh kombinasi *endpoint–role* yang relevan dengan skenario *Broken Access Control*. Nilai *coverage* yang tinggi membantu meminimalkan area yang belum tersentuh pengujian sehingga mengurangi risiko adanya kerentanan yang tersembunyi, serta melengkapi metrik performa lain (*precision*, *recall*, dan *accuracy*) dalam menilai efektivitas keseluruhan strategi pengujian otomatis yang diusulkan.

2.15 Evaluasi Model

Evaluasi model dilakukan untuk menilai seberapa baik sistem bekerja dalam mendeteksi kerentanan di dalamnya. Berikut diantaranya:

- **Confusion Matrix**: *Matrix* yang digunakan untuk mengukur seberapa tepat model dalam memprediksi hasil yang benar dan salah. Matriks ini membantu untuk memahami seberapa banyak kesalahan yang dilakukan model dalam mendeteksi kerentanan.
- **Precision**: Metrik ini mengukur sejauh mana model berhasil memprediksi dengan benar ketika menyatakan sesuatu itu ”positif” (misalnya, ada kerentanan), dibandingkan dengan berapa banyak prediksi yang ternyata salah.

- **Recall**: Metrik ini menunjukkan seberapa banyak kasus yang benar-benar positif berhasil ditemukan oleh model, dibandingkan dengan jumlah seluruh kasus positif yang ada.
- **F1-Score**: kombinasi dari *precision* dan *recall*, yang memberikan gambaran seimbang antara keduanya, terutama ketika ada ketidakseimbangan antara prediksi positif dan negatif.
- **Accuracy**: Mengukur seberapa sering model benar dalam memprediksi hasil, dibandingkan dengan jumlah total prediksi yang dilakukan.
- **Coverage**: Metrik ini mengukur sejauh mana model telah menguji seluruh kombinasi *endpoint* dan peran yang mungkin terjadi dalam aplikasi.
- **Time to Detect**: Mengukur waktu yang dibutuhkan model untuk mendeteksi kerentanan, yang penting untuk menilai efisiensi model dalam pengujian.

Metrik yang ada dapat membantu untuk lebih memahami bagaimana model bekerja dan sejauh mana model tersebut dapat diandalkan untuk mendeteksi masalah keamanan diaplikasi.