BAB 2
LANDASAN TEORI

2.1 Prosedur dan Alur Pengumpulan Skripsi di Universitas Multimedia Nusantara pada *Platform* Akademik

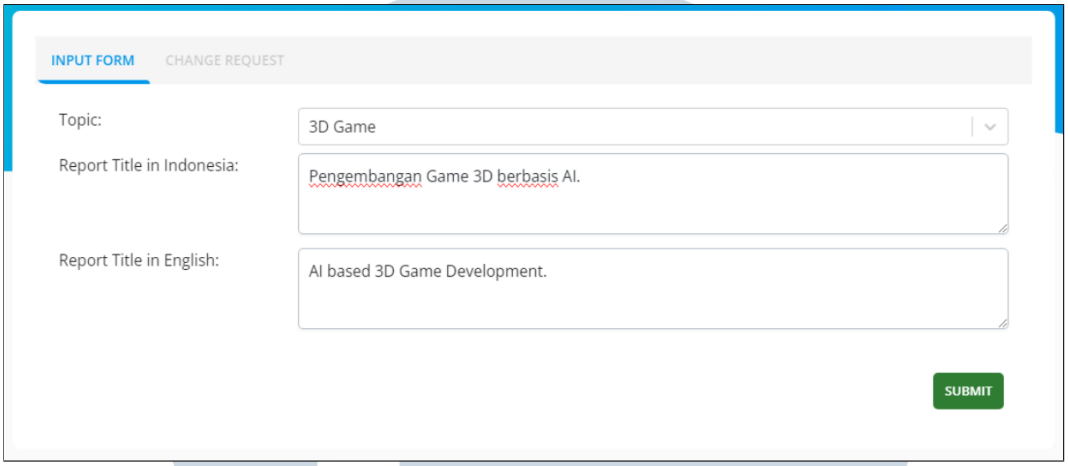
Prosedur penyusunan tugas akhir di Universitas Multimedia Nusantara, sebagaimana diatur dalam panduan *platform* Akademik, menjadi studi kasus untuk penelitian ini [8]. Alur ini dirancang untuk menjaga kualitas akademik, namun dalam praktiknya menyoroti beberapa titik yang rentan dari segi validitas dan keamanan dokumen skripsi. Proses tersebut diawali pendaftaran judul, pendaftaran sidang, sidang, revisi, dan pengunggahan laporan akhir dengan setiap tahapan ini melibatkan berbagai persetujuan yang secara tradisional mengandalkan tanda tangan manual pada dokumen fisik [8].

Menurut Manajer BIA Universitas Multimedia Nusantara, alur pengumpulan skripsi saat ini sudah menggunakan pendekatan digital, di mana mahasiswa mengirimkan *file* skripsi langsung ke *platform* Akademik yang nantinya divalidasi atau disetujui oleh dosen pembimbing [2]. Namun, BIA belum memiliki cara yang efektif dalam memastikan keaslian tanda tangan pada lembar persetujuan. Validasi hanya bergantung pada persetujuan (*approval*) dari dosen di dalam sistem, dengan asumsi bahwa jika dosen telah menyetujui, maka tanda tangan di dalam dokumen juga asli [2].

Menurut Dosen Informatika Universitas Multimedia Nusantara, alur penandatanganan dokumen skripsi memang masih dilakukan secara manual, bahkan sering kali mahasiswa mencoba meminta tanda tangan kepala program studi tanpa terlebih dahulu ditandatangani oleh dosen pembimbing dan penguji [9]. Beliau juga menerangkan bahwa selain itu, diperlukan sebuah mekanisme dimana dokumen skripsi yang ditolak, harus mengulangi alur pengumpulan dari awal yaitu dari dosen pembimbing, penguji, lalu terakhir kepala program studi [9].

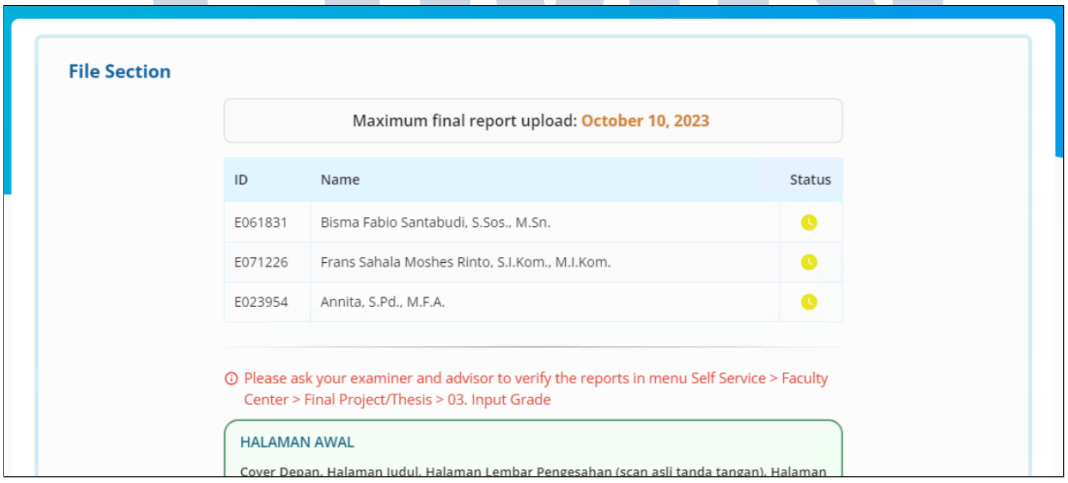
Merujuk pada panduan penggunaan *platform* Akademik untuk mahasiswa Universitas Multimedia Nusantara, alur pengumpulan skripsi di Universitas Multimedia Nusantara telah dilakukan secara digital, namun validasi utamanya masih bergantung pada persetujuan dosen dan integritas dokumen yang diunggah mahasiswa [10]. Proses digital ini diawali dengan mahasiswa memasukkan judul skripsi dalam dua bahasa (Indonesia dan Inggris) melalui portal Akademik.

Sistem juga menyediakan fitur untuk mengajukan perubahan judul, di mana status permintaan tersebut dapat dipantau.



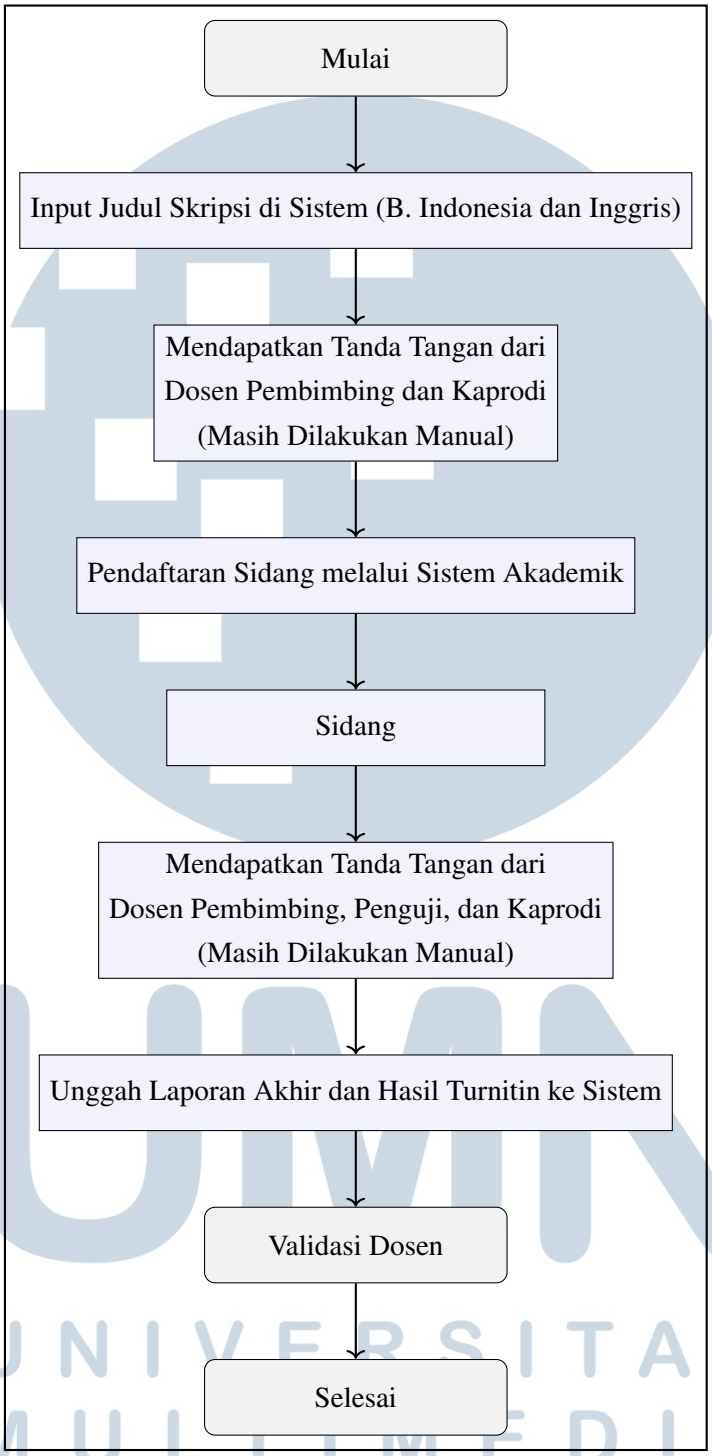
Gambar 2.1. Formulir input judul skripsi di portal akademik UMN.

Tahap krusial adalah pendaftaran sidang, yang dibagi menjadi tiga langkah utama: pemenuhan prasyarat, pengunggahan berkas, dan registrasi. Pertama, sistem Akademik secara otomatis memeriksa apakah judul skripsi telah terdaftar sebelum mahasiswa dapat melanjutkan. Kemudian, mahasiswa diwajibkan mengunggah berkas laporan skripsi dan hasil pemeriksaan Turnitin dalam format PDF. Berkas ini kemudian diajukan untuk ditinjau (*submit for review*) oleh dosen. Pada proses ini, mahasiswa di ekspektasikan telah mendapatkan semua tanda tangan dosen dan kepala program studi secara manual. Setelah semua berkas disetujui, mahasiswa dapat memilih periode dan jenis sidang yang tersedia untuk mendaftar.



Gambar 2.2. Antarmuka pengunggahan berkas untuk pendaftaran sidang, menunjukkan status peninjauan oleh dosen.

Proses yang paling menyoroti kelemahan dari segi validasi keamanan dokumen terjadi pada tahap akhir, yaitu pengunggahan laporan akhir (*Final Report*) seperti pada Gambar 2.2. Hal ini mengonfirmasi bahwa sistem Akademik, meskipun digital, masih bergantung pada dokumen pindaian (*scanned document*) untuk bukti persetujuan. Tidak ada mekanisme verifikasi untuk tanda tangan tersebut, dan validasi hanya berupa persetujuan akhir dari dosen di dalam sistem setelah seluruh berkas diunggah. Ini menegaskan celah yang disebutkan sebelumnya, yaitu keaslian tanda tangan tidak diverifikasi oleh sistem itu sendiri, melainkan diasumsikan valid jika dosen dan kepala program studi memberikan *approval* final pada portal.



Gambar 2.3. Alur Pengumpulan Skripsi UMN (versi digital dengan validasi dokumen)

Kelemahan pada *workflow* ini membuka celah manipulasi oleh mahasiswa. Permasalahan inilah yang menjadi fokus utama dan hendak dijawab oleh sistem yang diusulkan dalam penelitian ini.

2.2 PHP-Laravel

Laravel adalah framework PHP open-source yang dirancang untuk mempercepat dan mempermudah pengembangan aplikasi web dengan menerapkan pola arsitektur *Model–View–Controller* (MVC) serta menyediakan beragam fitur bawaan seperti *routing*, migrasi basis data, sistem otentikasi, dan *template engine* Blade. Laravel dikembangkan oleh Taylor Otwell sejak 2011 sebagai alternatif yang lebih lengkap dibandingkan framework sebelumnya [11]. Eloquent ORM pada Laravel memudahkan interaksi dengan basis data melalui sintaksis objek, sedangkan Artisan CLI menyediakan perintah otomatis untuk membuat kontroler, model, dan migrasi. Blade memungkinkan pembuatan tampilan dinamis dengan sintaks sederhana dan mendukung pewarisan template.

2.3 FPDI

FPDI (Free PDF Document Importer) adalah pustaka PHP yang menyediakan *Class* untuk mengimpor halaman dari dokumen PDF yang sudah ada dan menggunakan halaman tersebut sebagai template dalam FPDF atau pustaka sejenis lainnya. FPDI dikembangkan oleh Setasign dan dirilis di bawah lisensi MIT sejak versi 1.6 [12]. Penggunaan FPDI dimulai dengan memuat berkas PDF sumber, mengimpor halaman tertentu, lalu meletakkan halaman tersebut pada dokumen baru yang dihasilkan oleh FPDF. FPDI tidak memerlukan ekstensi PHP khusus selain pustaka PDF generator.

2.3.1 SHA-512

SHA-512 (*Secure Hash Algorithm 512-bit*) adalah salah satu fungsi *hash* kriptografis yang termasuk dalam keluarga SHA-2, yang dikembangkan oleh *National Security Agency* (NSA) dan dipublikasikan oleh *National Institute of Standards and Technology* (NIST) [13]. Algoritma ini menghasilkan nilai *hash* dengan panjang 512 *bit*, sehingga memberikan tingkat keamanan yang sangat tinggi terhadap serangan *brute-force* dan *collision attacks* [13]. SHA-512 dipilih dalam penelitian ini karena reputasinya yang kuat dalam menjaga integritas data pada berbagai aplikasi keamanan.

Cara kerja SHA-512 didasarkan pada prinsip-prinsip kriptografi untuk memastikan keamanan dan keandalan. Properti utamanya meliputi:

- *One-way Function*: Proses *hashing* dari data asli ke nilai *hash* mudah dilakukan, namun sangat sulit secara komputasi untuk mengembalikan nilai *hash* menjadi data aslinya [13].
- *Deterministic*: Data input yang sama akan selalu menghasilkan nilai *hash* yang sama persis, tidak peduli berapa kali prosesnya diulang.
- *Avalanche Effect*: Perubahan sekecil apapun pada data input, bahkan hanya satu *bit*, akan menghasilkan nilai *hash* yang sama sekali berbeda dan tidak dapat diprediksi. Properti ini memastikan bahwa modifikasi sekecil apapun pada dokumen dapat terdeteksi.
- *Collision Resistance*: Sangat sulit secara komputasi untuk menemukan dua data input berbeda yang menghasilkan nilai *hash* yang sama [13].

Secara garis besar, proses perhitungan SHA-512 mengikuti standar FIPS PUB 180-4 dan melibatkan beberapa tahapan [13]:

1. *Padding*: Data input (dalam kasus ini, isi dokumen PDF) ditambahkan -bit tambahan hingga panjang totalnya menjadi kelipatan 1024 -bit.
2. *Parsing*: Data yang sudah di-*padding* dibagi menjadi blok-blok berukuran 1024 -bit.
3. Inisialisasi: Algoritma dimulai dengan delapan nilai *hash* awal (H0 sampai H7) yang merupakan konstanta tetap berukuran 64 -bit.
4. Komputasi Iteratif: Sistem memproses setiap blok data secara berurutan. Sebuah fungsi kompresi yang kompleks diaplikasikan pada setiap blok bersama dengan nilai *hash* dari iterasi sebelumnya. Fungsi ini melibatkan 80 putaran operasi logika, pergeseran *bit*, dan penjumlahan modular.
5. Hasil Akhir: Setelah semua blok data selesai diproses, hasil akhir dari delapan nilai *hash* tersebut digabungkan untuk membentuk *digest* SHA-512 berukuran 512 -bit.

Pada sistem yang dibangun dengan *framework* PHP-Laravel, perhitungan *hash* SHA-512 tidak dilakukan secara manual. Sistem memanfaatkan fungsi *hashing* bawaan dari PHP untuk kemudahan dan keamanan. Fungsi yang digunakan adalah `hash()`, seperti yang terlihat pada implementasi di Bab 3.

```
$proposal->hash_value = hash('sha512', file_get_contents(
    $originalPdfPath));
```

Kode 2.1: Implementasi Perhitungan Hash SHA-512

Kode 2.1 menunjukkan bahwa sistem memanggil fungsi `hash()` dengan parameter pertama `'sha512'` untuk menentukan algoritma yang digunakan, dan parameter kedua adalah konten dari *file* PDF yang dibaca secara keseluruhan. Hasil dari fungsi ini adalah sebuah *string* heksadesimal 128 karakter (merepresentasikan 512 *bit*) yang kemudian disimpan ke dalam *database*.

2.4 XML *Advanced Electronic Signatures* (XAdES)

XML *Advanced Electronic Signatures* (XAdES) merupakan serangkaian ekstensi untuk standar *XML-DSig* yang dikelola oleh *European Telecommunications Standards Institute* (ETSI), dirancang untuk memenuhi kebutuhan akan tanda tangan digital yang bisa menyematkan metadata informasi [14]. Berbeda dengan tanda tangan PDF standar (PAdES) yang menyematkan data dalam format *binary*, XAdES menggunakan format XML yang terstruktur sehingga dapat dibaca oleh manusia maupun mesin. Keunggulan ini memungkinkan penyematan informasi tambahan yang kaya dan terverifikasi secara kriptografis di dalam tanda tangan itu sendiri [4].

Format XML yang terstruktur ini menjadi dasar dari fungsionalitas sistem yang diusulkan. Dalam konteks penelitian ini, beberapa elemen kunci dari struktur XAdES dimanfaatkan untuk mencatat informasi penting terkait proses pengesahan dokumen skripsi. Struktur metadata ini kemudian disematkan langsung ke dalam dokumen PDF. Berikut adalah rincian dari beberapa komponen penting yang digunakan:

- `ds:KeyInfo`: Elemen ini berfungsi untuk memberikan informasi mengenai kunci dan sertifikat yang digunakan untuk menandatangani dokumen. Di dalamnya terdapat elemen turunan seperti `X509Data`, yang berisi `X509Certificate`. Elemen `X509Certificate` memuat sertifikat digital publik, yang digunakan oleh verifikator untuk memastikan identitas penanda tangan adalah sah.

- `xades:SignedSignatureProperties`: Properti ini berisi informasi tambahan mengenai tanda tangan itu sendiri dan ikut diamankan secara kriptografis. Beberapa properti yang relevan dalam penelitian ini antara lain:

- `SigningTime`: Elemen ini mencatat informasi waktu dan tanggal kapan proses penandatanganan dilakukan. Informasi ini krusial untuk audit dan melacak kronologi persetujuan dokumen.
- `SigningCertificateV2`: Berbeda dengan hanya menyertakan sertifikat, properti ini mereferensikan sertifikat penanda tangan dengan menggunakan nilai *hash* dari sertifikat tersebut. Pendekatan ini memberikan jaminan integritas yang lebih kuat bahwa sertifikat yang digunakan tidak berubah.
- `DataObjectFormat`: Properti ini mendeskripsikan tipe data dari objek yang ditandatangani. Sebagai contoh, dalam implementasinya, properti ini akan menyatakan bahwa objek yang ditandatangani adalah sebuah dokumen dengan *MimeType* 'application/pdf'. Hal ini penting untuk mencegah ambiguitas saat proses verifikasi.

Kemampuan untuk menyematkan metadata yang kaya dan terstruktur inilah yang membuat XAdES dipilih dalam penelitian ini. Informasi seperti peran penanda tangan dan waktu penandatanganan dapat dicatat dan diverifikasi secara andal, yang merupakan sebuah fungsionalitas penting untuk alur kerja akademik dan kebutuhan audit.

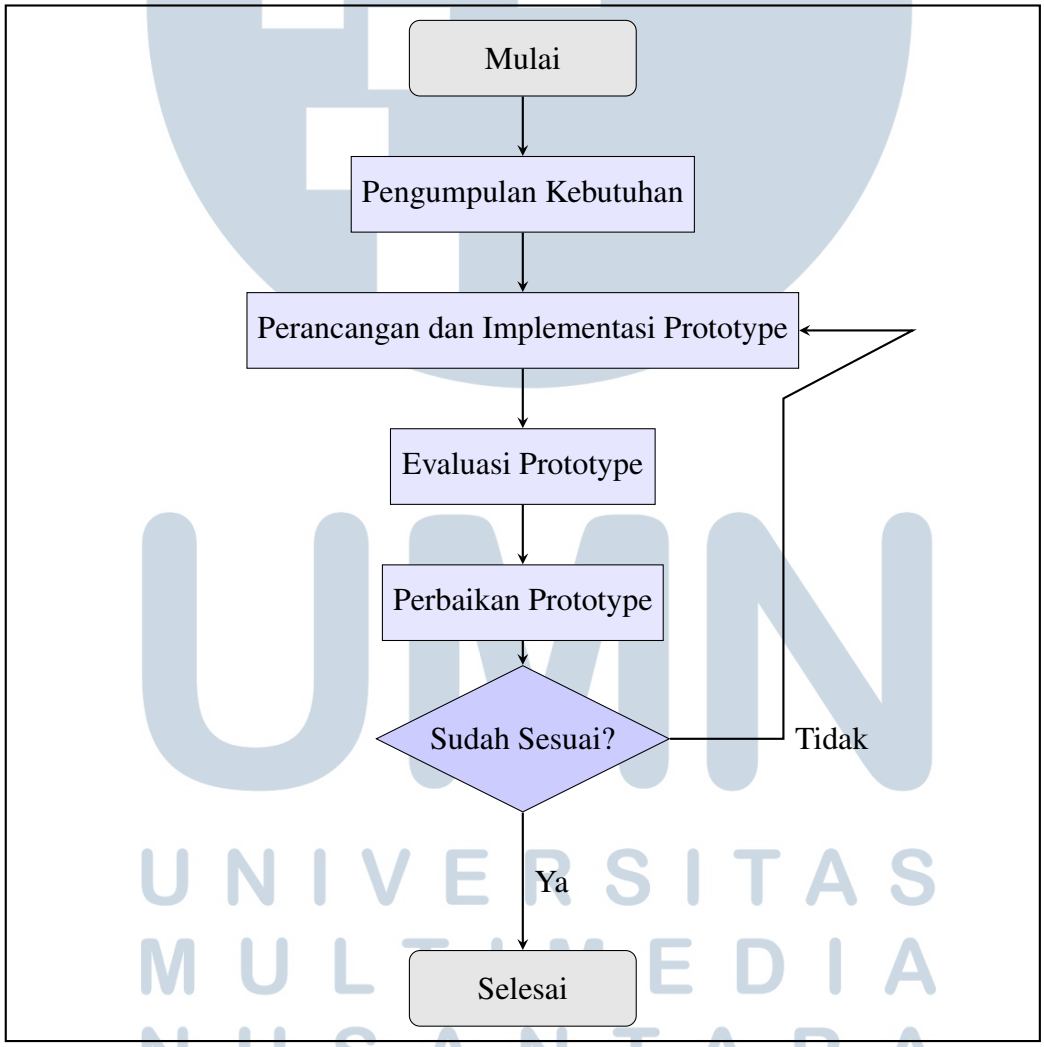
2.4.1 Metodologi *Prototyping*

Metodologi *prototyping* adalah salah satu metode pengembangan *software* yang memungkinkan pengembang untuk membuat versi awal dari sistem atau yang biasa disebut dengan *prototype* [15]. *Prototype* ini digunakan untuk mendemonstrasikan konsep, menguji desain, dan menemukan masalah serta solusi yang mungkin ada pada sistem yang akan dibangun [15].

Metode ini sangat berguna ketika kebutuhan sistem tidak sepenuhnya dipahami di awal, atau ketika pengguna akhir perlu terlibat secara aktif dalam proses pengembangan untuk memberikan masukan [15]. Dengan adanya *prototype*, pengguna dapat melihat dan merasakan bagaimana sistem akan bekerja, sehingga dapat memberikan umpan balik yang lebih akurat dan relevan [15]. Proses ini biasanya melibatkan beberapa tahapan utama, yaitu:

1. Pengumpulan Kebutuhan (*Requirement Gathering*): Pengembang dan pengguna akhir (mahasiswa, dosen, Kaprodi, BIA) bekerja sama untuk mengidentifikasi kebutuhan dasar sistem yang akan dibangun.

2. Perancangan dan Implementasi *Prototype*: Sebuah *prototype* dibuat untuk menunjukkan fungsionalitas inti dari sistem.
3. Evaluasi *Prototype*: Pengguna akhir mencoba *prototype* dan memberikan umpan balik.
4. Perbaikan *Prototype*: Berdasarkan umpan balik (*feedback*), *prototype* diperbaiki dan disempurnakan.
5. Iterasi: Tahap 3 dan 4 diulang hingga sistem memenuhi kebutuhan pengguna.



Gambar 2.4. Flowchart Metodologi Prototyping

Alur dari metodologi prototyping yang digunakan pada penelitian ini dapat dilihat pada Gambar 2.4.

2.5 *Common Vulnerability Scoring System* (CVSS)

Common Vulnerability Scoring System (CVSS) adalah sebuah *framework* yang digunakan untuk mengukur tingkat kerentanan keamanan *software* [6]. CVSS memberikan skor numerik dari 0.0 hingga 10.0, di mana skor yang lebih tinggi menunjukkan tingkat *vulnerability* yang lebih besar [16]. Tujuan utama dari CVSS adalah untuk menyediakan cara yang terstandarisasi dalam menilai *vulnerability*, memungkinkan organisasi untuk memprioritaskan sumber daya mereka secara efektif dalam menangani *threat*.

Berikut merupakan metrik (*metrics*) dari CVSS yang akan diujikan pada penelitian ini:

- *Attack Vector* (AV): Merefleksikan bagaimana kerentanan dapat dieksploitasi.
  - *Network* (N): Kerentanan dapat dieksploitasi dari jarak jauh melalui jaringan.
  - *Adjacent* (A): Pelaku harus berada pada segmen jaringan yang sama dengan target.
  - *Local* (L): Pelaku memerlukan akses lokal ke sistem target, seperti melalui *keyboard* atau konsol.
  - *Physical* (P): Pelaku memerlukan akses fisik ke komponen sistem.
- *Attack Complexity* (AC): Menggambarkan hambatan teknis yang harus diatasi pelaku agar serangan berhasil.
  - *Low* (L): Tidak ada hambatan khusus; pelaku dapat berulang kali menyerang target dengan hasil yang sama.
  - *High* (H): Pelaku memerlukan usaha lebih, misalnya harus mengatasi mekanisme keamanan.
- *Attack Requirements* (AT): Mencatat kondisi prasyarat pada sistem target yang harus ada agar serangan berhasil. Pada penelitian ini metrik ini tidak digunakan. Metrik ini menjadi relevan hanya ketika sebuah serangan memerlukan kondisi atau konfigurasi non-standar pada sistem target. Untuk banyak skenario serangan umum, prasyarat khusus seperti ini tidak diperlukan, sehingga nilai untuk metrik ini adalah *None* [7].
  - *None* (N): Tidak ada kondisi khusus yang diperlukan.

  - *Present* (P): Membutuhkan kondisi tertentu yang dapat dieksploitasi, seperti konfigurasi spesifik pada sistem target.

- *Privileges Required* (PR): Mengukur tingkat hak akses yang harus dimiliki pelaku sebelum dapat melancarkan serangan.
  - *None* (N): Tidak memerlukan hak akses apa pun.
  - *Low* (L): Memerlukan hak akses setingkat pengguna biasa.
  - *High* (H): Memerlukan hak akses setingkat administrator.
- *User Interaction* (UI): Menilai apakah serangan memerlukan interaksi dari pengguna selain pelaku.
  - *None* (N): Tidak memerlukan interaksi dari siapa pun.
  - *Passive* (P): Memerlukan interaksi minimal dari pengguna, seperti mengunjungi situs berbahaya.
  - *Active* (A): Pengguna harus melakukan tindakan spesifik, seperti membuka *file* berbahaya atau mengklik tautan.
- *Confidentiality* (C): Mengukur dampak terhadap kerahasiaan data jika kerentanan berhasil dieksploitasi.
  - *High* (H): Terjadi kebocoran informasi total. Penyerang dapat mengakses semua data pada sistem yang terdampak.
  - *Low* (L): Terjadi kebocoran informasi sebagian. Penyerang hanya mendapatkan akses ke beberapa data atau informasi terbatas.
  - *None* (N): Tidak ada data yang bocor sama sekali.
- *Integrity* (I): Mengukur dampak terhadap keaslian atau keutuhan data.
  - *High* (H): Terjadi kehilangan integritas total. Penyerang dapat memodifikasi semua data atau file pada sistem target, seringkali untuk menyembunyikan jejak serangan.
  - *Low* (L): Modifikasi data terbatas. Penyerang hanya dapat mengubah sebagian kecil data atau file.
  - *None* (N): Tidak ada modifikasi data sama sekali.

- *Availability* (A): Mengukur dampak terhadap ketersediaan sistem atau layanan.
  - *High* (H): Terjadi kehilangan ketersediaan total. Penyerang dapat membuat sistem atau layanan sama sekali tidak dapat diakses oleh pengguna yang sah (contoh: serangan Denial of Service).
  - *Low* (L): Kinerja sistem menurun atau layanan menjadi tidak stabil dan sering terganggu.
  - *None* (N): Tidak ada dampak sama sekali terhadap ketersediaan sistem.

Penilaian skor CVSS dilakukan dengan mencari kombinasi nilai dari setiap metrik dalam sebuah tabel vektor pencarian yang telah disediakan oleh FIRST.org [6]]. Setiap kombinasi vektor ini telah memiliki skor dan tingkat keparahan kualitatif yang telah dihitung sebelumnya. Proses ini menghilangkan kebutuhan untuk perhitungan manual dan memastikan konsistensi penilaian [6]. Tingkat keparahan kualitatif didefinisikan sebagai berikut:

- *None*: 0.0
- *Low*: 0.1 - 3.9
- *Medium*: 4.0 - 6.9
- *High*: 7.0 - 8.9
- *Critical*: 9.0 - 10.0

Skor CVSS secara dihitung berdasarkan beberapa sub-skor, yaitu *Exploitability* dan *Impact*. Untuk CVSS, komponen utamanya adalah *EQ1* hingga *EQ6* yang merepresentasikan berbagai kombinasi metrik. Contoh penyederhanaan formula untuk menghitung dampak gabungan (*Combined Impact*) dapat dilihat pada Rumus (2.1).

$$\text{ISS}_{\text{combined}} = 1 - ((1 - \text{ISS}_{\text{VC}}) \times (1 - \text{ISS}_{\text{VI}}) \times (1 - \text{ISS}_{\text{VA}})) \quad (2.1)$$

Dimana ISS (*Impact Sub-Score*) untuk setiap dampak (*Confidentiality*, *Integrity*, *Availability*) dihitung untuk sistem yang rentan dan sistem terdampak. Hasil dari berbagai perhitungan ini kemudian digabungkan untuk menghasilkan

skor akhir yang tunggal melalui serangkaian langkah perhitungan vektor yang kompleks.

Dalam penelitian ini, CVSS digunakan untuk mengevaluasi secara kuantitatif tingkat risiko dari berbagai skenario serangan yang telah diidentifikasi, memberikan dasar yang objektif untuk analisis keamanan *framework* yang diusulkan.

2.6 *Open Web Application Security Project* (OWASP) *Top* 10

Open Web Application Security Project (OWASP) merupakan sebuah organisasi nirlaba yang berfokus pada peningkatan keamanan perangkat lunak [17]. Salah satu proyek utama dari OWASP adalah daftar "OWASP Top 10", yang secara berkala diperbarui untuk mengidentifikasi dan mempublikasikan sepuluh risiko keamanan aplikasi web yang paling kritis [17]. OWASP Top 10 akan digunakan pada penelitian ini untuk mengevaluasi keamanan dan kerentanan sistem yang telah dibuat bersama dengan CVSS [7].

- A01:2021 – *Broken Access Control*: Kerentanan ini terjadi ketika batasan terhadap apa yang seharusnya dapat dilakukan oleh seorang pengguna tidak diterapkan dengan benar. Pelaku dapat mengeksploitasi celah ini untuk mendapatkan akses tidak sah ke fungsionalitas atau data, seperti mengakses akun pengguna lain, melihat *file* sensitif, atau mengubah data dan hak akses pengguna lain [18].
- A02:2021 – *Cryptographic Failures*: Kerentanan ini berfokus pada kegagalan terkait kriptografi yang sering kali berujung pada terungkapnya data sensitif [19]. Kegagalan ini dapat mencakup berbagai hal, mulai dari tidak digunakannya enkripsi untuk data, penggunaan algoritma yang lemah, hingga manajemen kunci yang buruk, yang dapat membuka peluang bagi pelaku untuk mencuri atau memodifikasi data [19].
- A03:2021 – *Injection*: Kerentanan *injection* terjadi ketika data yang tidak tepercaya dikirim ke sebuah *interpreter* sebagai bagian dari perintah. Data berbahaya dari pelaku dapat mengelabui *interpreter* untuk menjalankan perintah yang tidak diinginkan atau memberikan akses ke data tanpa otorisasi [20].

- A04:2021 – *Insecure Design*: Kategori ini berfokus pada risiko yang terkait dengan kelemahan pada tahap perancangan dan arsitektur sistem [21]. Kerentanan ini tidak disebabkan oleh kesalahan implementasi, melainkan oleh ketiadaan pengendalian keamanan yang efektif sejak awal perancangan [21].
- A05:2021 – *Security Misconfiguration*: Kesalahan konfigurasi keamanan dapat terjadi pada berbagai tingkatan, mulai dari sistem operasi hingga aplikasi [22]. Contohnya termasuk membiarkan fitur *default* yang tidak perlu tetap aktif atau menampilkan pesan *error* yang terlalu detail sehingga mengungkap informasi sensitif [22].
- A06:2021 – *Vulnerable and Outdated Components*: Kerentanan ini terjadi ketika aplikasi menggunakan komponen atau *library* pihak ketiga yang diketahui memiliki celah keamanan atau sudah usang [23]. Pelaku dapat secara aktif mengeksploitasi kerentanan yang sudah diketahui publik pada komponen tersebut untuk menyerang sistem [23].
- A07:2021 – *Identification and Authentication Failures*: Kegagalan dalam fungsi identifikasi dan autentikasi dapat memungkinkan pelaku untuk mengambil alih akun pengguna [24]. Contohnya termasuk mengizinkan serangan *brute force* karena tidak adanya pembatasan percobaan *login* atau manajemen sesi yang lemah [24].
- A08:2021 – *Software and Data Integrity Failures*: Kategori ini berfokus pada kegagalan yang berkaitan dengan integritas perangkat lunak dan data, seperti pembaruan perangkat lunak atau data kritis tanpa verifikasi integritas [25]. Salah satu contoh serius adalah ketika aplikasi menggunakan komponen dari sumber yang tidak tepercaya, yang memungkinkan potensi penyisipan kode berbahaya [25].
- A09:2021 – *Security Logging and Monitoring Failures*: Kegagalan dalam pencatatan (*logging*) dan pemantauan (*monitoring*) keamanan dapat menghambat kemampuan untuk mendeteksi dan merespons insiden keamanan [26]. Tanpa *log* yang memadai, administrator mungkin tidak menyadari bahwa sistem mereka telah disusupi [26].
- A10:2021 – *Server-Side Request Forgery* (SSRF): Kerentanan SSRF terjadi ketika aplikasi web mengambil sumber daya dari URL jarak jauh tanpa

memvalidasi URL yang diberikan pengguna [27]. Hal ini memungkinkan pelaku untuk memaksa server mengirimkan permintaan ke tujuan yang tidak terduga, yang dapat digunakan untuk memindai jaringan internal atau mengakses layanan internal [27].

2.7 *User Acceptance Testing* (UAT)

User Acceptance Testing (UAT) adalah tahap pengujian akhir dalam siklus pengembangan *software*, di mana sistem dievaluasi oleh pengguna akhir (*end-user*) dalam *environment* yang mensimulasikan skenario dunia nyata (pada kasus ini Universitas Multimedia Nusantara). Fokus utama UAT pada penelitian ini adalah memastikan *software* yang tepat telah dibuat untuk memenuhi kebutuhan pengguna, terutama pada kasus sistem Akademik Universitas Multimedia Nusantara. Tujuan utamanya adalah untuk memastikan bahwa *software* tersebut sesuai dengan tujuannya, selaras dengan kebutuhan bisnis, dan dapat diterima untuk digunakan oleh pengguna di Universitas Multimedia Nusantara.

UAT dilaksanakan dengan bentuk *black box testing*, di mana penguji mengevaluasi fungsionalitas sistem hanya berdasarkan masukan dan keluarannya, tanpa perlu mengetahui struktur kode internal [15]. Dalam prosesnya, pengguna menjalankan serangkaian skenario atau kasus uji (*test cases*) yang telah dirancang untuk menyimulasikan alur kerja bisnis dari awal hingga akhir. Keberhasilan UAT diukur melalui metrik kuantitatif dan kualitatif yang sering kali berakar pada *framework* teori *technology acceptance model* [15]:

- Tingkat Keberhasilan Tugas (*Task Success Rate*): Metrik ini mengukur efektivitas sistem dengan menghitung persentase tugas yang berhasil diselesaikan oleh pengguna. Formula untuk menghitungnya adalah:

$$\text{TSR} = \frac{\text{Jumlah Tugas yang Berhasil Diselesaikan}}{\text{Total Jumlah Tugas yang Dikerjakan}} \times 100\%$$

  Meskipun bukan merupakan pengukuran langsung, hasil dari metrik ini dapat secara signifikan memengaruhi *Perceived Usefulness* (Persepsi Kegunaan). Ketika pengguna secara dapat mencapai tujuan mereka menggunakan sistem. Oleh karena itu, TSR berfungsi sebagai bukti kuantitatif yang mendukung evaluasi konstruk *Perceived Usefulness*.

- Umpan Balik Kualitatif: Catatan, komentar, dan saran deskriptif yang

diberikan oleh pengguna selama sesi pengujian. Umpan balik ini sangat berharga untuk mengidentifikasi area perbaikan, masalah kegunaan, dan celah fungsionalitas yang mungkin terlewatkan selama fase pengujian sebelumnya.

Dalam konteks penelitian ini, UAT dipilih sebagai metodologi validasi utama karena bertujuan memastikan bahwa sistem yang dibangun tidak hanya aman dan stabil secara teknis, tetapi yang lebih penting, dapat diterima dan diadopsi secara efektif oleh pengguna akademik Universitas Multimedia Nusantara (mahasiswa, dosen, Kaprodi, BIA). Dengan demikian, UAT memberikan bukti bahwa solusi *software* yang dihasilkan benar-benar memberikan nilai tambah dan mendukung proses bisnis yang ada di lingkungan universitas.