BAB II

LANDASAN TEORI

2.1 Penelitian Terdahulu

Penelitian terdahulu dikaji untuk memahami berbagai pendekatan yang telah digunakan dalam pengembangan sistem informasi, pengelolaan data *sustainability*, perancangan basis data, serta implementasi layanan berbasis RESTful API. Kajian ini dilakukan untuk mengidentifikasi metode yang relevan, menemukan kesenjangan penelitian, serta menentukan kontribusi yang dapat diberikan oleh penelitian ini. Melalui analisis terhadap penelitian-penelitian yang memiliki keterkaitan dengan pengelolaan data *sustainability* dan pengembangan sistem berbasis basis data, diharapkan dapat diperoleh dasar yang kuat dalam merancang arsitektur basis data dan RESTful API yang mampu mendukung proses pelaporan *sustainability* secara lebih terintegrasi dan terstruktur. Hasil kajian penelitian terdahulu disajikan pada Tabel 2.1.

Tabel 2.1 Penelitian Terdahulu

Referensi	Metodologi	Hasil Penelitian
[16]	SDLC, *Website-based*, AHP	Membuktikan bahwa digitalisasi sistem audit internal memangkas waktu proses hingga 52,8% dibandingkan metode manual.
[17]	Studi literatur praktik industri menggunakan ELK Stack, Prometheus, JWT/OAuth, dan AI/Blockchain untuk audit API.	Audit eksternal (*logging* & RBAC) vital untuk keamanan REST API karena tidak ada fitur bawaan. Sangat relevan sebagai referensi standar keamanan dan integritas data pada arsitektur RESTful API serta basis data terpusat yang sedang kamu susun untuk sistem audit mutu.
[18]	SUS Framework, Audit Mutu Internal	Menekankan pentingnya interoperabilitas data pada sistem audit akademik agar mudah digunakan oleh auditor.

Referensi	Metodologi	Hasil Penelitian
[19]	DBSDLC (Fase Database Design)	Fokus pada krusialnya fase Requirement Collection untuk menghasilkan ERD yang akurat sebelum tahap koding.
[20]	RESTful, Laravel, RBAC, JWT	Mengimplementasikan akses kontrol berbasis peran (RBAC) yang krusial untuk memisahkan hak akses Auditor dan Auditee.
[21]	OpenAPI Spec, *Schema Tracking*	Menekankan pentingnya pelacakan perubahan struktur data API untuk menjaga integritas bukti digital dalam audit.
[22]	RESTful API, JSON, *Centralized Database*	Membuktikan bahwa arsitektur terpusat via API mampu menyatukan data dari berbagai sumber tanpa mengubah sistem asal.
[9]	Pemodelan data berbasis *fuzzy logic*, *semantic mapping*	Mendasari perancangan sistem basis data terpusat untuk memecahkan kendala integrasi data audit *sustainability* lintas unit kerja.
[23]	Eksperimen optimasi API (*Connection Pooling, Batch Processing, In-Memory Caching*).	Membuktikan bahwa Node.js dengan *connection pooling* stabil menangani konkurensi tinggi (hingga 321 req/s), sehingga andal dijadikan *back-end* sistem pelaporan yang rentan lonjakan akses.
[24]	Analisis komparatif kinerja (*Performance Benchmarking*) antara basis data Relasional (PostgreSQL) dan *Document-Oriented* (MongoDB).	Membuktikan skema relasional PostgreSQL mengeksekusi kueri analitik 15,1 kali lebih cepat dibanding NoSQL, menjadikannya fondasi efisien untuk implementasi *Unified Question Management*.

Berdasarkan penelitian terdahulu yang disajikan pada Tabel 2.1, terlihat adanya pergeseran fokus dalam pengembangan sistem informasi audit dan pelaporan, yakni dari digitalisasi manual menuju integrasi arsitektur yang *scalable* dan pemodelan basis data yang fleksibel. Secara metodologis, penggunaan kerangka kerja *Database System Development Life Cycle* (DBSDLC) sebagaimana diterapkan oleh [19] memberikan landasan yang valid dalam menghasilkan struktur data yang terintegrasi dan terdokumentasi dengan baik.

Dalam aspek teknis, penelitian yang dilakukan oleh [23] dan [24] memberikan bukti empiris bahwa pemilihan teknologi *event-driven* (Node.js) serta pemanfaatan skema dinamis (PostgreSQL JSONB) mampu memberikan efisiensi performa dan skalabilitas yang signifikan. Keunggulan Node.js dalam menangani konkurensi tinggi dan kemampuan PostgreSQL dalam memproses kueri analitik secara cepat

menjadi parameter krusial dalam membangun arsitektur *back-end* yang tangguh untuk sistem dengan volume data besar dan akses yang intensif.

Perbedaan mendasar penelitian ini dengan penelitian terdahulu terletak pada konteks implementasi dan pendekatan integrasi sistemnya. Jika penelitian terdahulu seperti [16], [17], [20] cenderung berfokus pada otomasi audit umum atau sistem *e-commerce*, penelitian ini menerapkan model arsitektur tersebut secara spesifik untuk memecahkan masalah fragmentasi data (*siloed data*) pada sistem pelaporan *sustainability* di lingkungan universitas. Oleh karena itu, penelitian ini menjadi krusial sebagai studi kasus penerapan DBSDLC yang terintegrasi dengan teknologi modern *Node.js-PostgreSQL* dalam satu ekosistem sistem informasi audit internal yang terpusat dan efisien.

2.2 Teori yang berkaitan

2.2.1 Sustainability

Konsep keberlanjutan dalam konteks institusi pendidikan tinggi kini tidak hanya terbatas pada penghijauan fisik, tetapi telah bertransformasi menjadi tata kelola sumber daya yang cerdas dan efisien [25], [26], [27]. Merujuk pada agenda global *Sustainable Development Goals* (SDGs), khususnya poin ke-9 mengenai industri, inovasi, dan infrastruktur, organisasi dituntut untuk membangun infrastruktur digital yang tangguh dan berkelanjutan [28]. Teori *Green Computing* atau *Green IT* menjadi relevan dalam hal ini, di mana optimasi perangkat lunak dan arsitektur basis data berperan langsung dalam menekan dampak lingkungan melalui efisiensi energi dan pengelolaan teknologi yang bijak [29], [30]. Basis data yang tidak teroptimasi akan memaksa unit pemrosesan bekerja lebih berat dan meningkatkan konsumsi energi pada data center universitas [31].

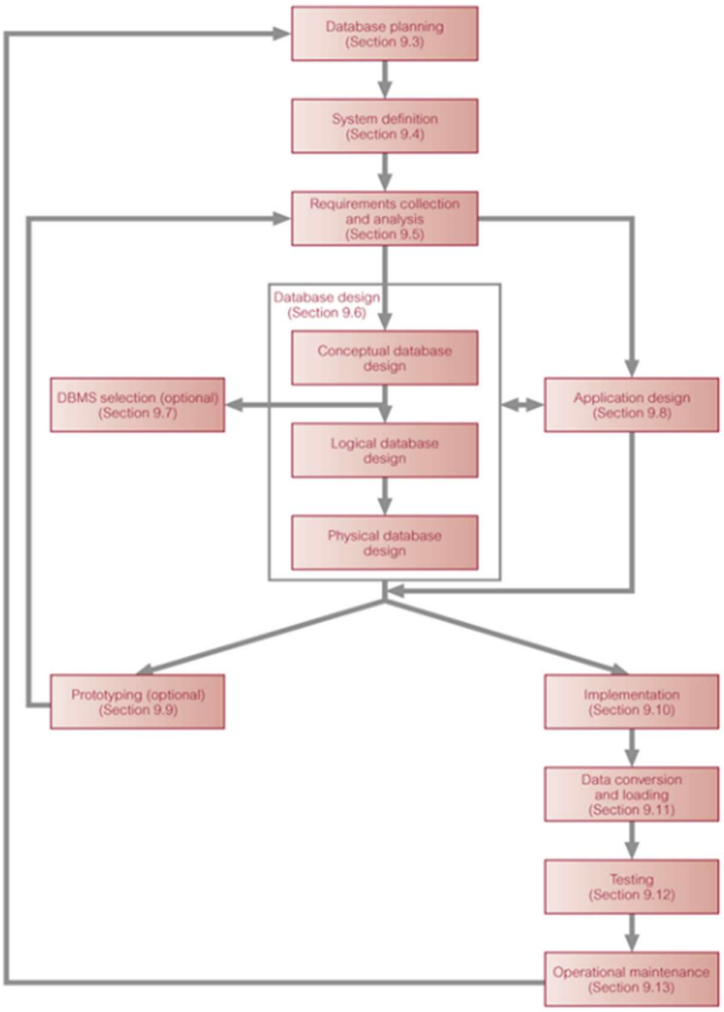
Keterkaitan konsep ini dengan sistem pelaporan data *sustainability* terletak pada standardisasi metrik keberlanjutan seperti UI GreenMetric, THE Impact Rankings, dan QS Sustainability. Implementasi arsitektur basis data terpusat melalui metode DBSDLC bukan sekadar upaya digitalisasi, melainkan langkah strategis organisasi untuk meminimalkan fragmentasi data. Dengan

mengintegrasikan data melalui RESTful API yang efisien, organisasi dapat mengurangi redundansi proses dan beban kerja server. Hal ini selaras dengan upaya pencapaian efisiensi teknis di mana infrastruktur digital dikelola sedemikian rupa untuk mendukung transparansi data tanpa memberikan dampak negatif yang besar terhadap lingkungan (jejak karbon digital). Dengan demikian, sistem pelaporan data *sustainability* yang dirancang berfungsi sebagai instrumen pemantauan sekaligus perwujudan nyata dari komitmen keberlanjutan institusi melalui efisiensi infrastruktur teknologi informasi.

2.3 Framework/Algoritma yang digunakan

2.3.1 Database System Development Life Cycle

Database System Development Life Cycle (DBSDLC) merupakan suaut metode pengembangan basis data yang mencakup beberapa tahapan mulai dari perancangan konseptual hingga implementasi fisik [32]. *Database System Development Lifecycle* memberikan gambaran relasi database untuk memudahkan pengembangan penelitian. Perancangan basis data dengan metode DBSDLC dapat meningkatkan pengelolaan data, kecepatan serta mengurangi data redudan [33]. Adapun beberapa tahap dalam *Database System Development Life Cycle* yaitu *database planning* untuk menentukan tujuan dari pembuatan *database*, *system definition* dengan mengidentifikasi pengguna yang berperan dalam sistem, *Requirement Collection and Analysis* dengan pengumpulan kebutuhan yang dibutuhkan dalam pembuatan *database*, *database design* sebagai dasar pengimplementasian *database*, *application design* sebagai perantara user dengan *database* hingga tahap implementasi. Penerapan metode ini sangat dibutuhkan dalam merancang basis data.



Gambar 2.1 Tahapan Database System Development Life Cycle

Sumber: [34]

2.4 Arsitektur dan Teknologi

2.4.1 Node.js

Node.js merupakan platform pengembangan yang memanfaatkan arsitektur berbasis *event* dan karakteristik *non-blocking* I/O dari bahasa JavaScript untuk meningkatkan utilisasi CPU pada sisi server web [35]. Teknologi ini menunjukkan keunggulan signifikan dalam skenario konkurensi tinggi, di mana arsitektur Node.js terbukti lebih unggul dibandingkan arsitektur Apache dalam hal waktu respons rata-rata, tingkat respons permintaan, dan *throughput* data [35]. Model konkurensi Node.js dapat memberikan performa yang lebih baik, dengan implementasi *multi-threaded* Node.js menunjukkan hasil superior dibandingkan *single-threaded* [35]. Selain itu, Node.js telah terbukti efektif dalam pengembangan RESTful API dan aplikasi web yang memerlukan pengolahan batch, memberikan fleksibilitas dalam konversi fitur interaktif menjadi skrip yang dapat dijalankan dalam mode batch [36], [37]. Kemampuan ini menjadikan Node.js sebagai solusi yang tepat untuk membangun layanan web berkinerja tinggi yang dapat menangani akses pengguna masif di era *big data* [35].

2.4.2 React.js

React.js merupakan *library* JavaScript yang digunakan untuk membangun antarmuka pengguna berbasis komponen *(component-based architecture)* yang efektif dalam menangani antarmuka fungsional seperti operasi pengelolaan data [38]. *Library* ini menggunakan pendekatan deklaratif, di mana setiap transisi status antarmuka pengguna dikelola secara otomatis oleh *framework*, sehingga menyederhanakan proses pengembangan aplikasi. Salah satu fitur inti yang mendasari efisiensi React.js adalah penggunaan Virtual DOM, yang memungkinkan proses *rendering* dilakukan secara efisien tanpa harus memuat ulang seluruh halaman, sehingga meningkatkan performa aplikasi secara keseluruhan terutama pada pembaruan antarmuka berskala kompleks [39]. Selain itu, React.js didukung oleh metode *reconciliation* dan React Hooks yang mempermudah pengelolaan status *(state)* pada antarmuka.

Dalam pengembangan aplikasi web modern, React.js tidak hanya mendukung penggunaan kembali kode *(reusability)*, tetapi juga memfasilitasi manajemen data yang krusial bagi sistem yang membutuhkan pembaruan *real-time*. Pemilihan React.js dalam penelitian ini didasarkan pada kemampuannya dalam memastikan alur data yang halus, konsistensi antarmuka, serta respons yang tepat terhadap interaksi pengguna. Dengan ekosistem yang kuat, React.js menjadi alat yang tepat untuk membangun dasbor visualisasi data yang responsif dan terintegrasi secara efisien dengan arsitektur *back-end* melalui komunikasi REST API [40].

2.4.3 Express.js

Express.js merupakan *web application framework* minimalis yang berjalan di atas *runtime environment* Node.js, yang digunakan sebagai alat pengembangan aplikasi web yang efisien. Dalam arsitektur sistem, Express.js berperan sebagai kerangka kerja untuk membangun *REST API* yang memfasilitasi komunikasi data terstruktur antara klien dan server. Karena *RESTful API* menjadi tulang punggung aplikasi modern, implementasi metode autentikasi yang handal, seperti penggunaan *OAuth* dan *JSON Web Tokens* (JWT) menjadi sangat krusial dalam kerangka kerja ini untuk membatasi akses, memverifikasi identitas pengguna, dan mencegah kebocoran data [41]. Selain aspek keamanan, pemilihan Express.js didasarkan pada keunggulan performanya; studi komparatif menunjukkan bahwa *framework* ini memiliki kinerja yang lebih baik dibandingkan *framework* berbasis JavaScript lainnya dalam pengukuran *Transactions Per Second* (TPS) pada struktur basis data transaksi [42]. Keandalan *framework* ini juga telah tervalidasi melalui pengujian *black box* yang menunjukkan hasil fungsionalitas yang valid pada setiap *endpoint* yang dikembangkan dalam sistem [43].

2.4.4 PostgreSQL

PostgreSQL merupakan sistem manajemen basis data relasional yang menunjukkan performa superior dan fleksibilitas tinggi dalam berbagai aplikasi enterprise. Penelitian [44] membuktikan bahwa PostgreSQL memiliki performa

13 kali lebih cepat dibandingkan MySQL pada operasi select (0,6-0,8 ms vs 9-12 ms untuk 1 juta *record*) dan mampu mempertahankan stabilitas performa saat menangani operasi yang kompleks. PostgreSQL juga mendukung migrasi dari sistem *database enterprise* seperti Oracle melalui transformasi struktur *database*, format data, dan objek yang dapat dieksekusi [45]. Sistem ini dapat diintegrasikan dengan *tools big data* seperti HBase dan Hive untuk menangani data terstruktur dan tidak terstruktur [46]. PostgreSQL terbukti sangat *adaptable* untuk aplikasi khusus, seperti penyimpanan data simulasi molekuler yang memenuhi standar FAIR dengan metadata yang ketat, serta dapat diperluas menjadi sistem terdistribusi untuk mengelola *dataset spatiotemporal* besar melalui ekstensi seperti *Distributed MobilityDB* [47], [48]. Kemampuan PostgreSQL dalam menangani *query* kompleks dengan *governance* performa yang dapat dikonfigurasi membuatnya ideal untuk lingkungan *enterprise* yang membutuhkan latensi data rendah dan kemampuan pemrosesan konkuren yang *robust* [49].

2.4.5 Centralized Database

Database terpusat merupakan arsitektur penyimpanan data yang menggabungkan informasi dari berbagai sumber ke dalam satu lokasi sentral untuk memfasilitasi akses, pengelolaan, dan analisis yang efisien. [50] Menunjukkan penerapan *database* terpusat dalam konteks pendidikan untuk mengatasi tantangan penyimpanan materi pembelajaran yang terfragmentasi, menggunakan kerangka kerja *Transformer* dan integrasi LLM untuk memastikan respons yang tepat dari *database* komprehensif. [51] Mengidentifikasi bahwa model terpusat secara khusus memfasilitasi penautan data, harmonisasi, dan interoperabilitas, meskipun model terfederasi menawarkan keunggulan dalam skalabilitas dan kepatuhan hukum. Implementasi praktis menunjukkan hasil signifikan, seperti yang didemonstrasikan oleh [52] dengan kerangka kerja *multi-layer* yang mencapai pengurangan 40% waktu eksekusi *query* dan peningkatan 25% efisiensi *preprocessing*. Penelitian [53] menekankan bahwa *database* terpusat

mendukung pengambilan keputusan yang andal melalui pengumpulan dan transformasi data yang cepat dari sumber heterogen.

2.4.6 Restful API

RESTful API (*Representational State Transfer Application Programming Interface*) merupakan arsitektur komunikasi data yang memungkinkan berbagai platform berbeda untuk berinteraksi secara ringan, efisien, dan skalabel melalui protokol HTTP [54]. Dalam perancangan sistem pelaporan data *sustainability* ini, RESTful API berperan sebagai jembatan strategis yang menghubungkan basis data terpusat di sisi server dengan aplikasi di sisi pengguna. Keunggulan arsitektur ini menjadi semakin optimal dengan penggunaan PostgreSQL sebagai sistem manajemen basis data relasional (*RDBMS*). PostgreSQL menyediakan struktur data yang sangat kuat untuk menangani kueri kompleks serta menjamin integritas data yang bersifat multi-metrik terutama pada *create, delete data*, serta klausa *join* [55], [56].

Integrasi antara PostgreSQL dan RESTful API dalam penelitian ini memastikan bahwa seluruh data *sustainability* dapat diakses secara dinamis tanpa perlu membangun infrastruktur *back-end* yang kaku. PostgreSQL menangani penyimpanan logis dan validasi data di level *database* [57], sementara RESTful API memastikan data tersebut tetap fleksibel untuk didistribusikan ke berbagai aplikasi di masa depan, mengingat API merupakan standar utama industri untuk menghubungkan layanan modern dengan basis data eksternal secara andal [58]. Dengan kombinasi ini, sistem tidak hanya mampu menyajikan data secara cepat, tetapi juga menjamin bahwa setiap transaksi data *sustainability* tercatat secara konsisten dan aman dalam skema terpusat.

2.5 Tools/software yang digunakan

2.5.1 Postman

Postman merupakan platform yang digunakan untuk desain, pengujian, distribusi, dokumentasi, dan mengelola API. Postman telah menjadi alat yang sangat penting dalam pengembangan dan pengujian API modern. Alat ini efektif dalam mendeteksi cacat secara dini, mengurangi upaya pengujian manual, dan

mendukung praktik Agile dan DevOps [59]. Postman memiliki kemampuan komprehensif dalam membuat, menguji, mengotomatisasi, dan mendokumentasikan API [60]. Perkembangan Postman sangat pesat, dari plugin browser Chrome sederhana menjadi solusi pengujian API lengkap yang digunakan oleh 5 juta pengembang dan lebih dari 100.000 perusahaan di seluruh dunia [61]. Postman juga berperan dalam memastikan validasi layanan backend dan deteksi dini masalah integrasi [59].

2.5.2 Apache Jmeter

Apache JMeter merupakan perangkat lunak *open-source* berbasis Java yang digunakan untuk melakukan *load testing* dan *performance testing* pada aplikasi web, layanan RESTful API, serta berbagai layanan berbasis jaringan. JMeter mampu mensimulasikan sejumlah pengguna secara bersamaan untuk mengukur metrik kinerja seperti *response time*, *throughput*, tingkat kesalahan (*error rate*), dan kapasitas sistem dalam menangani beban kerja tertentu. Karena fleksibilitas dan dukungannya terhadap berbagai protokol komunikasi, JMeter banyak digunakan dalam evaluasi performa aplikasi modern dan layanan berbasis API [62], [63].