BAB II
TINJAUAN PUSTAKA

2.1 Justifikasi Solusi

2.1.1 Penelitian Terdahulu dan Research Gap

Penelitian terkait penerapan *RFID* untuk inventori umumnya terbagi ke dalam dua solusi utama. Solusi pertama berfokus pada sistem *RFID* berbiaya rendah menggunakan *IoT* dengan memanfaatkan *microcontroller* seperti *ESP32*. Penelitian oleh Karmalkar et menunjukkan bahwa integrasi *RFID* dan *IoT* efektif untuk pelacakan dan pengelolaan aset [1]. Namun, cara ini tidak terintegrasi langsung dengan sistem *Point of Sale (POS)*.

Solusi kedua menargetkan ritel skala besar dengan integrasi langsung antara *RFID* dan sistem *POS*. Studi oleh Bertolini et al. serta solusi komersial seperti *ConnectPOS* menekankan sinkronisasi stok fisik dan data transaksi secara *real-time* [2], [3]. Meskipun efektif, pendekatan ini bergantung pada ekosistem *hardware* industri dan infrastruktur *enterprise* dengan biaya serta kompleksitas implementasi yang tinggi.

Berdasarkan tinjauan tersebut, masih terdapat *research gap* pada solusi inventori *RFID* yang berbiaya terjangkau namun tetap terintegrasi dengan sistem *POS* komersial. Penelitian ini mengisi celah tersebut melalui pendekatan arsitektur modular menggunakan host dengan integrasi *POS* melalui *REST API*, sehingga lebih sesuai untuk ritel skala kecil dan menengah.

2.1.2 Arsitektur Sistem

Arsitektur sistem pada penelitian ini menggunakan arsitektur *host-based*, di mana seluruh *system logic* dijalankan pada satu perangkat *host*. Arsitektur ini dipilih karena pengembangan *embedded system* berada di luar ruang lingkup dan waktu penelitian yang tersedia.

Dengan arsitektur *host-based*, sistem dapat langsung diintegrasikan dengan *RFID reader* tanpa memerlukan perancangan *firmware* dan

pengelolaan lingkungan *embedded*. Hal ini membuat penelitian berfokus pada perancangan alur sistem, integrasi data, dan evaluasi proses *stock opname*.

Konsekuensi dari pemilihan arsitektur ini adalah penempatan *backend*, *database*, dan *user interface* pada satu perangkat *host*. Rancangan arsitektur sistem dibahas lebih lanjut pada Bab III, sementara detail implementasi dijelaskan pada Bab IV.

2.1.3 Arsitektur *Software*

Arsitektur *software* sistem menggunakan model *three-tier* yang memisahkan *presentation layer*, *application logic layer*, dan *data layer*. Pemisahan ini menjaga struktur sistem tetap modular dan memudahkan pengelolaan komponen dengan tanggung jawab yang berbeda.

Pada arsitektur ini, interaksi pengguna berada pada *presentation layer*, alur proses dan integrasi sistem berada pada *system logic layer*, sedangkan penyimpanan data inventaris berada pada *data layer*. Dengan struktur ini, setiap lapisan dapat dikembangkan dan diuji secara terpisah tanpa ketergantungan langsung.

Dalam sistem *stock opname RFID*, komunikasi *hardware* dan integrasi dengan sistem eksternal ditempatkan sepenuhnya pada *system logic layer*. *Presentation layer* hanya berfokus pada interaksi pengguna tanpa menangani detail teknis komunikasi atau pengelolaan data internal.

Model *three-tier* memberikan fleksibilitas pengembangan selama kontrak data antar lapisan dijaga. Struktur ini sesuai dengan ruang lingkup penelitian yang berfokus pada validasi konsep dan pengembangan sistem secara bertahap.

2.1.4 Arsitektur *Hardware*

Arsitektur *hardware* sistem dirancang menggunakan arsitektur *non-embedded*, di mana *RFID reader* tidak berfungsi sebagai unit pemrosesan mandiri, melainkan sebagai sumber data yang terhubung langsung ke perangkat *host*. Arsitektur ini dipilih karena ruang lingkup

dan durasi penelitian tidak mencakup pengembangan *embedded system* secara menyeluruh, termasuk perancangan *firmware*, pengujian komunikasi, dan proses *debugging* yang kompleks.

Dalam konteks penelitian ini, fokus diarahkan pada perancangan dan validasi konsep sistem *stock opname RFID*, bukan pada pengembangan *embedded hardware*. Oleh karena itu, penggunaan perangkat *host* berupa laptop sebagai pusat pemrosesan dinilai lebih sesuai untuk mendukung proses pengembangan yang cepat dan terkontrol. Strategi pengembangan ini mengikuti prinsip *Rapid Application Development (RAD)* yang menekankan iterasi cepat, pengujian berulang, dan penyempurnaan sistem dalam waktu terbatas.

Sistem menggunakan *RFID reader UHF* yang terhubung ke perangkat *host* melalui *interface USB* dengan komunikasi serial. Pemilihan *RFID UHF* dilakukan pada level arsitektur karena mendukung pembacaan banyak tag dalam satu sesi pemindaian, yang sesuai dengan kebutuhan proses *stock opname*. Perangkat *hardware* digunakan secara terbatas sebagai pembaca tag dan pengirim data mentah, tanpa menjalankan *system logic* atau penyimpanan data.

Dengan menempatkan seluruh pemrosesan pada *software* di perangkat *host*, sistem dapat dikembangkan dan diuji dengan lebih fleksibel tanpa terikat pada keterbatasan sumber daya *embedded hardware*. Arsitektur *non-embedded* ini menjaga fokus penelitian pada evaluasi alur pembacaan *RFID*, pengolahan data inventaris, dan integrasi sistem dalam konteks operasional ritel.

2.1.5 Arsitektur Komunikasi

Arsitektur komunikasi sistem dirancang menggunakan skema *multi-protocol* untuk menangani perbedaan karakteristik data dan kebutuhan komunikasi pada setiap lapisan sistem. Sistem melibatkan komunikasi antara *web interface, backend*, dan perangkat *RFID*, yang masing-masing menggunakan mekanisme komunikasi yang berbeda.

Komunikasi antara *frontend* dan *backend* menggunakan *HTTP REST* dan *WebSocket*. *HTTP REST* digunakan untuk pertukaran data yang bersifat transaksional, seperti pengambilan data inventaris dan pengelolaan konfigurasi sistem. *WebSocket* digunakan untuk mengirim data hasil pembacaan *RFID* secara *real-time* agar *frontend* dapat menerima pembaruan selama proses *stock opname* berlangsung.

Penggunaan *WebSocket* menyediakan koneksi yang tetap terbuka antara *frontend* dan *backend*, sehingga data pembacaan tag dapat dikirim segera setelah diterima dari *hardware* tanpa menunggu permintaan tambahan dari klien. Mekanisme ini menurunkan latensi tampilan data dan menjaga kesesuaian antara kondisi fisik inventaris dan informasi yang ditampilkan.

Pada sisi *backend*, komunikasi dengan *RFID reader* dilakukan melalui komunikasi serial menggunakan format data biner sesuai dengan protokol perangkat. *Backend* berfungsi sebagai perantara yang menerima data dari *hardware*, memprosesnya, dan meneruskan hasilnya ke *frontend* melalui protokol *web*.

Dengan pemisahan fungsi masing-masing protokol, arsitektur komunikasi yang diterapkan menjaga sistem tetap efisien dan terstruktur. Struktur ini juga memungkinkan perubahan pada salah satu jalur komunikasi tanpa memengaruhi keseluruhan sistem.

2.1.6 Distribusi *system logic*

Distribusi *system logic* ditetapkan untuk mencegah penumpukan tanggung jawab pemrosesan pada satu sisi aplikasi dan menjaga alur kerja tetap responsif selama proses *stock opname*. Pembagian ini ditentukan pada tingkat arsitektur sistem, bukan pada detail implementasi, sehingga setiap lapisan memiliki peran yang jelas dan tidak saling bergantung secara langsung.

Penempatan logika inti pada *backend* digunakan untuk menangani proses yang memerlukan konsistensi data, pengelolaan *state* yang disimpan, serta interaksi dengan *hardware* dan sistem eksternal. *Frontend*

difokuskan pada interaksi pengguna dan penyajian informasi yang relevan dengan sesi yang sedang berjalan, tanpa menangani proses inti sistem.

Distribusi ini menjaga pemisahan tanggung jawab antar lapisan dan memungkinkan pengembangan *frontend* dan *backend* dilakukan secara terpisah selama kontrak komunikasi dijaga. Dengan struktur tersebut, perubahan pada *user interface* tidak berdampak langsung pada logika inti sistem, dan sebaliknya.

2.1.7 Pemilihan *Database*

Karakteristik penggunaan sistem inventaris *RFID* pada penelitian ini menghasilkan kebutuhan *database* yang spesifik. Sistem digunakan oleh satu pengguna pada satu lokasi operasional, dengan volume data inventaris pada skala ribuan *record*. Sistem juga dirancang untuk dijalankan secara lokal tanpa bergantung pada koneksi internet, sehingga proses *deployment* dan pengelolaan dibuat sesederhana mungkin.

Berdasarkan kebutuhan tersebut, sistem tidak memerlukan dukungan *concurrent access* yang tinggi maupun mekanisme distribusi data antar lokasi. Fokus utama *database* adalah menyimpan data inventaris dan aktivitas operasional secara konsisten dalam satu *runtime environment.*

Beberapa alternatif *database engine* dipertimbangkan, yaitu *SQLite*, *MySQL, dan PostgreSQL*. *Database client–server* seperti *MySQL* dan *PostgreSQL* menyediakan dukungan *concurrent access* yang tinggi serta fitur manajemen data yang lebih kompleks, namun memerlukan *server* terpisah, konfigurasi tambahan, dan proses *deployment* yang lebih rumit.

Sebaliknya, *SQLite* merupakan *database engine serverless* dan *embedded* yang tidak memerlukan instalasi atau pengelolaan *database server* terpisah. Seluruh data disimpan dalam satu berkas yang portabel dan mudah dicadangkan, sesuai dengan kebutuhan sistem yang dijalankan secara lokal.

Berdasarkan pertimbangan tersebut, *SQLite* dipilih sebagai *database engine* yang digunakan dalam penelitian ini. *SQLite* dinilai memadai

untuk mendukung skala data dan pola penggunaan sistem inventaris pada toko ritel skala kecil hingga menengah. Dukungan properti *ACID* menjaga konsistensi data selama sistem beroperasi.

Keterbatasan *SQLite* dalam hal *concurrent access* dan ketiadaan fitur lanjutan seperti *stored procedure* diterima sebagai *trade-off* yang tidak berdampak signifikan, mengingat sistem dirancang untuk penggunaan *single-user* pada satu lokasi. Pemilihan *SQLite* sesuai dengan tujuan penelitian yang menekankan kesederhanaan implementasi, kemudahan *deployment*, dan efisiensi pengembangan sistem.

2.2 Tinjauan Teori

2.2.1 Inventaris dan *Stock Opname*

Inventori merupakan kumpulan barang yang dimiliki dan dikelola oleh suatu entitas untuk mendukung aktivitas operasional dan penjualan. Dalam konteks ritel, inventori tidak hanya merepresentasikan jumlah barang yang tersedia, tetapi juga kondisi aktual barang yang berada di lokasi penyimpanan atau area penjualan. Ketidaksesuaian antara data inventori pada sistem dan kondisi fisik di lapangan dapat berdampak langsung pada proses penjualan, pengambilan keputusan, dan efisiensi operasional [4].

Stock opname adalah proses pencocokan antara data inventori yang tercatat pada sistem dengan kondisi fisik barang yang tersedia. Proses ini dilakukan secara berkala untuk mendeteksi selisih stok, kehilangan barang, kesalahan pencatatan, atau perubahan kondisi inventori yang tidak tercatat oleh sistem transaksi. *Stock opname* berperan sebagai mekanisme kontrol untuk menjaga akurasi data inventori dan memastikan data yang digunakan oleh sistem manajemen ritel tetap dapat dipercaya [5].

Dalam praktiknya, *stock opname* menekankan pada identifikasi fisik barang dan pencatatan hasil penghitungan ulang. Proses ini bersifat operasional dan sering kali memerlukan interaksi langsung dengan rak penyimpanan atau *area display*. Oleh karena itu, *stock opname* dipandang

sebagai aktivitas yang terpisah dari proses transaksi harian, namun memiliki keterkaitan langsung dengan kualitas data inventori yang digunakan oleh sistem informasi ritel.

2.2.2 Metode *Stock Opname*

Metode *stock opname* dapat dibedakan berdasarkan cara identifikasi barang dan mekanisme pencatatan data inventori. Perbedaan metode ini memengaruhi tingkat akurasi, waktu pelaksanaan, serta beban operasional selama proses penghitungan stok.

Metode *stock opname* manual dilakukan dengan menghitung barang secara fisik dan mencatat hasilnya secara langsung, baik pada formulir kertas maupun ke dalam aplikasi sederhana. Metode ini mudah diterapkan dan tidak memerlukan perangkat khusus, namun sangat bergantung pada ketelitian petugas. Pada skala inventori yang besar, metode manual cenderung memakan waktu lama dan rentan terhadap kesalahan pencatatan serta duplikasi data [6].

Metode berbasis *barcode* menggunakan label *barcode* sebagai identitas barang yang dipindai menggunakan *scanner*. Setiap barang perlu dipindai satu per satu untuk dicatat ke dalam sistem. Metode ini meningkatkan akurasi pencatatan dibandingkan metode manual dan mempercepat proses *input* data, namun tetap memerlukan *line-of-sight* antara *scanner* dan label *barcode*. Selain itu, proses pemindaian masih bersifat individual sehingga waktu pelaksanaan meningkat seiring jumlah *item* yang dihitung [7].

Metode *stock opname RFID* menggunakan *tag RFID* yang memungkinkan identifikasi barang secara *contactless*. Pembacaan *tag* dapat dilakukan tanpa kontak langsung dan tidak selalu memerlukan *line-of-sight*, sehingga beberapa *item* dapat terdeteksi dalam satu sesi pemindaian. Metode ini mengurangi interaksi fisik dengan barang dan memungkinkan proses penghitungan stok dilakukan secara lebih efisien, terutama pada lingkungan ritel dengan jumlah item besar dan frekuensi *stock opname* yang tinggi [7].

2.2.3 Teknologi *RFID*

Radio Frequency Identification (RFID) merupakan teknologi identifikasi otomatis yang memanfaatkan gelombang radio untuk mengenali objek yang dilengkapi *RFID tag* secara *contactless*. Sistem *RFID* umumnya terdiri dari tiga komponen utama, yaitu *RFID tag*, *RFID reader*, dan *antenna*. *RFID tag* menyimpan informasi identitas objek, *reader* memancarkan dan menerima sinyal radio, sedangkan *antenna* menjadi media transmisi sinyal antara *reader* dan *tag* [7].

Proses kerja *RFID* dimulai ketika *RFID reader* memancarkan gelombang radio melalui *antenna*. *RFID tag* yang berada dalam jangkauan merespons sinyal tersebut dengan mengirimkan data identitas yang tersimpan. Data ini diterima oleh *reader* dan diteruskan ke *software* sistem untuk diproses lebih lanjut. Pada sistem *RFID* pasif, sumber energi *tag* berasal dari gelombang radio *reader* sehingga *tag* tidak memerlukan sumber daya internal [7].

Teknologi *RFID* memungkinkan identifikasi objek dilakukan tanpa kontak fisik dan tanpa ketergantungan *line-of-sight*. Karakteristik ini membedakan *RFID* dari teknologi identifikasi visual seperti *barcode*. *RFID* juga mendukung pembacaan banyak *tag* dalam satu sesi pemindaian, sehingga sesuai untuk lingkungan yang membutuhkan identifikasi cepat dan berulang, seperti manajemen inventori dan *stock opname* [8].

Data hasil pembacaan *RFID* diproses oleh backend untuk disimpan, dianalisis, atau ditampilkan kepada pengguna. *RFID* berfungsi sebagai lapisan akuisisi data yang menghubungkan kondisi fisik objek di lapangan dengan sistem digital yang mengelola informasi tersebut.

2.2.4 Klasifikasi *RFID*

Teknologi *RFID* diklasifikasikan berdasarkan rentang frekuensi operasi karena perbedaan frekuensi memengaruhi karakteristik pembacaan, jarak jangkau, dan konteks penggunaan. Secara umum, *RFID* dibagi menjadi

tiga kategori, yaitu *Low Frequency (LF), High Frequency (HF),* dan *Ultra High Frequency (UHF)* [7].

RFID LF beroperasi pada rentang frekuensi sekitar 125–134 kHz dengan jarak baca pendek dan kecepatan transfer data rendah. Teknologi ini lebih toleran terhadap gangguan material tertentu dan umum digunakan pada aplikasi identifikasi sederhana seperti kontrol akses dan pelacakan hewan.

RFID HF beroperasi pada frekuensi 13.56 MHz dan banyak digunakan pada kartu pintar serta sistem pembayaran *contactless*. Jarak baca *RFID HF* berada pada kisaran sentimeter hingga puluhan sentimeter dengan kecepatan transfer data yang lebih baik dibandingkan *RFID LF*. Teknologi ini masih memerlukan jarak dekat antara *reader* dan *tag*.

RFID UHF beroperasi pada rentang frekuensi 860–960 MHz dan mendukung jarak baca lebih jauh serta pembacaan banyak tag dalam satu sesi pemindaian. Karakteristik ini menjadikan *RFID UHF* sesuai untuk aplikasi inventori, logistik, dan identifikasi barang dalam jumlah besar. Namun, performa pembacaan dapat dipengaruhi oleh kondisi lingkungan dan material di sekitar *tag*, sehingga konfigurasi pembacaan perlu diperhatikan [8].

Penelitian ini menggunakan teknologi *RFID UHF* (860–960 MHz) karena sesuai dengan kebutuhan *stock opname* yang menekankan efisiensi waktu. Teknologi *HF/NFC* tidak digunakan karena memerlukan pemindaian satu per satu pada setiap barang, sehingga tidak memberikan peningkatan efisiensi yang signifikan dibandingkan metode manual. Sebaliknya, *RFID UHF* memungkinkan pemindaian satu rak dalam satu proses tanpa memerlukan posisi tag yang presisi.

2.2.5 Standar *RFID (EPC Gen2 / ISO 18000-6C)*

Standar *RFID* digunakan untuk memastikan interoperabilitas antara *RFID tag, reader*, dan *software* sistem dari berbagai vendor. Pada sistem *RFID UHF*, standar yang umum digunakan adalah *EPC Gen2* yang diadopsi secara internasional sebagai *ISO/IEC 18000-6C*. Standar ini

mendefinisikan mekanisme *air interface*, format data, serta prosedur *anti-collision* untuk pembacaan banyak *tag* dalam satu sesi [9].

EPC Gen2 mengatur struktur *EPC (Electronic Product Code)* yang disimpan pada *RFID tag*. *EPC* berfungsi sebagai pengenal unik untuk membedakan setiap unit barang. Struktur *EPC* dirancang ringkas dan mendukung pengalamatan global, sehingga sesuai untuk aplikasi identifikasi barang pada lingkungan ritel dan logistik [9].

ISO/IEC 18000-6C juga mendefinisikan perintah komunikasi antara *reader* dan *tag*, termasuk proses *inventory, read*, dan *write*. Mekanisme *anti-collision* memungkinkan *reader* mengelola respons banyak *tag* secara terkoordinasi sehingga data dapat dikumpulkan tanpa konflik sinyal [10].

Dengan adanya standar ini, sistem *RFID UHF* dapat dikembangkan tanpa ketergantungan pada implementasi vendor tertentu. *RFID reader* yang mendukung *EPC Gen2* atau *ISO/IEC 18000-6C* dapat berinteraksi dengan berbagai jenis *tag* yang sesuai standar, sehingga integrasi sistem dan pengembangan aplikasi *RFID* pada level *software* menjadi lebih mudah.

2.2.6 Arsitektur *Client–Server*

Arsitektur *client–server* merupakan model arsitektur sistem terdistribusi yang memisahkan peran antara pihak *client* dan *server*. *Client* bertugas mengirim permintaan dan menampilkan hasil kepada pengguna, sedangkan *server* menangani pemrosesan permintaan, pengelolaan data, dan penyediaan layanan yang dibutuhkan oleh *client* [11].

Dalam model *client–server*, komunikasi umumnya dilakukan melalui mekanisme *request–response*. Client mengirimkan permintaan melalui jaringan menggunakan protokol komunikasi tertentu, kemudian *server* memproses permintaan tersebut dan mengirimkan kembali respons. Mekanisme ini memungkinkan pembagian beban kerja dan pengelolaan sumber daya terpusat pada sisi *server*.

Arsitektur *client–server* banyak digunakan pada sistem *web* karena memudahkan pengelolaan dan pengembangan sistem. Perubahan pada sisi *server* dapat dilakukan tanpa memodifikasi seluruh *client* selama *interface* komunikasi tetap dijaga. Pemisahan peran ini menjadi dasar pengembangan aplikasi yang melibatkan interaksi pengguna dan pengolahan data secara terpisah [12].

2.2.7 *Three-Tier* Architecture

Three-tier architecture merupakan model arsitektur *software* yang membagi sistem ke dalam tiga lapisan, yaitu *presentation layer*, *system logic layer*, dan *data layer*. Setiap lapisan memiliki tanggung jawab yang berbeda dan berinteraksi melalui *interface* yang jelas, sehingga struktur sistem lebih terorganisasi dan mudah dikelola [13].

Presentation layer menangani interaksi pengguna dan penyajian informasi, termasuk pengelolaan tampilan dan *input*. Lapisan ini tidak menangani pemrosesan inti maupun akses langsung ke *database*, sehingga perubahan pada *user interface* tidak memengaruhi *system logic*.

System logic layer menangani alur proses sistem, validasi data, serta koordinasi interaksi antar komponen. Lapisan ini menjadi penghubung antara *presentation layer* dan *data layer*, sehingga aturan proses dan kontrol akses terpusat pada satu lapisan.

Data layer bertanggung jawab atas penyimpanan dan pengelolaan data. Lapisan ini mencakup struktur data dan mekanisme akses *database*. Pemisahan data layer membantu menjaga konsistensi data dan memudahkan pengelolaan perubahan skema. *Three-tier architecture* banyak digunakan pada aplikasi modern karena mendukung modularitas dan pengembangan komponen secara terpisah [12].

2.2.8 *HTTP*

Hypertext Transfer Protocol (HTTP) merupakan protokol komunikasi yang digunakan secara luas pada aplikasi *web* untuk pertukaran data antara *client* dan *server. HTTP* bekerja dengan model *request–response*, di

mana *client* mengirimkan permintaan ke *server* dan *server* memberikan respons sesuai dengan sumber daya atau layanan yang diminta [14].

HTTP bersifat *stateless*, artinya setiap permintaan diproses secara independen tanpa mempertahankan informasi sesi secara implisit di sisi *server*. Karakteristik ini menyederhanakan desain sistem dan memungkinkan server menangani banyak permintaan dari berbagai *client* secara efisien. Informasi sesi, jika diperlukan, dikelola melalui mekanisme tambahan seperti *cookies* atau *token*.

Dalam aplikasi *web*, *HTTP* digunakan untuk berbagai operasi pertukaran data, termasuk pengambilan data, pengiriman data, serta pemanggilan layanan *API*. Fleksibilitas dan dukungan luas dari berbagai *platform* menjadikan *HTTP* sebagai dasar utama komunikasi pada sistem *web* modern [15].

2.2.9 *REST*

Representational State Transfer (REST) merupakan gaya arsitektur untuk perancangan *web service* yang memanfaatkan protokol *HTTP* sebagai media komunikasi. *REST* mendefinisikan cara *client* dan *server* berinteraksi melalui representasi *resource* yang diidentifikasi menggunakan *Uniform Resource Identifier (URI)*. Setiap resource diakses dan dimanipulasi melalui operasi standar *HTTP* seperti *GET, POST, PUT,* dan *DELETE* [16].

REST menekankan prinsip *stateless communication*, di mana setiap permintaan dari *client* harus memuat seluruh informasi yang diperlukan untuk diproses oleh *server*. *Server* tidak menyimpan konteks sesi antar permintaan, sehingga desain sistem menjadi lebih sederhana dan mudah diskalakan. Representasi data yang dipertukarkan umumnya menggunakan format terstruktur seperti *JSON* atau *XML*.

Dalam pengembangan aplikasi *web*, *REST* banyak digunakan sebagai dasar perancangan *API* karena strukturnya yang sederhana dan konsisten. Dengan memanfaatkan mekanisme *HTTP* yang sudah ada, *REST*

memungkinkan integrasi antar sistem dilakukan dengan standar dan mudah dipahami tanpa ketergantungan pada implementasi *software* tertentu [17].

2.2.10 *WebSocket*

WebSocket merupakan protokol komunikasi yang menyediakan koneksi *bidirectional* antara *client* dan *server* melalui satu koneksi *TCP* yang tetap terbuka. Berbeda dengan *HTTP* yang menggunakan pola *request–response*, *WebSocket* memungkinkan *client* dan *server* saling mengirim data kapan pun setelah koneksi berhasil dibentuk [18].

Koneksi *WebSocket* diawali dengan proses *handshake* menggunakan *HTTP*, kemudian beralih ke protokol *WebSocket* untuk pertukaran data selanjutnya. Setelah koneksi aktif, data dapat dikirim dalam bentuk pesan tanpa perlu membuka ulang koneksi untuk setiap pertukaran informasi. Mekanisme ini mengurangi *overhead* komunikasi dan mendukung aliran data yang *continuous*.

Dalam aplikasi web, *WebSocket* digunakan untuk kebutuhan komunikasi *real-time* atau *near real-time*, seperti pembaruan data langsung, notifikasi, dan *streaming data*. Dengan koneksi yang tetap terbuka, *WebSocket* memungkinkan sistem menangani perubahan data secara dinamis tanpa bergantung pada permintaan tambahan dari *client* [19].

2.2.11 *OAuth 2.0*

OAuth 2.0 merupakan standar *authorization framework* yang digunakan untuk memberikan akses terbatas pada aplikasi pihak ketiga terhadap sumber daya yang dilindungi, tanpa harus membagikan kredensial pengguna secara langsung. *OAuth 2.0* memisahkan peran antara *resource owner, client, authorization server,* dan *resource server*, sehingga proses otorisasi dapat dilakukan secara terkontrol dan aman [20].

Pada *OAuth 2.0*, *client* memperoleh izin akses melalui mekanisme pemberian *access token* yang dikeluarkan oleh *authorization server*. *Access token* ini digunakan oleh *client* untuk mengakses sumber daya

pada *resource server* sesuai dengan *scope* yang telah disetujui. Mekanisme ini memungkinkan pembatasan hak akses dan masa berlaku izin secara fleksibel.

OAuth 2.0 mendukung beberapa *authorization flow* yang disesuaikan dengan jenis aplikasi dan konteks penggunaan. Secara umum, *OAuth 2.0* digunakan pada aplikasi *web* dan layanan *API* untuk mengelola akses antar sistem secara terstandar, sekaligus mengurangi risiko penyalahgunaan kredensial pengguna [21].