

## BAB II

### TINJAUAN PUSTAKA

#### 2.1 Teori Terkait

Penelitian ini dikembangkan berdasarkan kajian komprehensif terhadap berbagai literatur dan penelitian terdahulu yang relevan. Kajian ini bertujuan untuk mengetahui sejauh mana permasalahan serupa telah diselesaikan serta menemukan celah penelitian (*research gap*). Pada aspek sistem akses parkir berbasis *Internet of Things* (IoT) dan integrasi identitas digital, penelitian mengenai otomatisasi tempat parkir terintegrasi yang dilengkapi aplikasi *mobile* dan mikrokontroler telah berhasil diimplementasikan sebelumnya [12]. Selain itu, pengembangan aplikasi *mobile* untuk melihat ketersediaan parkir secara *real-time* berbasis simulasi sensor IoT dan komputasi awan (*cloud*) juga telah membuktikan bahwa teknologi ini dapat mempermudah pengawasan area fasilitas umum secara efisien[13].

Meskipun sistem perangkat keras dan sensor fisik telah banyak diterapkan, aspek keamanan visual tetap membutuhkan pendalaman. Untuk menutupi celah keamanan tersebut, teknologi pengenalan plat nomor menggunakan *machine learning* mulai banyak diteliti. Deteksi plat nomor menggunakan algoritma YOLO terbukti sangat andal dalam mengenali area objek pada kendaraan secara cepat [14]. Pengenalan pola (*pattern recognition*) tersebut kemudian dimaksimalkan untuk mengekstrak teks plat kendaraan menggunakan metode *Optical Character Recognition* (OCR) [15].

Keandalan algoritma YOLO sebagai metode deteksi objek juga telah banyak diterapkan di berbagai bidang secara presisi [16], didukung dengan implementasi *Convolutional Neural Network* (CNN) sebagai basis utama untuk mendeteksi dan mengklasifikasikan fitur citra secara visual [17].

Selanjutnya, untuk mengintegrasikan perangkat keras IoT dan model kecerdasan buatan tersebut ke dalam genggamannya pengguna, pengembangan arsitektur perangkat lunak yang tangguh sangat dibutuhkan. Dalam membangun ekosistem digital, perancangan *backend* sangat penting untuk efisiensi pengelolaan dan penyebaran informasi secara terpusat [18]. Rancangan website *Service API* dan dokumentasi REST API mutlak dibutuhkan agar komunikasi perpindahan data antarsistem berjalan lancar [19]. Dengan infrastruktur ini, pengembangan aplikasi antarmuka untuk monitoring sistem berbasis IoT dapat dilakukan dengan aman dan terukur [20].

Berdasarkan tinjauan pustaka di atas, terlihat bahwa masing-masing penelitian terdahulu masih berjalan terpisah pada fokus otomasi sensor keras, kecerdasan buatan, atau arsitektur lunaknya saja. Oleh karena itu, penelitian yang diusulkan ini memosisikan diri sebagai penyempurna dengan menerapkan metode Validasi Ganda (*dual-validation*). Sistem akan menyatukan verifikasi identitas fisik melalui RFID KTM dengan verifikasi visual plat nomor kendaraan melalui kamera (YOLO dan OCR) secara serentak yang seluruhnya dijumpai melalui arsitektur aplikasi berbasis Flutter dan FastAPI.

## 2.2 Landasan Teori

### 2.2.1 Sistem Parkir Berbasis Sistem Informasi dan IoT

*Internet of Things* (IoT) merupakan sebuah konsep komputasi di mana objek fisik disematkan sensor, perangkat lunak, dan teknologi lainnya dengan tujuan untuk menghubungkan dan bertukar data dengan perangkat atau sistem lain melalui jaringan internet. Pemanfaatan IoT dalam sistem parkir memungkinkan otomatisasi pendataan dan pengawasan secara *real-time* [2][21]. Dalam penerapannya, perangkat IoT seperti mikrokontroler dihubungkan dengan basis data terpusat. Penggunaan Kartu Tanda Mahasiswa (KTM) yang dilengkapi teknologi *Radio Frequency Identification* (RFID) bertindak sebagai tag unik bagi setiap pengguna. Saat KTM didekatkan pada RFID reader, sistem akan langsung membaca ID mahasiswa menggunakan gelombang radio dan memprosesnya ke *server*. Pendekatan teknologi ini terbukti mampu mempercepat proses verifikasi identitas mahasiswa di gerbang parkir jika dibandingkan dengan metode pencatatan atau pengecekan karcis manual [3][22]. Integrasi sistem cerdas berbasis IoT ini sangat efektif dalam meminimalkan tingkat kelalaian manusia (*human error*), mencegah antrean panjang, serta meningkatkan standar pengawasan dan keamanan kendaraan [12].

### 2.2.2 Arsitektur Client–Server pada Aplikasi Parkir

Arsitektur *client-server* adalah sebuah model jaringan yang memisahkan peran antara penyedia sumber daya atau layanan (*server*) dan pihak peminta layanan (*client*). Dalam konteks pengembangan sistem parkir cerdas ini, sisi *client* direpresentasikan oleh aplikasi *mobile* yang diakses oleh pengguna (mahasiswa dan petugas), sedangkan sisi *server* merupakan pusat pengolahan logika bisnis (*back-end*) dan *database*. Pemisahan ini sangat fundamental karena memungkinkan aplikasi klien untuk berfokus murni pada penyajian antarmuka visual yang responsif, sementara seluruh beban komputasi berat seperti penerimaan log sensor IoT dan verifikasi *machine learning* ditangani secara asinkron di *server*. Komunikasi antara *client* dan *server* umumnya dijumpai oleh *Application Programming Interface* (API). Penggunaan arsitektur yang terdistribusi ini menjadi kunci penting untuk menjamin stabilitas kecepatan respons API dan menjaga skalabilitas sistem, terutama untuk menangani lonjakan jumlah pengguna yang mengakses aplikasi secara bersamaan pada jam-jam sibuk kampus [10][11].

### 2.2.3 Flutter sebagai Frontend Aplikasi

Flutter merupakan *Software Development Kit* (SDK) *open-source* yang dikembangkan oleh Google untuk membangun antarmuka pengguna aplikasi yang dikompilasi secara asli (*natively compiled*) dari satu basis kode tunggal. Sebagai Framework *cross-platform*, Flutter memungkinkan pengembang untuk membangun dan

merilis aplikasi pada perangkat Android maupun iOS secara efisien menggunakan bahasa pemrograman Dart. Flutter mengadopsi arsitektur yang sangat bergantung pada *widget*, di mana setiap elemen visual merupakan komponen yang dapat dikustomisasi secara mendalam. Penerapan Flutter pada aplikasi layanan terintegrasi memberikan pengalaman pengguna yang sangat dinamis, interaktif, dan mempermudah integrasi fitur monitoring status serta perubahan data secara *real-time* [7][8]. Selain itu, Flutter memiliki mesin rendering grafisnya sendiri yang berkinerja tinggi, sehingga keunggulan ini memastikan aplikasi pendataan parkir dapat melakukan transisi antar-halaman (*routing*) dengan sangat lancar dan stabil di berbagai spesifikasi resolusi maupun perangkat [9].

#### **2.2.4 FastAPI sebagai Backend Aplikasi dan Basis Data**

FastAPI adalah sebuah Framework web modern berbasis bahasa pemrograman Python yang dirancang khusus untuk membangun antarmuka pemrograman aplikasi (RESTful API) dengan performa tinggi. Keunggulan utama FastAPI terletak pada dukungan bawaannya terhadap pemrograman asinkron (*asynchronous*), yang memungkinkannya memproses ribuan permintaan (*request*) secara konkuren tanpa memblokir proses lain yang sedang berjalan. Karakteristik ini membuat FastAPI sangat ideal dan efisien sebagai *back-end* sistem IoT, karena kemampuannya menangani banyak *request* data sensor gerbang secara bersamaan dengan waktu *respons*

yang sangat cepat [18][19]. Selain itu, FastAPI memiliki fitur validasi data otomatis terintegrasi yang meminimalisir kesalahan format pengiriman data dari perangkat keras. Pengelolaan basis data relasional yang terpusat melalui web API ini memastikan pencatatan riwayat *log* dari sensor IoT maupun hasil deteksi kamera berjalan dengan konsisten, terekam secara komprehensif, dan aman dari manipulasi pihak luar [20].

### 2.2.5 Machine learning untuk Pengenalan Plat Nomor Kendaraan

Pengenalan plat nomor kendaraan secara otomatis atau *Automatic Number Plate Recognition* (ANPR) merupakan salah satu implementasi tingkat lanjut dari teknologi *computer vision* dan *machine learning*. Dalam sistem ini, algoritma YOLO (*You Only Look Once*) digunakan sebagai model pendeteksi objek (*object detection*). Berbeda dengan algoritma deteksi konvensional, YOLO memproses seluruh citra gambar dalam satu kali komputasi (*single forward pass*) menggunakan *Convolutional Neural Network* (CNN) untuk melokalisasi koordinat objek secara serentak. Oleh karena itu, algoritma YOLO memiliki tingkat akurasi dan kecepatan *Frame Per Second* (FPS) yang sangat tinggi untuk mendeteksi objek dinamis seperti kendaraan secara *real-time* [4][5][14]. Setelah lokasi plat nomor berhasil dideteksi oleh YOLO, area citra tersebut akan dipotong (*crop*) dan dikombinasikan dengan metode *Optical Character Recognition* (OCR). OCR bertugas mengenali pola visual

pada gambar dan mengekstrak karakter teks di dalamnya menjadi sekumpulan data digital otomatis, yang selanjutnya akan dikirim ke *server* untuk divalidasi [6][15].

#### 2.2.6 UML (Unified Modeling Language)

*Unified Modeling Language* (UML) adalah bahasa standar yang digunakan untuk membuat diagram atau model perangkat lunak. UML memungkinkan pengembang atau *developer* untuk memvisualisasikan, mendefinisikan, membangun, dan mendokumentasikan sistem perangkat lunak secara jelas dan terstruktur. Analogi yang sering digunakan adalah seperti seorang arsitek bangunan yang membuat denah sebagai acuan bagi perusahaan konstruksi; demikian pula seorang *arsitek* perangkat lunak membuat diagram UML agar pengembang dapat membangun sistem sesuai dengan rancangan yang telah ditetapkan. Dengan memahami notasi dan kosakata UML, seseorang dapat lebih mudah memahami alur dan struktur sistem, serta menjelaskan desain perangkat lunak kepada pihak lain yang terlibat dalam pengembangan.

*Unified Modeling Language* (UML) sendiri termasuk salah satu metode pemodelan visual yang banyak digunakan dalam perancangan perangkat lunak berbasis objek. berfungsi sebagai standar penulisan atau semacam *blueprint*, yang di dalamnya mencakup bisnis proses, penulisan kelas, dan berbagai elemen sistem lainnya dalam bahasa yang spesifik. Terdapat beragam diagram yang sering digunakan

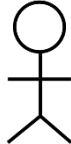
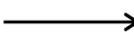
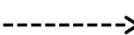
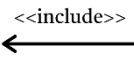
dalam pengembangan sistem untuk menggambarkan struktur, perilaku, dan interaksi antar elemen sistem.

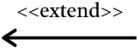
### 1. Use Case Diagram

Menggambarkan fungsionalitas yang diharapkan dari sebuah sistem dan merepresentasikan sebuah interaksi antara aktor dan sistem.

Didalam *use case* diagram terdapat (6 enam aktor) yang merupakan sebuah gambaran entitas dari manusia atau sebuah sistem yang melakukan pekerjaan di sistem.

Tabel 2. 1 Use Case Diagram

| No | Gambar                                                                              | Simbol              | Keterangan                                                                                                |
|----|-------------------------------------------------------------------------------------|---------------------|-----------------------------------------------------------------------------------------------------------|
| 1. |  | Aktor               | Mewakili peran seorang <i>system</i> yang lain, atau alat ketika berkomunikasi dengan <i>use case</i> .   |
| 2. |  | <i>Use Case</i>     | Mewakili abstraksi dan interaksi antara sebuah <i>system</i> dan <i>actor</i> .                           |
| 3. |  | <i>Association</i>  | Abstraksi penghubung antar <i>actor</i> dengan <i>use case</i> .                                          |
| 4. |  | <i>Generalisasi</i> | Menunjukkan spesialisasi <i>actor</i> untuk dapat berpartisipasi dengan <i>use case</i> .                 |
| 5. |  | <i>Include</i>      | Menunjukkan bahwa suatu <i>use case</i> seluruhnya merupakan fungsionalitas dari <i>use case</i> lainnya. |

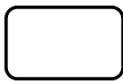
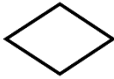
| No | Gambar                                                                            | Simbol        | Keterangan                                                                                |
|----|-----------------------------------------------------------------------------------|---------------|-------------------------------------------------------------------------------------------|
| 6. |  | <i>Extend</i> | Menunjukkan bahwa suatu <i>use case</i> tambahan fungsional dari <i>use case</i> lainnya. |

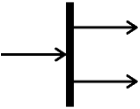
## 2. Activity Diagram

Menggambarkan alur kerja dari suatu proses dan urutan aktivitas, yang dimodelkan sebagai alur kerja dari satu aktivitas ke aktivitas berikutnya atau dari satu aktivitas ke aktivitas lainnya.

Di dalam *activity diagram* terdapat beberapa simbol yang digunakan untuk menggambarkan alur aktivitas dan proses kerja dalam sistem.

Tabel 2. 2 Activity Diagram

| No | Gambar                                                                              | Simbol              | Keterangan                                                                            |
|----|-------------------------------------------------------------------------------------|---------------------|---------------------------------------------------------------------------------------|
| 1. |  | <i>Initial node</i> | Merupakan awal aktivitas yang akan dibuat                                             |
| 2. |  | <i>Activity</i>     | Memperlihatkan bagaimana masing-masing kelas antarmuka saling berinteraksi            |
| 3. |  | <i>Decicion</i>     | Menggambarkan suatu keputusan atau tindakan yang harus diambil pada kondisi tertentu. |

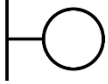
| No | Gambar                                                                            | Simbol                     | Keterangan                                                                                                    |
|----|-----------------------------------------------------------------------------------|----------------------------|---------------------------------------------------------------------------------------------------------------|
| 5. |  | <i>Fork Join node</i>      | Digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel untuk dari dua kegiatan paralel jadi satu. |
| 6. |  | <i>Fork node</i>           | Satu aliran pada tahap yang berubah menjadi beberapa aliran.                                                  |
| 7. |  | <i>Activity Final node</i> | Merupakan akhir aktivitas yang telah dibuat.                                                                  |

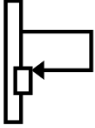
### 3. Sequence Diagram

Menggambarkan interaksi antar objek didalam dan disekitar sistem yang berupa *message* yang digambarkan terhadap waktu.

Di dalam sequence diagram terdapat beberapa simbol yang digunakan untuk alur komunikasi pesan antar objek dalam sistem.

Tabel 2. 3 Sequence Diagram

| No | Gambar                                                                              | Simbol                | Keterangan                                                                 |
|----|-------------------------------------------------------------------------------------|-----------------------|----------------------------------------------------------------------------|
| 1. |  | <i>Entity Class</i>   | Menggambarkan awal dari sebuah sistem, menjadi dasar penyusunan basisdata. |
| 2. |  | <i>Boundary Class</i> | Menggambarkan komunikasi antara sistem seperti <i>form</i> .               |
| 3. |  | <i>Control Class</i>  | Penghubung antara <i>boundary</i> dengan <i>table</i> .                    |

| No | Gambar                                                                            | Simbol            | Keterangan                                                         |
|----|-----------------------------------------------------------------------------------|-------------------|--------------------------------------------------------------------|
| 4. |  | <i>Recursive</i>  | Mengirimkan pesan yang akan dikirim ke dirinya sendiri             |
| 5. |  | <i>Activation</i> | Menghubungkan proses durasi aktivasi sebuah proses                 |
| 6. |  | <i>Life Line</i>  | Menghubungkan tiap-tiap objek yang mana terdapat <i>activation</i> |
| 7. |  | <i>Message</i>    | Sebagai pengirim pesan ke tiap-tiap <i>class</i>                   |

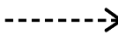
#### 4. Class Diagram

Mengambaran struktur dan deskripsi dari *class*, *package* dan objek yang saling berhubungan seperti diantaranya pewarisan, asosiasi dan lainnya.

Di dalam *class* diagram terdapat beberapa simbol yang digunakan untuk hubungan dan struktur data dalam sistem.

Tabel 2. 4 Class Diagram

| No | Gambar                                                                              | Simbol | Keterangan                                                              |
|----|-------------------------------------------------------------------------------------|--------|-------------------------------------------------------------------------|
| 1. |  | Class  | Menggambarkan kumpulan objek-objek dengan atribut dan operasi yang sama |

| No | Gambar                                                                              | Simbol                | Keterangan                                                                                                                            |
|----|-------------------------------------------------------------------------------------|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 2. |    | Nary Association      | Sebagai upaya menghindari asosiasi dengan lebih dari 2 objek                                                                          |
| 3. |    | <i>Collaboration</i>  | Urutan aksi-aksi yang ditampilkan oleh sistem yang menghasilkan suatu hasil yang tertukar bagi suatu <i>actor</i>                     |
| 4. |    | <i>Generalization</i> | Menggambarkan dimana hubungan objek <i>descendent</i> berperilaku dan struktur data objek yang di atasnya objek induk <i>ancestor</i> |
| 5. |  | <i>Realization</i>    | Operasi yang benar-benar dilakukan oleh suatu objek                                                                                   |
| 6. |  | <i>Dependency</i>     | Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri akan mempengaruhi elemen yang tidak mandiri.                         |