BAB II

TINJAUAN LITERATUR

2.1. PENELITIAN TERKAIT

Penelitian oleh (Chandra et al., 2024) berjudul "Perancangan dan Implementasi RESTful API untuk Aplikasi Mobile Pembelajaran Flora dan Fauna pada Google Cloud Platform" telah berhasil membuat aplikasi untuk menghubungkan antara aplikasi mobile dengan server sehingga dapat mendukung fungsionalitas dari aplikasi mobile tersebut. Aplikasi *backend* yang dikembangkan dalam penelitian ini digunakan untuk membuat API berbasis REST yang kemudian di-*deploy* pada layanan Google Cloud Provider. Penelitian serupa juga dilakukan oleh (Maulana et al., 2024) yang mengembangkan sistem *backend* untuk aplikasi rekomendasi resep makanan menggunakan Express.js dengan metode pengembangan waterfall, aplikasi kemudian di-*deploy* di layanan GCP dan dapat berfungsi dengan baik serta berjalan sesuai spesifikasi yang diharapkan. Penelitian selanjutnya oleh (Falah et al., 2023) yang melakukan perancangan *microservice* berbasis REST menggunakan Node.js dan layanan Google Cloud Platform namun dengan metodologi SDLC Agile Model yaitu XR (Extreme Programming) dengan melakukan uji performa testing dapat memperoleh performa yang baik dengan rata-rata waktu tanggap sebesar 60,41 ms dan server mampu menerima rata-rata *hits* 40,07 per detik. Hasil penelitian ini menunjukkan performa yang sangat baik dengan spesifikasi pengembangan sistem menggunakan Node.js dan layanan Google Cloud Provider.

Node.js diketahui memiliki performa yang baik, penelitian berjudul "Perbandingan Rest Api Menggunakan Node.js Dan PHP Pada Aplikasi Pemilihan Umum" oleh (Haryadi et al., 2023) memperoleh hasil perbandingan yaitu implementasi api yang dibuat menggunakan Node.js memiliki rata-rata hasil yang stabil dan konsisten dibanding dengan api yang dikembangkan menggunakan PHP. Pada penelitian berjudul "Analisis Perbandingan Performa Web Service REST Menggunakan Framework Laravel, Django, dan Node.js pada Aplikasi Berbasis Website", dilakukan pengujian dengan memberikan beban data dan request untuk mengevaluasi kinerja masing-masing *framework*. Hasil penelitian menunjukkan bahwa Node.js dengan *library* Express.js memiliki keunggulan dalam tingkat keberhasilan serta lebih unggul dibandingkan Django dan Laravel dalam hal kecepatan dalam menangani banyaknya *request* yang diberikan.

2.2. Backend

Backend adalah bagian dari suatu perangkat lunak atau *software* dan atau situs web yang berfokus pada logika bisnis, penyimpanan dan pengelolaan data, serta interaksi dengan database dan server (Kurniawan et al., 2023). Istilah ini sering digunakan pada arsitektur terpisah, dimana aplikasi dipisah menjadi dua bagian yaitu *frontend* yang mengelola tampilan aplikasi serta berinteraksi dengan pengguna dan *backend* yang berhubungan dengan pengelolaan dan modifikasi database serta server

Terlepas dari arsitektur yang digunakan, *backend* dapat dibuat menggunakan bahasa pemrograman populer seperti Node.js (*JavaScript Runtime*), Python, Java,

GO Lang dan PHP. Untuk membantu pengembangan lebih cepat umumnya para *programmer* menggunakan *library* dan atau *framework* untuk membuat sebuah aplikasi *backend*. Beberapa *framework* dan *library* populer yang digunakan antara lain:

1. PHP: Codeigneter dan Laravel
2. Node.js (JavaScript): Express.js, Astro.js, Next.js dan NestJS
3. Java: Spring Boot
4. Python: Django, Flask dan FastAPI

2.3. API

API (*Application Programming Interface*) atau antarmuka pemrograman aplikasi merupakan seperangkat aturan atau protokol yang menjembatani aplikasi perangkat lunak untuk saling berkomunikasi satu sama lain baik untuk bertukar data, fitur dan fungsionalitas (Goodwin, 2024). Penggunaan API dalam kehidupan sehari-hari sangatlah luas, API bukan hanya menangani interaksi dan komunikasi sistem pada layanan *website* namun lebih dari itu. API sebagai contoh pada komputer yang digunakan saat ini, koleksi dari API yang disebut *toolkit* misalnya, digunakan untuk menghubungkan sistem operasi (OS) dengan *middleware*. Beberapa perusahaan teknologi ternama seperti Meta, Google, dan Yahoo turut mempublikasikan berbagai API mereka untuk mendorong developer lainnya agar mengembangkannya. Ini mendorong berbagai perkembangan teknologi di internet seperti *content delivery*, teknologi AR, *wearable technology,* akses peta, cuaca, dan berbagai situs yang dapat diakses melalui perangkat *mobile*, yang kemudian

menjadi bagian besar dari pemanfaatan API (Gillis, 2024). Walaupun API sejatinya merupakan seperangkat aturan dan protokol yang membuat antar sistem dapat berkomunikasi, dalam perkembangannya yang semakin pesat dengan kontribusi dari berbagai raksasa internet, tuntutan kebutuhan yang menjadikan orang berlomba-lomba mengembangkan versi API nya sendiri. Hal ini telah mengarah pada pengembangan dan penggunaan berbagai gaya, protokol, standar dan bahasa tertentu.

2.4. RESTful API

Arsitektur API merupakan kumpulan aturan-aturan, protokol dan *tools* yang menentukan bagaimana komponen pada *software* saling berinteraksi (Gavrilenko, 2023). Lebih lanjut terdapat enam jenis arsitektur yang umum digunakan pada pengembangan API yaitu REST, GraphQL, WebSocket, Webhook, RPC (gRPC) dan SOAP. Dari keenam arsitektur tersebut, REST merupakan yang paling populer dan fleksibel digunakan dalam pengembangan aplikasi *client*-server berbasis web (CBNCloud, 2023). REST atau *Representational State Transfer* dirancang oleh Roy Fielding dalam tesisnya pada tahun 2000, merupakan arsitektur yang paling banyak digunakan saat ini. Arsitektur inilah yang sering digunakan untuk berkomunikasi *client-server* melalui protokol HTTP. REST menggunakan protokol dan aturan yang baku membuatnya mudah dipahami dan diimplementasikan. Namun, REST bukanlah sebuah protokol, format *file* atau rancangan kerja pengembangan aplikasi, REST sebenarnya adalah sekumpulan aturan atau batasan yang disebut "*Fielding Constraints*" (Richardson & Amundsen, 2013). Roy Fielding mengemukakan

batasan REST yaitu bersifat *stateless*, terdapat pemisahan client-server, *cacheable,* antarmuka yang konsisten dan seragam serta modularitas (Lauret, 2024). Hal ini yang menjadikannya solusi yang tepat untuk mengembangkan API berbasis web. Web *services* API yang sesuai dengan gaya arsitektur REST inilah yang kemudian disebut sebagai REST API atau RESTful API (Masse, 2011).

Alasan pemilihan REST bukan hanya merupakan faktor protokol HTTP, pemilihan RESTful API dalam pembuatan web *service* ini dengan alasan kemudahan dalam melakukan implementasi, komunitas yang sudah besar dan aplikasi masih dalam kategori kecil. Jika dibandingkan dengan GraphQL yang diimplementasikan pada sistem dengan kompleksitas yang lebih tinggi (altexsoft, 2020).

2.5. Node.js

Node.js adalah lingkungan pengembangan JavaScript di luar browser yang bersifat gratis, *open-source* dan lintas platform yang memungkinkan pengembang membuat server, aplikasi web dan *command line tools* (OpenJS Foundation, 2025).

Berdasarkan studi perbandingan pembuatan RESTful API menggunakan Node.js dan PHP oleh (Haryadi et al., 2023) menunjukkan bahwa RESTful API yang dibuat dengan Node.js memiliki kecepatan respon yang lebih baik, stabil dan juga konsisten dibandingkan dengan bahasa pemrograman PHP. Selain itu, Node.js memiliki komunitas yang besar dan aktif, bahkan perusahaan besar seperti Paypal, eBay, LinkedIn, Netflix dan Yahoo! juga menggunakan Node.js dalam produksinya (Hadinata & Stianingsih, 2024), ditambah dengan banyaknya ketersediaan modul

dan paket yang dapat membantu dalam mempercepat pengembangan aplikasi nantinya. Selanjutnya, penelitian ini juga akan melakukan pengembangan API menggunakan salah satu library Node.js yaitu Express.js dimana memiliki fleksibilitas yang baik serta pengaturan proyek yang dapat disesuaikan dengan kebutuhan pengembang

2.6. JSON

JSON atau JavaScript Object Notation merupakan sebuah format *pair* pertukaran data berbasis *Key-value* yang ringan. Hal ini dikarenakan mudah dibaca dan di urai dan di buat. Struktur JSON bersifat universal, sehingga sekarang ini hampir semua bahasa pemrograman modern mendukung format pertukaran dengan format data JSON (*ECMA-404 - Ecma International*, 2017). JSON secara bawaan telah didukung oleh JavaScript atau Node.js sehingga akan menjadi pilihan yang tepat jika menggunakannya pada aplikasi REST API berbasis Node.js.

Di dalam penulisan JSON terdapat beberapa jenis value, yaitu (Fadillah, 2024):

Tabel 2.1. Format Data JSON

Jenis	Contoh	Deskripsi

Object	{
	“alatTulis”:{
	“alatSatu”:”Penggaris”,
	“alatDua”:”Penghapus”,
		Object JSON mirip dengan objek pada JavaScript yaitu pasangan antara *key*

	}	dan *value* yang dipisahkan dengan koma.
Array	{ “siswa”: [{”namaDepan” : ”Bagas”, ”namaBelakang”:”Haryanto”}, {”namaDepan” : ”Ujang”, ” namaBelakang”:”Denim”}]}	Array merupakan kumpulan value yang tersusun mengikuti urutan khusus, setiap array dibatasi dengan kurung siku ([]).
String	{”nama” : ”Ucup Peterpen”}	String merupakan value yang tersusun atas unicode yang diapit oleh tanda kutip (“”).
Number	{”usia” : 21}	Number atau Angka merupakan tipa data value angka.
Null	{”pasangan” : Null}	Null adalah nilai kosong yang mengindikasikan tidak ada informasi yang tertera pada key tersebut.

Boolean	{”isDone” : false}	Boolean merupakan value yang berisi dua jawaban yaitu true dan false.

2.7. UML

UML merupakan salah satu standar bahasa yang digunakan untuk mendefinisikan *requirement,* membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek (Hasanah & Untari, 2020). UML versi 2.3 dikategorikan kedalam 3 jenis yaitu:

A. Structure Diagrams

Structure diagram merupakan kumpulan diagram yang digunakan untuk menggambarkan struktur statis dari sistem yang dimodelkan. Beberapa contoh dari kategori ini adalah *Class Diagram, Objects Diagram* dan *Component Diagrams.*

B. Behaviour Diagrams

Behaviour Diagrams merupakan kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada suatu sistem. Contoh dari kategori ini ialah *use case diagram, Activity Diagram* dan *State Machine Diagram.*

C. Interaction Diagrams

Interaction Diagram merupakan kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem. Contoh dari kategori ini adalah *Sequence Diagram, Communication Diagram, Timing Diagram* dan *Interaction Overview Diagram.*

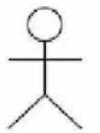
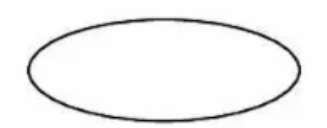
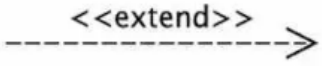
Penelitian ini menggunakan empat buah jenis diagram UML yaitu *Use Case Diagram*, *Activity Diagram*, *Sequence Diagram* dan *Class Diagram*.

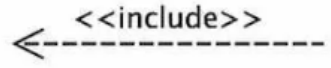
2.7.1. Use Case Diagram

Use Case Diagram digunakan untuk menentukan aktivitas yang dilakukan oleh aktor di sistem. Sebuah *use case* model menggambarkan apa yang dilakukan sistem tanpa menjelaskan bagaimana sistem melakukannya (Kendall & Kendall, 2019). *Use case diagram* dapat digunakan untuk menggambarkan interaksi antara user dengan sistem (Falah et al., 2023). *Use Case Diagram* memiliki empat buah jenis hubungan yaitu *communicates*, *includes*, *extends* dan *generalizes.* Berikut ini adalah notasi yang digunakan dalam hubungan *Use Case:*

Tabel 2.2. Notasi *Use Case Diagram*

Simbol	Nama	Deskripsi

	*Actor*	Aktor merupakan entitas yang berkomunikasi dengan sistem.
	*Use Case*	*Use Case* merupakan tindakan atau aksi yang dilakukan aktor dengan tujuan tertentu.
	Association Relathionship	Dilambangkan dengan garis tanpa kepala panah.
	*Extend Relathionship*	*Extend* menunjukkan bahwa suatu *use case* dapat menambah fungsi lain dalam kondisi tertentu.

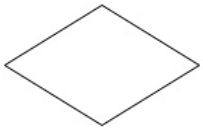
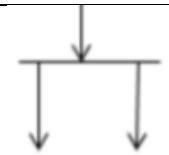
	*Include Relathionship*	*Include* digunakan untuk menyatakan bahwa sebuah *use case* merupakan suatu bagian fungsi yang dipanggil oleh *us e case* lain.
	*Generalization Relathionship*	*Generalizes* atau *Generalization* menjelaskan spesialisasi aktor yang dapat berpartisipasi dalam use case tertentu.

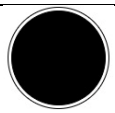
2.7.2. Activity Diagram

Activity Diagram atau Diagram aktivitas menggambarkan alur langkah-langkah dalam suatu proses, termasuk urutan aktivitas, aktivitas yang berjalan bersamaan, dan keputusan yang diambil. Biasanya, diagram ini dibuat untuk satu *use case* dan menunjukkan berbagai kemungkinan

jalannya proses (Kendall & Kendall, 2019). Tabel 2.3 dibawah ini merupakan penjelasan tiap simbol atau notasi yang digunakan di dalam *activity diagram.*

Tabel 2.3. Notasi *Activity Diagram*

Simbol	Nama	Deskripsi
	*Start Node*	Memulai Proses
	*Action Node*	*Action Node* digunakan untuk merepresentasikan aktivitas atau tugas spesifik yang dilakukan dalam suatu proses
	*Decision Node*	*Decision Node* digunakan untuk memilih satu dari beberapa jalur berdasarkan suatu kondisi atau keputusan.
	*Control Flow*	Digunakan untuk menentukan alur atau kontrol alur dari aktivitas
	*Fork*	Notasi *fork* digunakan untuk membagi satu alur menjadi

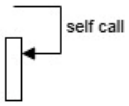
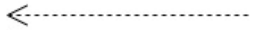
		beberapa aktivitas yang berjalan secara paralel
	*Join*	Digunakan untuk menggabungkan kembali alur yang sebelumnya bercabang secara paralel
	*End State*	Mengakhiri proses

2.7.3. Sequence Diagram

Sequence diagram atau diagram urutan dibuat berdasarkan analisis *use case* dan digunakan dalam perancangan sistem untuk menentukan interaksi, hubungan, serta metode yang digunakan oleh objek dalam sistem. Diagram ini membantu menggambarkan pola umum dari aktivitas atau interaksi dalam sebuah *use case* (Kendall & Kendall, 2019). Tabel 2.4 dibawah ini merupakan penjelasan tiap simbol atau notasi yang digunakan di dalam *sequence diagram*.

Tabel 2.4. Notasi *Sequence Diagram*

Simbol	Nama	Deskripsi
	*Objek*	Nama objek terletak diatas dan setiap objek memiliki garis lifeline.

	*Activation Boxes*	*Activations bars* pada lifeline menggambarkan durasi keaktifan partisipan dalam memproses pesan.
	*Actor*	Sebuah aktor mewakili entitas yang berinteraksi dengan sistem dan objek dan letaknya berada di luar sistem.
	Lifeline	Lifeline adalah garis yang berada dibawah elemen objek atau partisipan.
	*Self Message*	*Self message* muncul ketika sebuah objek perlu mengirimkan pesan kedirinya sendiri.
	Messages	Komunikasi antar objek digambarkan menggunakan *messages.* Setiap messages muncul secara berurutan pada *lifeline*
	*Return* atau *Reply Message*	*Reply Messages* mengembalikan messages dari penerima ke pengirim.

2.7.4. Class Diagram

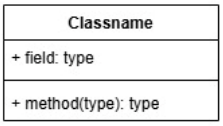
Class Diagram menggambarkan hubungan antar kelas dan menjelasakan struktur sistem yang dibuat (Kendall & Kendall, 2019). Diagram ini menunjukkan kelas-kelas dalam sistem beserta atribut, metode (fungsi), serta hubungan antar kelas seperti asosiasi, pewarisan (*inheritance*), agregasi, dan komposisi.

Kelas dibagi kedalam empat kategori yaitu *entity*, *interface*, *abstract*, dan *control.* Dimana *entity* adalah kelas yang berhubungan dengan objek dalam sistem, seperti pada ERD, yang biasanya merepresentasikan data dari entitas nyata seperti orang, benda, atau konsep, misalnya *user*, *product*, atau *order*. *Interface* atau *boundary* yang merupakan bagian yang berinteraksi dengan pengguna dan *control* yang merupakan pengontrol alur aktivitas pada sistem, dan semua kelas terkoneksi pada suatu kelas kontrol.

Berikut ini adalah notasi dari hubungan yang ada pada *class diagram:*

Tabel 2.5. Notasi *Class Diagram*

Simbol	Nama	Deskripsi
	Asosiasi	Hubungan antar kelas tanpa kepemilikan
	Agregasi	Agregasi adalah hubungan "has-a" (bagian-dari) yang menunjukkan

		bahwa suatu kelas memiliki referensi ke kelas lain.
	Komposisi	Komposisi adalah bentuk khusus dari agregasi di mana kelas yang dimiliki sangat bergantung pada kelas pemiliknya
	Kelas	Kelas adalah representasi abstrak dari suatu objek dalam pemrograman berorientasi objek (OOP).
	Pewarisan	Pewarisan adalah hubungan antara kelas superclass (induk) dan subclass (anak), di mana subclass mewarisi atribut dan metode dari superclass.

2.8. GCP Services

GCP platform adalah layanan komputasi milik Google yang menyediakan banyak layanan meliputi infrastruktur *cloud*, layanan penyimpanan, analisis *big data*, *machine learning* dan server untuk mengembangkan aplikasi dengan skala kecil hingga *enterprise,* GCP juga telah memiliki server regionnya di Indonesia yang tentunya dapat menjangkau seluruh wilayah Indonesia dengan sangat baik (Cloud Ace Indonesia, 2021). Sehingga aplikasi nantinya diharapkan dapat bekerja

secara optimal dan berfungsi dengan baik dimanapun dan kapanpun ketika diakses. Beberapa jenis layanan GCP berdasarkan kategori adalah sebagai berikut:

1. *Compute* adalah layanan untuk menjalankan dan mengelola aplikasi serta beban kerja komputasi. Contoh layanan *compute* antara lain Compute Engine*,* APP Engine dan Cloud Functions*.*
2. *Storage* dan Database adalah layanan untuk menyimpan, mengelola, dan mengakses data dalam berbagai format, baik terstruktur maupun tidak terstruktur. Contoh dari layanan *storage* dan database antara lain Cloud Storage, Cloud SQL dan Cloud Bigtable.
3. *Networking* adalah layanan untuk membangun dan mengelola infrastruktur jaringan guna mendukung komunikasi yang cepat, aman, dan andal di *cloud*. Contoh dari layanan ini adalah Cloud VPC, Cloud Load Balancing, Cloud CDN dan Cloud DNS