BAB II

TINJAUAN PUSAKA

2.1 Landasan Teori

Landasan teori berfungsi sebagai dasar konseptual dalam buku ini dengan menguraikan teori-teori serta teknologi yang digunakan. Pada proyek ini, konsep yang dibahas meliputi sistem *monitoring* sebagai kerangka utama pengawasan, serta Bea dan Cukai yang menjadi konteks implementasi pengembangan. Selanjutnya, dijelaskan pula teknologi pendukung yang digunakan, mulai dari konsep *website* sebagai media aplikasi, model *Waterfall* sebagai metode pengembangan sistem, serta arsitektur *modern* seperti *microservices* dan *microfrontend* untuk mendukung skalabilitas dan modularitas aplikasi. Selain itu, komponen teknis seperti API *services* untuk komunikasi data, ReactJS pada sisi *frontend*, Spring Boot pada sisi *backend*, hingga PostgreSQL sebagai sistem manajemen basis data juga akan dibahas sebagai landasan dalam pembangunan sistem.

2.1.1. Sistem Monitoring

Monitoring adalah proses mengumpulkan dan menganalisis informasi terhadap indikator yang ditetapkan secara sistematis tentang suatu proses atau kegiatan, sehingga dapat dilakukan tindakan evaluasi untuk penyempurnaan kegiatan selanjutnya. Dalam melakukan kegiatan *monitoring*, tak jarang menggunakan *tool* seperti sistem *monitoring* dalam berbagai bentuk, baik berbasis web maupun *mobile*. Sistem *monitoring* adalah sebuah sistem yang dirancang untuk melakukan monitor/pengawasan pada aktivitas tertentu secara *realtime*. Sistem *monitoring* telah

diterapkan di berbagai bidang, seperti dalam memantau jaringan komputer, server aplikasi, keuangan, pendidikan, hingga kinerja pegawai. [10]

Dengan adanya sistem *monitoring*, diharapkan dapat mendeteksi masalah sedini mungkin agar bisa segera ditangani. Selain itu, sistem *monitoring* juga bertujuan untuk meningkatkan efisiensi dan mempermudah pengecekan batas waktu pada pekerjaan.

2.1.2. Bea dan Cukai

Gambar II.1 Lambang DJBC

(sumber: www.beacukai.go.id/assets/img/bea-cukai.png) [11]

Bea dan cukai adalah dua komponen berbeda, yaitu bea adalah pungutan yang dikenakan terhadap barang-barang yang diimpor dan/atau diekspor; dan cukai adalah pungutan yang dikenakan terhadap barang yang memiliki karakteristik tertentu, seperti pada rokok/hasil tembakau lainnya, minuman beralkohol, dan etil alkohol.

Pemerintah mendelegasikan tugas dan fungsi dalam melaksanakan kebijakan di bidang kepabeanan dan cukai ini kepada DJBC. Tidak hanya berperan sebagai *tax collector*, DJBC juga melindungi masyarakat Indonesia dari peredaran barang-barang impor berbahaya, melindungi industri dalam negeri dari persaingan yang tidak sehat dengan industri sejenisnya dari luar negeri, memberantas penyelundupan barang ilegal, serta menjalankan tugas titipan dari instansi lainnya yang memiliki kepentingan dengan lalu lintas barang secara internasional. [12]

Dalam melaksanakan tugas dan fungsinya, DJBC memiliki aplikasi yang disebut dengan CEISA 4.0. Kepanjangan dari *Customs-Excise Information System and Automation*, merupakan sistem informasi untuk mengotomasi dan meningkatkan efisiensi layanan kepabeanan dan cukai. CEISA 4.0 terdiri dari berbagai macam aplikasi internal, salah satunya adalah Sistem Informasi Kesekretariatan, yang menunjang kinerja pegawai DJBC di bidang administrasi. Di dalamnya terdiri dari beberapa modul, seperti Presensi, Cuti, Penugasan, dan lain sebagainya.

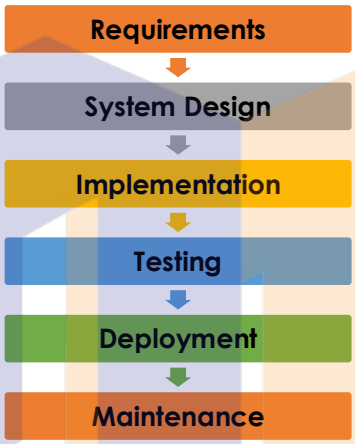
2.1.3. Website

Website adalah kumpulan dari halaman web yang saling terhubung. Halaman muka yang dilihat saat membuka sebuah *website* disebut dengan *homepage*, memberikan navigasi ke halaman-halaman lainnya dan berfungsi sebagai identitas utama dari *website*. Sebuah website ditempatkan pada sebuah server web yang dapat diakses melalui jaringan internet ataupun intranet. Dan gabungan dari semua *website* yang dapat diakses melalui internet ini disebut dengan *World Wide Web* atau lebih dikenal dengan singkatan WWW.

Tim Berners-Lee, penemu WWW menggambarkan web sebagai sistem terbuka yang memungkinkan siapa saja untuk mengakses dan menautkan informasi dari berbagai sumber. Dalam artikelnya Ia menyatakan, "*Untuk menjadi bagian dari web, aplikasi atau halaman, Anda harus dapat ditautkan dan memungkinkan akses oleh klien mana pun*". Ia menekankan pentingnya keterbukaan dan aksesibilitas dalam pengembangan web. [13]

2.1.4. Waterfall

Model *Waterfall* dalam *Software Development Life Cycle* (SDLC) adalah pendekatan pengembangan perangkat lunak yang bersifat linear dan berurutan. Setiap fase dalam model ini harus diselesaikan sepenuhnya sebelum melanjutkan ke fase berikutnya, tanpa adanya tumpang tindih antar fase.



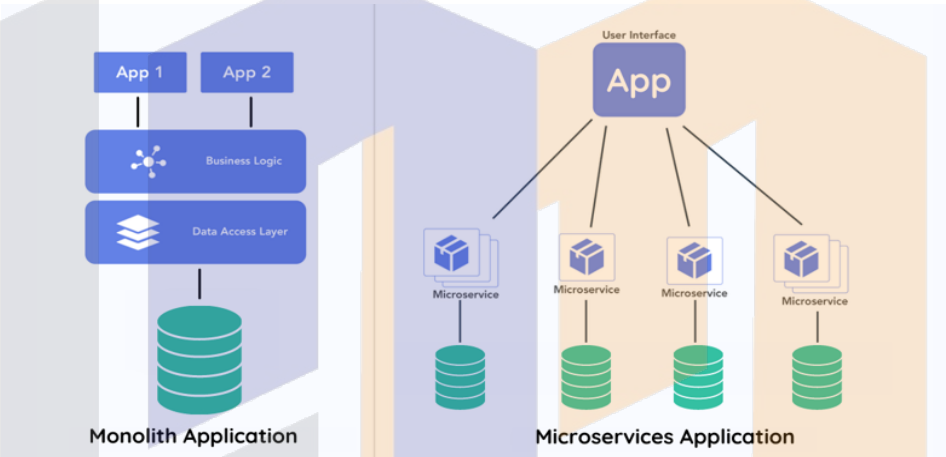
Gambar II.2 Metode *Waterfall*

Fase-fase utama dalam model *Waterfall* meliputi:

a. Analisis Kebutuhan (*Requirements*): Mengumpulkan dan mendokumentasikan semua kebutuhan sistem dari pemangku kepentingan.
b. Desain Sistem (*System Design*): Merancang arsitektur dan spesifikasi teknis berdasarkan kebutuhan yang telah dikumpulkan.
c. Implementasi (*Implementation*): Mengembangkan dan mengkodekan sistem sesuai dengan desain yang telah dibuat.
d. Pengujian (*Testing*): Melakukan pengujian untuk memastikan bahwa sistem berfungsi sesuai dengan spesifikasi dan bebas dari kesalahan.
e. Penerapan (*Deployment*): Menginstal dan mengkonfigurasi sistem di lingkungan produksi untuk digunakan oleh pengguna akhir.
f. Pemeliharaan (*Maintenance*): Melakukan perawatan, perbaikan, dan pembaruan sistem setelah diterapkan untuk memastikan kinerjanya tetap optimal.

Model ini cocok digunakan untuk proyek dengan kebutuhan yang jelas dan tidak berubah-ubah, serta di mana pemahaman penuh tentang sistem dapat dicapai sebelum pengembangan dimulai. Namun, kekakuan model ini membuatnya kurang fleksibel dalam menghadapi perubahan kebutuhan selama proses pengembangan. [14]

2.1.5. Microservices



Gambar II.3 Arsitektur *Microservices*
(sumber: www.dev.to) [15]

Microservices adalah pendekatan arsitektur dalam pengembangan perangkat lunak di mana aplikasi dipecah menjadi layanan-layanan kecil yang berdiri sendiri dan berkomunikasi melalui API yang terdefinisi dengan baik. Setiap layanan biasanya berfokus pada kapabilitas bisnis tertentu dan dapat dikembangkan, diuji, serta diterapkan secara independen oleh tim kecil yang otonom. [16]

Karakteristik dari arsitektur *Microservices* antara lain:

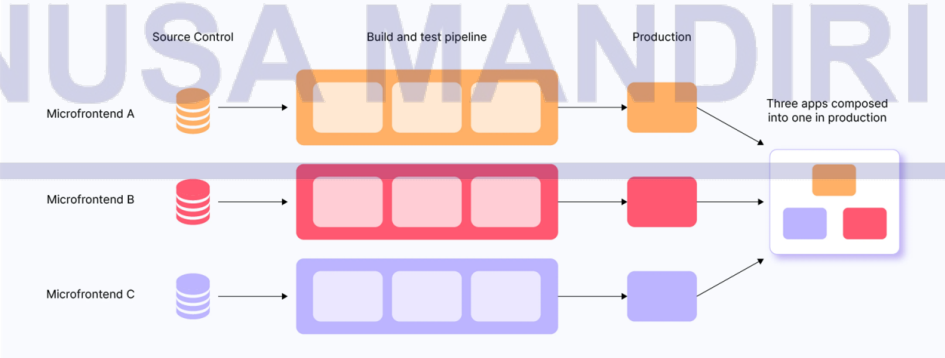
a. Modularitas: Aplikasi dibangun dari bagian-bagian kecil (modul) yang masing-masing menjalankan fungsi spesifik. Setiap modul bisa dikembangkan, diuji, serta dikelola secara terpisah tanpa mengganggu modul lainnya.
b. Kemandirian: Setiap komponen layanan dalam *microservices* dapat dibuat dan dijalankan secara independen. Masing-masing modul dapat menggunakan

teknologi yang berbeda, termasuk bahasa pemrograman, sistem basis data, dan *framework*.

c. Komunikasi melalui API: Modul-modul dapat saling berinteraksi melalui *Application Programming Interface* (API), biasanya dengan menggunakan protocol HTTP, REST, atau *message broker* seperti Kafka. Hal ini memungkinkan kolaborasi antar modul meski menggunakan teknologi yang beragam.
d. Skalabilitas Terpisah: Setiap modul bisa ditingkatkan kapasitasnya secara individual sesuai beban kerja. Jika satu modul mengalami lonjakan permintaan, hanya modul tersebut yang perlu diskalakan tanpa mengubah modul lainnya.
e. Pengembangan dan Penyebaran Terpisah: Modul-modul dapat dikembangkan dan di-*deploy* secara mandiri. Hal ini memungkinkan tim pengembang untuk bekerja secara paralel tanpa menunggu proses dari modul lain selesai. [17]

Secara keseluruhan, arsitektur *microservices* menawarkan pendekatan yang fleksibel dan skalabel untuk pengembangan aplikasi modern, namun memerlukan perencanaan dan pengelolaan yang cermat untuk mengatasi kompleksitas yang muncul.

2.1.6. Microfrontends



Gambar II.4 Arsitektur *Microfrontends*
(sumber: www.infracloud.io) [18]

Microfrontends adalah pendekatan arsitektur dalam pengembangan antarmuka pengguna (UI) di mana aplikasi web dipecah menjadi bagian-bagian kecil yang dapat dikembangkan, diuji, dan diterapkan secara independen oleh tim yang berbeda. Pendekatan ini mirip dengan konsep *microservices* di sisi *backend*, tetapi diterapkan pada *frontend*.

Karakteristik dari arsitektur *Microfrontends* antara lain:

a. Independency: Setiap bagian dari *frontend* dikembangkan, diuji, dan dideploy secara mandiri oleh tim yang berbeda. Mirip dengan konsep *microservices*, namun diterapkan pada level antarmuka pengguna (*User Interface*).
b. Technology Agnostic: Setiap *microfrontend* dapat dibangun dengan teknologi berbeda, selama mereka tetap bisa diintegrasikan dengan lancar ke aplikasi utama.
c. Isolated Development and Deployment: Karena terisolasi, setiap *microfrontend* bisa dikembangkan dan di-*deploy* tanpa harus menunggu bagian lain selesai. Ini mendukung *continuous delivery* dan *independent scaling*.
d. Composition at Runtime or Build Time: *Microfrontends* bisa dikombinasikan saat *build time* (misalnya menggunakan *module federation* atau monorepo), atau saat *runtime* (menggunakan iframe, Web Components, atau lainnya).
e. Resilience: Jika satu *microfrontend* mengalami error, tidak harus memperbaiki seluruh aplikasi, aplikasi lain tetap bisa berjalan tanpa error. [19]

Arsitektur *microfrontends* menawarkan pendekatan modular untuk pengembangan *frontend*, memungkinkan skalabilitas dan fleksibilitas yang lebih besar. Namun, penting untuk mempertimbangkan tantangan potensial dan memastikan praktik terbaik diikuti untuk memitigasi masalah yang mungkin timbul.

2.1.7. API Services

Gambar II.5 Ilustrasi API *Service*

(sumber: www.quintagroup.com) [20]

API (*Application Programming Interface*) *services* adalah seperangkat aturan dan protokol yang memungkinkan dua perangkat lunak berkomunikasi satu sama lain. API mendefinisikan cara permintaan data dilakukan, format data yang digunakan, dan bagaimana respons dikembalikan. API Service merujuk pada layanan yang disediakan melalui API, memungkinkan aplikasi atau sistem lain untuk mengakses fungsionalitas atau data tertentu.

Karakteristik dari API *Services* antara lain:

a. Integrasi Layanan Pihak Ketiga: Misalnya, sebuah aplikasi dapat menggunakan Google Maps API untuk menampilkan peta tanpa harus mengembangkan fitur pemetaan sendiri.
b. Pengembangan *Microservices*: Perusahaan seperti WSO2 menawarkan platform *middleware* untuk integrasi yang gesit, manajemen API, serta manajemen identitas dan akses.
c. Manajemen API: Kong Inc. menyediakan platform *open-source* dan layanan *cloud* untuk mengelola, memantau, dan menskalakan API serta *microservices*. [21]

2.1.8. Spring Boot

Gambar II.6 Logo Spring Boot

(sumber: www.spring.io) [22]

Spring Boot adalah kerangka kerja *open-source* berbasis Java yang dirancang untuk mempermudah pengembangan aplikasi Spring yang berdiri sendiri dan siap produksi. Dengan pendekatan *convention over configuration*, Spring Boot mengurangi kebutuhan konfigurasi manual, memungkinkan pengembang untuk memulai proyek dengan cepat dan efisien.

Karakteristik dari Spring Boot antara lain:

a. Server Web Tersemat: Menyediakan server aplikasi seperti Tomcat, Jetty, atau Undertow yang terintegrasi, sehingga aplikasi dapat dijalankan tanpa perlu mengatur server eksternal.
b. Starter POMs: Menyediakan kumpulan dependensi yang telah dikonfigurasi sebelumnya untuk memudahkan integrasi dengan berbagai pustaka dan modul.
c. Konfigurasi Otomatis: Mendeteksi pustaka yang ada di *classpath* dan secara otomatis mengonfigurasi aplikasi sesuai dengan kebutuhan.
d. Fitur Siap Produksi: Menyediakan fungsionalitas seperti metrik, pemeriksaan kesehatan, dan konfigurasi eksternal untuk mempermudah pemantauan dan pengelolaan aplikasi. [23]

Dengan fitur-fitur tersebut, Spring Boot memungkinkan pengembang untuk fokus pada logika bisnis tanpa terbebani oleh konfigurasi yang kompleks, sehingga mempercepat proses pengembangan dan penerapan aplikasi.

2.1.9. ReactJS

Gambar II.7 Logo ReactJS

(sumber: www. react.dev) [24]

React (juga dikenal sebagai React.js atau ReactJS) adalah pustaka JavaScript open-source yang digunakan untuk membangun antarmuka pengguna berbasis komponen. Dikembangkan oleh Meta (sebelumnya Facebook) dan komunitas pengembang, React mempermudah pembuatan aplikasi web yang dinamis dan responsif.

Karakteristik dari ReactJS antara lain:

a. Pendekatan Berbasis Komponen: Memungkinkan pengembang untuk membangun UI menggunakan komponen yang dapat digunakan kembali, sehingga meningkatkan modularitas dan efisiensi kode. [25]
b. Virtual DOM: Menggunakan representasi virtual dari DOM untuk meminimalkan manipulasi DOM langsung, yang meningkatkan kinerja aplikasi. [26]
c. JSX (JavaScript XML): Sintaks mirip HTML yang mempermudah penulisan struktur UI dalam kode JavaScript. [27]

React dikembangkan menggunakan konsep komponen untuk menyusun UI, sehingga lebih mudah dikelola dan digunakan kembali. React juga menggunakan Virtual DOM untuk mempercepat pembaruan tampilan secara efisien.

2.1.10. PostgreSQL

Gambar II.8 Logo PostgreSQL

(sumber: www.postgresql.org) [28]

PostgreSQL sering disebut sebagai Postgres, adalah sistem manajemen basis data relasional (*Relational Database Management System* atau RDBMS) *open-source* yang menekankan pada ekstensibilitas dan kepatuhan terhadap standar SQL. Dikembangkan oleh PostgreSQL Global Development Group, PostgreSQL mendukung transaksi dengan properti ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*), serta fitur seperti tampilan yang dapat diperbarui secara otomatis, tampilan materialisasi, *trigger*, *foreign key* dan *stored procedures*.

Karakteristik dari PostgreSQL antara lain:

a. Ekstensibilitas: PostgreSQL memungkinkan pengguna menambahkan tipe data, fungsi, operator dan metode *custom index*, menjadikannya sangat dapat disesuaikan untuk berbagai kebutuhan aplikasi.
b. Kepatuhan terhadap Standar SQL: PostgreSQL sangat mematuhi standar SQL, memastikan kompatibilitas dan interoperabilitas dengan alat dan aplikasi lain.
c. Dukungan untuk Transaksi ACID: PostgreSQL memastikan integritas data melalui implementasi properti ACID, yang penting untuk aplikasi yang memerlukan transaksi yang andal.
d. Replikasi dan Ketersediaan Tinggi: PostgreSQL mendukung replikasi streaming dan solusi ketersediaan tinggi lainnya, memungkinkan skalabilitas dan keandalan yang lebih besar. [29]

2.2. Penelitian Terkait

Untuk mendukung pengembangan aplikasi SMARD, penulis melakukan studi pusaka terhadap beberapa penelitian terkait yang memiliki kesamaan, baik dari pendekatan teknologi, maupun metodologi pengembangan. Penelitian terkait sangat penting untuk dijadikan landasan dalam pengembangan sebuah sistem, serta untuk mengetahui kelebihan dan kekurangan dari sistem terdahulu. Penelitian-penelitian berikut dijadikan acuan dalam pembangunan SMARD.

Tabel II.1 Tabel Perbandingan Penelitian Terkait

Tahun	Judul	Penulis	Persamaan	Perbedaan
2023	Desain Sistem *Monitoring* dan Evaluasi Pemberian Tugas pada Bea Cukai Kantor Wilayah Sumatera Bagian Timur Berbasis Web [30]	Muhamad Lutfhan Nugraha Sani, Sri Hariani Eko Wulandari dan Ayuningtyas	1. Alur data yang hampir mirip, karena terkait dengan instansi yang sama walaupun berbeda kantor. 2. Perancangan sistem dibuat dengan berbasis website, dengan mekanisme sistem yang sebagian besar sama.	1. Penelitian hanya berupa rancangan bangun, belum teraplikasikan ke sebuah sistem. 2. Role hanya ada di 2 level user, sedangkan pada SMARD ada berbagai macam user dengan fungsi masing-masing dan terdapat 3 level user. 3. Tidak terdapat dashboard untuk *monitoring*.
2024	Sistem Informasi *Monitoring* dan Evaluasi Kinerja Service Center menggunakan Performance Dashboard [31]	Kelvin Pradana, Agus Prasetyo Utomo, dan Novita Mariana	Menyediakan dashboard untuk visualisasi *monitoring* kinerja.	Sistem tidak difokuskan pada perekaman tindak lanjut, hanya seputar kinerja operasional layanan.
2025	Pengembangan Sistem Informasi *Monitoring* Harian Magang Industri Pendidikan Teknik Mesin menggunakan Model 4D [32]	Rasyid Sidik, Azizah Nurul Husnaini, Riina Syivarulli, Muhammad Kholil, dan Ngatou Rohman	1. Terdapat pencatatan aktivitas harian dan evaluasi. 2. Tersedianya perekaman tindak lanjut, ditampilkan dalam bentuk log kegiatan.	Pendekatan pengembangan yang digunakan menggunakan model 4D (*Define, Design, Develop, Disseminate*).

Berdasarkan perbandingan pada tabel di atas, dapat disimpulkan bahwa meskipun ketiga penelitian terdahulu memiliki fokus pada sistem *monitoring* berbasis web, masing-masing diarahkan untuk konteks yang berbeda. Penelitian pertama yang paling mendekati dari sisi institusi, namun belum secara khusus mengatur tata kelola arahan strategis dari Direktur Jenderal. SMARD hadir sebagai solusi spesifik untuk kebutuhan *monitoring* arahan Direktur Jenderal di seluruh lingkungan DJBC secara terintegrasi, terdokumentasi, dan berkala. Dengan demikian, penelitian ini melengkapi kekosongan yang belum diangkat dalam penelitian sebelumnya.