Pengembangan Back-End dan Arsitektur Sistem Pada Aplikasi Key's Laundry

Rayhan Daffa Aulya Putra
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
rayhdf@student.telkomuniversity.ac.id

Roswan Latuconsina
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
roswan@telkomuniversity.ac.id

Astri Novianty
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
astrinov@telkomuniversity.ac.id

***Abstrak* — Key's Laundry merupakan penyedia jasa laundry di daerah sekitar Universitas Telkom. Key's Laundry menggunakan sistem pencatatan manual dan digital. Pencatatan dua kali ini dilakukan karena sistem digital yang ada tidak memenuhi proses bisnis. Dengan sistem pencatatan tersebut memerlukan usaha yang lebih besar bagi seorang pencatat. Untuk meningkatkan efisiensi usaha, dibutuhkan sebuah sistem pencatatan digital yang memenuhi seluruh spesifikasi yang dibutuhkan sesuai proses bisnis. Solusi dari permasalahan tersebut adalah dengan dibuatnya aplikasi Android Key's Laundry. Dalam pengembangan aplikasi Android tersebut, diperlukan *back-end* untuk berinteraksi dengan *database* dan menangani seluruh logika untuk data. Back-end harus memiliki waktu *down* yang minimal, agar tidak mengganggu proses bisnis. Oleh karena itu, arsitektur yang digunakan adalah arsitektur *microservice*. Arsitektur tersebut memisahkan komponen menjadi servis-servis kecil, dan jika salah satu servis *down*, maka tidak akan terpengaruh ke servis lain, dan pengguna masih dapat menggunakan bagian aplikasi yang tidak terpengaruh oleh waktu *down*. Untuk mengembangkan *back-end* memerlukan *tools* dan *framework*, serta layanan untuk men-*deploy*. Pengembangan menghasilkan *back-end* dengan arsitektur *microservice* dengan sebuah API *gateway* untuk menggabungkan *endpoint* dari tiap servis.**

***Kata kunci*— Key's Laundry, *back-end*, arsitektur sistem, *database*, *deployment*, FastAPI**

I. PENDAHULUAN

UMKM (Usaha Mikro Kecil dan Menengah) merupakan unit usaha yang memiliki tenaga kerja 5 hingga 99 orang dalam operasi bisnisnya [1]. UMKM merupakan bagian penting untuk masyarakat sebagai penyedia layanan lokal dan menurunkan angka pengangguran.

Key's Laundry merupakan UMKM laundry yang menyediakan layanan jasa di bidang cuci mencuci pakaian yang berada di daerah sekitar Universitas Telkom. Untuk membantu dalam operasional sehari-hari, Key's Laundry menggunakan sistem manajemen manual dan digital.

Meskipun Key's Laundry sudah menggunakan sistem manajemen digital, operasional bisnis laundry masih kurang efektif karena sistem manajemen manual dan digital yang digunakan secara bersamaan menyebabkan usaha yang harus dilakukan bagi pencatat lebih banyak jika dibandingkan dengan hanya menggunakan sistem manajemen digital sepenuhnya. Oleh karena itu, Key's Laundry memerlukan sistem digital yang dengan spesifikasi yang sesuai dengan proses bisnisnya untuk memaksimalkan efisiensi sumber daya.

Aplikasi Key's Laundry merupakan aplikasi Android khusus untuk bisnis Key's Laundry. Aplikasi Key's Laundry yang dibuat memiliki spesifikasi lengkap sesuai yang diperlukan oleh proses bisnis. Proses bisnis yang memerlukan pencatatan digital dengan aplikasi berupa pencatatan data pelanggan dan layanan, pembuatan pesanan, memperbarui status pesanan, pembuatan dan pencetakan nota fisik dan digital, pencatatan kehadiran dan gaji karyawan, dan pembuatan laporan keuangan untuk sebuah periode.

Dalam pengembangan aplikasi Key's Laundry, diperlukan *back-end* untuk menangani interaksi dengan *database* untuk menyimpan data, serta menangani seluruh logika data atau memproses data untuk digunakan di *front-end*. Selain itu, aplikasi membutuhkan arsitektur sistem yang memudahkan pengembang, serta meminimalkan waktu *down* agar tidak mengganggu proses bisnis.

II. KAJIAN TEORI

A. Web API

Web API adalah API yang menggunakan HTTP (Hypertext Transfer Protocol) untuk pengiriman data. HTTP adalah protokol komunikasi yang memungkinkan pengguna internet untuk mengirimkan berbagai jenis media, seperti tulisan, gambar, atau JSON [2]. HTTP memiliki beberapa *methods* sesuai penggunaannya, dan yang sering digunakan adalah GET dan POST, yaitu untuk menerima dan mengirim data.

B. FastAPI

FastAPI adalah *framework* web untuk membangun API dengan Python berdasarkan *type hints* Python [3]. Sebagai framework, FastAPI dilengkapi beberapa fitur yang dapat membantu mempercepat pengembangan. FastAPI dapat secara otomatis membuat dokumentasi terhadap *endpoint* yang telah dibuat, dan pengembang dapat mengujinya langsung dengan Swagger UI.

C. Docker

Docker adalah suatu *platform* terbuka untuk membangun, mengirim, dan menjalankan aplikasi dalam *containerized environment* [4]. Docker mengemas aplikasi menjadi *container* dan aplikasi tersebut dapat dijalankan di mesin manapun yang memiliki Docker Engine. Mesin dengan Docker Engine dapat berupa mesin lokal atau mesin layanan *cloud* yang dapat mendeploy *container* Docker.

D. Google Cloud Run

Google Cloud Run merupakan layanan *serverless computing* untuk melakukan *deployment* sebuah *container* yang telah dibangun oleh Docker ataupun alat containerisasi lainnya menjadi sebuah *service* [5]. Google Cloud Run mendukung penskalaan berdasarkan jumlah permintaan pengguna ke *service*.

E. Arsitektur *Microservice*

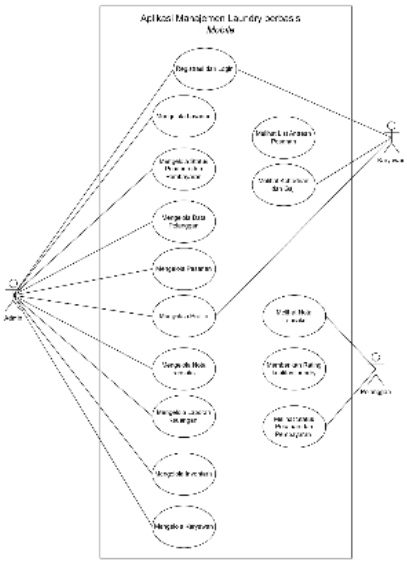
Microservice adalah arsitektur sistem yang memisahkan komponen-komponennya, seperti servis-servis kecil yang bekerja sama [2]. Dengan memisahkan komponen-komponen, jika sebuah servis mengalami *down time*, servis lainnya tidak akan terpengaruh, dan fitur-fitur lainnya dalam aplikasi tetap dapat diakses. Keuntungan *microservice* lainnya yaitu mudah untuk pengembangan, karena satu servis dapat menggunakan *framework* yang berbeda dengan servis lainnya.

III. METODE

Untuk melakukan pengembangan back-end dan arsitektur sistem, dibutuhkan perancangan desain sistem sebagai gambaran untuk aplikasi yang akan dibuat. Rancangan berupa mendesain arsitektur sistem dan mengembangkan sistem sesuai arsitektur yang telah dibuat.

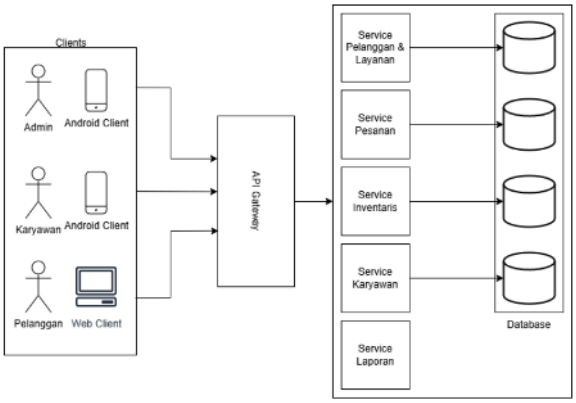
A. Arsitektur Sistem

Untuk menggambarkan arsitektur sistem, maka harus menentukan apa saja fungsi-fungsi yang diperlukan melalui *use case*. Setiap kelompok *use case* umum akan dipisahkan menjadi servis sendiri.



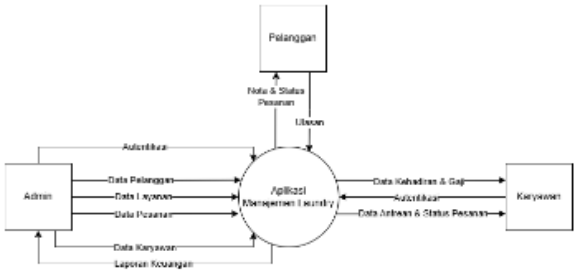
GAMBAR 1
(*Use Case Diagram*)

Berdasarkan *use case*, terdapat tiga jenis pengguna, dan masing-masing memiliki *use case* sendiri atau *use case* yang digunakan bersama. Arsitektur sistem yang akan digunakan adalah arsitektur *microservice* dengan API gateway. Arsitektur ini cocok untuk ketika ingin memisahkan fungsi-fungsi *back-end* menjadi servis atau aplikasi *self-contained*. Dengan ini, jika salah satu servis *down*, maka servis lainnya tidak terpengaruh, atau jika salah satu servis memiliki *traffic* yang tinggi, maka hanya servis tersebut yang diskalakan.



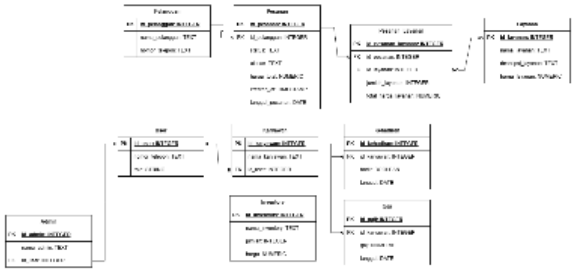
GAMBAR 2
(Arsitektur Sistem)

Terdapat enam servis sesuai fungsionalitasnya masing-masing. Servis pertama mengatur penyimpanan data pelanggan dan layanan ke database, serta pengiriman data dari database ke klien. Servis kedua membuat pesanan berdasarkan data yang pelanggan dan layanan yang telah dibuat, menghitung total harga pesanan, dan membuat serta melayani nota digital dalam bentuk web kepada pelanggan. Servis inventaris dan servis karyawan menyimpan data terkait ke database, serta mengirim datanya ke klien. Servis laporan berfungsi untuk membuat laporan keuangan berdasarkan data-data yang ada di database. Servis ini tidak menyimpan data ke database, hanya mengambil data dan mengirimnya ke klien. Data yang dikirim berupa seluruh data keuangan yang ada yaitu data pesanan, gaji dari karyawan, dan pengeluaran dari inventaris. Dalam salah satu *endpoint*-nya, servis ini juga dapat menyusun laporan dalam bentuk Excel untuk disimpan secara lokal di perangkat klien. Dengan penjelasan tersebut, dapat menggambarkan bagaimana data mengalir pada sistem.



GAMBAR 3
(Context Diagram)

Context Diagram memberi gambaran umum bagaimana data mengalir pada sebuah sistem. Dengan gambaran ini, kita dapat merancang *database* yang diperlukan serta relasi-relasi antar tabel dalam *database*.



GAMBAR 4
(*Entity Relationship Diagram*)

Diagram Relasi Entitas menggambarkan tabel-tabel *database* yang diperlukan, serta relasi antar tabel berupa *one-to-many* atau *one-to-one*, serta relasi *foreign key* yang diperlukan untuk meminimalkan pengulangan data.

B. Pengembangan Sistem

Untuk mengembangkan sistem sesuai arsitektur yang telah dipilih, dilakukan beberapa tahap. Tahap-tahap yang dilakukan yaitu melakukan pengembangan *back-end* API dengan *framework*, serta melakukan *deployment*.

Pertama, melakukan pengembangan back-end API dengan framework. Framework yang digunakan adalah FastAPI, karena FastAPI dapat mempercepat pengembangan dengan fitur dokumentasi otomatis. Selain itu, FastAPI memiliki performa tinggi dan dapat didukung *library* Python lainnya jika diperlukan. Pengembangan back-end berupa membuat endpoint yang akan dipakai, memproses data sebelum mengirim ke database atau klien, dan mengirim data ke klien atau database. Back-end yang dibuat berupa servis-servis kecil yang digabungkan dengan servis API gateway. API gateway juga berperan sebagai servis yang berinteraksi dengan klien Android.

Kedua, melakukan *deployment* back-end yang telah dibuat. Untuk men-*deploy* dengan mudah, menggunakan Docker untuk melakukan kontainerisasi, lalu melakukan *deployment* dengan Google Cloud Run. Dengan Docker, aplikasi dapat dijadikan *containerized application* dan dapat dijalankan di mesin manapun yang memiliki Docker Engine. Selain itu, Google Cloud Run mendukung *scaling*, dan dapat mengatur jumlah *instance* yang ada berdasarkan *traffic* dari pengguna.

IV. HASIL DAN PEMBAHASAN

Hasil pengembangan *back-end* berupa *back-end* API dalam bentuk *microservice* yang melayani dan menangani seluruh interaksi dengan klien dan *database*, serta seluruh pemrosesan data yang perlu dilakukan. Back-end ini berhasil dibuat untuk digunakan dengan klien Android melalui integrasi dengan aplikasi Key's Laundry.

A. Hasil *Endpoint*

Berikut hasil *endpoint* pada tiap servis yang ada yang digabungkan di servis API *gateway*:

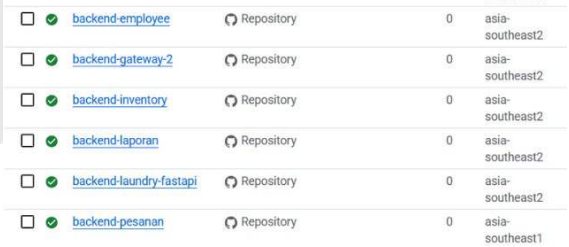
TABEL 1
(Daftar *Endpoint*)

Servis	HTTP Method	Endpoint
Pelanggan dan Layanan	GET	/pelanggan/
	POST	/pelanggan/
	GET	/pelanggan/{pelanggan_id}
	PUT	/pelanggan/{pelanggan_id}
	DELETE	/pelanggan/{pelanggan_id}
	GET	/layanan/
	POST	/layanan/
	GET	/layanan/{layanan_id}
	PUT	/layanan/{layanan_id}
	DELETE	/layanan/{layanan_id}
Pesanan	GET	/pesanan/
	POST	/pesanan/
	GET	/pesanan/{pesanan_id}
	PATCH	/pesanan/{pesanan_id}
	DELETE	/pesanan/{pesanan_id}
	GET	/pesanan/nota/{pesanan_id}
Inventory	GET	/inventory/
	PUT	/inventory/
	GET	/inventory/{id_inventory}
	POST	/inventory/{id_inventory}
	DELETE	/inventory/{id_inventory}
Karyawan	GET	/kehadiran/
	POST	/kehadiran/
	GET	/kehadiran/{id_kehadiran}
	PUT	/kehadiran/{id_kehadiran}
	DELETE	/kehadiran/{id_kehadiran}
	GET	/kehadiran/{id_karyawan}
	GET	/gaji/
	POST	/gaji/
	GET	/gaji/{id_gaji}
	PUT	/gaji/{id_gaji}
	DELETE	/gaji/{id_gaji}
	GET	/gaji/karyawan/{id_karyawan}
	PATCH	/gaji/{id_gaji}/konfirmasi
Laporan	GET	/reports/employees/
	GET	/reports/attendance/
	GET	/reports/sales/
	GET	/reports/daily-sales/
	GET	/reports/services/{service_id}
	GET	/reports/services-summary
	GET	/reports/customers/
	GET	/reports/inventory/
	GET	/reports/excel/

Tabel di atas menunjukkan seluruh *endpoint* yang dibuat, berdasarkan servisnya. Sebagian *endpoint* hanya menghandal penyimpanan data ke *database* dan pengiriman data ke klien. *Endpoint* lainnya menangani pemrosesan dan logika data sebelum dikirim ke klien atau disimpan ke database, seperti data pesanan yang dihitung total harganya terlebih dahulu, pembuatan laporan dalam bentuk Excel, atau dan perhitungan keuangan. Selain itu, tiap *endpoint* memiliki validasi data atau model data yang terkirim dengan *type hint* Python oleh FastAPI.

B. Hasil *Deployment*

Deployment back-end dilakukan melalui Google Cloud Run untuk menjalankan Docker *container*. Kode *back-end* diunggah ke Github untuk dijadikan *remote repository*, lalu dihubungkan dengan Google Cloud Run untuk dilakukan *build* dan *deploy*, serta secara otomatis mengatur *trigger* yang akan mengecek secara otomatis ketika ada perbaruan kode dan melakukan *build* dan *deploy* baru.



GAMBAR 5
(*Deployment* Servis)

Gambar di atas merupakan hasil *deployment* tiap servis dalam bentuk Docker *container*. Dengan menggabungkan tiap servis menjadi satu melalui gateway, *back-end* siap digunakan oleh klien Android dan melakukan integrasi Android dengan *back-end*.

V. KESIMPULAN

Pengembangan *back-end* dan arsitektur sistem untuk aplikasi Key's Laundry berhasil dilakukan dengan arsitektur *microservice*, serta menggunakan FastAPI sebagai *framework*, Docker untuk kontainerisasi, dan Google Cloud Run untuk menjalankan aplikasi *back-end* di *cloud*. Tiap fitur atau *use case* telah terpenuhi sesuai proses bisnis yang ada untuk memudahkan pencatatan dan meminimalkan usaha pencatat. Tiap *use case* memiliki servis dan *endpoint* masing-masing, dan dapat langsung diintegrasikan dengan Android untuk digunakan oleh klien.

REFERENSI

[1] K. Ismail, M. Rohmah dan D. A. P. Putri, "Peranan UMKM dalam Penguatan Ekonomi Indonesia," *Jurnal Pendidikan dan Ilmu Ekonomi Akuntansi,* vol. VII, pp. 208-217, 2023.

[2] J. H. Peralta, Microservices APIs, Manning, 2023.

[3] FastAPI, "FastAPI Documentation," [Online]. Available: https://fastapi.tiangolo.com/. [Diakses 5 Juli 2025].

[4] L. E. Butarbutar, S. Davina dan V. F. Tambunan, "Implemetasi Docker Untuk Cloud Computing Pada Feroda Linux: Studi Kasus Python Dalam Kontainer," *Jurnal Studi Multidisipliner,* vol. VIII, no. 11, 2024.

[5] Z. Z. Maulidia dan L. Venica, "Serverless Computing: Analisis Cloud Run dan App Engine dalam Profile Website Deployment," *SISTEMASI: Jurnal Sistem Informasi ,* vol. XIII, no. 2, pp. 492-505, 2024.