Arsitektur Perangkat Lunak

Departemen Informatika, Universitas Pradita, Indonesia

2023

Daftar Isi

Bab 1

Pendahuluan

Alfa Yohannis

1.1 Materi

1. Introduction
2. Client-Server Architecture
3. Model-View-Controller Architecture
4. Model-View-ViewModel Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps

AAAA [**?**]

Bab 2

Arsitektur Client-Server

Alfa Yohannis

2.1 Latar Belakang

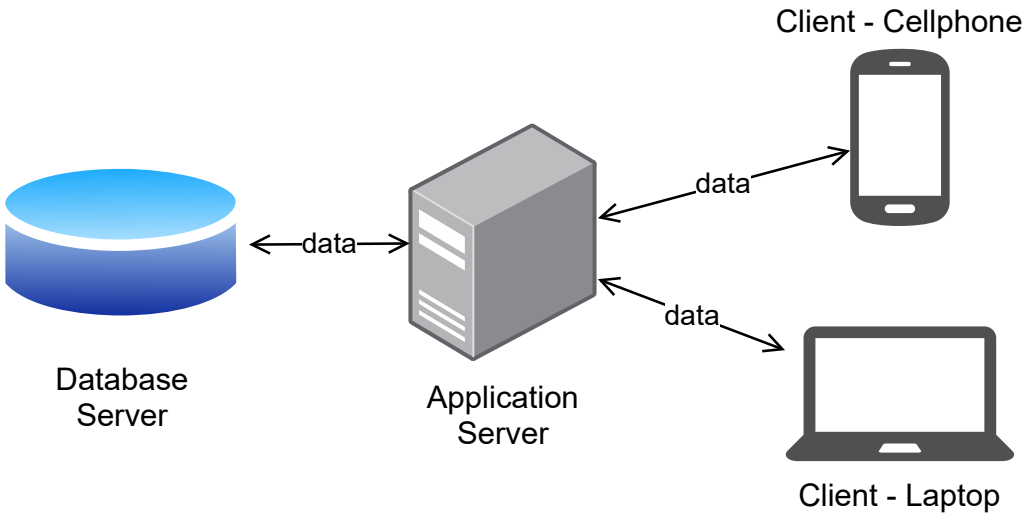
Pada awal komputer bermula sebagai suatu kesatuan, tidak terpisah-pisah. Perangkat lunak hanya berjalan pada satu unit komputer tersebut. Secara perlahan, ada bagian komputer yang dapat terpisah secara fisik dan menjalankan tanggung jawab tertentu. Sebagai contoh, data storage terpisah dari komputer utama. Lalu, beberapa fungsionalitas akhirnya terpisah dan membutuhkan mesin tersendiri. Misalnya, komputer yang didedikasikan untuk menyimpan data atau yang kita sebut sebagai *database server*. Di sisi lain, jaringan komputer juga berkembang dan kemudian menjadi sesautu yang umum. Komputer-komputer saling berkomunikasi satu sama yang lain, dan setiap komputer dapat memiliki peran-peran tertentu yang memungkinkan lahirnya sistem terdistribusi.

2.2 Arsitektur Client-Server

Suatu sistem *client-server* terdiri dari satu *server* dan satu *client* atau lebih. *Server* biasanya memiliki kemampuan komputasi dan penyimpanan data yang lebih cepat dan banyak dibanding *client*. Oleh karena itu, *client* menugaskan *server* untuk melakukan komputasi tertentu dan menerima hasilnya atau sekedar menarik data dari *server*.

Terdapat 2 jenis *client-server architecture*: *two-tier architecture* dan *three-tier architecture*. Two tier-architecture umumnya hanya terdiri dari *desktop application* yang berada di sisi klien dan *database* yang berada di sisi server. Contoh lain adalah *web browser* yang memuat *web application* dan *web server*

untuk melakukan *backend computation*. Arsitektur tersebut dapat diperluas menjadi *three-tier architecture*, dengan menambahkan *database server* seperti yang ditampilkan pada Gambar 14.2.



Gambar 2.1: Skema dari 3-tier client-server arsitektur.

2.3 Kelebihan dan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur client-server:

2.3.1 Kelebihan

Keuntungan dari menerapkan arsitektur client-server adalah:

- Kemampuan komputasi (dan penyimpanan data) dapat diakses dari berbagai lokasi berjauhan dan oleh banyak komputer/pengguna.
- Komputasi-komputasi yang membutuhkan kinerja tinggi dapat didelegasikan ke server.
- Data dapat disentralisasikan sehingga meningkatkan konsistensi data dan mengurangi duplikasi data.
- Sistem dapat menerapkan *horizontal scaling* untuk skalabilitas. Horizontal scaling adalah meningkatkan kinerja komputer dengan penambahan komputer agar beban komputasi dibagi ke komputer-komputer

yang tersedia. Misalnya, awalnya terdapat 10 000 requests perhari yang ditangani oleh suatu *application server*. Jika *application server* ditambah, maka beban tersebut dibagi di antara kedua *server* tersebut. Vertical scaling adalah meningkat kinerja suatu komputer dengan menaikkan spefikasi komputer tersebut, misalnya dengan menggunakan prosesor yang lebih cepat atau meningkatkan kapasitas memori.

2.3.2 Kekurangan

Konsekuensi dari penerapan arsitektur client-server adalah sistem jadi lebih kompleks untuk dikelola:

- Biaya akan meningkat karena terdapat komponen/mesin tambahan yang perlu dikelola.
- Faktor keamanan juga perlu diperhatikan karena server dan client beroperasi dalam suatu jaringan komputer yang mana rawan terhadap *cyber attack*.
- Perlunya koordinasi antar-komputer, misalnya komunikasi sinkron dan asinkron serta komputasi parallel.
- Kompatibilitas antara *server* dan *client* maupun sesama klien.
- Masalah-masalah yang umum terdapat pada jaringan komputer etwork problems, misalnya *network latency*, kesalahan dalam konfigurasi jaringan, dsb.

2.4 Contoh Kasus

2.4.1 Deskripsi

Jelaskan contoh kasus yang dipaparkan berkaitan dengan arsitektur yang dimaksud pada bab ini. Contoh kasus harus memperjelas arsitektur yang dimaksud.

2.4.2 Penjelasan Implementasi

Jelaskan bagian-bagian kode program, basisdata, atau konfigurasi yang signifikan terhadap arsitektur yang dimaksud.

2.5 Kesimpulan

Rangkum dan ulangi (beri penekanan pada) hal-hal kunci dari arsitektur yang dimaksud.

Bab 3

Arsitektur MVC (Model-View-Controller)

Alfa Yohannis

3.1 Latar Belakang

Pada mulanya pengembangan perangkat lunak menyatukan fungsi-fungsi dari *graphical user interface* (GUI) dan pengelolaan data ke dalam satu kode tanpa memisahkan mereka sesuai dengan perhatian (*concerns*) mereka masing-masing. Konsekuensinya, pola tersebut akan menimbulkan masalah ketika *developer* diminta untuk membangun aplikasi skala besar, misalnya aplikasi yang menolong pengguna berinteraksi dengan dataset yang besar dan kompleks. Kode program akan menjadi lebih tidak terstruktur (*spaghetti code*) dan sulit untuk dipahami. Sebagai solusi, kode program perlu dibagi ke dalam komponen-komponen sesuai dengan perhatian mereka (*separation of concerns*). Arsitektur Model-View-Controller (MVC) kemudian diajukan untuk membagi kode program ke dalam tiga abstraksi utama: *model*, *view*, dan *controller*.

3.2 Arsitektur Model-View-Controller

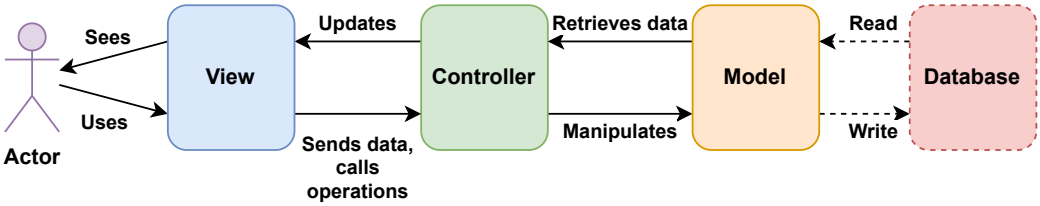
Arsitektur MVC adalah pola arsitektur untuk pengembangan *Graphical User Interface* (GUI). Arsitektur tersebut membagi logika progam menjadi 3 bagian yang saling terhubung: Model, View, dan Controller. Skema dari MVC dapat dilihat pada Gambar 3.1..

Model ditujukan untuk berinteraksi dengan data: menyimpan, memperharui, menghapus, dan menarik data dari database. Model juga digunakan

untuk menggagregasi data sesuai dengan logika bisnis yang dijalankan.

View merupakan presentasi yang ditampilkan ke pengguna yang dengannya pengguna dapat berinteraksi. Misalnya, halaman web, GUI desktop, diagram, *text fields*, *buttons*, dsb.

Controller bertugas untuk menerima input dari pegguna melalui *view* dan meneruskan input tersebut ke model untuk disimpan atau diproses lebih lanjut. Controller juga menarik data dari *model* dan memembetuknya demikian rupa sehingga siap untuk dikirimkan ke *view* untuk ditampilkan ke pengguna.



Gambar 3.1: Arsitektur Model-View-Controller (MVC).

3.3 Kelebihan dan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur MVC:

3.3.1 Kelebihan

Keuntungan dari menerapkan arsitektur MVC adalah:

- Pemisahan presentasi dan data membolehkan model ditampilkan di banyak *view* secara bersamaan.
- View bersifat *composable* artinya view dapat dibangun dari berbagai atau berisi *subviews/fragments*.
- Controller satu dapat diganti (*switchable*) dengan controller lain pada saat *runtime*.
- Developer dapat membuat berbagai macam mekanisme pemrosesan data dari input ke output dengan mengkombinasikan berbagai macam fungsionalitas yang dimiliki oleh views, controllers, dan models.
- *Data engineers*, *backend* dan *frontend developers* masing-masing dapat fokus mengerjakan tugas utama mereka. Misal, *data engineers* hanya mengerjakan tugas yang berkaitan dengan data, sendangkan *frontend developers* fokus ke *user interface*.

3.3.2 Kekurangan

Konsekuensi dari penerapan arsitektur MVC adalah sebagai berikut:

- Derajat kompleksitas kode program bertambah karena kode harus dibagi ke dalam tiga abstraksi yang berbeda.
- *Developers* harus mengikuti aturan ketat tertentu dalam mendefinisikan *controllers*, *models*, dan *views*.
- Secara relative, MVC lebih sulit dipahami dikarenakan struktur bawaannya.
- Terlalu berlebihan (*overkill*) untuk aplikasi sederhana.
- Cocok untuk pembangunan Graphical User Interface tetapi belum tentu cocok untuk pengembangan aplikasi atau komponen yang lain.
- Adanya lapisan-lapisan abstraksi dapat mengurangi kinerja (*performance*) aplikasi.

3.4 Contoh Kasus

3.4.1 Deskripsi

Jelaskan contoh kasus yang dipaparkan berkaitan dengan arsitektur yang dimaksud pada bab ini. Contoh kasus harus memperjelas arsitektur yang dimaksud.

3.4.2 Penjelasan Implementasi

Jelaskan bagian-bagian kode program, basisdata, atau konfigurasi yang signifikan terhadap arsitektur yang dimaksud.

Listing 3.1: Model dari Rate.

```
1  import javax.persistence.Entity;
2  import javax.persistence.Id;
3  import javax.persistence.IdClass;
4
5  @Entity
6  @IdClass(RateId.class)
7  public class Rate {
8    @Id
9    private String fromCurrency;
```

```
10    @Id
11    private String toCurrency;
12    private Double rate;
13    ...
14    Rate(String fromCurrency, String toCurrency, Double
          rate) {
15      ...
16    }
17    ...
18  }
```

Listing 3.2: RateRepository.

```
1  import java.util.Collection;
2  import org.springframework.data.jpa.repository.Query;
3  import org.springframework.data.repository.
       CrudRepository;
4
5  public interface RateRepository extends CrudRepository<
       Rate, Integer> {
6    @Query("SELECT r FROM Rate r WHERE r.fromCurrency = ?1
         and r.toCurrency = ?2")
7    Collection<Rate> findFirstByFromCurrencyAndToCurrency(
         String fromCurrency, String toCurrency);
8
9    @Query("SELECT DISTINCT(r.fromCurrency) FROM Rate r")
10   Collection<String> findAllFromCurrency();
11
12   @Query("SELECT DISTINCT(r.toCurrency) FROM Rate r")
13   Collection<String> findAllToCurrency();
14 }
```

3.5 Kesimpulan

Rangkum dan ulangi (beri penekanan pada) hal-hal kunci dari arsitektur yang dimaksud.

Bab 4

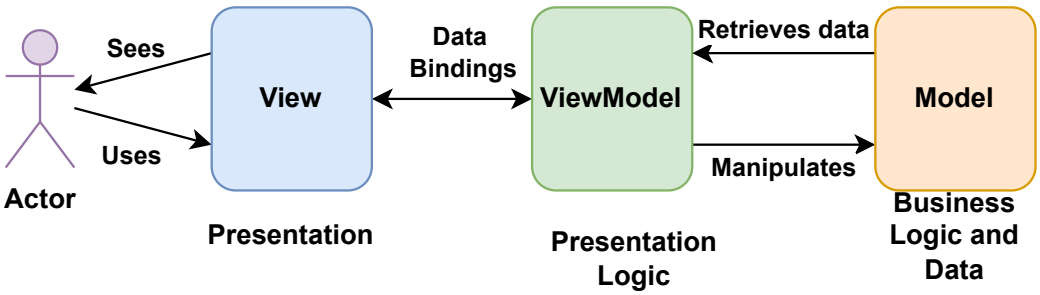
Arsitektur MVVM (Model-View-ViewModel)

Alfa Yohannis

4.1 Latar Belakang

Pada mulanya, dalam pengembangan perangkat lunak, kode yang bertanggung jawab terhadap data, logika bisnis, dan tampilan (Graphical User Interface) bercampur jadi satu, tidak ada pemisahan abstraksi. Pola Model-View-Controller kemudian muncul memisahkan kode program ke dalam 3 abstraksi utama berdasarkan perhatian mereka: *model* untuk data, *view* untuk tampilan, dan *controller* untuk logika bisnis. Hanya saja, MVC tidak memiliki abstraksi yang secara eksplisit mengelola *states* dari tampilan (*views*). Pola MVP (Model-View-Presenter) kemudian diajukan di mana komponen *Presenter*-nya bertanggung jawab mengelola logika presentasi dari *views*. Walaupun demikian,kode program yang mengelola sinkronisasi antara views dan state dari logika presentasi mereka masih harus dibuat secara manual.

Keunikan dari Model-View-ViewModel adalah pola tersebut memiliki komponen *binder* yang mengotomasi komunikasi/sinkronisasi antara view dengan properties yang ada pada *view model*. Nilai-nilai pada *view* ditautkan dengan properties pada view model sehingga perubahan nilai pada salah komponen di view (misalnya perubahan pada *textbox*) akan memperbarui juga nilai pada *property*-nya di *view model* yang ditautkan pada komponen tersebut. Adanya binder mengurangi jumlah kode yang harus ditulis oleh developer secara manual untuk melakukan sinkronisasi antara *view* dan *view model*.



Gambar 4.1: Arsitektur Model-View-ViewModel (MVVM).

4.2 Arsitektur Model-View-ViewModel

- Separation of the view layer by moving all GUI code to the view model via data binding.
- UI developers don't write the the GUI, instead a markup language is used.
- The separation of roles allows UI designers to focus on the UX design rather than programming of the business logic.
- A proper separation of the view from the model is more productive, as the user interface typically changes frequently and late in the development cycle based on end-user feedback.
- Data bindings and properties are used to synchronise the relevant values in the view and the view model, that represents the state of the view, so that they are always the same.
- It eliminates or minimises application logic that directly manipulates the view.

4.3 Kelebihan dan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur MVVM:

4.3.1 Kelebihan

Keuntungan dari menerapkan arsitektur MVVM adalah:

- Separation of the view layer by moving all GUI code to the view model via data binding.

- UI developers don't write the the GUI, instead a markup language is used.

- The separation of roles allows UI designers to focus on the UX design rather than programming of the business logic.

- A proper separation of the view from the model is more productive, as the user interface typically changes frequently and late in the development cycle based on end-user feedback.

- Data bindings and properties are used to synchronise the relevant values in the view and the view model, that represents the state of the view, so that they are always the same.

- It eliminates or minimises application logic that directly manipulates the view.

4.3.2 Kekurangan

Konsekuensi dari penerapan arsitektur MVVM adalah sebagai berikut:

- It can be overkill for small projects.

- Generalizing the viewmodel upfront can be difficult for large applications.

- Large-scale data binding can lead to lower performance.

- It's best for UI development but might not the best for other types of developments and applications.

4.4 Contoh Kasus

4.4.1 Deskripsi

Jelaskan contoh kasus yang dipaparkan berkaitan dengan arsitektur yang dimaksud pada bab ini. Contoh kasus harus memperjelas arsitektur yang dimaksud.

4.4.2 Penjelasan Implementasi

Jelaskan bagian-bagian kode program, basisdata, atau konfigurasi yang signifikan terhadap arsitektur yang dimaksud.

Listing 4.1: Model dari Rate.

```
1  import javax.persistence.Entity;
2  import javax.persistence.Id;
3  import javax.persistence.IdClass;
4
5  @Entity
6  @IdClass(RateId.class)
7  public class Rate {
8    @Id
9    private String fromCurrency;
10   @Id
11   private String toCurrency;
12   private Double rate;
13   ...
14   Rate(String fromCurrency, String toCurrency, Double
       rate) {
15     ...
16   }
17   ...
18 }
```

Listing 4.2: RateRepository.

```
1  import java.util.Collection;
2  import org.springframework.data.jpa.repository.Query;
3  import org.springframework.data.repository.
     CrudRepository;
4
5  public interface RateRepository extends CrudRepository<
     Rate, Integer> {
6    @Query("SELECT r FROM Rate r WHERE r.fromCurrency = ?1
       and r.toCurrency = ?2")
7    Collection<Rate> findFirstByFromCurrencyAndToCurrency(
       String fromCurrency, String toCurrency);
8
9    @Query("SELECT DISTINCT(r.fromCurrency) FROM Rate r")
10   Collection<String> findAllFromCurrency();
11
12   @Query("SELECT DISTINCT(r.toCurrency) FROM Rate r")
13   Collection<String> findAllToCurrency();
```

```
14 }
```

4.5 Kesimpulan

Rangkum dan ulangi (beri penekanan pada) hal-hal kunci dari arsitektur yang dimaksud.

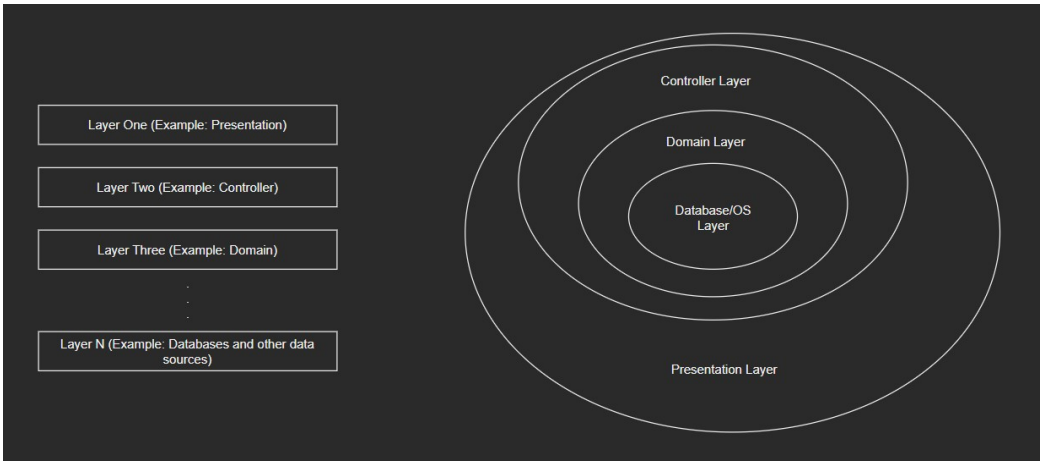
Bab 5

Layered Architecture

Austin Nicholas Tham, Darren Valentio, Muhammad

5.1 Definisi *Layered Architechture*

Pola arsitektur layered adalah pola n-tiered di mana komponen disusun dalam lapisan horizontal. Ini adalah metode tradisional untuk merancang sebagian besar perangkat lunak dan dimaksudkan untuk pengembangan mandiri sehingga semua komponen saling berhubungan tetapi tidak saling bergantung.



Seperti yang ditunjukkan pada gambar, layering biasanya dilakukan dengan mengemas fungsionalitas khusus aplikasi di lapisan atas, penyebaran fungsionalitas spesifik menjadi lapisan bawah dan fungsionalitas yang membentang di seluruh domain aplikasi di lapisan tengah. Jumlah lapisan dan bagaimana lapisan-lapisan ini disusun ditentukan oleh kompleksitas masalah dan solusinya.

Di sebagian besar arsitektur berlapis, ada beberapa lapisan (atas ke bawah):

- **The application layered:** Berisi layanan spesifik aplikasi.
- **The business layer:** Menangkap komponen yang umum di beberapa aplikasi.
- **The middleware layer:** Lapisan ini mengemas beberapa fungsi seperti pembangun GUI, antarmuka ke basis data, laporan, dan dll.
- **The database/System Software Layer:** Berisi OS, database, dan antarmuka ke komponen perangkat keras tertentu.

5.2 Latar Belakang

Penilaian untuk setiap karakteristik berdasarkan kecenderungan alami untuk implementasi tipikal pola layered.

- Kemampuan untuk merespon dengan cepat terhadap lingkungan yang terus berubah. (monolitik)
- Bergantung pada implementasi pola, penyebaran bisa menjadi masalah. Satu perubahan kecil ke komponen dapat memerlukan redeployment seluruh aplikasi.
- Pengembang dapat memberikan pengujian singkat untuk menguji aplikasi sebelum klien menggunakannya
- Mudah dikembangkan karena polanya sudah terkenal dan tidak terlalu rumit untuk melakukan implementasinya.

5.3 Pros Cons

5.3.1 Pros

- Mudah untuk diuji karena komponen-komponennya termasuk lapisan khusus sehingga dapat diuji secara terpisah.
- Sederhana dan mudah diimplementasikan karena secara alami, sebagian besar aplikasi bekerja berlapis-lapis

5.3.2 Cons

- Tidak mudah untuk melakukan perubahan pada lapisan tertentu karena aplikasi merupakan unit tunggal.

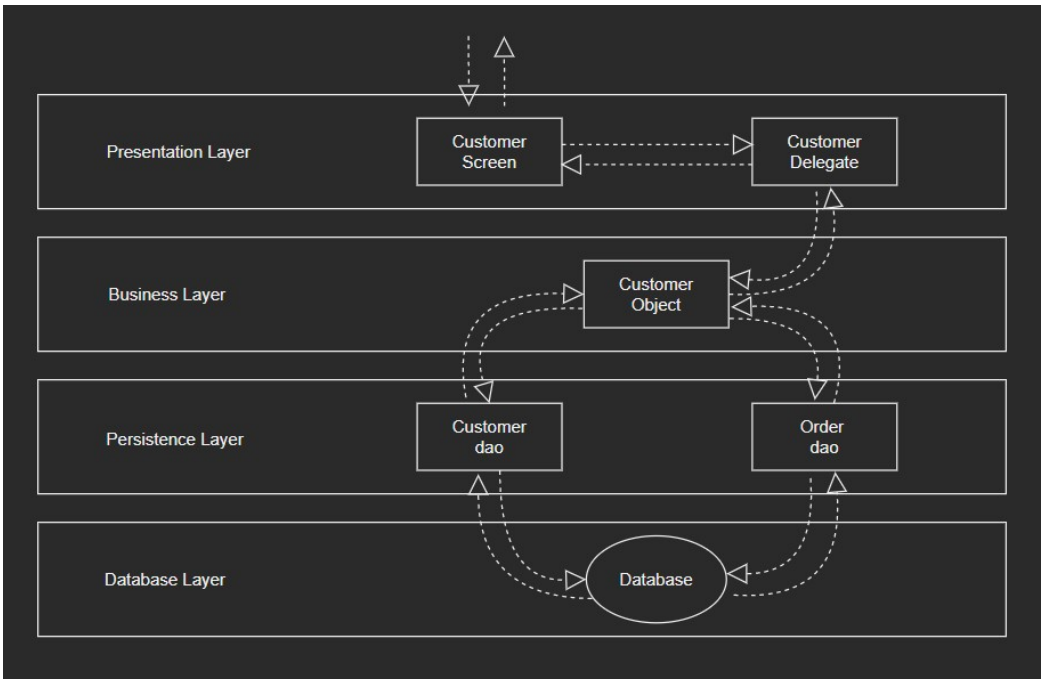
- Kopling antar lapisan cenderung membuatnya lebih sulit. Hal ini membuatnya sulit untuk diukur.

- Harus digunakan sebagai unit tunggal sehingga perubahan ke lapisan tertentu berarti seluruh sistem harus dipekerjakan kembali.

- Semakin besar, semakin banyak sumber daya yang dibutuhkan untuk permintaan untuk melewati beberapa lapisan dan dengan demikian akan menyebabkan masalah kinerja.

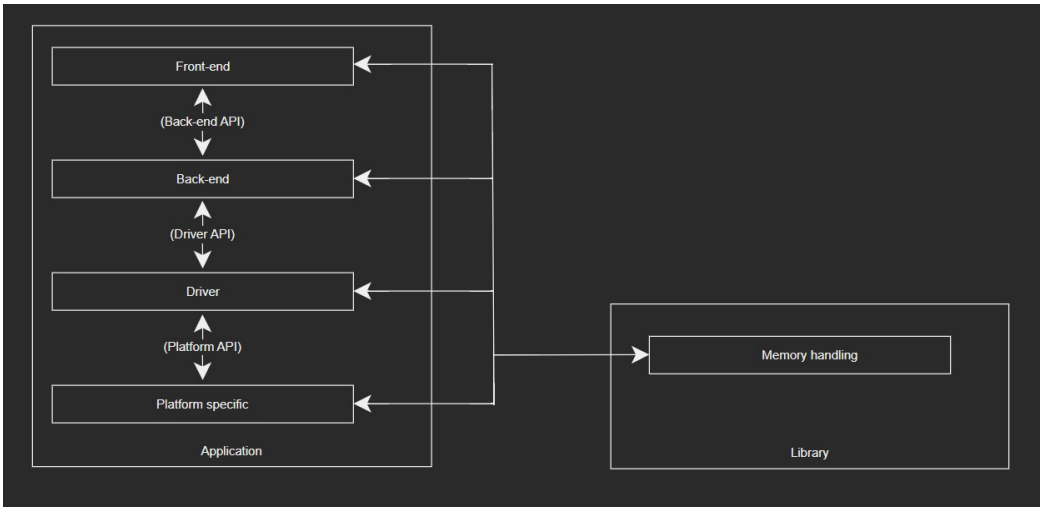
5.4 Software Architechture Pattern

Ini adalah pola arsitektur paling umum di sebagian besar aplikasi tingkat perusahaan. Ini juga dikenal sebagai pola n-tier, dengan asumsi n jumlah tingkatan. Contoh Skenario:



5.5 Design Patterns

Anggap mock-up software design, susunan "stack" nya seperti layered architecture:

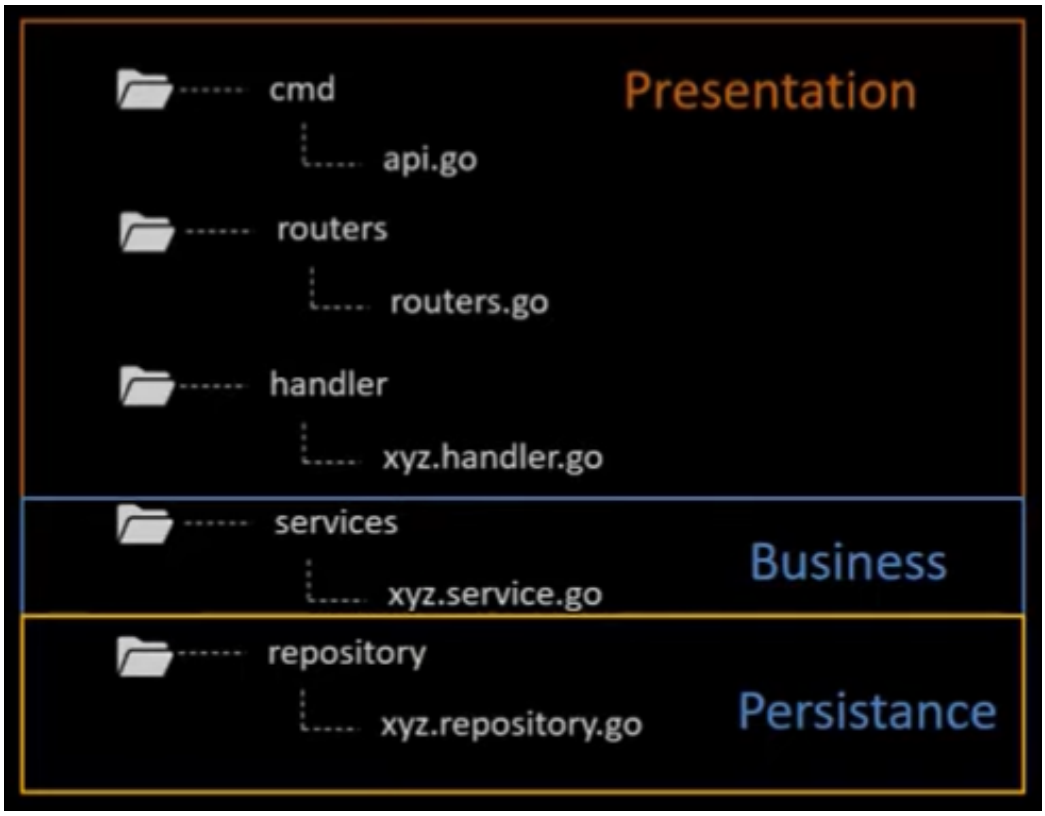


Setiap layer dari aplikasi terpisah dengan cara penggunaan metode API, namun yang masih saling berhubungan adalah memory handling , karena setiap komunikasi layer akan membawa/mengirim data sehingga akan terjadi alokasi memory dan pada akhirnya membutuhkan memory handling.

Ada 4 bagian dari layered architecture yang di mana setiap layer memiliki hubungan antara komponen yang ada di dalamnya dari atas ke bawah yaitu:

- **The presentation layer:** Semua bagian yang berhubungan dengan layer presentasi.
- **The business layer:** Berhubungan dengan logika bisnis.
- **The persistence layer:** Berguna untuk mengurusi semua fungsi yang berhubungan dengan objek relasional
- **The database layer:** Tempat penyimpanan semua data layer.

5.5.1 Contoh penerapan layered architecture:



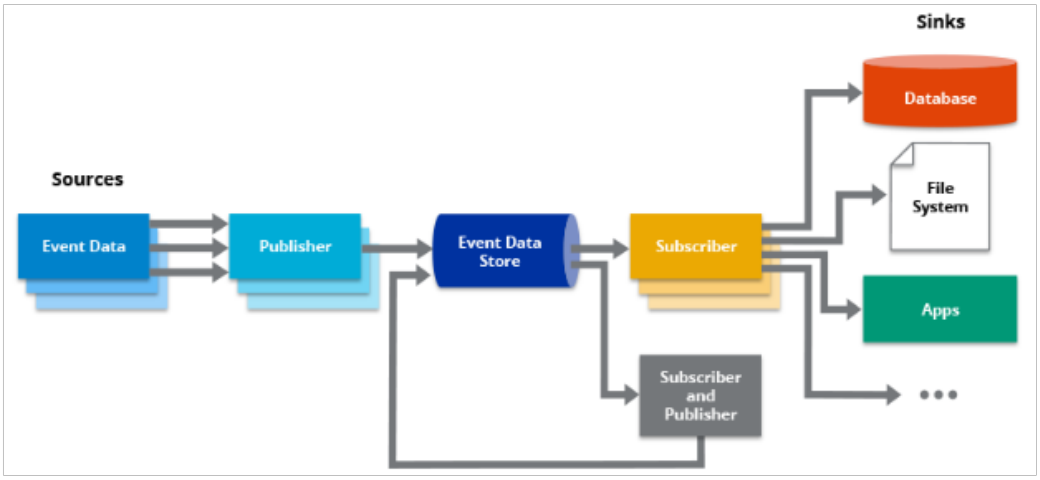
Bab 6

Event-Driven Architecture

Delvin, Gabrielle Sheila Sylvagno, Danica Recca Danendra

6.1 Event-Driven Architecture

6.1.1 Event-Driven Architecture



Gambar 6.1: Skema Diagram EDA

Event-driven architecture (EDA) atau arsitektur berbasis peristiwa adalah paradigma desain perangkat lunak yang memanfaatkan peristiwa *(event)* sebagai dasar interaksi dan integrasi antara komponen-komponen perangkat lunak.EDA berfokus pada peristiwa yang terjadi pada waktu tertentu, seperti

permintaan pengguna atau respons sistem terhadap permintaan tersebut. Komponen-komponen perangkat lunak dalam arsitektur ini saling berkomunikasi melalui peristiwa-peristiwa yang terjadi, sehingga memungkinkan sistem untuk beroperasi secara asinkron.EDA sering digunakan dalam pengembangan aplikasi skala besar dan sistem berbasis layanan *(service-oriented architecture/SOA)* untuk memastikan penggunaan sumber daya yang efektif dan efisien. Arsitektur ini juga dapat membantu meminimalkan waktu respon dan meningkatkan skalabilitas sistem.Berikut ini diagram EDA yang ditampilkan pada 6.1

6.2 Kelebihan dan Kekurangan

6.2.1 Kelebihan

Berikut ini adalah kelebihan dari EDA:

- Asinkron: EDA memungkinkan komponen sistem beroperasi secara asinkron, yaitu mereka dapat beroperasi secara independen tanpa harus menunggu komponen lainnya untuk menyelesaikan tugasnya.
- Pemicu: EDA didasarkan pada penggunaan peristiwa sebagai pemicu untuk memicu tindakan atau respons. Ketika peristiwa terjadi, EDA akan memicu tindakan yang sesuai dengan peristiwa tersebut.
- Publikasi dan Langganan: EDA menggunakan model publikasi-langganan *(publish-subscribe)* dimana sebuah komponen menghasilkan peristiwa *(publisher)* dan komponen lainnya yang tertarik *(subscriber)* dapat menerima dan menangani peristiwa tersebut.
- Terdistribusi: EDA memungkinkan komponen sistem tersebar di berbagai mesin atau jaringan, sehingga memudahkan pengembangan sistem yang *scalable* dan tahan bencana.
- Flesksibel dan modular: EDA memisahkan komponen-komponen sistem sehingga mereka dapat beroperasi secara independen dan dapat digunakan kembali dalam berbagai aplikasi atau sistem yang berbeda.
- Responsif: EDA memungkinkan sistem merespons permintaan dengan cepat, karena komponen sistem dapat beroperasi secara independen dan merespons peristiwa secara asinkron.
- Berorientasi pada pesan: EDA menggunakan pesan sebagai sarana untuk berkomunikasi antar komponen sistem. Pesan dapat mengandung data atau informasi yang diperlukan oleh komponen lain dalam sistem.

- Skalabel: EDA dapat diimplementasikan pada sistem yang memiliki tingkat skala dan kompleksitas yang berbeda-beda, mulai dari sistem skala kecil hingga sistem skala besar dan terdistribusi.

6.2.2 Kekurangan

Berikut ini adalah kekurangan dari EDA:

- Kompleksitas: EDA bisa menjadi sangat kompleks karena banyaknya komponen dan interaksi antar komponen dalam sistem. Hal ini dapat membuat pengembangan dan pemeliharaan sistem menjadi lebih sulit.
- Kesulitan dalam pemantauan dan manajemen: Dalam EDA, setiap peristiwa dapat dicatat dan dilacak, namun hal ini bisa menyebabkan sulitnya pemantauan dan manajemen sistem jika terdapat banyak peristiwa yang terjadi pada waktu yang sama.
- Kemungkinan kesalahan: Karena EDA melibatkan banyak komponen yang berinteraksi satu sama lain, maka kemungkinan terjadinya kesalahan atau bug dalam sistem juga semakin besar. Hal ini dapat menyebabkan kerusakan sistem atau bahkan kegagalan total dalam sistem. /item Tidak cocok untuk sistem yang simpel: EDA biasanya digunakan pada sistem yang kompleks dan memerlukan integrasi dengan berbagai sistem atau aplikasi lainnya. Sehingga EDA mungkin tidak cocok untuk sistem yang simpel atau terbatas dalam kompleksitasnya.

6.3 Contoh Penerapan

6.3.1 Perbankan

Sistem perbankan: EDA dapat digunakan untuk membangun sistem perbankan yang responsif dan skalabel. Contohnya adalah ketika seorang pelanggan melakukan transfer uang, hal ini memicu peristiwa (event) yang kemudian membuat sistem mengirimkan notifikasi kepada penerima transfer bahwa uang telah diterima.

6.3.2 E-commerce

Aplikasi e-commerce: EDA dapat digunakan dalam aplikasi e-commerce untuk mempercepat proses pembelian. Ketika seorang pelanggan menyelesaikan pembelian, peristiwa ini dapat memicu sistem untuk mengirim notifikasi ke bagian pengiriman dan bagian keuangan untuk memproses pesanan.

6.3.3 Internet of Thing (IoT)

Internet of Things (IoT): EDA juga dapat digunakan dalam sistem IoT, di mana banyak sensor dan perangkat harus berinteraksi dengan sistem pusat. Contohnya adalah ketika suhu di suatu ruangan melebihi batas normal, peristiwa ini dapat memicu sistem untuk mengirim notifikasi ke teknisi untuk memperbaiki perangkat pendingin ruangan.

6.3.4 Manajemen Rantai Pasokan

Sistem manajemen rantai pasokan: EDA dapat digunakan dalam sistem manajemen rantai pasokan untuk memantau pergerakan barang dari satu titik ke titik lainnya. Ketika sebuah produk telah dikirim, peristiwa ini dapat memicu sistem untuk mengirim notifikasi ke penerima produk tentang waktu pengiriman yang dijadwalkan.

6.3.5 Manajemen Proyek

Sistem manajemen proyek: EDA dapat digunakan dalam sistem manajemen proyek untuk memantau perkembangan proyek dan memperingatkan manajer proyek ketika terjadi masalah atau penundaan. Contohnya, ketika seorang anggota tim menyelesaikan tugas mereka, peristiwa ini dapat memicu sistem untuk memperbarui proyek secara otomatis dan memberikan notifikasi kepada manajer proyek.

Bab 7

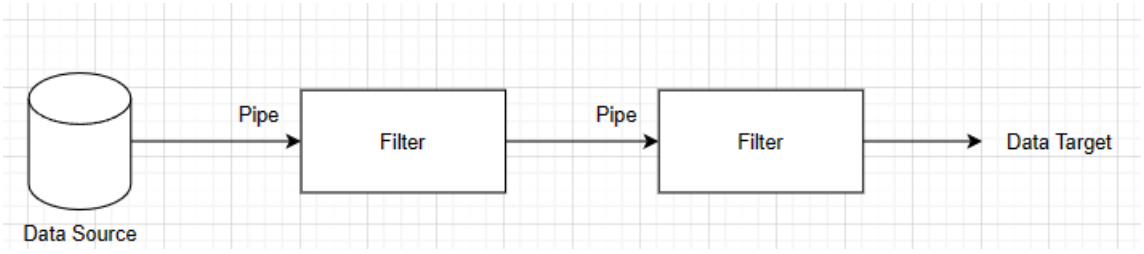
Pendahuluan

Alfa Yohannis, Rizki Wahyudi, Tommy Chitiawan, Mandalan

7.1 Definisi

Pipa dan Filter adalah pola arsitektur lain, yang memiliki entitas independen yang disebut filter (komponen) yang melakukan transformasi pada data dan memproses masukan yang mereka terima, dan pipa , yang berfungsi sebagai penghubung aliran data yang sedang diubah, masing-masing terhubung ke komponen berikutnya di dalam pipa.

7.2 Pipe and Filter Architecture Schema



Seperti yang Anda lihat pada diagram, data mengalir dalam satu arah. Ini dimulai dari sumber data, tiba di port input filter tempat pemrosesan dilakukan pada komponen, dan kemudian, diteruskan melalui port outputnya melalui pipa ke filter berikutnya, dan akhirnya berakhir di sasaran data.

7.3 Kelebihan

- Memastikan sambungan komponen, filter yang longgar dan fleksibel.
- Kopling longgar memungkinkan filter diubah tanpa modifikasi ke filter lain.
- Konduktif untuk pemrosesan paralel.
- Filter dapat diperlakukan sebagai kotak hitam. Pengguna sistem tidak perlu mengetahui logika di balik kerja setiap filter.
- Dapat digunakan kembali. Setiap filter dapat dipanggil dan digunakan berulang kali.

.

7.4 Kekurangan

- Penambahan sejumlah besar filter independen dapat mengurangi kinerja karena overhead komputasi yang berlebihan.
- Bukan pilihan yang baik untuk sistem interaktif.
- Sistem pipa-dan-pemasang mungkin tidak cocok untuk perhitungan jangka panjang.

7.5 penerapan dalam aplikasi

- Sistem pengolahan data: Pipe and filter dapat digunakan untuk mengambil data dari berbagai sumber dan memprosesnya melalui serangkaian filter untuk menghasilkan output yang diinginkan.
- Sistem pengolahan gambar: Pipe and filter dapat digunakan untuk memproses gambar atau video yang diambil dari kamera dengan menggunakan berbagai filter untuk menghasilkan gambar yang lebih baik atau memberikan efek khusus.
- Sistem pencarian: Pipe and filter dapat digunakan untuk memproses data pencarian yang diberikan oleh pengguna dan memfilter data untuk menghasilkan hasil pencarian yang relevan.

- Sistem pemrosesan audio: Pipe and filter dapat digunakan untuk memproses audio dan melakukan pengolahan suara seperti pengurangan kebisingan, pengaturan volume, dan pemotongan audio.
- Sistem pemrosesan teks: Pipe and filter dapat digunakan untuk memproses teks dan melakukan pengolahan bahasa alami seperti analisis sentimen dan pengenalan entitas. content...

Bab 8

Pendahuluan

Alfa Yohannis, Charlie Chaplin

8.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps

Bab 9

Pendahuluan

Alfa Yohannis, Charlie Chaplin

9.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps

Bab 10

Pendahuluan

Alfa Yohannis, Charlie Chaplin

10.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps

Bab 11

Orchestration-driven Service-oriented Architecture

Hansel Ricardo, Jonathan Erik Maruli Tua, Yefta Tanuwijaya

11.1 Definisi

Orchestration-driven Service-oriented Architecture (ODSOA) adalah suatu pendekatan arsitektur perangkat lunak yang bertujuan untuk memfasilitasi pengembangan dan integrasi sistem yang kompleks dengan cara menggunakan layanan (services) yang terdistribusi dan terpisah secara fisik namun saling terkait secara fungsional.

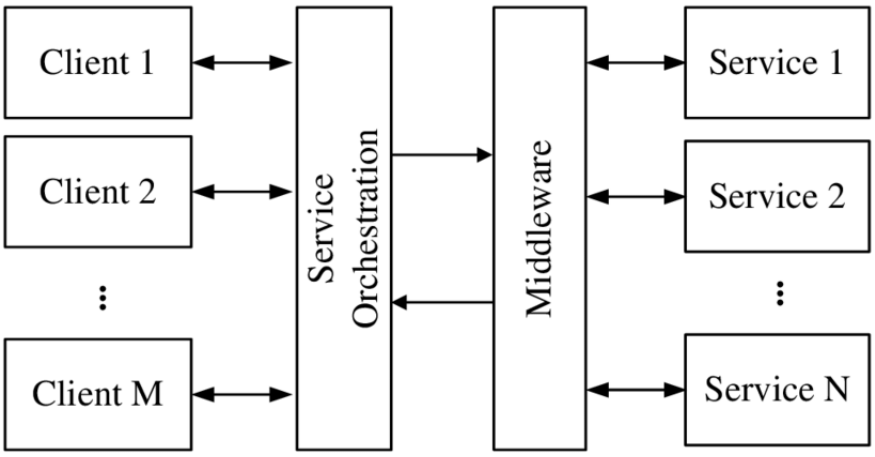
ODSOA menempatkan Orkestrasi (Orchestration) sebagai elemen kunci untuk mengelola interaksi antara layanan. Orkestrasi dapat didefinisikan sebagai proses otomatis yang mengkoordinasikan dan mengatur eksekusi layanan secara teratur untuk mencapai tujuan bisnis tertentu.

Dalam ODSOA, layanan disediakan sebagai fungsi-fungsi modular yang dapat digunakan oleh aplikasi dan sistem lain untuk memperoleh fungsionalitas tambahan. Layanan ini biasanya disediakan secara independen oleh unit bisnis atau departemen yang berbeda dan dapat diakses melalui jaringan.

ODSOA memiliki beberapa keuntungan, antara lain: skalabilitas, fleksibilitas, dan interoperabilitas. Skalabilitas memungkinkan sistem untuk diukur dan meningkatkan kapasitasnya dengan mudah. Fleksibilitas memungkinkan pengguna untuk menyesuaikan layanan sesuai kebutuhan mereka tanpa harus mengubah keseluruhan arsitektur. Interoperabilitas memungkinkan sistem untuk berinteraksi dengan sistem lain yang menggunakan standar yang sama.

Secara keseluruhan, ODSOA dapat membantu perusahaan dalam mempercepat pengembangan dan integrasi aplikasi serta meningkatkan efisiensi dan efektivitas bisnis secara keseluruhan.

11.2 Orchestration-driven Service-oriented Architecture Schema



Orchestration-driven Service-oriented Architecture (ODSOA) Schema adalah suatu model yang menggambarkan arsitektur ODSOA secara visual, yang mencakup komponen-komponen utama dan interaksi antara mereka. Beberapa komponen utama dalam schema ODSOA antara lain:

- Layanan (Services): Komponen inti dari ODSOA adalah layanan, yang merupakan unit fungsional yang terdistribusi secara terpisah namun saling terkait secara fungsional. Layanan ini dapat digunakan oleh aplikasi dan sistem lain untuk memperoleh fungsionalitas tambahan.
- Orkestrasi (Orchestration): Orkestrasi merupakan proses otomatis yang mengkoordinasikan dan mengatur eksekusi layanan secara teratur untuk mencapai tujuan bisnis tertentu. Orkestrasi dapat mengatur urutan dan kondisi yang harus dipenuhi oleh layanan.
- Bus Layanan (Service Bus): Bus Layanan adalah infrastruktur yang memfasilitasi komunikasi antara layanan dalam arsitektur ODSOA. Bus Layanan dapat mengatur dan mengarahkan permintaan dan respons antara layanan.
- Repositori Layanan (Service Repository): Repositori Layanan adalah tempat untuk menyimpan informasi terkait dengan layanan yang terse-

dia, seperti deskripsi, spesifikasi teknis, dan interdependensi antara layanan. Repositori Layanan memungkinkan pengguna untuk mencari dan menemukan layanan yang dibutuhkan.

- Klien (Client): Klien adalah aplikasi atau sistem yang menggunakan layanan untuk memperoleh fungsionalitas tambahan. Klien mengirim permintaan ke layanan dan menerima respons dari layanan.

- Penyedia Layanan (Service Provider): Penyedia Layanan adalah unit bisnis atau departemen yang menyediakan layanan untuk digunakan oleh aplikasi dan sistem lain. Penyedia Layanan bertanggung jawab untuk mengembangkan dan menjaga layanan yang disediakan.

Dalam ODSOA Schema, interaksi antara komponen-komponen tersebut direpresentasikan dengan panah yang menghubungkan mereka. Misalnya, panah dari klien ke layanan menunjukkan bahwa klien menggunakan layanan tersebut, sedangkan panah dari layanan ke bus layanan menunjukkan bahwa layanan terdaftar dalam infrastruktur bus layanan. Dengan ODSOA Schema, pengguna dapat dengan mudah memahami arsitektur ODSOA secara visual dan mengidentifikasi komponen-komponen utama dan interaksi antara mereka.

11.3 Kelebihan

- Skalabilitas: ODSOA memungkinkan sistem untuk diukur dan meningkatkan kapasitasnya dengan mudah. Layanan dapat dikonfigurasi ulang atau ditambahkan ke infrastruktur dengan mudah, tanpa mempengaruhi sistem keseluruhan. Hal ini memudahkan perusahaan untuk menyesuaikan sistem mereka dengan perubahan kebutuhan bisnis.

- Fleksibilitas: ODSOA memungkinkan pengguna untuk menyesuaikan layanan sesuai kebutuhan mereka tanpa harus mengubah keseluruhan arsitektur. Dengan demikian, perusahaan dapat dengan mudah memodifikasi fungsionalitas sistem dan mengintegrasikan solusi baru tanpa mempengaruhi sistem keseluruhan.

- Interoperabilitas: ODSOA memungkinkan sistem untuk berinteraksi dengan sistem lain yang menggunakan standar yang sama. Hal ini memungkinkan perusahaan untuk berintegrasi dengan sistem lain dengan mudah dan memperluas fungsionalitas sistem mereka.

- Reusabilitas: Layanan dalam ODSOA adalah modular dan dapat digunakan kembali oleh aplikasi dan sistem lain. Hal ini memungkinkan perusahaan untuk mengembangkan sistem dengan cepat dan efisien.
- Pemisahan Tugas: ODSOA memisahkan tugas-tugas sistem menjadi layanan yang terpisah secara fisik namun saling terkait secara fungsional. Hal ini memudahkan manajemen sistem dan memungkinkan perusahaan untuk mengoptimalkan penggunaan sumber daya.

11.4 Kekurangan

- Kompleksitas: Arsitektur ODSOA dapat menjadi sangat kompleks, terutama ketika menangani banyak layanan yang berbeda dan memerlukan integrasi yang kompleks. Oleh karena itu, perusahaan memerlukan tingkat keahlian teknis yang tinggi untuk mengimplementasikan dan mengelola arsitektur ini dengan efektif.
- Keamanan: Arsitektur ODSOA dapat menimbulkan masalah keamanan karena penggunaannya yang melibatkan layanan dari banyak sistem dan vendor. Oleh karena itu, perusahaan harus memperhatikan masalah keamanan yang terkait dengan integrasi dan melakukan tindakan yang tepat untuk mengurangi risiko keamanan.
- Pengelolaan versi: Dalam ODSOA, perubahan pada satu layanan dapat mempengaruhi layanan lainnya. Oleh karena itu, pengelolaan versi menjadi penting untuk memastikan bahwa perubahan yang dibuat pada layanan tidak mengganggu kinerja sistem secara keseluruhan.
- Biaya: Implementasi arsitektur ODSOA memerlukan biaya yang tinggi karena melibatkan pengembangan, integrasi, dan manajemen layanan yang kompleks. Oleh karena itu, perusahaan harus mempertimbangkan biaya ini sebelum mengimplementasikan arsitektur ini.
- Ketergantungan terhadap vendor: Terkadang perusahaan tergantung pada vendor tertentu untuk memasok layanan tertentu. Jika vendor tersebut menghentikan layanannya, maka perusahaan perlu mencari alternatif layanan dari vendor lain atau bahkan harus mengubah arsitektur sistem secara keseluruhan.

11.5 Penerapan dalam Aplikasi

Berikut adalah contoh penerapannya dalam sebuah aplikasi

Bab 12

Pendahuluan

Alfa Yohannis, Charlie Chaplin

12.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps

Bab 13

Pendahuluan

Alfa Yohannis, Charlie Chaplin

13.1 Materi

1. Introduction
2. Client-Server Architecture
3. Monolith vs. Distributed Architecture
4. Model-View-Controller Architecture
5. Layered Architecture
6. Event-Driven Architecture
7. Pipeline / Pipe-and-Filter Architecture
8. Service-based (Serverless) Architecture
9. Microkernel Architecture
10. Space-based Architecture
11. Orchestration-driven Service-oriented Architecture
12. Microservices Architecture
13. Containers
14. DevOps

Bab 14

DevOps

Hendra Lijaya, Oktavianus Hendry Wijaya

14.1 Pengertian

DevOps merupakan metode pengembangan software dengan mengkolaborasikan *software developer* dengan *IT operation*.

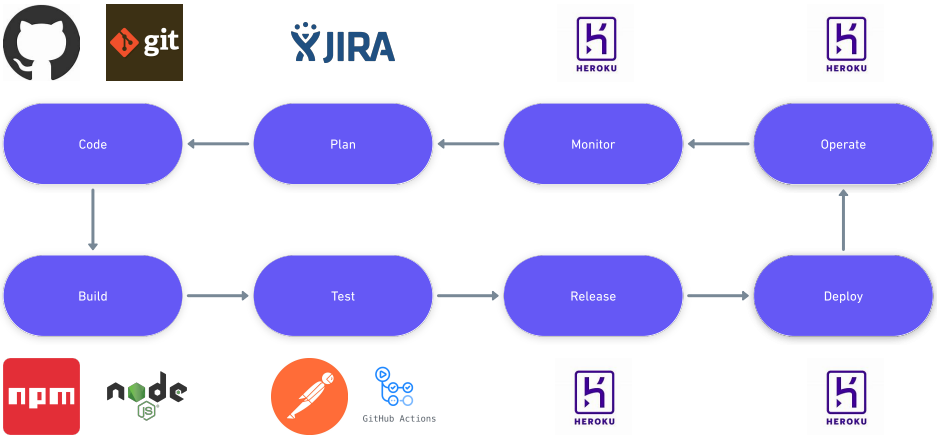
14.2 Fungsi

Tujuan akhir atau *goal* dari DevOps adalah untuk menciptakan lingkungan kolaborasi yang berkelanjutan untuk membawa software menjadi lebih berkualitas, lebih cepat, dan dapat diandalkan.

14.3 Arsitektur

- *Plan*
  Tahap paling awal dalam SDLC (*Software Development Life Cycle*). Mulai dari tahap pengumpulan data, membuat *roadmap*, menetapkan tujuan, *timelines* serta mengidentifikasi *resources* yang diperlukan untuk menyelesaikan sebuah proyek.

- *Code*
  Pada tahap ini, developer mulai menulis kode untuk mengembangkan *software* berdasarkan requirement yang telah dikumpulkan pada tahap *Plan*.



Gambar 14.1: Arsitektur DevOps.

- *Build*
  Pada tahap ini, kode di *compile*, dan di *package* kedalam format yang bisa di *deliver*. Tujuan tahap ini membuat kode yang telah di *compile* agar dapat melakukan tahap *Testing* dan *Release*.

- *Test*
  Pada tahap ini, perangkat lunak diuji untuk memastikan bahwa perangkat lunak sudah memenuhi requirement dan fungsinya sudah berjalan tanpa ada *bug*.

- *Release*
  Pada tahap ini, perangkat lunak sebelum di *deploy* ke *staging* atau *production environment* dapat dilakukan data migrasi, konfigurasi dan lainnya. Tujuannya adalah untuk memastikan bahwa perangkat lunak tersedia dan dapat digunakan oleh audiens yang dituju.

- *Deploy*
  Pada tahap ini, perangkat lunak di *deploy* ke *production* ataupun *staging* bisa menggunakan tools atau *automation script*. Proses ini mencakup juga instalasi library, dan konfigurasi server dan lainnya.

- *Operate*
  Pada tahap ini, perangkat lunak sudah beroperasi di *production environment* dan dapat dikelola dan dipantau untuk memastikan kinerjanya seperti yang diharapkan.

- *Monitor*
  Pada tahap ini, pengumpulan dan analisis data dari log sistem. Informasi ini dapat digunakan untuk mengidentifikasi area untuk perbaikan, mengoptimalkan kinerja, dan menginformasikan upaya pengembangan perangkat lunak kedepannya.

14.4 Kelebihan Kekurangan

Berikut adalah kelebihan dan kekurangan arsitektur DevOps:

14.4.1 Kelebihan

Keuntungan dari menerapkan arsitektur DevOps adalah:

- DevOps menjadi pilihan yang bagus untuk *development* dan *deployment* aplikasi yang cepat

- Merespon lebih cepat ke perubahan market untuk meningkatkan *business growth* (pertumbuhan bisnis)

- Data dapat disentralisasikan sehingga meningkatkan konsistensi data dan mengurangi duplikasi data.

- DevOps meningkatkan profit bisnis dengan mengurangi waktu *delivery software* dan biaya.

- DevOps menghilangkan proses deskriptif sehingga memberikan kejelasan mengenai *development* dan *delivery product*.

- Meningkatkan *experience* dan kepuasan *customer*.

- DevOps menyederhanakan kolaborasi dan menggunakan semua *tools* di cloud untuk diakses pengguna.

- Meningkatkan keterlibatan dan produktivitas tim.

14.4.2 Kekurangan

Kekurangan dari penerapan arsitektur DevOps adalah:

- DevOps *professional* atau *expert* masih belum umum ditemukan.
- *Developing* dengan DevOps mahal.
- Penerapan DevOps baru ke dalam industri sulit untuk dikelola dalam waktu singkat.
- Kurangnya pengetahuan mengenai DevOps dapat menyebabkan masalah pada *Continuous Integration* dari *project automation*.

14.5 Perbedaan DevOps dan nonDevOps

Perbedaan besar dari *development* dengan DevOps dan nonDevOps yaitu:

1. Kolaborasi
   Pada *Software development* nonDevOps, developer dan tim operation bekerja secara terpisah. Sedangkan pada DevOps, kedua hal tersebut bekerja secara kolaboratif dalam 2 tim yang berbeda dengan berbagi pengentahuan dan skill sehingga memastikan proses software development dapat disederhanakan.

2. *Continuous Integration and Delivery (CI/CD)*
   DevOps menerapkan CI/CD yang dimana melibatkan sistem *automation* pada proses *software development* mulai dari *building* dan *testing* hingga ke *deployment* dan *maintenance*. Dengan menerapkan ini, perubahan dapat di tes dan diintegrasikan ke software secara cepat dan efisien mungkin.

3. *Automation*
   DevOps bergantung secara penuh pada automation untuk meningkatkan efisiensi dan mengurangi error. Tools seperti *management configuration*, *continuous integration*, dan *continuous delivery* memungkinkan tim untuk mengautomatis proses yang awalnya manual dan memastikan konsistensi.

4. *Monitoring*
   Tim yang menerapkan DevOps menggunakan *tools* untuk *monitoring* dan analitik untuk mengumpulkan data mengenai performa pada software saat *production*. Hal ini membantu tim dalam mengidentifikasi

dan menyelesaikan isu dengan cepat sehingga mengurangi downtime dan meningkatkan *user experience* secara keseluruhan.

5. Agile Development
   DevOps berdasarkan pada prinsip *agile development* yang dimana fleksibilitas, adaptabilitas, dan kolaborasi sangat ditekankan. Tim DevOps memprioritaskan dalam *delivery* dalam perubahan kecil dan perubahan *incremental* dengan cepat dibandingkan perilisan monolitik yang bersifat besar.

Kesimpulannya adalah DevOps bersifat lebih kolaboratif, *automated*, dan *agile* pada proses *software development* yang menekankan *continuous integration and delivery, automation,* dan *monitoring*.

14.6 Tools

Tools yang digunakan dalam pembuatan DevOps:

1. Git - GitHub Action
   GitHub Action adalah fitur dari *platform* GitHub yang memungkinkan developer untuk mengautomasi *workflows* dan *build, test*, dan *deploy* kode langsung dari *platform* GitHub. GitHub Action menyediakan *library* dari *pre-built actions* yang dapat digunakan untuk membangun *workflows* dan juga kemampuan untuk membuat action kustom menggunakan JavaScripts atau Docker containers. *Workflows* dapat dipicu/di*trigger* oleh *events* seperti push kode, request pull, atau pembuatan perilisan baru.
   Keuntungan menggunakan GitHub:

   - Terintegrasi dengan GitHub
   - Workflows yang dapat dikustomisasi
   - Reusability
   - Kolaborasi
   - Skalabilitas
   - Gratis

   Kesimpulan, GitHub Action merupakan *tools* yang sangat berguna untuk *automating software development workflows*, menyediakan developer fleksibilitasm, kustomisasi, dan platform yang terintegrasi untuk *building, testing,* dan *deploy* kode

2. Heroku
   Heroku merupakan *platform cloud* yang memungkinkan developer untuk *build, deploy*, dan mengelola aplikasi secara cepat dan mudah. Heroku mendukung beberapa Bahasa pemrograman seperti Java, Ruby, Node.js, Python, PHP, dan Go. Heroku menyediakan platform yang dikelola secara penuh sehingga developer tidak perlu mengkhawatirkan mengenai mengelola infrastruktur, sistem operasi, dan server.
   Heroku didasarkan pada arsitektur yang berbasis *container* dan menggunakan Dyno untuk menjalankan aplikasi. Dyno merupakan *container* linux yang ringan dan terisolasi yang berjalan diatas platform Heroku. Dyno di*design* untuk menjalankan satu proses atau layanan yang membantu meningkatkan performa, skalabilitas, dan ketahanan.
   Fitur-fitur Heroku:

   - *Command Line Interface* (CLI)
   - *Web Based Dashboard*
   - Beragam *add-ons* dan *extensions*
   - *Support continuous integration and continuous delivery (CI/CD) workflows.*

   Adapun kekurangan dari Heroku yaitu:

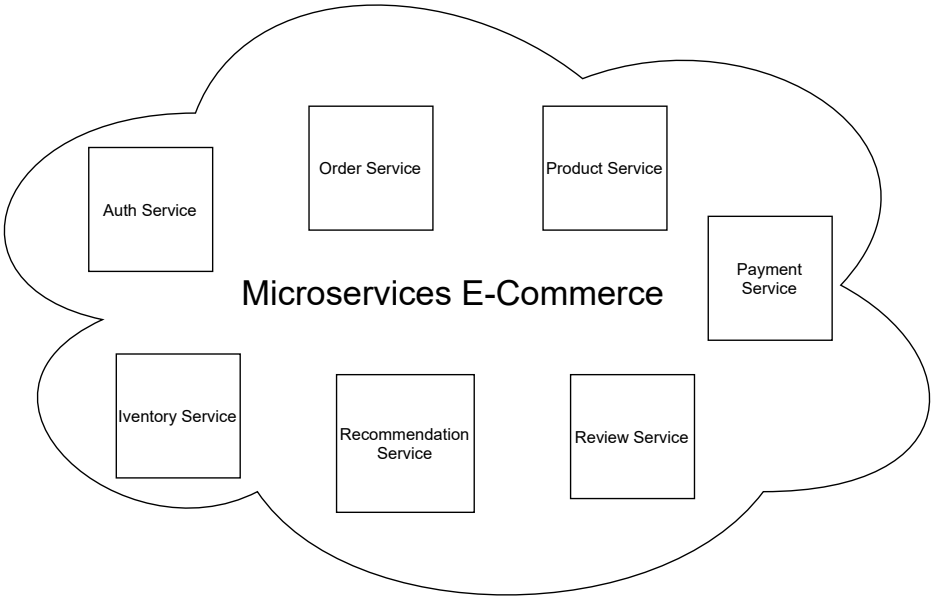
   - Kustomisasi yang terbatas.
   - Bergantung pada *add-on third party.*
   - Memerlukan biaya dan kartu kredit.

3. Postman
   Postman merupakan tools software yang sering digunakan oleh developer untuk *test*, dokumentasi, dan berbagi API. API atau *Application Programming Interfaces* memungkinkan software aplikasi yang berbeda untuk berkomunikasi melalui pertukaran data antara satu dengan yang lain.
   Dengan menggunakan Postman, developer dapat dengan mudah membuat dan mengeksekusi *HTTP requests*, yang memungkinkan mereka untuk mencoba API dan memastikan API berjalan dengan benar. Postman juga menyediakan berbagai fitur yang memudahkan pendokumentasian API, termasuk kemampuan untuk membuat dokumentasi API dan membuat *code snippets* dalam berbagai variasi bahasa pemrograman.
   Fitur utama pada Postman:

- *Collections*: Postman memungkinkan developers untuk mengorganisasikan *request* ke sebuah *collections*, yang dimana memudakan dalam *grouping request* berdasarkan fungsionalitas.
- *Environments*: Postman mendukung penggunaan *environments* yang memungkinkan developer untuk pendefinisian variabel dan *values* yang berbeda untuk *testing environments* yang berbeda (seperti *development, testing,* atau *production*).
- *Test automation*: Postman memungkinkan developer untuk mengautomatisasi *testing* dengan membuat *scripts* yang dapat dijalankan sebagai bagian dari *test*. Hal ini memudahkan dalam memastikan API berjalan secara benar dan memudahkan mencari isu diawal dalam proses *development*.
- *Collaboration*: Postman memudahkan dalam berkolaborasi dengan developer lain dengan memungkinkan berbagai *collection, environment,* dan *documentation* dengan yang lain.
- *Integrations*: Postman terintegrasi dengan bermacam-macam tools dan layanan lain, seperti GitHub, Jira, Slack, yang memudahkan dalam memasukkan Postman kedalam *workflows* yang sudah ada.

14.7 Contoh Kasus

Contoh Kasus Aplikasi *E-Commerce* Biasanya pada aplikasi *e-commerce* yang menggunakan arsitektur *microservice*. Beberapa *service* pada *microservice* yaitu *Authentication Service, product catalog service, order management service, payment service, and shipping service,* yang masing-masing bertanggung jawab untuk fungsi tertentu. Dengan menggunakan DevOps dapat memberikan beberapa kemudahan pada developer dengan beberapa fitur.
Fitur utama DevOps:
CI/CD Pipeline digunakan untuk melakukan *compile code*, *unit testing*, *integration testing*, *packaging*, *deployment* dan *monitoring*. Setiap kali developer melakukan *push* pada *repository* git, *CI/CD Pipeline* otomatis melakukan *compile code*, menjalankan *unit tests*, *deploy* perubahan pada *staging environment* untuk *integration testing*. Jika *integration test* dilewati, maka akan di *deploy* di *production environment*.
Menggunakan DevOps dapat membuat developer fokus pada fitur aplikasi yang ingin dibangun sehingga tidak terlalu lama memikirkan setup infrastruktur untuk *testing* dan lainnya.



Gambar 14.2: Studi Kasus *E-Commerce Microservice*.

14.8 Code

Code dapat dilihat di link berikut

Bibliography