

Bahasa Pemrograman

Silver - Chapter 0 - Topic 5

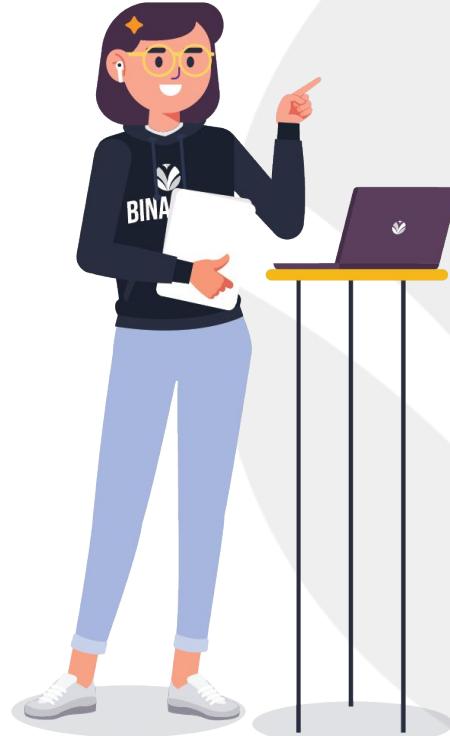
**Selamat datang di Chapter 0 Topic 5
online course dari
Binar Academy!**



Hai teman-teman! 😊

Di topic sebelumnya, kamu udah belajar mengenai sejarah singkat aplikasi dan apa sih aplikasi itu. Di sini, kamu bakal belajar lebih dalam lagi tentang **isi aplikasi dan siapa aja orang-orang di balik layarnya**.

Kalau gitu, yuk langsung aja kita kepoin~



Apa yang bakal kamu pelajari di Topic 5 ini?

Pada kesempatan kali ini, kamu akan mengenal lebih dalam mengenai:

- 1) Bedain bahasa pemrograman, framework, dan library
- 2) Cara nentuin tech stack yang cocok buat produkmu
- 3) Perbandingan tech stack di client-side dan server-side
- 4) Tech stack di database



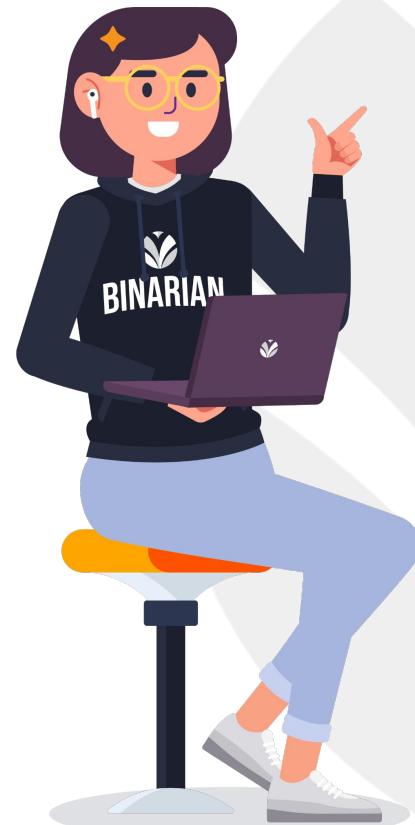
Kotlin adalah salah satu bahasa pemrograman dalam dunia Android app development

Selain dikenal stabil, **popularitasnya makin melambung sehabis Google ngumumin Kotlin jadi bahasa pemrograman resmi buat bikin aplikasi Android di event Google I/O 2017.**



Nah, jangan kejebak sama Android Studio. Android Studio itu bukan bahasa pemrograman, tapi salah satu tools yang termasuk kategori integrated development environment (IDE).

React Native bisa dipakai buat bikin aplikasi Android. Tapi ingat, React Native itu termasuk framework, bukan bahasa pemrograman.



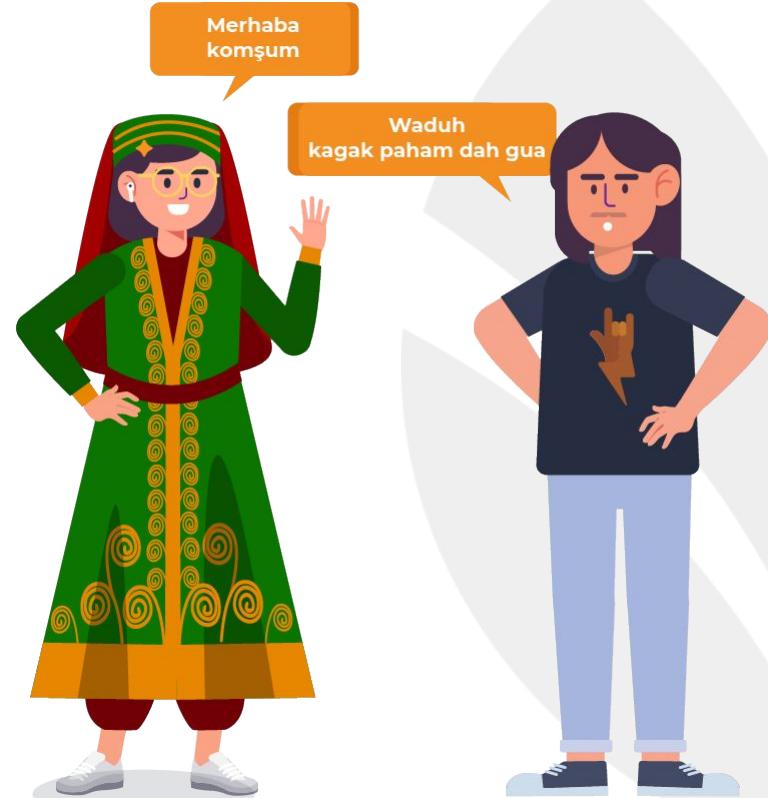
Ngomong-ngomong soal framework nih, kalau framework bukan bahasa pemrograman, trus framework itu apa?

Pertama, bahasa pemrograman itu kayak bahasa di dunia nyata. Di dalamnya ada serangkaian elemen kayak daftar istilah (vocabulary), struktur (grammar), dan interpretasi.



Bedanya, kalau kamu ngomong bahasa Inggris dengan grammar acak-acakan, selama lawan bicaramu bisa nangkep maksudmu, jadinya ga ada masalah.

Tapi **di bahasa pemrograman, grammar (alias syntax) harus 100% sesuai aturan biar komputer bisa menginterpretasi apa yang kita maksudkan.**



WHOPPER.



BIG MAC



Kita main analogi lagi ya. Kamu suka makan burger?

Dua franchise burger yang terkenal, yaitu Burger King dan McDonald's. Mereka berdua sama-sama franchise yang jual burger, tapi punya vocabulary yang beda.

Contoh: Di Burger King ada Whopper, di McDonald's ada Big Mac.

Resep yang dipakai buat bikin Whopper dan Big Mac kurang lebih sama. Tapi, mereka berdua punya cara masaknya sendiri-sendiri. Cara masak yang dipakai Burger King dan McDonald's itulah yang membedakan rasa dan hasilnya.

Nah, dalam kasus ini, **resep** disebut **library**, dan **cara masak** disebut **framework**.



Tadi sempat disinggung kalau banyak yang bilang framework itu kumpulan library. Nah, ini gak sepenuhnya benar, tapi gak sepenuhnya salah.

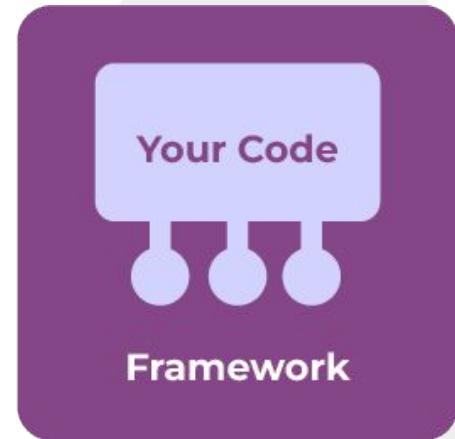
Library itu sekelompok kode yang ditulis programmer yang bisa dipakai sebagai source buat programmer lain. Tujuannya biar programmer lain gausah ribet ngoding dari nol lagi.



Sedangkan kalau framework itu lebih ke "cara penulisan" yang gak cuma terdiri dari beberapa kelompok kode, tapi buat keseluruhan aplikasi.

Tujuannya buat menyelesaikan masalah struktural dan arsitektural pas ngoding.

Sama kayak kasus burger tadi, dalam panduan cara masak Burger King dan McDonald's pasti dijelasin juga isi resep-nya. Tapi belum tentu kumpulan resep bisa jadi panduan cara masak, kan?



Beda bahasa pemrograman pasti beda juga frameworknya.

Dan umumnya, satu bahasa punya banyak framework, tergantung tujuannya.

Balik ke burger, Burger King dan McDonald's gak cuma jual burger, tapi juga ada menu lain kayak minuman, kentang goreng, dan lain-lain yang pastinya juga butuh framework yang beda tiap menunya.



Simpelnya, yang "paling gampang" dibikin itu library, makanya jumlahnya udah pasti paling banyak. Kalau yang paling susah, jelas bahasa pemrograman karena butuh bikin struktur yang kompleksnya gak ketulungan, makanya jumlahnya paling sedikit di antara 3 pilihan tadi.

Tambahan nih sob, nggak semua produk harus pakai framework. Anggap aja kalau ada yang mau beli seledri di Burger King, kan bisa langsung dijual tanpa harus ada cara masak khusus. Hehe.



Eh tapi perlu diingat sob, bahasa pemrograman gak semua diciptakan khusus buat client-side atau server-side. Beberapa bahasa server-side bisa dipakai buat client-side development juga lho, tentu dengan tambahan add-on atau pakai framework tertentu.

Tapi pengetahuan dasar tentang bahasa yang populer di client dan server side ini penting buat ngukur kelebihan dan kekurangan masing-masing tech stack.

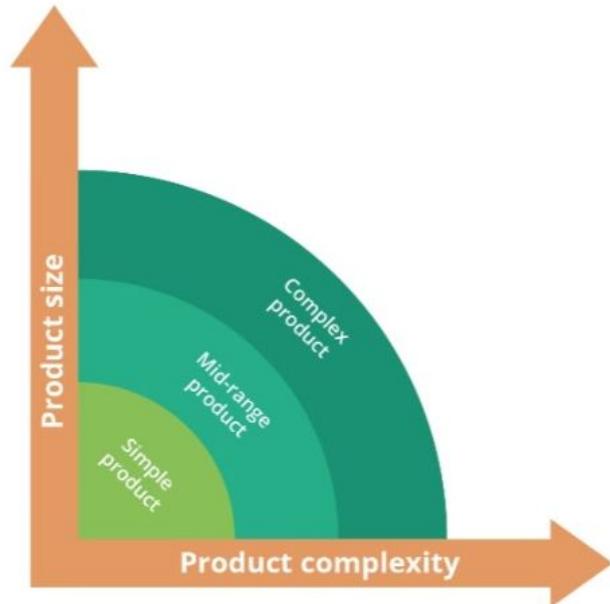


**Nah, buat nentuin enaknya
produkmu pakai tech stack apa, ada
caranya nih**

Biar gampang, coba kita timbang-timbang pakai 5 kriteria:

- Type of product
- Time to Market (TTM)
- Development cost
- Security
- Scalability



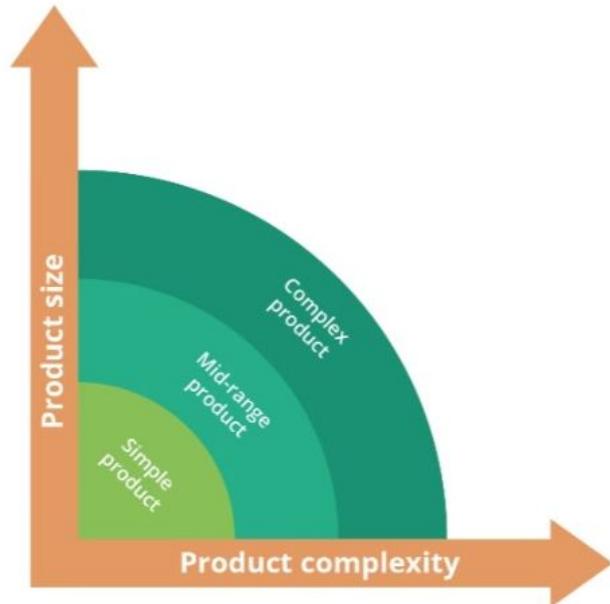


- **Type of product**

Di sini, kamu harus tahu gambaran web app yang mau kamu bikin nantinya bakal kayak gimana. Nah, biar gampang, ada kategorisasi sederhana:

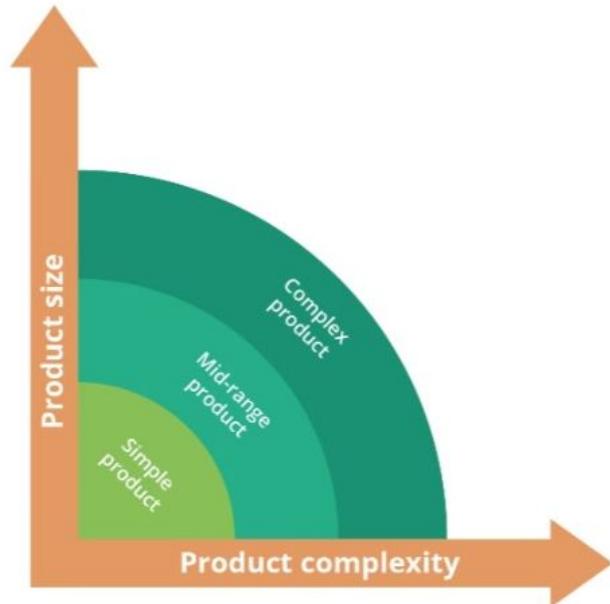
- a) **Simple product**

Kalau spesifikasi produkmu gak ribet dan harus jadi cepet macem bikin blog pribadi, landing page, atau katalog online shop; kamu bisa pakai CMS atau Wordpress.



b) Mid-range product

Kalau yang ini biasanya pembuatan web yang mengharuskan kita buat pakai framework biar bisa diintegrasikan antar platform. Contohnya kayak bikin web e-commerce atau web perusahaan. Nah, kalau mulai susah gini, Java bisa jadi opsi.



b) Complex product

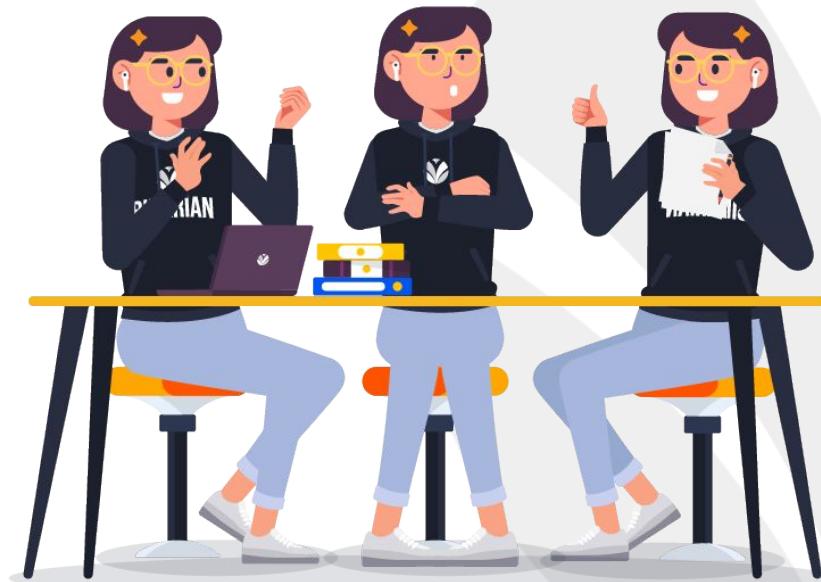
Kamu mau bikin super app yang terintegrasi ke banyak app lain? Kalau gitu masuk ke kategori ini. Yang jelas, kamu bakal butuh kombinasi dari beberapa bahasa dan teknologi sekaligus. Contohnya kayak bikin Facebook atau Google.

Tapi, tipe di atas baru dari sisi produk aja dan belum dari sisi bisnis

Padahal, pertimbangan bisnis juga penting buat nentuin tech stack apa yang kita pilih.

Sebagai calon engineer, kamu harus bisa ngasih pertimbangan teknis ke tim bisnis: **Mau bikin mobile app apa web duluan?**

Tambahan: Buat MVP, sebaiknya dirilis di 1 platform dulu sambil gali feedback dari user.



Mobile first

Mending bikin responsive web app yang fit ke semua ukuran dan dimensi. UI-nya simple dan diprioritaskan ukuran mobile dulu dibanding laptop. **Kelebihannya selain murah, juga lebih gampang iterasinya**, karena fleksibel jadi bisa dijadikan tool buat validasi produkmu.

Mobil first cocok buat produk yang kontennya ditargetkan buat segmen umum, yang link kontennya sering/gampang dishare, atau kontennya bisa sekalian dipakai buat SEO.

Contoh: Vimeo, Scribd, Kumparan.



Mobile only

Nah, resiko mobile only adalah **harus bikin native mobile app, makanya perilisannya pun harus di Play/App Store**. Karena langsung didownload, user punya ekspektasi lebih soal UI dan stabilitasnya.

Mobile only cocok buat produk yang bakal akses fitur device kayak kamera atau GPS. Enaknya, experience user bisa didongkrak lewat push notifications atau micro purchase.

Contoh: Instagram, GO-JEK, Subway Surfer.



Typeform

Mobile later

Gak semuanya butuh produk berbasis mobile buat MVP. Ada sebagian yang prefer ke web tradisional. Kenapa? **Karena UI-nya kompleks dan banyak fitur upload file (excel atau jpg size besar), usernya banyak berinteraksi sama produk kita lewat laptop/PC.**

Mobile later cocok buat SaaS yang targetnya perusahaan. Jatuhnya, kalau ada app-nya pun, cuma buat suplemen.

Contoh: Typeform, MailChimp, Hootsuite.





Sambil cerita soal Netflix ya

Dulu, Netflix pakai Java secara penuh di server-side dan JavaScript di client-side. Sekarang, **sebagian besar server-side mereka ditulis pakai NodeJS karena Netflix mengubah webnya jadi single page app.**

Di tahun 2018, mereka mulai mindahin data center fisik mereka ke cloud. Arsitekturnya juga ikut berubah karena awalnya satu aplikasi raksasa berbasis Java, trus diganti jadi gabungan beberapa service yang dipecah kecil-kecil.

Nah sekarang kita masuk ke kriteria yang kedua

- **Time to Market (TTM)**

Makin cepet aplikasi jadi, makin kecil juga biaya development yang dikeluarkan. Atau kamu dituntut buat bikin MVP yang waktunya mepet? Coba pertimbangkan beberapa hal di bawah:

- a) **Banyak open-source library-nya gak?**

Kalau kamu gak punya banyak waktu, mending cari tech stack yang banyak open-source library-nya biar kamu ga perlu bikin semuanya dari nol.



b) Bisa diintegrasikan sama third-party gak?

Ada tech stack yang ekslusif, tapi ada juga yang bisa diperkuat sama sumber-sumber dari luar atau third-party. Makin terintegrasi, makin gampang kamu masukin sumber yang asalnya bukan dari tech stack aslinya.

Lanjut brader!



- c) **Tech stack yang kamu pilih banyak yang pakai gak?**

Makin banyak yang pakai, makin banyak library yang dibikin, artinya makin berkembang juga komunitas teknologinya. Ini bikin kamu enak cari sumber belajar tiap kamu nemu masalah pas lagi ngoding.



Kriteria ketiga, apalagi kalau bukan biaya development

- **Development cost**

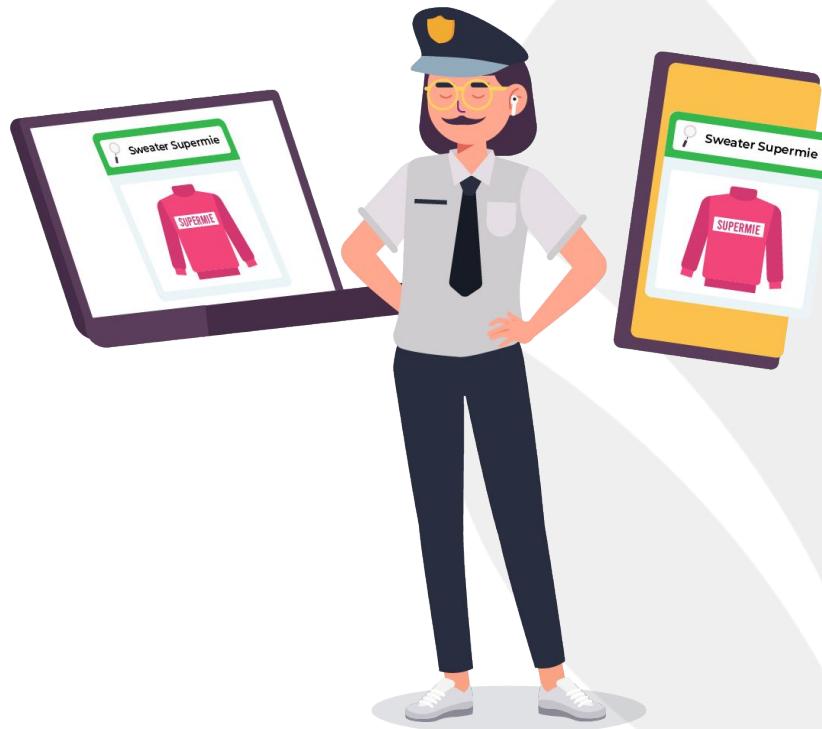
Coba petakan mulai dari awal sampai akhir proses maintenance. Contohnya nih, seiring berjalaninya waktu, usermu makin banyak juga. Nah, kira-kira database servermu juga bakal nambah banyak loadnya. Dari sini, coba kalkulasi harga sewa servernya.



Tapi, yang termurah belum tentu yang paling bagus lho

- **Security**

Pernah nonton Mr. Robot? Perusahaan sekaliber E Corp ditembus keamanannya dan customernya pun merugi, gak cuma itu krisis keuangan jadi akibatnya.



Nah, gak mau nasib yang sama terjadi di aplikasi yang kamu bikin susah payah, kan? **Perlu diingat bahwa gak ada bahasa pemrograman yang ngasih jaminan 100% aman dari serangan.** Tapi, tiap tech stack pasti nyediain guideline tentang security. Jadi, di sinilah pentingnya cari tech stack yang bisa ngasih tahu seluk beluk keamanan teknologinya.



Kriteria terakhir adalah scalability

- **Scalability**

Artinya, kita gak mau kan produk kita stuck alias gitu-gitu aja? Kita juga butuh tools yang adaptif buat mengakomodir perubahan zaman dan segala kebutuhannya.



Secara umum, scalability bisa dibagi jadi 2 jenis:

a) Horizontal scalability

Berhubungan sama makin banyaknya user yang pakai produkmu. Contohnya pertimbangan waktu milih jenis server database.

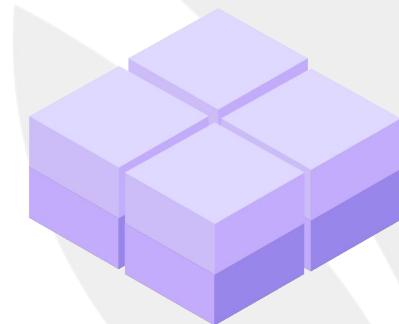
b) Vertical scalability

Kalau ini hubungannya sama kemungkinan nambah fitur baru ke dalam arsitektur produkmu tanpa merusak performanya.

SQL
Vertical Scalability



NoSQL
Horizontal Scalability



Jadi, tech stack yang paling bagus apa dong?

Semua tech stack punya kelebihan dan kekurangannya masing-masing. Adanya cuma yang paling cocok.

Dan kayak milih pasangan: yang paling cocok artinya yang bisa menyesuaikan kriteriamu 😊

Ada penjelasan menarik dari salah satu ex-Tech Lead di Google tentang pertimbangan milih bahasa pemrograman apa yang sebaiknya dipelajari.



Ada penjelasan menarik dari Patrick Shyu, ex-Tech Lead di Google dan Facebook

Isinya tentang pertimbangan milih bahasa pemrograman apa yang sebaiknya dipelajari daripada cuma ngikut trend teknologi terbaru.



[How to choose a programming language \(for your tech stack\)](#)



Coba sekarang kita masuk ke karakteristik tech stack di server-side

- **Ruby (Framework: Ruby on Rails)**

Punya ekosistem development yang cocok buat bikin trading platform, data services, social network, dan marketplace.

Contoh: Airbnb, GitHub, dan Bloomberg.



- **Python (Framework: Django, Flask, dll)**

Sering dipakai buat mengolah data yang gede dan machine learning. Keuntungannya bisa handle ribuan request per menit.

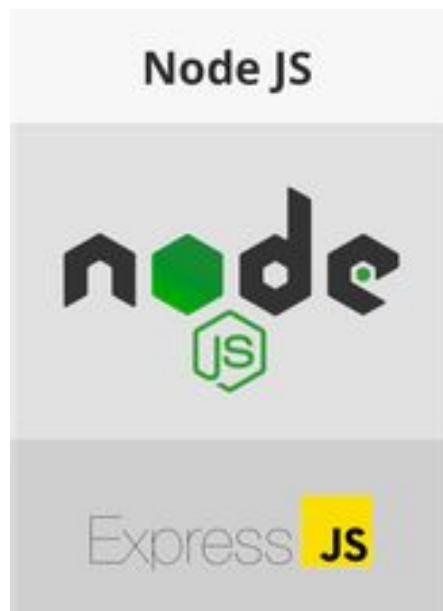
Contoh: Instagram, Spotify, dan Facebook.



- **PHP (Framework: Laravel, CodeIgniter, dll)**

Cocok buat bikin website ringan atau software-as-a-service (SaaS) dengan waktu development yang sangat terbatas.

Contoh: Wikipedia, Wordpress, dan Flickr.



- **Node Js (Framework: Express.JS, dll)**

Node.js ini unik, karena bukan termasuk kategori bahasa pemrograman. Sederhananya, Node.js adalah sebuah platform buat server-side programming yang mengandalkan bahasa JavaScript. Node.js cocok buat bikin real time apps, karena pertukaran data yang bagus.

Contoh: Netflix, LinkedIn, dan Medium.



- **Java (Framework: Spring, dll)**

Dengan slogan sakti "write once, work everywhere" khasnya, Java jadi salah satu bahasa pemrograman yang paling populer di dunia. Kenapa? Karena Java itu bahasa yang bisa dipakai di hampir semua platform, OS, dan devices, apalagi kebanyakan aplikasi Android pakai bahasa yang berbasis Java.

Contoh: Google, Intel, dan eBay.

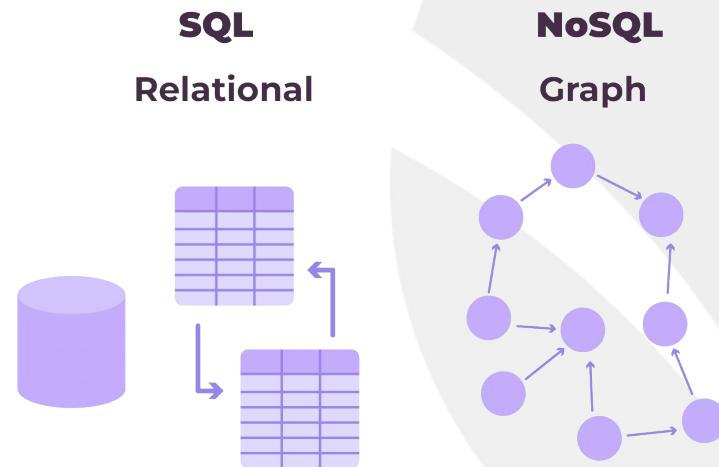
Nah, kalau database ini beda lagi

Secara garis besar, **database dibagi jadi dua yaitu SQL (Structured Query Language) dan NoSQL (Not only SQL).**

Karakteristik SQL adalah structured, artinya databasenya pun berbentuk tabel dan datanya saling berhubungan satu sama lain.

Kalau NoSQL, skema datanya lebih dinamis dan cenderung dipakai buat dokumentasi. Cek perbandingan lengkapnya pada link di bawah ini.

[SQL vs NoSQL Database Differences Explained with few Example DB](#)



Apa aja jenis tech stack di database SQL dan NOSQL yang direkomendasikan?

Untuk SQL ada:

- **MySQL**, punya kemampuan handal dan scalable. Cocok buat transaksi yang saling berhubungan, kayak bank dan asuransi.
- **PostgreSQL**, sering dibilang open-source version-nya Oracle karena fitur punya fitur analytics. Cocok dipakai buat industri finance, manufacturing, dan research project.



Untuk NoSQL ada:

- **MongoDB**, bersifat open-source dan punya customer service yang bagus. Selain itu, MongoDB punya fitur geospasial yang bikin cocok buat hitung jarak dan lokasi.
- **Redis**, populer karena kecepatan dan durabilitasnya. Uniknya, Redis nyimpan data utama di memory server lewat cache web, biar bisa dikonfigurasi dan ditransfer ke disk server.



Cukup sudah pembahasan di topic 5 ini

Kita rangkum yuk udah belajar apa aja!

- 1) Bedain bahasa pemrograman, framework, dan library
- 2) Cara nentuin tech stack yang cocok buat produkmu
- 3) Perbandingan tech stack di client-side dan server-side
- 4) Tech stack di database



Kamu udah sampai di penghujung
Chapter 0. Luar biasa! 🎉

Siap buat mulai materi yang lebih teknis
di Chapter 1 nanti?

