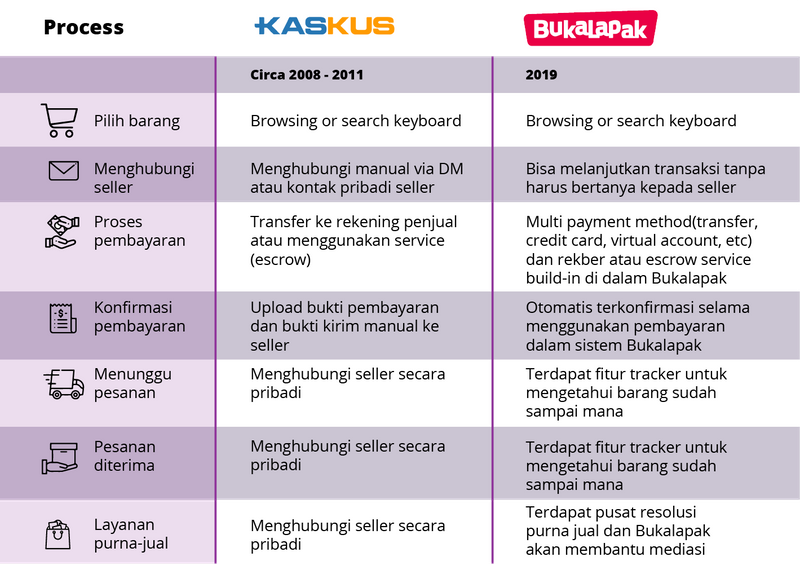
# **CHAPTER 0**

## **Topic 1**

perbandingan proses belanja zaman Kaskus FJB circa 2008 sama Bukalapak di tahun 2019 di bawah. dulu masih banyak interaksi antara seller dan buyer yang masih manual, alias harus japri. Keamanannya juga kadang belum terjamin.Waktu itu, satu-satunya penyelamat adalah review dan bukti resi pengiriman.



Apple merilis App Store di Juli 2008. Yang menarik, 1 tahun sebelumnya mereka ngundang para third party developer buat bikin program khusus di iPhone. Sebelumnya, aplikasi cuma jadi fitur dari handphone. Inget Snake gak? Sebenernya Snake juga aplikasi, tapi akhirnya cuma jadi fitur buat HP Nokia. Nah, inisiatif Apple ini lah yang bikin iPhone berbeda pada masanya, karena aplikasi yang terdapat di iPhone jadi berlimpah. Efeknya, tiap iPhone user bisa pakai iPhone buat tujuan yang beda-beda, kayak nge-game, fokus ke produktivitas, atau sekadar update berita terbaru.

Cuma dalam seminggu App Store dirilis, ada 10 juta download untuk 800 lebih aplikasi yang tersedia di dalam App Store! Hal ini bisa dicapai oleh Apple karena mereka udah pasang kuda-kuda duluan: ngundang third party developers buat bikin aplikasi yang kompatibel sama iPhone setahun sebelum App Store dirilis. Apple juga bolehin para pengembang aplikasi iPhone meraup untung dengan skema bagi hasil 70:30! Sekitar 70% dari jumlah yang dibayarkan downloader tiap download aplikasi berbayar didistribusikan ke pengembang, dan sisa 30% diberikan ke Apple.

Ngeliat Apple sukses besar, kompetitor juga mulai banyak.

a) Android Market (sekarang Google Play Store) dirilis Oktober 2008

b) Blackberry World dirilis April 2009

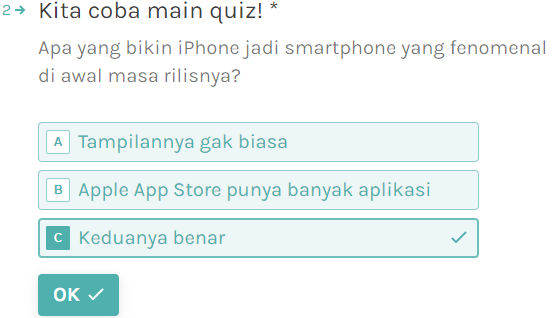
c) Windows Store dirilis Oktober 2012

Tapi, karena persaingan bisnis, banyak yang akhirnya gugur di medan perang. Contoh: Android Market harus ganti nama jadi Google Play Store. Langkah rebranding ini dipilih biar bisa tetep bersaing sama Apple.

Kesuksesan Apple mempopulerkan mobile application gak cuma karena bagus produknya, tapi ini juga diiringi oleh strategi pemasaran yang oke punya. Bulan Januari 2009, Apple ngerilis campaign "There's an app for that" yang bikin app jadi kata yang paling sering diucap di 2010. Campaign yang dibuat Apple viral banget waktu itu. Sampai dikit-dikit orang bilang "Kan ada appnya" tiap kali mau ngelakuin sesuatu.

Desember 2009, Angry Birds rilis sebagai game berbayar di App Store dan jadi fenomena. Game yang dirilis Rovio Entertainment Ltd. ini udah didownload lebih dari 3 milyar kali.Di masa itu, orang-orang bisa ngabisin waktu berjam-jam buat main Angry Birds, bisa sambil ngemil, naik bus, atau pas nongkrong. Angry Birds bertahan 6 bulan di puncak chart App Store. Padahal, pas itu udah ada 160,000 app lain yang tersedia di App Store.

Android Market yang udah berubah jadi Google Play Store juga pelan-pelan sukses sampai punya 1 juta aplikasi di Juli 2013. Mereka juga hasilin 50 juta download di seluruh dunia waktu itu. Nah, di zaman sekarang ini, banyak perusahaan besar berbasis aplikasi yang juga mengakuisisi perusahaan lain. Kayak Facebook yang akuisisi Instagram dan Whatsapp buat ngembangin bisnisnya.



Bukan cuma tampilan smartphone pertama yang menjual akurasi touchscreen-nya, iPhone juga menang di pasar karena banyaknya fitur yang diperoleh melalui App Store. Waktu itu, hp ber-touchscreen umumnya pakai stylus, tapi Apple pede menjual iPhone tanpa stylus. Dan karena aplikasi dulu belum booming, para pengguna jatuh hati pada iPhone karena beragamnya 'fitur' yang ditawarkan. "God gave us 10 styluses—let's not invent another," kata Steve Jobs pas rilis iPhone generasi pertama.

Dengan kolaborasi antar disiplin ilmu, mobile app jadi hal yang gak bisa dilepasin dalam hidup kita sekarang.Dan dari sudut pandang user maupun developer, kehadiran mobile app adalah solusi yang lebih menguntungkan dibanding web. Aplikasi bisa disesuaikan sama kebutuhan bisnis dan lebih fleksibel dari sisi use-case-nya. Dan buat user, pengalaman pakai aplikasi juga cenderung lebih smooth ketimbang pakai website.

Nah jadi waktu sebuah bisnis mau memutuskan pakai website atau aplikasi buat jadi gerbang masuk customer, ada beberapa pertimbangan.

Mulai dari segi biaya, security, practicality, sampai feasibility.

Misal begini:

a) User harus login gak sih buat melakukan kegiatan di app/web?

b) User butuh tahu perkembangan kegiatannya secara cepat dan real time gak sih?

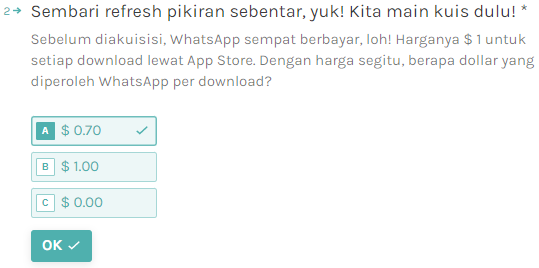
c) Informasi pribadi user itu sangat sensitif, jadi butuh pengamanan yang kayak gimana sih?

Pertanyaan-pertanyaan di atas yang idealnya menjadi beberapa pertimbangan buat nentuin: harus banget bikin aplikasi, atau bikin website aja udah cukup.

## **Topic 2**

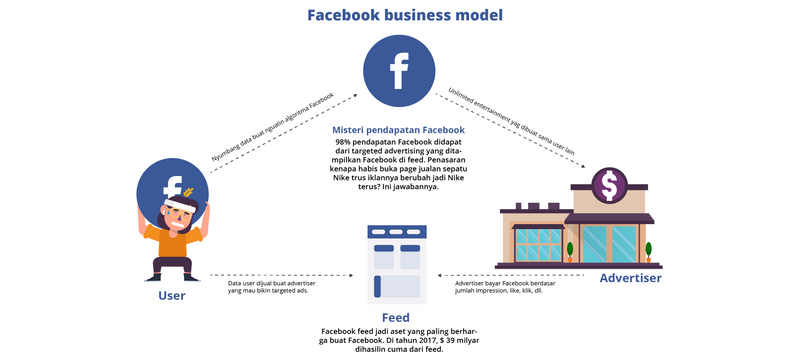
1) Cara-cara bisnis berbasis internet bikin revenue  
2) Konsep sederhana frontend, backend, dan API  
3) Perbedaan karakteristik project dan product

Kalau di topic sebelumnya kita udah belajar sejarah kenapa aplikasi bisa booming, sekarang kita masuk lebih dalem tentang salah satu pemain besarnya.



Whatsapp meraup $ 0.70 karena Apple bikin skema pembagian 70:30 di App Store! Artinya, tiap transaksi yang dilakukan oleh pengguna Apple di App Store, Apple bakal dapet 30%-nya. Dalam kasus ini, tiap kali WhatsApp didownload, Apple bakal dapet $ 0.30 dan WhatsApp $ 0.70.

Sekarang kita ambil contoh di Facebook. Bisnis mana sih yang gak pengin ada pendapatan? Nah, walaupun bikin akunnya bisa gratis, tapi Facebook dapet duit dari para advertiser. Gimana ceritanya? Kamu yang punya akun personal emang gak bayar pakai duit. Tapi kamu "bayar" ke Facebook adalah dengan hal-hal sederhana yang kamu lakukan. Kayak ngisi biodata, beraktivitas kayak ngelike atau ngeview, apa aktif di forum. Yang bayar duit ke Facebook disebut advertiser yang pengin ngiklanin produknya ke kamu. Cerdas ya Facebook!



Intinya, baik Facebook atau bisnis berbasis internet lainnya perlu punya pendapatan (revenue) buat bayar operasional, kayak bayar gaji karyawan, nyewa kantor, dsb. Cara mereka cari uang (revenue stream) juga unik, karena cara bekerja satu platform dan lainnya juga berbeda.



Tadi bahas sisi bisnis dari Facebook. Nah, sekarang kita bahas sisi teknis tapi pakai startup lokal, yuk! Di sesi ini, kita coba bongkar dapur engineering yang ada di Tokopedia. Per April 2019, ada 4,200 nakama (sebutan pegawai Tokopedia) yang bekerja baik di dapur atau depan layar marketplace hijau ini. Banyak banget, ya? Tapi kayak gimana sih pembagiannya?

Tapi sebelum masuk ke Tokopedia, kita mulai pakai analogi yang lumayan familiar dulu kali ya. Anggep begini: Ceritanya, kalian lagi laper banget dan pengen makan burger nih.

Kamu (user) masuk ke restoran dan pilih meja makan. Gak lama duduk, kamu didatengin sama pelayan.

👩‍💼"Silakan menunya."

🙋‍♀️"Makasih, Mbak. Saya pesen beef burger gak pakai acar, satu porsi ya."

👩‍💼"Oke, Kak. Ada tambahan lain?"

🙋‍♀️"Udah, itu aja."

👩‍💼"Silakan tunggu sebentar. Terima kasih."

🙋‍♀️"Sama-sama."

Sehabis milih menu, pesenanmu bakal dicatet di sebuah kertas orderan.

Habis itu, pelayan tadi bakal nerusin catetan pesenanmu ke koki di dapur.

👩‍💼"Sist, meja no. 10 pesen beef burger tanpa acar, satu porsi."

👩‍🍳 "Oke siap!"

Koki yang ada di dapur masakin beef burger kamu di sana. Mulai dari menyiapkan roti bun, patty, tomat, dll.

Masakan udah jadi dan siap dihidangkan. Koki di dapur manggil pelayan lagi.

👩‍🍳 "Udah jadi nih!"

👩‍💼"Oke tengkyuu."

Pelayan bawain pesenanmu ke meja makan. Burger udah siap dimakan.

👩‍💼"Silakan Kak pesanannya: beef burger tanpa acar."

🙋‍♀️"Oke. Terima kasih, Mbak."

Prosesnya simpel, kan?

Nah, cara kerja mobile app kurang lebih kayak proses di restoran tadi.

1) Meja makan tempat kamu pesen tadi disebut frontend.

2) Dapur tempat koki bikin burgermu disebut backend.

3) Pelayan yang bolak-balik tadi adalah penghubung frontend dan backend yang disebut API.

4) Kertas orderan dan beef burger disebut data.

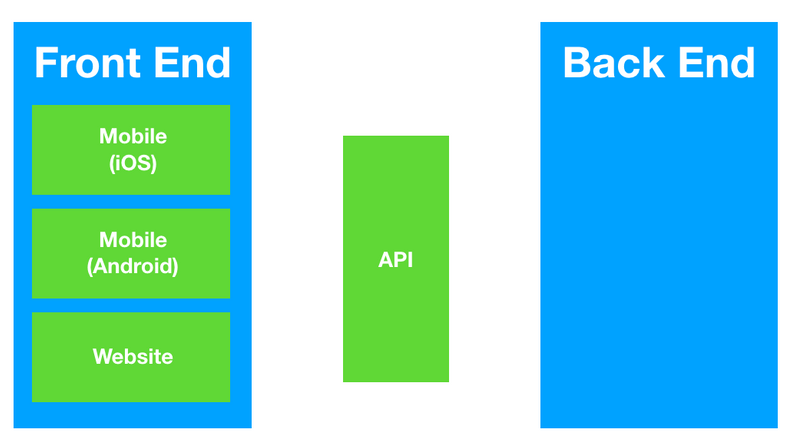
Balik lagi ke Tokopedia ya. Kalau kamu buka app-nya dan search "sweater supermi" itu artinya kamu ngetik di frontend-nya Tokopedia. Ssst.. mau dari website apa mobile app, tempat di mana user bisa berinteraksi sama program, sebutannya frontend.

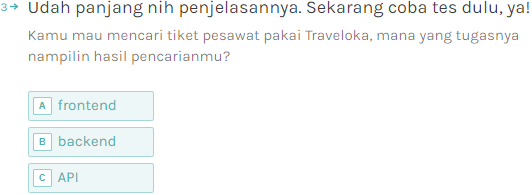
Nah, trus pas kamu tap button Submit atau tekan Enter di kolom bar pencarian, ada sesuatu yang terjadi. Frontend ngirim request ke backend: "Ada yang cari sweater supermi, nih!" Request ini disampaikan secara otomatis oleh API dan ditujukan ke backend-nya Tokopedia.

Di sisi lain, backend nerima request dari API kalau ada yang lagi cari "sweater supermi". Nah, habis itu backend Tokopedia mulai langsung ngambilin "sweater supermi" di database, trus API nganterin barangmu ke frontend.

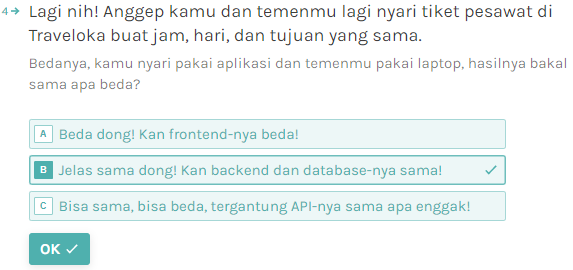
Terakhir, "sweater supermi" bakal ditampilin di frontend dalam bentuk hasil pencarian yang ditunjukan melalui visual di web atau app Tokopedia yang kamu buka.

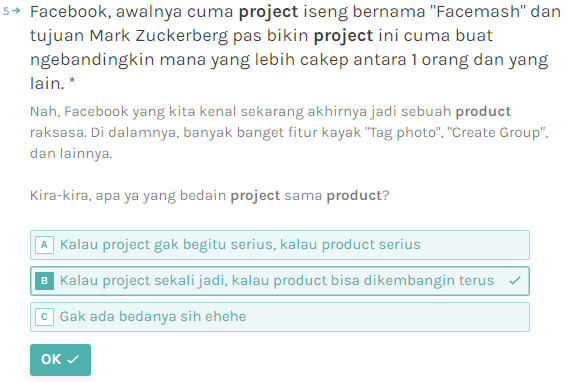
Dengan adanya API, arsitekturnya jadi lebih simpel dan cara kerjanya juga jadi gampang. Daripada harus bikin 3 backend dan 3 frontend yang berbeda, cukup bikin 1 backend yang ngomong ke 3 frontend yang berbeda, melalui API.



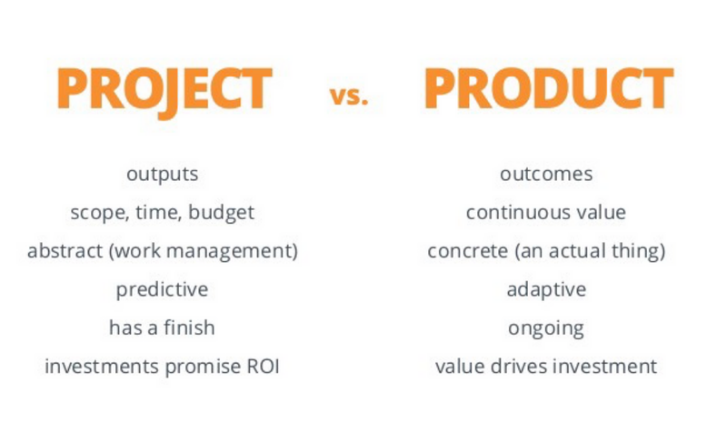


Frontend lah yang tugasnya nampilin hasil pencarianmu. Walaupun pada prosesnya, frontend emang manggil data penerbangan di backend, dan itu juga harus lewat API, tapi yang kamu lihat di layar adalah frontend.

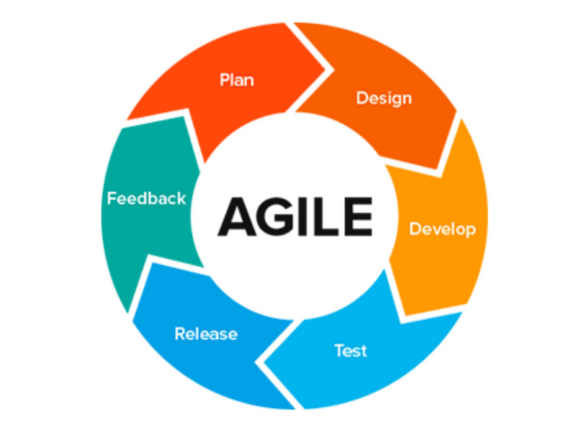




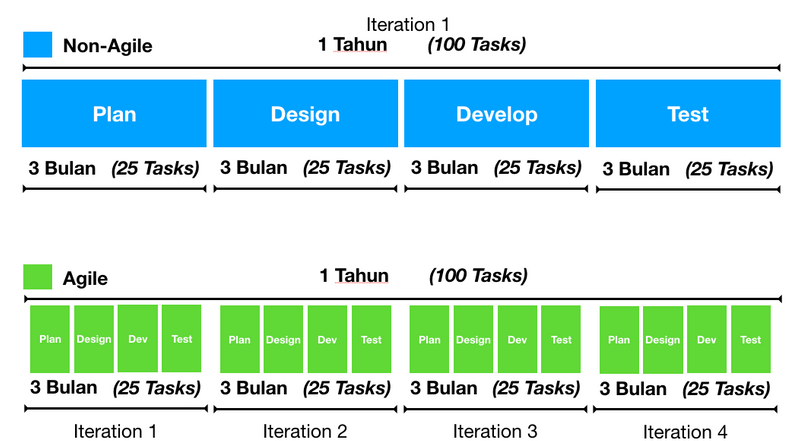
Perbedaan paling keliatan dari project dan product yaitu soal rentang waktu. Project biasanya punya batas waktu dan target yang spesifik. Project biasanya dibikin buat nyelesein kondisi tertentu. Contoh: Binar Academy ngajar di Batam dengan target sekian lulusan. Begitu target tercapai, kalau project-nya gak diperpanjang, berarti udah selesai. Sementara product, biasanya ada "barang konkret" yang dipakai buat nyelesein masalah seseorang. Karena gak ada 'selesai'-nya, maka product bakal selalu berkembang. Contoh: Binar Academy bikin aplikasi biar kamu bisa belajar dari mana aja. Awalnya appnya cuma bisa buat buka soal, lama-lama bisa buat buka video. Di bawah ini adalah bedanya project dan product.



Dalam suatu product, yang terpenting adalah komponen tim dan metode kerjanya. Nah, buat perusahaan berbasis internet dan aplikasi, metode kerja yang umum dipakai adalah metode Agile. Agile ini metode kerja yang berulang-ulang. Artinya, begitu selesai, bakal selalu ada evaluasi, perbaikan, riset, plan buat fitur baru, dan gitu terus...



Agile ini bukan berarti cepat, tapi artinya gesit atau lincah. Agile dibutuhin biar bisa terus adaptasi sama perubahan. Coba liat gambar di bawah. Bayangin kalau kita cuma punya waktu 1 tahun untuk bikin suatu aplikasi. Nah, kalau gak pake Agile, kita cuma bisa bikin 1 kali. Sementara kalau pakai agile, kita bisa bikin sampai 4 kali.



Dalam Agile, ada yang disebut Scrum

Nah, kalau mau bikin product yang oke, metode kerja Scrum bakal mayan sering dipakai. Secara garis besar, Scrum cuma ngebagi tim jadi 3 fungsi pekerjaan yang setara pentingnya:

1) **Product owner**: role ini yang menentukan akan membuat product apa.

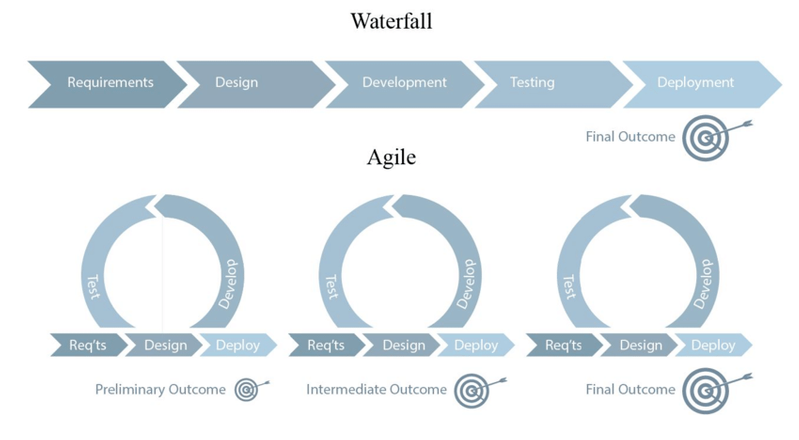
2) **Scrum master**: servant leader, alias yang selalu bantuin tim dan mastiin masalah mereka selesai.

3) **Team member**: mastiin produknya bisa sesuai dengan kriteria dari Product owner.

## **Topic 3**

1) Metodologi kerja Agile-Scrum dan bedanya sama Waterfall  
2) Pembuatan roadmap oleh product owner  
3) Persiapan, riset, dan desain oleh UI/UX designer  
4) Pembangunan database, API, dan client-server model oleh backend engineer  
5) Proses aktualisasi desain oleh frontend engineer  
6) Pengertian tentang app testing oleh QA engineer  
7) Bedanya product owner sama scrum master  
8) Kolaborasi pembuatan konsep sampai software staging di dapur app development  
9) Tahapan rilis produk dan environment yang ada dalam SDLC

Agile termasuk salah satu metodologi kerja pembuatan software, atau disebut software development life cycle (SDLC). Tujuan SDLC yaitu buat bikin software dengan kualitas terbaik, tapi dengan biaya terendah, di waktu yang paling cepet. FYI, jenis SDLC gak cuma Agile aja lho.



Jadi, sebelum masuk ke zaman aplikasi yang bisa lebih fleksibel diupdate, software konvensional umumnya dikerjain pakai metode SDLC Waterfall. Fase-fase di metodologi kerja Waterfall berbentuk linear dan dikerjakan dalam rentang waktu tertentu buat dapet hasil final.

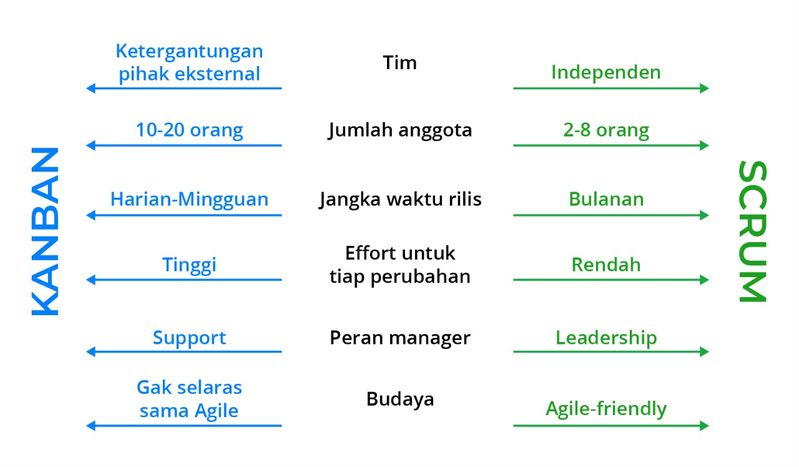
Tapi, karena smartphone makin ngetrend, perkembangan software juga makin lincah. Dulu, tiap update versi baru harus rilis CD installer atau download bergiga-giga, sekarang cukup update lewat Play Store atau App Store. Nah, makanya butuh metode kerja baru buat menyesuaikan dan dari sini lahirlah Agile. Kelebihan utama Agile adalah produk yang kita bikin bisa lebih cepet menangkap feedback dari market.

Di Agile, secara garis besar ada dua metode turunan yaitu Scrum dan Kanban. Apa bedanya?

a) **Scrum** biasa dipakai buat product development. Contoh: bikin aplikasi.

b) **Kanban** dipakai buat project/product yang melibatkan banyak pihak dari luar tim. Contoh: manufaktur perakitan mobil.

Tapi, kita gak bakal bahas Kanban. Kita bakal fokus ke Scrum karena nantinya kamu dilatih buat jadi seorang software engineer.



Beda sama Waterfall yang linear, Scrum punya fase yang terus-menerus dilakukan. Siklus yang ada di Scrum ini sering disebut juga Software Development Life Cycle (SDLC). Karena terus diulang, di Scrum ada satuan waktu konstan dan disebut **sprint**. Satu sprint biasanya berdurasi 1 minggu sampai maksimal 1 bulan.

Kita mulai dari dasarnya dulu ya:

a) Satu sprint terdiri dari beberapa **task** yang harus diselesein keseluruhan tim

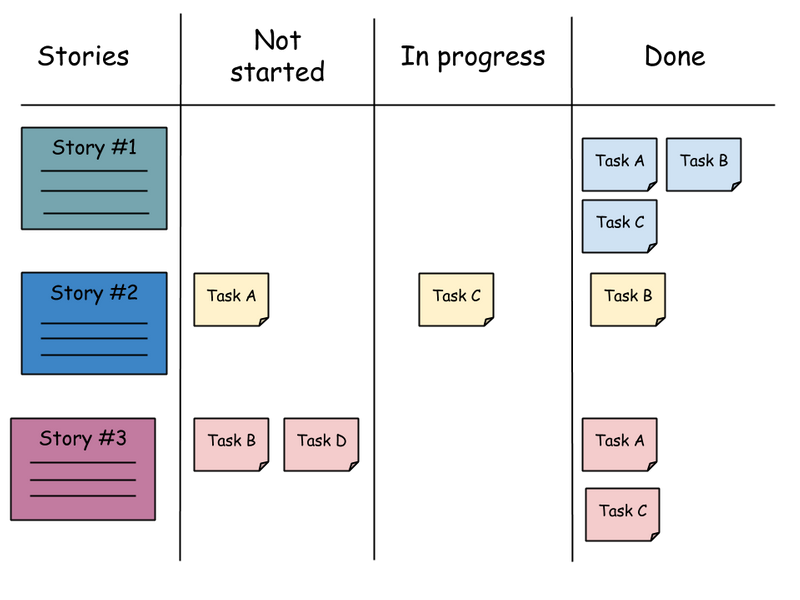
b) Seluruh task yang sudah dirumuskan dimasukkan ke **product backlog**

c) Sebelum sprint dimulai, ada sesi **sprint planning** yang mana seluruh anggota tim menentukan **jumlah task untuk sprint yang akan datang** pada masing-masing anggota

d) Task yang sedang dikerjakan dimasukkan ke bagian **in progress**, sementara task yang sudah selesai akan dimasukkan ke bagian **done**

e) Apabila ada task yang tidak selesai pada satu sprint, maka task itu akan dimasukkan ke **sprint backlo**g untuk sprint selanjutnya

Cara ngelakuinnya simpel, bisa pakai board yang kayak di bawah:

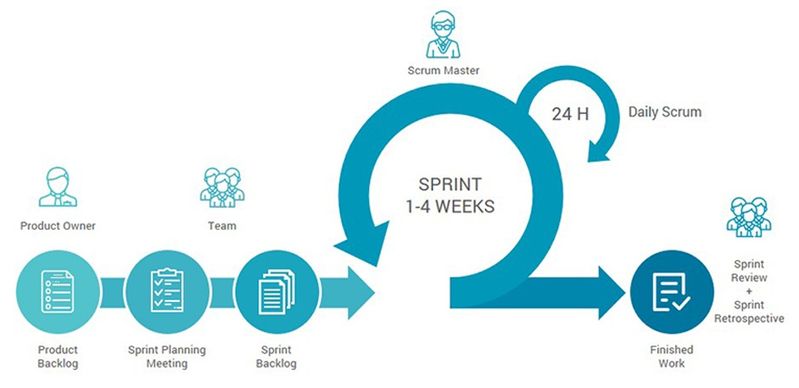


Itu tadi bentuk board dan cara bikinnya, tapi ada juga sesi-sesi khusus yang scrum banget.

a) Tiap hari, ada **scrum daily** atau **standup meeting**. Meeting harian ini tujuannya buat ngelaporin task apa yang udah dikerjain, obstacle yang ditemui, target task hari ini, dll.

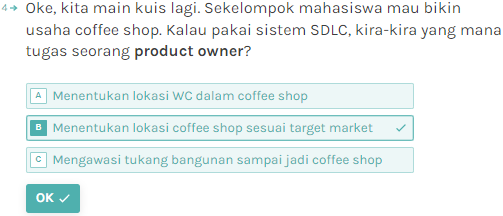
b) Sehabis selesai sprint, ada sesi khusus buat ngebahas pencapaian tim yang disebut **sprint review**.

c) Selain ngebahas pencapaian, ada juga pembahasan non-teknis kayak tingkat kebahagiaan, kekurangan, dll yang dirangkum di sesi yang dinamain **sprint retrospective**.

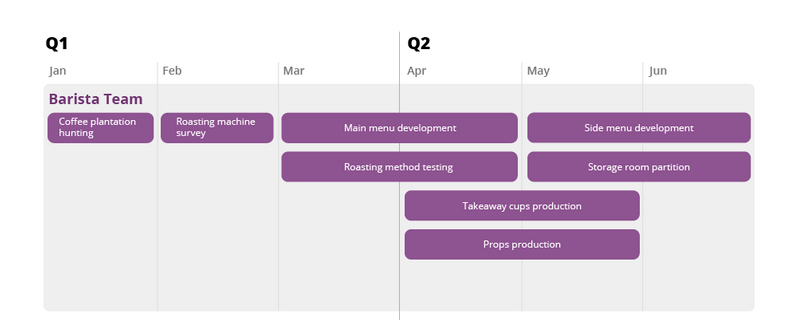




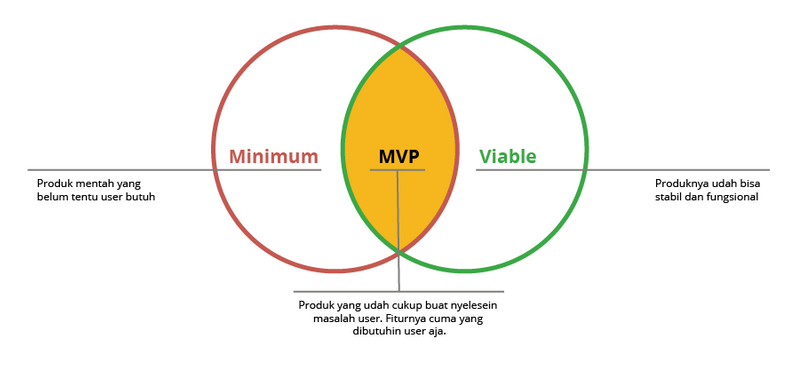
Product owner bertugas merumuskan masalah yang ada di dunia nyata, trus cari solusi digital yang bisa bantu selesein masalah itu. Tapi gak segampang itu, product owner harus bisa mimpin riset buat validasi masalah, riset produk dan pasar, sampai bikin strategi produk biar produk yang dibikin sesuai sama kebutuhan pasar, atau biasa disebut product-market fit.



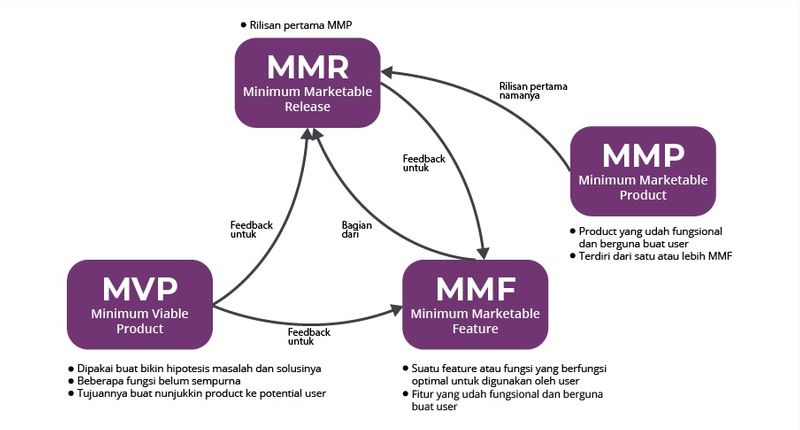
Kalau sebuah coffee shop adalah produknya, product owner bertugas buat nentuin gimana strategi biar produknya ini laku sesuai target pasar, dan salah satu elemennya adalah penentuan channel, atau dalam hal ini, penentuan lokasi. Selain itu, harusnya product owner juga bisa ngatur dalam jangka panjang coffee shop ini ga cuma jualan kopi, tapi bisa jadi tempat sharing komunitas, perpustakaan, atau jual merchandise juga. Nah, rencana untuk menuju ke objective jangka panjang ini lah yang namanya roadmap plan.



Tapi, kebanyakan aplikasi nggak dimulai dengan modal besar. Sebagai contoh, Instagram juga dulunya cuma bisa dipakai di iOS dengan fitur yang cuma "feed" aja. Tapi lambat laun, fiturnya berkembang jadi selengkap sekarang. Produk dengan fitur minimal ini disebut Minimum Viable Product (MVP). Walaupun fitur dalam produk itu udah cukup fungsional, tujuan MVP sebetulnya adalah pengumpulan feedback dari sampling user: produk yang dibuat bisa nyelesein masalah mereka atau enggak.



Produk yang masih MVP, artinya belum bisa dipasarkan secara luas, atau belum bisa disebut (Minimum Marketable Product) MMP. Ada beberapa tahap yang harus dilalui sebelum sebuah produk bisa dirilis secara luas (Full release).



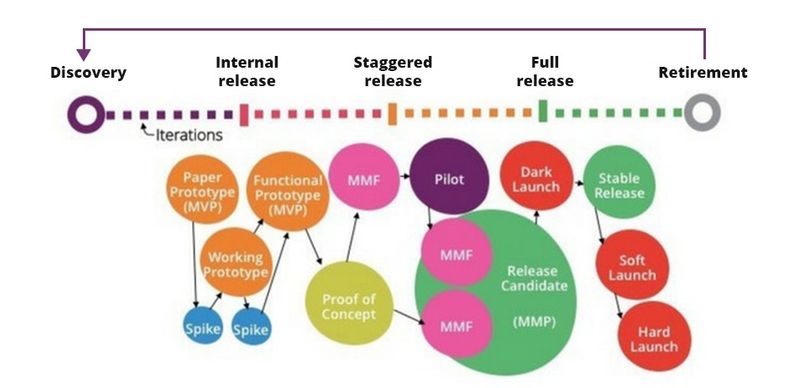
Proses buat sampai jadi **MMP** juga panjang. Dibutuhkan siklus riset-development-validasi yang berulang — disebut iterasi — sebelum masuk ke tahap rilis.

Untuk merilis pun juga ada 3 jenis:

a) **Internal release**: Dirilis ke internal tim produk untuk dilakukan unit test. Output yang diharapkan yaitu technical feedback.

b) **Staggered release**: Dirilis ke publik secara terbatas dan bertahap. Output yang diharapkan yaitu performance feedback , untuk mengetahui performa dan keandalan produk saat digunakan banyak pengguna

c) **Full release**: Dirilis ke publik secara luas. Di versi ini, produk yang dirilis seharusnya sudah diterima pasar, fungsional, dan stabil secara teknis.

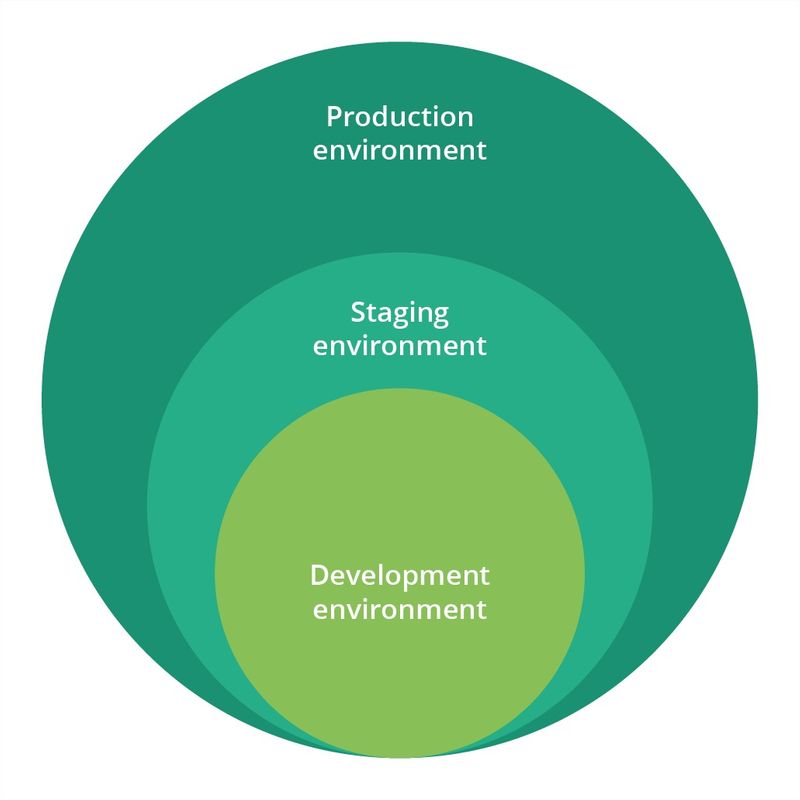


Selain rilis, ada juga yang namanya **environment**. Gampangnya, sebuah environment adalah kalangan yang terdiri dari beberapa role. Umumnya **environment** ada 3 jenis. Apa aja ya?

a) **Development environment** : para software engineer menuliskan kode mereka di environment ini. Fase ini bersifat dinamis, sehingga segala perubahan terjadi sangat cepat karena masih dalam masa pengerjaan, dan dapat langsung dicoba

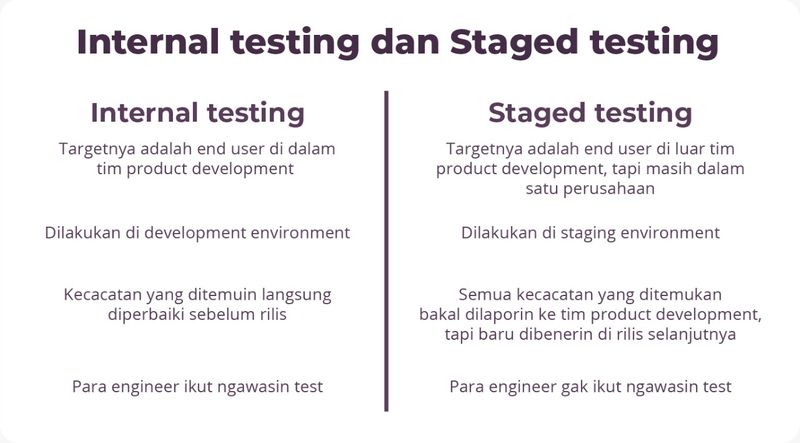
b) **Staging environment** : di fase ini berisi versi software yang akan dirilis ke publik. Disinilah biasanya software sudah bisa diuji coba untuk skala internal perusahaan, sambil menunggu persetujuan dari stakeholder sebelum dirilis. Perbaikan dan perubahan kecil masih diperbolehkan terjadi di fase ini

c) **Production environment** : fase ini berisi versi software yang siap dirilis ke publik. Pada fase ini pula tidak dikehendaki adanya perubahan atau pergantian produk, melainkan harus menjadwalkan perubahan tersebut untuk dirilis pada versi berikutnya





Boleh dibilang peristiwa ini berlangsung di staging environment dan produknya berada dalam fase MVP. Pengetesan dilakukan ke semua pegawai (ada di staging environment). Tapi, peracik kopi (setara sama engineer) udah gak bisa ngubah rasa kopi karena udah waktunya rilis. Jadi, tugas PO pas udah di staging environment adalah mikirin gimana strateginya biar produknya bisa nyelesein masalah user dengan lebih efektif, biar terjadi product-market fit. Nah, kalau masih ditemuin kekurangan dari sisi kegunaan produk, perbaikan bisa dilakukan di rilis berikutnya.



UI/UX designer ternyata gak sepele dan gak terbatas di desain aja, lho. Lebih dari itu, tugas utama seorang UI/UX designer bisa dibagi jadi 3 garis besar:

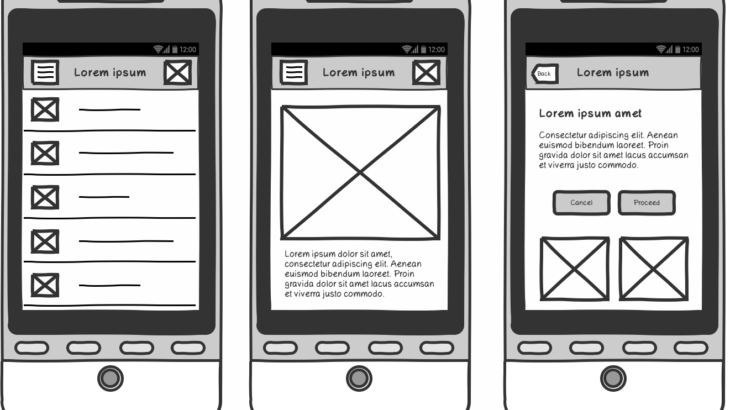
a) Memetakan ide dan user behavior ke dalam rancangan flow aplikasi. Output dari proses ini biasanya berbentuk flowchart, wireframe, dan lo-fi design.

b) Melakukan serangkaian riset/test biar flowchart yang dibikin gak membingungkan dan bisa ngasih experience yang bagus buat user. Output dari proses ini disebut user journey matrix.

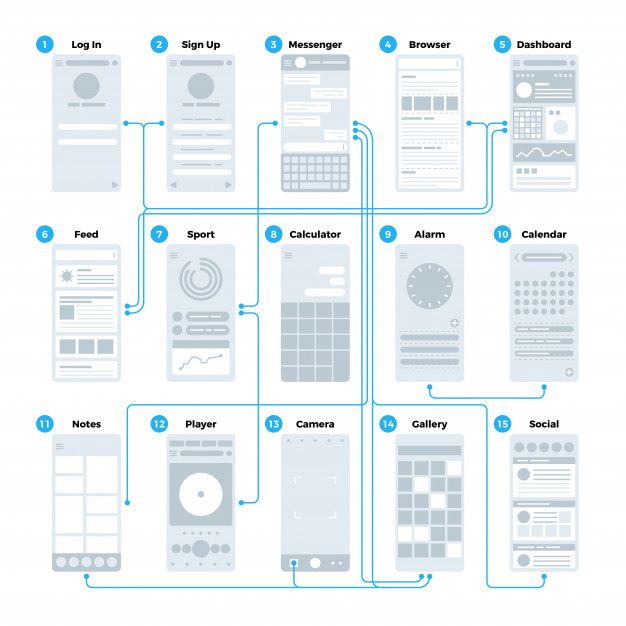
c) Bikin desain interface yang gak cuma estetis, tapi juga intuitif. Artinya, user bisa paham kegunaan masing-masing fitur tanpa harus ribet menjelaskan. Output dari proses ini disebut hi-fi design.

Sebelum jadi lo-fi design, sebaiknya seorang UI/UX designer bikin yang namanya flowchart. **Flowchart** ini berisi logika perjalanan user di dalam produk kita yang disampaikan secara sederhana tanpa gambaran interface

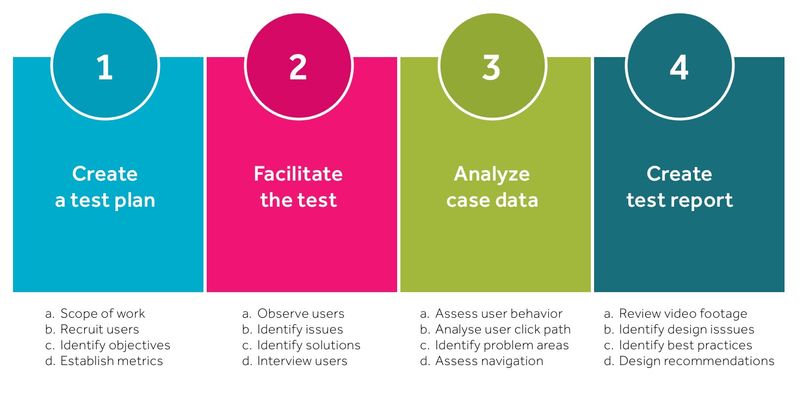
Setelah flowchart jadi, ini waktunya bikin rancangan kasar per screen. Rancangan ini dinamakan wireframe. **Wireframe** berisi komponen yang ada di tiap screen-nya. Biasanya, rancangan interface kasar ini digambar secara sederhana dengan kotak-kotak beserta ukuran/dimensinya.



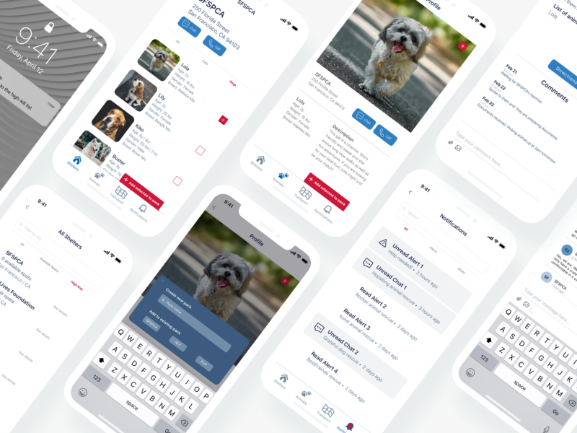
Kalau wireframe per screen udah jadi, sekarang waktunya buat sambungin wireframe sesuai logika yang dibikin di flowchart. Rancangan yang udah setengah jadi ini dinamain **low-fidelity design** **(Lo-Fi).** Lo-Fi design umumnya berbentuk digital dan dibuat pakai tools kayak Sketch, Zeplin, atau Figma. Isinya wireframe yang udah terstruktur sesuai sama perjalanan user dari awal sampai akhir penggunaan produk.

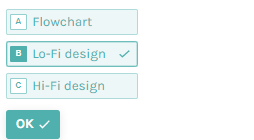
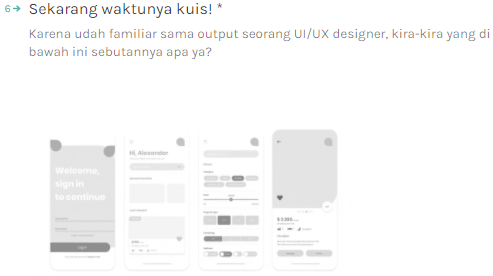


Lo-Fi design ini dijadikan bahan untuk user testing. Test ini diawasi oleh UI/UX designer dan tujuannya buat menggali potensi user behavior sewaktu nanti app-nya udah jadi. Langkah-langkah di tabel di bawah nantinya didokumentasikan ke dalam user journey matrix. Dan dari matrix itu, kelemahan dalam flow bisa kelihatan: Bisa nggak sih user mengoperasikan appnya? Ngerti gak sih user sama flow yang udah dibikin?

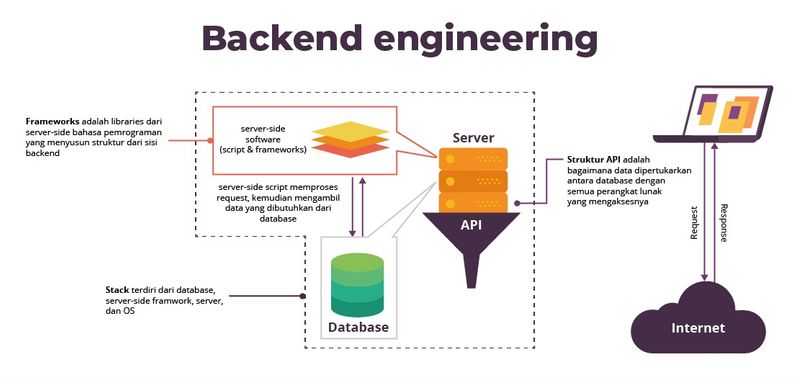


Sehabis semua feedback dikumpulkan, sekarang waktunya buat bikin **high-fidelity design (Hi-Fi).** Sederhananya, sebuah Hi-Fi design adalah Lo-Fi yang udah dilengkapi aset visual dan interaksi kayak icon, ilustrasi, typeface, dll.





Dalam SDLC, seorang **backend engineer** bertugas buat mengurus server, bikin skema database, dan bikin API. Intinya, bikin arsitektur buat eksekusi fitur biar sesuai sama logika flowchart yang disepakati. Seorang **backend engineer** yang baik harus bisa menerjemahkan keinginan UI/UX designer yang ada di flowchart ke dalam bahasa yang lebih teknis. Output dari proses ini adalah **application programming interface** atau biasa disingkat **API**. Inget analogi API yang ada di topic sebelumnya? Nah dalam SDLC, API adalah protokol komunikasi yang fungsinya sebagai pengatur pertukaran database yang berlangsung antara server dan client.

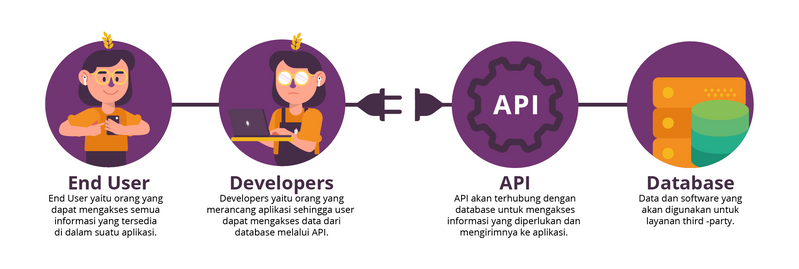


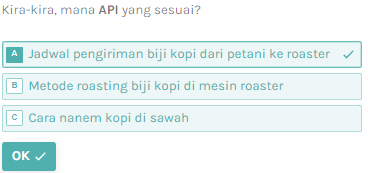
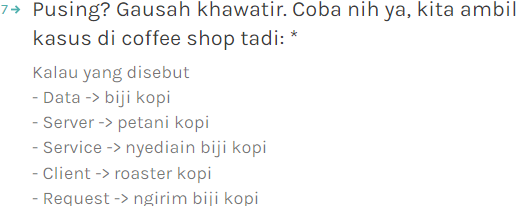
perdalam dikit soal server dan client.

a) Yang disebut **server**, adalah pihak yang menyediakan service. (Server -> Yang ngasih service, yang nyediain service; hehe). Gampangnya lagi, server selalu ada di backend.

b) Kalau **client**, kayak klien di dunia nyata, adalah service requester, alias yang minta service. Nah, kalau client biasanya ada di frontend.

Sederhananya, isi dari **API** adalah "perjanjian" yang dibuat antara server dan client tentang cara penyebutan data, data apa aja yang disalurkan, sampai ke cara pengambilan datanya.





Dalam konteks ini, biji kopi berperan sebagai data yang harus dikirimkan petani kopi ke roaster. Memang, metode roasting biji kopi dan cara nanem kopi bisa jadi API yang sama-sama dibikin sama backend engineer. Tapi kamu harus inget service yang diminta adalah nyediain biji kopi, jadi proses yang terjadi adalah pengiriman biji kopi dari petani ke roaster. Inget, beda service, beda pula API-nya.

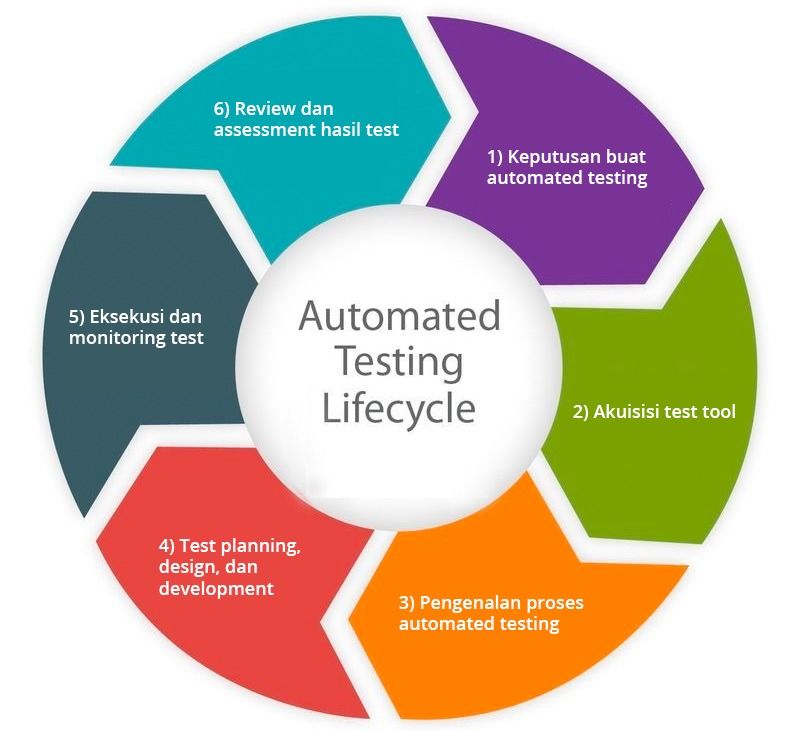
Dalam SDLC, seorang frontend engineer bertugas buat bikin apa yang udah didesain UI/UX designer dalam Hi-Fi design bisa jadi kenyataan. Sebelumnya, di Hi-Fi design belum ada satu kode pun yang bisa diimplementasi. Di sini lah tempat frontend engineer bekerja. Nantinya dia harus bisa menghubungkan desain yang udah dibikin sama UI/UX designer dan API yang udah dibikin backend engineer.

Output yang dibikin dari hasil kolaborasi UI/UX designer, backend engineer, sama frontend engineer disebut prototype. Prototype sederhananya adalah Hi-Fi design bikinan UI/UX designer yang udah diisi coding sama frontend engineer dan udah terhubung ke API yang dibikin sama backend engineer. Alias: desain yang udah bisa dipencet-pencet.



Ngomong-ngomong soal unit testing, di sini peran **QA engineer** dibutuhin. **QA** itu singkatan dari **Quality Assurance**, alias, Si Penjamin Kualitas. Tugas seorang QA engineer cukup berat, yaitu memastikan produk yang dirilis minim masalah, seperti bug dan error. Buat minimalisir masalah, QA engineer bertanggungjawab buat bikin serangkaian tes yang boleh dilakukan manual atau automated.

Kenapa boleh dilakukan manual atau automated? Setiap tes punya karakteristik sendiri-sendiri. Gak semua jenis tes bisa diotomatisasi dan beberapa jenis lain bisa melelahkan kalau dilakukan manual.



Apa yang biasa dites sama QA engineer? Buanyaaak banget yang bisa dilakuin QA engineer buat mastiin bahwa produknya bener-bener anti cacat. Gak heran, kebanyakan QA engineer bersifat teliti dan perfeksionis.



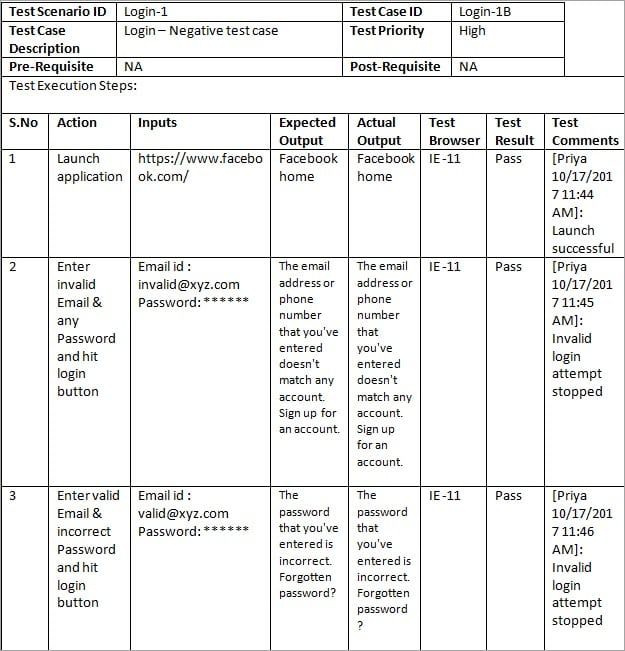
Setiap tes yang dilakukan oleh QA engineer harus didokumentasiin secara rapi biar bisa dibaca backend dan frontend engineer. Jadi, banyak product owner yang dulunya QA engineer karena:

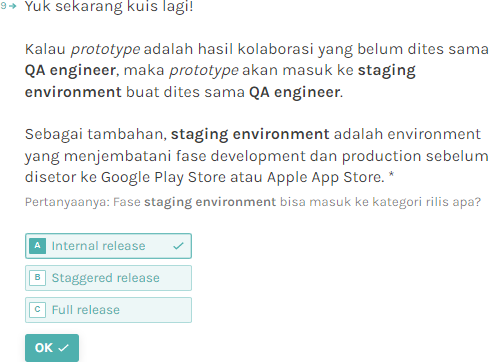
a) Tahu luar-dalem permasalahan yang biasa kejadian di SDLC dan tahu cara minimalisirnya.

b) Ngerti bahasa pemrograman yang dipakai di pembuatan API oleh backend engineer.

c) Perhatian terhadap detail dan tahu cara dokumentasi teknis yang baik.

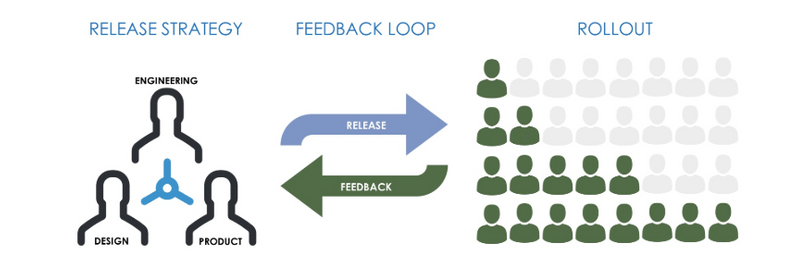
Di bawah ini contoh dokumentasi teknis oleh QA engineer.





Staging environment masih termasuk ke internal release, karena yang bisa akses cuma anak dalem kantor. Cara rilisnya adalah dengan package system (APK atau iPA). Biasanya rilis kayak gini dipakai buat produk yang belum disetor ke Play Store atau App Store. Ada kasus lain: bagi-bagi APK ini dilakukan karena belum dapet approval dari Play Store atau App Store.

Begitu udah dilakukan test dan siap rilis, ada proses khusus yang harus dilakukan sebelum full release. Namanya staggered release. Staggered release ini ciri khasnya yaitu udah bisa diakses publik, tapi masih terbatas. Kenapa dinamain staggered? Karena rilis ini dilakukan secara bertahap (rollout). Rilis kayak gini tujuannya buat ngetes seberapa stabil produk kita kalau dipakai sejumlah user. Contoh: Fitur dark mode di Instagram baru bisa dipakai di 5% user di hari pertama, 20% hari berikutnya, dst. sampai semua user bisa pakai.





Staggered release ada di production environment karena udah disetor ke Play Store atau App Store dan bisa dinikmatin publik meski terbatas.

Apa bedanya product owner sama scrum master? Kan sama-sama leader? Beda dong. Mereka juga punya tanggung jawab lain:

a) **Scrum master**: Gimana caranya keharmonisan dan kerja sama tim terjaga.

b) **Product owner**: Gimana caranya produknya bisa product-market fit.

Contoh lain mungkin kayak pemain bola. Scrum master kayak kapten tim dan product owner kayak pelatihnya. Si kapten juga termasuk pemain, tapi leadership yang dia punya dipakai buat nyemangatin kalau ada yang drop atau ingetin temennya yang lagi bengong. Tapi kalau pelatih, tugasnya adalah bikin timnya menang lewat taktik, strategi, pergantian pemain, dll.

## **Topic 4**

1. Definisi tech stack dan tools

2. Berbagai jenis tech stack buat Frontend engineer

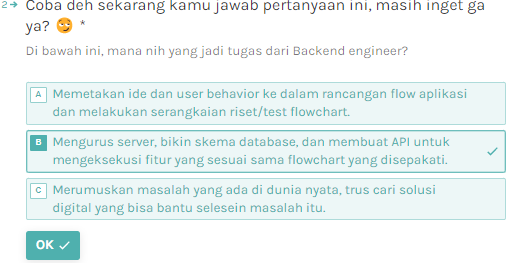
3. Berbagai jenis tools buat Frontend engineer

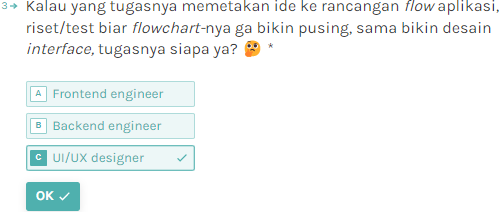
4. Berbagai jenis tech stack buat Backend engineer

5. Berbagai jenis tools buat Backend engineer

7. Perbedaan Frontend engineer dengan Mobile engineer

8. Berbagai jenis tech stack buat Mobile engineer





di topik ini kita akan membedah tech stack dan tools yang biasa dipakai buat menjalankan tugas mereka. Oke, sekarang kita bayangkan ada di sebuah pabrik mobil:

a) Product owner adalah pemilik pabrik

b) Engineer adalah teknisi, dan mobil adalah produk aplikasi yang akan dibuat.

Maka, tech stack ini adalah metode yang digunakan untuk membuat mobil sementara tools adalah peralatan yang membantu menjalankan metode.

Tech stack yang wajib dikuasai oleh seorang frontend engineer sendiri itu ada tiga, yakni HTML, CSS, dan JavaScript. Karena sering berkutat sama tampilan depan dari sebuah aplikasi, tech stack yang dipakai harus bisa membantu frontend engineer buat bikin desain, konten, gambar, navigasi, sampe link yang berkaitan.

HTML itu singkatan dari Hypertext Markup Language yang bisa digunakan sebagai "alas" untuk membuat dan menyusun bagian paragraf, heading, dan link menggunakan tag kaya contoh di bawah pada halaman depan aplikasi. Gampangnya, HTML itu kerangka mobil sementara paragraf, heading, dan linknya itu kap, bumper, dan pintu mobil.

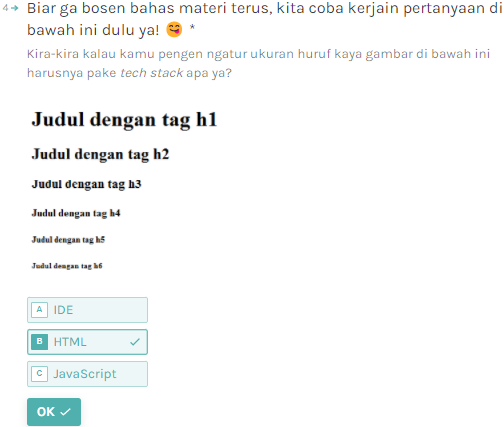
Kalo tadi HTML itu kerangka mobil, maka CSS adalah cat dan coating yang bikin eksterior mobil makin ciamik. CSS sendiri singkatan dari Cascading Styling Sheet yang berfungsi buat bikin tampilan aplikasi yang menarik mulai dari tata letak, pemilihan warna, sampai ukuran huruf yang pas biar enak dipandang.

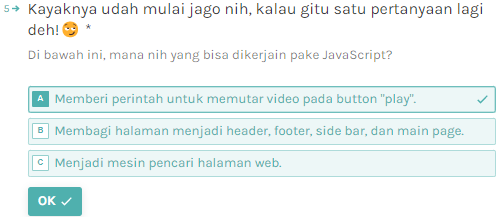
Biar pintu mobil bisa dibuka, jendela bisa diturunin, dan pintu bagasi bisa dinaikkan, kita butuh sistem yang ngatur itu semua. Sistem ini yang disebut dengan JavaScript. JavaScript ini adalah bahasa pemrograman intrepreter yang berfungsi agar tampilan depan aplikasi lebih interaktif, dinamis dan pastinya bisa dipake sama penggunanya.

Tools yang biasa dipake sama frontend engineer dibagi lagi jadi tiga, yaitu tools yang dipake pas bikin teksnya, pas ngecek hasil teksnya, sama buat ngontrol kalo ada masalah pada teksnya di kemudian hari. Tools yang dipake buat nulis teks dari tech stack disebut dengan text editor atau IDE (Integrated Development Environment). Text editor yang biasa dipake sama Frontend engineer ada Atom, Sublime, dan Visual Studio Code.

Butuh tools yang bisa menampilkan teks tersebut biar kelihatan hasilnya kaya gimana. Tools ini yang disebut sebagai browser development tools. Browser development ini biasanya pake browser yang biasa kamu pake buat internetan kaya Mozilla Firefox dan Google Chrome.

butuh tools yang disebut version control (Git) yang tugasnya buat membantu nge-track kalau ada perubahan dalam kodingan kita.





Tech stack yang dipake sama backend engineer ini biasanya dikelompokkan jadi 4 golongan tergantung fungsinya masing-masing.

1) Web server

2) Web framework

3) ORM (Object Relation Mapping)

4) Database: SQL (Structured Query Language)

**Web server** ini gampanganya adalah kurir yang nyalurin permintaan dari browser kaya mozilla tadi ke sumber data yang dibutuhin. Kalau udah dapet data yang diminta, dia juga yang bakal ngenterin data ini kembali ke browser buat ditampilin ke pengguna. Web server yang biasanya dipakai sih ada Apache, Nginx, sama Microsoft IIS (Internet Information Service).

Kalo **web framework** gunanya buat ngatur berbagai fungsi sama konsep biar membentuk satu sistem tertentu. Tujuannya biar aplikasi yang dibuat terstruktur dengan rapi. Pilihan web framework ini sebenernya banyak banget tapi yang sering dipake tuh ada Express by JavaScript, Django by Python, Laravel by PHP, dan Ruby on Rails by Ruby.

**Database** berupa kumpulan data yang disimpan secara sistemik di dalam sistem komputer. Database inilah yang jadi sumber data buat para pengguna aplikasi. Tech stack yang biasa dipakai buat me-manage database antara lain mySQL, PostgreSQL, MariaDB untuk yang pakai SQL, dan MongoDB untuk yang ga pakai SQL.

Yang terakhir ada **ORM**. Doi ini bertugas untuk ngehubungin web framework sama database.

Kalau di dunia nyata, modif mobil tuh prinsipnya gak bisa sembarangan. Nah, modif itu gak cuma ngubah bentuk luarnya aja tapi juga harus ada penyesuaian biar gak ganggu fungsi-fungsinya.Terus, modif mobil ini biasanya butuh teknisi khusus yang paham caranya modif biar gak konslet. Kalau di pembuatan aplikasi, tugas teknisi ini diisi sama **mobile engineer**.

Meskipun sama-sama fokus ke tampilan depan web, frontend engineer sama mobile engineer itu beda. Kalau frontend engineer tugasnya ngurusin tampilan depan web, Mobile engineer ini tugasnya lebih spesifik. Dia fokus buat bangun tampilan web biar ramah buat pengguna aplikasi smartphone. Mobile engineer ini bakal dibutuhin banget pas produk udah berada di tahap **production environment** alias aplikasinya udah masuk di Play Store atau App Store.

Ini nih, daftar tech stack yang biasa dipakai sama Mobile engineer

1. PWA (Progressive Web App)

2. Native

3. Hybrid

4. JavaScript Native

5. Cross-compiled

**PWA (Progressive Web App)** ini tuh jadi metode yang paling sederhana karena cuma bikin tampilan website jadi mirip sama tampilan sebuah aplikasi. Kalian pernah ga mengunjungi satu website dari browser di smartphone kalian tapi isi websitenya interaktif banget dan berasa kaya lagi pake aplikasi? Nah, metode itu yang disebut sama PWA.

Lanjut ke **Native** yak! Doi ini sistemnya misahin metode buat bikin mobile apps di sistem operasi Android dan iOS.

Terus, spesialisasi buat mobile engineer ini kebanyakan dibagi dua, spesialis Android dan spesialis iOS. Kenapa dibagi? Karena bahasa dan tech stack yang dipake sama keduanya beda alias native. Kalo Android pake Kotlin, iOS pake Swift.

Intermezzo dikit. Sekarang ini, Android udah jadi yang paling banyak di dunia jauh ngalahin iOS. Karena Android ini biasanya pake bahasa Kotlin, otomatis tren penggunaan bahasa ini juga ikutan naik.

Yang ketiga ada **Hybrid**. Sama kaya namanya, Hybrid ini campuran dari dua tech stack antara PWA sama Native. Kalau tools yang biasa dipake di metode Hybrid ini ada Apache Cordova atau Ionic’s Capacitor

lanjut ke **JavaScript Native**. Tech stack yang ini bisa dibilang yang paling universal karena kalau kita pake metode ini, kita bisa bikin mobile apps yang bisa dipake di sistem operasi Android dan iOS sekaligus. Framework yang biasa dipake tuh ada Native Script atau React Native. Tools buat framework ini ada Visual Studio Code dan Android Studio.

Terakhir nih, ada tech stack yang namanya **cross-compiled**. Kalau tadi PWA cuma bikin website jadi punya tampilan kaya aplikasi, cross-compiled ini yang ngubah website jadi aplikasi beneran. Salah satu contoh dari cross-compiled ini ada Xamarin yang dikeluarin sama Microsoft.

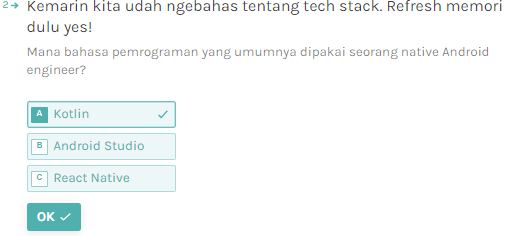
## **Topic 5**

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.

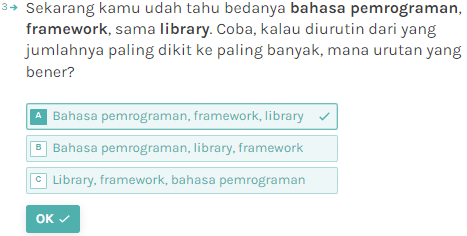
4) Tech stack di database



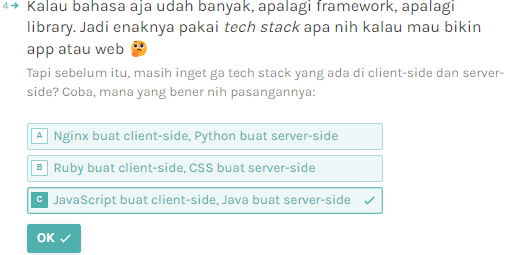
Kotlin adalah salah satu bahasa pemrograman dalam dunia Android app development. Selain dikenal stabil, popularitasnya makin melambung sehabis Google ngumumin Kotlin jadi bahasa pemrograman resmi buat bikin aplikasi Android di event Google I/O 2017.

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.

Nah 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.



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 dikit di antara 3 pilihan tadi.



bahasa pemrograman gak semua diciptain 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.

Nah, buat nentuin enaknya produkmu pakai tech stack apa, ada caranya nih. Biar gampang, coba kita timbang-timbang pakai 5 kriteria:

1) Type of product

2) Time to Market (TTM)

3) Development cost

4) Security

5) Scalability

Yang pertama, adalah **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 diintegrasi antar platform. Contohnya kayak bikin web e-commerce atau web perusahaan. Nah, kalau mulai susah gini, Java bisa jadi opsi.

c) **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.

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.

a) **Mobile first**

Mending bikin responsive web app yang fit ke semua ukuran dan dimensi. UI-nya simple dan diprioritasin ukuran mobile dulu dibanding laptop. Kelebihannya selain murah, juga lebih gampang iterasinya, karena fleksibel jadi bisa dijadiin tool buat validasi produkmu. Cocok buat: Produk yang kontennya ditargetin buat segmen umum, yang link kontennya sering/gampang dishare, atau kontennya bisa sekalian dipakai buat SEO. Contoh: Vimeo, Scribd, Kumparan.

b) **Mobile only**

Nah, resiko mobile doang 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. 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.

c) **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. Cocok buat: SaaS yang targetnya perusahaan. Jatuhnya, kalau ada app-nya pun, cuma buat suplemen. Contoh: Typeform, MailChimp, Hootsuite.



Kriteria yang kedua, yaitu **TTM (Time To Market).** Makin cepet aplikasi jadi, makin kecil juga biaya development yang dikeluarin. Atau kamu dituntut buat bikin MVP yang waktunya mepet? Coba pertimbangin bbrp 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 diintegrasi 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.

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.

d) **Gampang ngetesnya** gak?

Sederhananya, makin gampang dites, makin singkat waktu developmentnya. Cari tech stack yang support sama berbagai macam alat tes yang banyak tersedia.

Kriteria ketiga, apalagi kalau bukan **biaya development**. Coba petakan mulai dari awal sampai akhir proses maintenance. Contohnya nih, seiring berjalannya 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. Ada pertimbangan lain, yaitu kriteria keempat: **security**. tiap tech stack pasti nyediain guideline tentang security. Jadi, di sini lah pentingnya cari tech stack yang bisa ngasih tahu seluk beluk keamanan teknologinya.

Kriteria terakhir adalah **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:

1) **Horizontal scalability**

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

2) **Vertical scalability**

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

**karakteristik tech stack di server-side.**

1) 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.

2) 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

3) PHP (Framework: Laravel, CodeIgniter, dll)

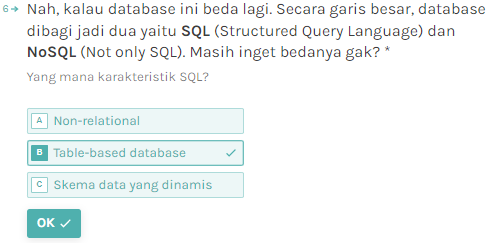
Cocok buat bikin website ringan atau software-as-a-service (SaaS) dengan waktu development yang sangat terbatas. Contoh: Wikipedia, Wordpress, dan Flickr.

4) 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.

5) 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.



jenis tech stack di database SQL dan NOSQL yang direkomendasikan?

**SQL**

1) **MySQL** punya kemampuan handal dan scalable. Cocok buat transaksi yang saling berhubungan, kayak bank dan asuransi.

2) **PostgreSQL** sering dibilang open-source version-nya Orcale karena fitur punya fitur analytics. Cocok dipakai buat industri finance, manufacturing, dan research project.

**NoSQL**

1) **MongoDB** bersifat open-source dan punya customer service yang bagus. Selain itu, MongoDB punya fitur geospasial yang bikin cocok buat hitung jarak dan lokasi.

2) Redis **populer** karena kecepatan dan durabilitasnya. Uniknya, Redis nyimpen data utama di memory server lewat cache web, biar bisa dikonfigurasi dan ditransfer ke disk server.