DIMY TEKNOLOGI

PROGRAMMER QUIZ

horizontal line

**Kategori soal yang tidak memiliki tanda bintang (\*) tidak wajib untuk diisi, tetapi jika diisi akan menjadi nilai tambah**

**(Beres)**

Warmup test\*

1. Dibawah ini terdapat beberapa link untuk melakukan tes dasar algoritma, test ini bisa dilakukan dengan bahasa apa saja (diutamakan Node Js, Go, Python, Java, Kotlin, PHP).

Test dilakukan menggunakan platform HackerRank (<https://www.hackerrank.com>), register atau login terlebih dahulu sebelum mengakses link dibawah. Kirimkan file code hasil test.

Berikut adalah beberapa link test yang harus dikerjakan :

1. Simple Array Sum

<https://www.hackerrank.com/challenges/simple-array-sum>

1. Compare the Triplets

<https://www.hackerrank.com/challenges/compare-the-triplets>

1. A Very Big Sum

<https://www.hackerrank.com/challenges/a-very-big-sum>

1. StairCase

<https://www.hackerrank.com/challenges/staircase>

# Design pattern & System Optimization

1. Aplikasi modern dituntut untuk banyak melakukan banyak task dalam suatu request, sebagai contoh ketika user melakukan register pada suatu aplikasi maka aplikasi tersebut harus mengirim email konfirmasi kepada user, atau memasukan user ke mailing list.

Kegiatan diatas bisa menyebabkan sistem membutuhkan waktu yang lebih untuk memproses semua task tersebut, sedangkan user tidak bisa menunggu terlalu lama sampai task tersebut selesai dilakukan.

Bagaimana solusi yang anda tawarkan untuk menangani hal tersebut ?

# Code Knowledge & Optimization

1. Apa fungsi dan kegunaan dari code refactoring ?
2. Apakah komentar dalam kode berguna ?, sebagian orang menyarankan komentar dalam kode sebaiknya dihindari. Apakah anda setuju dengan hal tersebut ?, jelaskan alasannya.
3. What makes good code good?

Clean code, meminimalisir adanya komentar pada penulisan code, terstruktur, menerapkan prinsip solid, DRY

# Web development

1. Jelaskan tentang API versioning dan kegunaannya, dan bagaimana anda akan memanage API versioning pada Web Services ?

Jawaban :

API versioning adalah praktik mempertahankan versi yang berbeda dari sebuah API. Ini memungkinkan pengembang untuk memperkenalkan perubahan pada API tanpa mempengaruhi aplikasi yang sudah ada yang bergantung pada versi sebelumnya. API versioning penting untuk memastikan keberlanjutan dan stabilitas antarmuka program yang digunakan oleh pengguna.

1. Pemilihan Pendekatan: Tentukan pendekatan versioning yang sesuai untuk kebutuhan Anda, seperti URL versioning, header versioning, query parameter 2. 2. versioning, atau content negotiation. Pilih pendekatan yang paling sesuai dengan arsitektur dan kebutuhan aplikasi Anda.

3. Konsistensi Penamaan Endpoint: Pastikan bahwa endpoint API memiliki pola penamaan yang konsisten dan mudah dipahami, terutama jika menggunakan URL versioning. Misalnya, /v1/users dan /v2/users untuk versi pertama dan kedua dari endpoint pengguna.

4. Manajemen Routing: Mengatur routing pada layer web server atau framework Anda untuk mengarahkan permintaan ke versi API yang tepat berdasarkan URL, header, atau parameter query yang menentukan versi.

5. Penanganan Versi di Kode: Di dalam kode aplikasi, buat struktur yang memisahkan logika dan pemrosesan untuk setiap versi API. Ini memungkinkan pengembangan dan pemeliharaan yang terpisah untuk setiap versi, menjaga kode yang bersih dan terorganisir.

6. Dokumentasi dan Informasi Versi: Sediakan dokumentasi yang jelas dan komprehensif tentang setiap versi API, termasuk perbedaan antara versi, pembaruan, dan pemutakhiran yang terkait. Juga, sediakan informasi tentang bagaimana pengguna dapat mengidentifikasi dan menggunakannya dengan benar.

7. Dukungan Berlanjut: Saat mengenalkan versi baru, pastikan untuk memberikan dukungan berlanjut untuk versi yang sudah ada. Hal ini termasuk memperbaiki bug kritis, menjaga kompatibilitas, dan memberikan informasi yang memadai kepada pengguna tentang transisi ke versi yang lebih baru.

8. Komunikasi dan Depresiasi Versi Lama: Jika Anda berencana untuk mendepresiasi versi API tertentu, komunikasikan dengan jelas kepada pengguna atau pengembang terkait. Berikan pemberitahuan yang memadai, waktu yang cukup untuk beralih, dan panduan tentang versi baru yang direkomendasikan.

# Database

1. How would you explain the recent rise in interest in NoSQL?

# Code collaboration & Team management

1. How would you deal with legacy code?
2. What is the biggest difference between Agile and Waterfall?

# Software development knowledge\*

1. Why is writing software difficult? What makes maintaining software hard?

jawaban

1. Evolution of Requirements: Software needs to adapt to changing business needs and user requirements over time. As new features are added, existing features enhanced, and bugs fixed, maintaining consistency and ensuring compatibility can be challenging.

2. Technical Debt: Over time, software can accumulate technical debt, which refers to compromises made during development that may result in suboptimal or inefficient code. Addressing technical debt and refactoring code without disrupting the existing functionality can be complex.

3. Legacy Systems: Maintenance often involves working with legacy systems or codebases that have been developed by multiple individuals or teams over an extended period. Understanding and modifying existing code can be difficult, especially if it lacks proper documentation or is poorly structured.

4. Regression Issues: Modifying existing code can introduce unintended side effects or regressions, where fixing one issue causes another problem. Proper testing and regression testing strategies are crucial to minimize the risk of introducing new bugs.

5. Versioning and Compatibility: As software interacts with various dependencies, libraries, and platforms, maintaining compatibility across different versions and environments can be challenging. Upgrading dependencies or migrating to newer versions requires careful planning and testing.