**ইন্টারভিউ প্রশ্নাবলী**

**1. PHP & OOP (Object-Oriented Programming)**

**1.1. PHP Basics & Advanced:**

1.1.1. PHP এর \*\*strict\_types\*\* declare করার সুবিধা কী এবং কখন এটি ব্যবহার করবেন?

1.1.2. PHP-তে \*\*Generators\*\* কী এবং কিভাবে এটি মেমরি অপ্টিমাইজেশনে সাহায্য করে?

1.1.3. \*\*Type Hinting\*\* এবং \*\*Return Type Declarations\*\* PHP তে কোডের রিডেবিলিটি ও মেইনটেনেন্স কিভাবে উন্নত করে?

1.1.4. PHP-তে \*\*magic methods\*\* (যেমন: \_\_call, \_\_get, \_\_set) কী এবং তাদের ব্যবহারিক প্রয়োগ উদাহরণসহ ব্যাখ্যা করুন।

1.1.5. \*\*Error handling\*\* এবং \*\*Exception handling\*\* এর মধ্যে পার্থক্য কী? আপনার প্রজেক্টে কিভাবে এগুলো ম্যানেজ করেছেন?

1.1.6. \*\*PSR standards\*\* কী এবং আপনার কোডিং স্টাইলে এর প্রভাব কেমন?

1.1.7. PHP তে \*\*মেথড চেইনিং (Method Chaining)\*\* কী? উদাহরণসহ ব্যাখ্যা করুন এবং এর সুবিধা কী?

1.1.8. PHP \*\*Namespaces\*\* বলতে কী বোঝায়? কেন এবং কিভাবে এটি ব্যবহার করবেন?

1.1.9. PHP তে \*\*access modifiers\*\* (public, protected, private) কী কী? প্রতিটি প্রকার উদাহরণসহ বর্ণনা করুন।

1.1.10. PHP তে `require()` এবং `include()` ফাংশনের মধ্যে পার্থক্য কী? কখন কোনটি ব্যবহার করবেন?

1.1.11. PHP তে `require\_once()` এবং `include\_once()` এর ব্যবহার কী?

1.1.12. PHP তে \*\*NULL\*\* বলতে কী বোঝায়?

1.1.13. PHP তে $variable এবং $$variable এর মধ্যে পার্থক্য কী? উদাহরণসহ ব্যাখ্যা করুন।

**1.2. OOP Concepts:**

1.2.1. \*\*OOP\*\* কী এবং এর ৪টি মূল নীতি কী কী?

1.2.2. \*\*ক্লাস\*\* এবং \*\*অবজেক্টের\*\* মধ্যে পার্থক্য কী?

1.2.3. \*\*ইনহেরিটেন্স\*\* কী এবং আপনি এটি কিভাবে ব্যবহার করেছেন?

1.2.4. \*\*Abstract Class\*\* এবং \*\*Interface\*\* এর মধ্যে পার্থক্য কী?

1.2.5. \*\*Polymorphism\*\* কী? PHP তে একটি উদাহরণ দিন।

1.2.6. PHP তে \*\*Traits\*\* কী? কেন এগুলো ব্যবহার করবেন?

1.2.7. \*\*Magic methods\*\* (\_\_construct, \_\_get, \_\_set) কী?

1.2.8. \*\*Public\*\*, \*\*protected\*\*, এবং \*\*private\*\* এর মধ্যে পার্থক্য কী?

12.9. \*\*Late static binding\*\* কী?

1.2.10. PHP তে \*\*Dependency Injection\*\* কিভাবে কাজ করে?

---

**2. Composer & PHP Ecosystem**

**2.1. Composer Basics**:

2.1.1. \*\*Composer\*\* কী? PHP ইকোসিস্টেমে এর ভূমিকা কী?

2.1.2. \*\*Composer\*\* কিভাবে কাজ করে? এর প্রধান কাজগুলো কী কী?

2.1.3. \*\*`composer.json`\*\* এবং \*\*`composer.lock`\*\* ফাইলের মধ্যে পার্থক্য কী?

2.1.4. \*\*`composer install`\*\* এবং \*\*`composer update`\*\* কমান্ডের মধ্যে পার্থক্য কী? কখন কোনটি ব্যবহার করবেন?

2.1.5. \*\*Autoloading\*\* এর ক্ষেত্রে Composer কিভাবে সাহায্য করে? \*\*PSR-4 autoloading\*\* ব্যাখ্যা করুন।

2.1.6. \*\*Vendor folder\*\* এর উদ্দেশ্য কী?

2.1.7. Composer ব্যবহার করে কিভাবে একটি নতুন PHP প্রজেক্ট শুরু করবেন এবং প্রয়োজনীয় প্যাকেজ যুক্ত করবেন?

2.1.8. \*\*Packagist\*\* কী এবং Composer এর সাথে এর সম্পর্ক কী?

**2.2. Composer Advanced:**

2.2.1. Composer ব্যবহার করে কিভাবে একটি \*\*Custom Package\*\* তৈরি করবেন এবং আপনার প্রোজেক্টে যুক্ত করবেন?

2.2.2. \*\*Composer scripts\*\* কী এবং কিভাবে এগুলো ব্যবহার করবেন?

2.2.3. Composer দিয়ে \*\*Dev Dependencies\*\* এবং \*\*Production Dependencies\*\* কিভাবে ম্যানেজ করবেন?

2.2.4. যদি \*\*Composer install\*\* ব্যর্থ হয়, তাহলে আপনি কিভাবে ডিবাগ করবেন?

2.2.5. Composer এর \*\*`dump-autoload`\*\* কমান্ডের কাজ কী? কখন এটি ব্যবহার করা হয়?

2.2.6. আপনার \*\*CI/CD pipeline\*\* এ Composer কিভাবে ব্যবহার করেন?

---

**3. Laravel**

**3.1. Laravel Core:**

3.1.1. Laravel \*\*Request Lifecycle\*\* বিস্তারিত ব্যাখ্যা করুন।

3.1.2. \*\*Service Providers\*\* এবং \*\*Service Container\*\* Laravel এর আর্কিটেকচারে কি ভূমিকা পালন করে?

3.1.3. \*\*Eloquent ORM\*\* এ \*\*N+1 problem\*\* কী এবং কিভাবে এটি সমাধান করবেন?

3.1.4. \*\*Middleware\*\* কী? আপনার প্রোজেক্টে ব্যবহার করা একটি কাস্টম Middleware এর উদাহরণ দিন।

3.1.5. Laravel এর \*\*Queue System\*\* কিভাবে কাজ করে? আপনার \*\*Data Sync Service\*\* এ কিভাবে এটি ব্যবহার করেছেন এবং এর সুবিধা কী?

3.1.6. \*\*Artisan Commands\*\* এর ব্যবহার এবং গুরুত্ব আলোচনা করুন।

3.1.7. Laravel এর \*\*Authentication\*\* এবং \*\*Authorization\*\* কিভাবে ইমপ্লিমেন্ট করেন? (যেমন: Gates, Policies, Sanctum/Passport)।

**3.2. Advanced Laravel:**

3.2.1. Laravel \*\*Service Container\*\* কী?

3.2.2. \*\*Service Provider\*\* এবং \*\*Facade\*\* এর মধ্যে পার্থক্য কী?

3.2.3. \*\*Middleware\*\* কী এবং আপনি এটি কিভাবে ব্যবহার করেন?

3.2.4. Laravel এ \*\*Queues\*\* এবং \*\*Jobs\*\* কিভাবে ব্যবহার করেন?

3.2.5. \*\*Failed jobs\*\* এবং \*\*retries\*\* কিভাবে হ্যান্ডেল করবেন?

3.2.6. Laravel এ \*\*Policies\*\* এবং \*\*Gates\*\* কী?

3.2.7. \*\*Form Request Validation\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

3.2.8. \*\*Observers\*\* কী এবং কখন এগুলো ব্যবহার করবেন?

3.2.9. \*\*Repository Pattern\*\* কী এবং আপনি এটি কিভাবে ব্যবহার করেছেন?

3.2.10. Laravel এবং \*\*Lumen\*\* এর মধ্যে পার্থক্য কী?

3.2.11. Laravel \*\*Events\*\* এবং \*\*Listeners\*\* কিভাবে কাজ করে? কখন এটি ব্যবহার করা উচিত?

3.2.12. Laravel \*\*Broadcasting\*\* কিভাবে \*\*real-time features\*\* ইমপ্লিমেন্টেশনে সাহায্য করে?

3.2.13. Laravel এ \*\*Caching\*\* কিভাবে ব্যবহার করেন? (যেমন: Redis cache, File cache)।

3.2.14. আপনার \*\*Ebaskat\*\* প্রোজেক্টে \*\*Cron Jobs\*\* কিভাবে সেটআপ এবং অপ্টিমাইজ করেছেন?

3.2.15. Database \*\*Migrations\*\* এবং \*\*Seeders\*\* এর গুরুত্ব এবং আপনার অভিজ্ঞতা আলোচনা করুন।

**4. Javascript (General & Advanced)**

**4.1. JS Core Concepts:**

4.1.1. \*\*Execution Context\*\* এবং \*\*Lexical Environment\*\* JavaScript এ কিভাবে কাজ করে?

4.1.2. \*\*Closures\*\* কী? এটি কিভাবে \*\*private variables\*\* এবং \*\*data encapsulation\*\* এ সাহায্য করে? একটি বাস্তব উদাহরণ দিন।

4.1.3. \*\*Hoisting\*\* কী? \*\*var\*\*, \*\*let\*\*, \*\*const\*\* এর সাথে \*\*hoisting\*\* এর পার্থক্য আলোচনা করুন।

4.1.4. \*\*Event Delegation\*\* কী এবং কখন এটি ব্যবহার করা উচিত? এর সুবিধা কী?

4.1.5. \*\*Prototypal Inheritance\*\* এবং \*\*Class-based Inheritance\*\* এর মধ্যে পার্থক্য ব্যাখ্যা করুন। JavaScript এ কিভাবে \*\*inheritance\*\* কাজ করে?

4.1.6. \*\*this\*\* কীওয়ার্ড এর বিভিন্ন \*\*binding rules\*\* (Implicit, Explicit, New, Lexical) উদাহরণসহ ব্যাখ্যা করুন।

**4.2. Asynchronous JS & ES6+:**

4.2.1. \*\*var\*\*, \*\*let\*\*, এবং \*\*const\*\* এর মধ্যে পার্থক্য কী?

4.2.2. উদাহরণসহ \*\*Closure\*\* কী তা ব্যাখ্যা করুন।

4.2.3. \*\*Hoisting\*\* ব্যাখ্যা করুন।

4.2.4. \*\*Synchronous\*\* এবং \*\*Asynchronous JS\*\* এর মধ্যে পার্থক্য কী?

4.2.5. \*\*Promises\*\* কী এবং আপনি কিভাবে ত্রুটি (errors) হ্যান্ডেল করেন?

4.2.6. \*\*Event Loop\*\* এবং \*\*Call Stack\*\* কী?

4.2.7. `map()`, `filter()`, `reduce()` বনাম `forEach()` ব্যাখ্যা করুন।

4.2.8. \*\*Arrow functions\*\* কী?

4.2.9. JS এ \*\*Destructuring\*\* কী?

4.2.10. \*\*Event Delegation\*\* ব্যাখ্যা করুন।

4.2.11. \*\*Promises\*\* এবং \*\*async/await\*\* এর মধ্যে পার্থক্য কী? জটিল Asynchronous কোড হ্যান্ডেল করার জন্য কোনটি আপনার পছন্দ এবং কেন?

4.2.12. \*\*Event Loop\*\*, \*\*Call Stack\*\*, \*\*Web APIs\*\*, \*\*Callback Queue\*\* কিভাবে Asynchronous JavaScript কোড এক্সিকিউট করে?

4.2.13. \*\*ES6 Modules\*\* (import/export) কিভাবে কাজ করে এবং এর সুবিধা কী?

4.2.14. \*\*Generators\*\* এবং \*\*Iterators\*\* JavaScript এ কী? কখন এগুলো ব্যবহার করবেন?

4.2.15. \*\*Proxy\*\* এবং \*\*Reflect\*\* API গুলো কী এবং তাদের ব্যবহার কী?

---

**5. NodeJS & NestJS**

**5.1. NodeJS (General):**

5.1.1. Node.js এর \*\*Event Loop\*\* কিভাবে কাজ করে তা বিস্তারিত ব্যাখ্যা করুন। \*\*microtasks\*\* এবং \*\*macrotasks\*\* এর মধ্যে পার্থক্য কী?

5.1.2. \*\*Blocking\*\* এবং \*\*Non-blocking I/O\*\* এর মধ্যে পার্থক্য উদাহরণসহ বুঝিয়ে দিন।

5.1.3. Node.js এ \*\*Asynchronous programming\*\* কিভাবে হ্যান্ডেল করেন? \*\*Callbacks\*\*, \*\*Promises\*\*, \*\*Async/Await\*\* এর সুবিধা-অসুবিধা আলোচনা করুন।

5.1.4. Node.js \*\*Streams\*\* কী এবং কখন আপনি এটি ব্যবহার করবেন? \*\*Readable\*\*, \*\*Writable\*\*, \*\*Duplex\*\* এবং \*\*Transform\*\* স্ট্রিমগুলি উদাহরণসহ ব্যাখ্যা করুন।

5.1.5. `process.nextTick()` এবং `setImmediate()` এর মধ্যে পার্থক্য কী?

5.1.6. Node.js অ্যাপ্লিকেশনকে \*\*Scalable\*\* করার জন্য আপনি কী কী পদ্ধতি অবলম্বন করবেন? (যেমন: Clustering, PM2, Load Balancing)।

**5.2. NestJS (Framework Specific):**

5.2.1. \*\*NodeJS\*\* কী? এটি কোথায় ব্যবহৃত হয়?

5.2.2. \*\*NodeJS\*\* এবং \*\*Express\*\* এর মধ্যে পার্থক্য কী?

5.2.3. \*\*NestJS\*\* কী এবং কেন Express এর চেয়ে এটি ব্যবহার করবেন?

5.2.4. NestJS এ \*\*Modules\*\*, \*\*Providers\*\*, \*\*Controllers\*\* ব্যাখ্যা করুন।

5.2.5. NestJS এ \*\*Exceptions\*\* কিভাবে হ্যান্ডেল করবেন?

5.2.6. NestJS এ একটি \*\*Guard\*\* কী?

5.2.7. NestJS এ একটি \*\*Interceptor\*\* কী?

5.2.8. NestJS এ \*\*JWT authentication\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

5.2.9. NestJS এ \*\*Dependency Injection\*\* কী?

5.2.10. NestJS এ \*\*Decorators\*\* (যেমন: @Controller, @Injectable) কী?

5.2.11. আপনার \*\*AT SMS\*\* প্রোজেক্টে কিভাবে \*\*Microservices\*\* আর্কিটেকচার NestJS দিয়ে ইমপ্লিমেন্ট করেছেন? কোন \*\*transport layer\*\* (যেমন: TCP, Redis, Kafka) ব্যবহার করেছেন এবং কেন?

5.2.12. NestJS এ \*\*Custom Decorators\*\* বা \*\*Custom Pipes\*\* কিভাবে তৈরি করবেন? একটি ব্যবহারিক উদাহরণ দিন।

5.2.13. NestJS অ্যাপ্লিকেশনে \*\*Unit Testing\*\* এবং \*\*End-to-End Testing\*\* কিভাবে করেন?

5.2.14. NestJS এ \*\*Configuration Management\*\* কিভাবে করেন? (যেমন: `ConfigModule`, `.env` ফাইল)।

5.2.15. \*\*WebSockets\*\* ব্যবহার করে NestJS এ \*\*Real-time communication\*\* কিভাবে ইমপ্লিমেন্ট করেছেন?

---

**6. React JS & Next.js**

**6.1. React JS:**

6.1.1. \*\*Virtual DOM\*\* কী এবং কিভাবে React এটি ব্যবহার করে পারফরম্যান্স অপ্টিমাইজ করে?

6.1.2. \*\*React Hooks\*\* কী? `useState`, `useEffect`, `useContext`, `useCallback`, `useMemo`, `useRef` এর ব্যবহার এবং উদ্দেশ্য ব্যাখ্যা করুন।

6.1.3. \*\*Controlled Components\*\* এবং \*\*Uncontrolled Components\*\* এর মধ্যে পার্থক্য কী? কখন কোনটি ব্যবহার করবেন?

6.1.4. \*\*Context API\*\* কী? কখন এটি \*\*prop drilling\*\* এর বিকল্প হিসেবে ব্যবহার করবেন?

6.1.5. \*\*Component Lifecycle\*\* কি এবং \*\*functional components\*\* এ `useEffect` কিভাবে Lifecycle methods এর কাজ করে?

6.1.6. \*\*Error Boundaries\*\* কী এবং কিভাবে React অ্যাপ্লিকেশনে এটি ইমপ্লিমেন্ট করবেন?

6.1.7. React এ \*\*Performance Optimization\*\* এর জন্য আপনি কী কী কৌশল ব্যবহার করেন?

6.1.8. \*\*Higher-Order Components (HOCs)\*\* এবং \*\*Render Props\*\* প্যাটার্নগুলো ব্যাখ্যা করুন। Hooks কিভাবে এগুলোকে আরও সহজ করেছে?

6.1.9. React এ \*\*Components\*\* কী?

6.1.10. \*\*Functional\*\* এবং \*\*Class Component\*\* এর মধ্যে পার্থক্য কী?

6.1.11. Hooks: `useState`, `useEffect`, `useContext` ব্যাখ্যা করুন।

6.1.12. \*\*State Lifting\*\* কী?

6.1.13. \*\*Prop drilling\*\* কী এবং কিভাবে এটি এড়ানো যায়?

**6.2. Next.js:**

6.2.1. \*\*SSR\*\*, \*\*CSR\*\*, এবং \*\*SSG\*\* এর মধ্যে পার্থক্য কী?

6.2.2. Next.js এ `getServerSideProps` কী?

6.2.3. Next.js এ \*\*Authentication\*\* কিভাবে হ্যান্ডেল করবেন?

6.2.4. আপনার Laravel প্রোজেক্টে কেন \*\*Inertia.js\*\* ব্যবহার করেছেন?

6.2.5. \*\*React Router\*\* এবং \*\*Route Protection\*\* ব্যাখ্যা করুন।

6.2.6. \*\*Server-Side Rendering (SSR)\*\*, \*\*Static Site Generation (SSG)\*\*, এবং \*\*Incremental Static Regeneration (ISR)\*\* এর মধ্যে পার্থক্য কী? প্রতিটি ব্যবহারের সেরা পরিস্থিতি ব্যাখ্যা করুন।

6.2.7. Next.js এ \*\*Data Fetching\*\* এর বিভিন্ন পদ্ধতি (যেমন: `getServerSideProps`, `getStaticProps`, `getStaticPaths`, client-side fetching) বিস্তারিত আলোচনা করুন।

6.2.8. \*\*File-system based routing\*\* Next.js এ কিভাবে কাজ করে? \*\*Dynamic routes\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

6.2.9. \*\*API Routes\*\* Next.js এ কিভাবে কাজ করে? কখন এগুলো ব্যবহার করবেন?

6.2.10. Next.js \*\*Middleware\*\* কী এবং এর ব্যবহার কী?

6.2.11. `next/image` এবং `next/head` কম্পোনেন্টগুলির ব্যবহার এবং সুবিধা কী?

6.2.12. \*\*App Router\*\* এবং \*\*Pages Router\*\* এর মধ্যে পার্থক্য কী? আপনি কেন একটির পরিবর্তে অন্যটি বেছে নেবেন?

6.2.13. Next.js অ্যাপ্লিকেশন \*\*Deployment\*\* এর জন্য কিভাবে প্রস্তুত করেন?

---

**7. SaaS & Microservice Architecture**

**7.1. SaaS (Software as a Service):**

7.1.1. \*\*Multi-tenancy\*\* কী? \*\*Multi-tenant SaaS\*\* অ্যাপ্লিকেশনের জন্য \*\*Database Schema Design\*\* এ আপনি কী কী বিষয় বিবেচনা করবেন? (যেমন: Shared database, separate schema, separate database)।

7.1.2. \*\*AT SMS\*\* এর মতো একটি \*\*SaaS\*\* প্ল্যাটফর্মে \*\*data isolation\*\* এবং \*\*security\*\* কিভাবে নিশ্চিত করেছেন?

7.1.3. SaaS অ্যাপ্লিকেশনের \*\*scaling strategies\*\* কী কী?

7.1.4. SaaS অ্যাপ্লিকেশন \*\*upgrades\*\* এবং \*\*maintenance\*\* কিভাবে হ্যান্ডেল করেন যাতে ইউজারদের কাজের বিঘ্ন না ঘটে?

**7.2. Microservice:**

7.2.1. \*\*SaaS\*\* কী? ডেভেলপমেন্টে মূল চ্যালেঞ্জগুলো কী কী?

7.2.2. \*\*Monolithic\*\* এবং \*\*Microservice Architecture\*\* এর মধ্যে পার্থক্য কী?

7.2.3. \*\*Microservices\*\* কিভাবে একে অপরের সাথে যোগাযোগ করে?

7.2.4. Microservices এ \*\*Service Failures\*\* কিভাবে ম্যানেজ করবেন?

7.2.5. \*\*Service Discovery\*\* কী?

7.2.6. \*\*Multi-service architecture\*\* এ \*\*Auth\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

7.2.7. \*\*Services\*\* এবং \*\*Logs\*\* কিভাবে মনিটর করবেন?

7.2.8. Microservices এ \*\*Database per Service\*\* কিভাবে হ্যান্ডেল করবেন?

7.2.9. \*\*Domain-Driven Design (DDD)\*\* এর সুবিধাগুলো কী কী?

7.2.10. আপনার \*\*AT SMS microservices\*\* এর ডিজাইন ব্যাখ্যা করুন।

7.2.11. \*\*Monolithic\*\* থেকে \*\*Microservices\*\* আর্কিটেকচারে স্থানান্তরের প্রধান কারণ এবং চ্যালেঞ্জগুলি কী কী?

7.2.12. \*\*Inter-service communication\*\* এর জন্য আপনি কী কী পদ্ধতি ব্যবহার করেছেন? (যেমন: Synchronous REST/gRPC, Asynchronous message queues/event bus)।

7.2.13. \*\*Service Discovery\*\* এবং \*\*API Gateway\*\* প্যাটার্নগুলি Microservices এ কিভাবে সাহায্য করে?

7.2.14. \*\*Distributed Transactions\*\* এবং \*\*Data Consistency\*\* Microservices এ কিভাবে ম্যানেজ করেন? (যেমন: Saga Pattern, eventual consistency)।

7.2.15. Microservices এর জন্য \*\*monitoring\*\*, \*\*logging\*\* এবং \*\*tracing\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

7.2.16. আপনার \*\*AT SMS\*\* প্রোজেক্টে প্রতিটি \*\*Microservice\*\* এর জন্য \*\*domain boundary\*\* কিভাবে নির্ধারণ করেছেন?

7.2.17. Microservices এর \*\*deployment\*\* এবং \*\*orchestration\*\* কিভাবে ম্যানেজ করেন? (যেমন: Docker, Kubernetes)।

---

**8. Databases (MySQL, MongoDB, PostgreSQL, Redis)**

**8.1. General Database Concepts:**

8.1.1. \*\*ACID properties\*\* কী এবং \*\*relational databases\*\* এ এর গুরুত্ব কী?

8.1.2. \*\*Database Normalization\*\* এবং \*\*Denormalization\*\* এর মধ্যে পার্থক্য কী? কখন কোনটি ব্যবহার করবেন?

8.1.3. \*\*Indexing\*\* কী? বিভিন্ন ধরণের ইনডেক্স এবং তাদের ব্যবহারের সুবিধা-অসুবিধা আলোচনা করুন।

8.1.4. \*\*Database Transactions\*\* কী? \*\*Isolation Levels\*\* গুলো ব্যাখ্যা করুন।

8.1.5. \*\*Database Security\*\* এর জন্য আপনি কী কী পদক্ষেপ গ্রহণ করেন?

8.1.6. \*\*Database Backup\*\* এবং \*\*Recovery Strategies\*\* সম্পর্কে আপনার অভিজ্ঞতা কী?

**8.2. MongoDB (NoSQL):**

8.2.1. \*\*Normalization\*\* কী? Normal Form এর প্রকারভেদ কী কী?

8.2.2. \*\*INNER JOIN\*\*, \*\*LEFT JOIN\*\*, \*\*RIGHT JOIN\*\* এর মধ্যে পার্থক্য কী?

8.2.3. \*\*Indexes\*\* কী? কিভাবে তারা পারফরম্যান্স অপ্টিমাইজ করে?

8.2.4. Laravel এ \*\*Raw Queries\*\* কিভাবে লিখবেন?

8.2.5. MySQL এ \*\*ACID Transaction\*\* কী?

8.2.6. \*\*MongoDB vs MySQL\*\* – কখন কোনটি ব্যবহার করবেন?

8.2.7. \*\*Embedded\*\* এবং \*\*Referenced Documents\*\* এর মধ্যে পার্থক্য কী?

8.2.8. MongoDB তে \*\*Aggregation Pipeline\*\* কী?

8.2.9. Laravel এ \*\*Redis Caching\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

8.2.10. Laravel/NestJS এ \*\*Redis Queues\*\* কিভাবে ব্যবহার করবেন?

8.2.11. \*\*MongoDB\*\* কখন \*\*relational database\*\* এর চেয়ে বেশি উপযোগী?

8.2.12. \*\*Document-oriented model\*\* এর সুবিধা কী?

8.2.13. \*\*Sharding\*\* এবং \*\*Replication\*\* MongoDB তে কিভাবে কাজ করে এবং কেন এগুলি ব্যবহার করা হয়?

8.2.14. \*\*Aggregation Pipeline\*\* কী এবং কিভাবে এটি ব্যবহার করে জটিল ডেটা প্রসেস করবেন?

8.2.15. \*\*Embedded Documents\*\* এবং \*\*Referenced Documents\*\* এর মধ্যে পার্থক্য কী? কখন কোনটি ব্যবহার করবেন?

**8.3. MySQL & PostgreSQL (Relational):**

8.3.1. \*\*SQL Join\*\* এর বিভিন্ন প্রকার (INNER, LEFT, RIGHT, FULL) উদাহরণসহ ব্যাখ্যা করুন।

8.3.2. \*\*Stored Procedures\*\*, \*\*Functions\*\* এবং \*\*Triggers\*\* এর মধ্যে পার্থক্য কী?

8.3.3. \*\*Database Views\*\* কী এবং তাদের ব্যবহার কী?

8.3.4. \*\*Query Optimization\*\* এর জন্য আপনি কী কী টুলস এবং কৌশল ব্যবহার করেন? (যেমন: EXPLAIN কমান্ড)।

8.3.5. \*\*Indexes\*\* কিভাবে \*\*query performance\*\* উন্নত করে এবং অতিরিক্ত ইনডেক্সিং এর অসুবিধা কী?

8.3.6. MySQL এর \*\*InnoDB\*\* এবং \*\*MyISAM\*\* storage engines এর মধ্যে পার্থক্য কী?

**8.4. Redis:**

8.4.1. \*\*Redis\*\* কী এবং এটি কিভাবে \*\*in-memory data store\*\* হিসেবে কাজ করে?

8.4.2. \*\*Redis\*\* এর বিভিন্ন \*\*data structures\*\* (Strings, Hashes, Lists, Sets, Sorted Sets) উদাহরণসহ ব্যাখ্যা করুন।

8.4.3. \*\*Caching\*\* ছাড়াও \*\*Redis\*\* এর অন্যান্য ব্যবহার কী কী? (যেমন: Session Management, Real-time analytics, Leaderboards, Message Broker)।

8.4.4. Redis \*\*persistence\*\* অপশনগুলো (RDB, AOF) কী এবং তাদের মধ্যে পার্থক্য কী?

8.4.5. \*\*Redis\*\* ব্যবহার করে \*\*rate limiting\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

---

**9. REST API, JWT, OAuth2**

**9.1. RESTful API:**

9.1.1. \*\*RESTful API\*\* কী? মূল নীতিগুলো কী কী?

9.1.2. \*\*PUT\*\*, \*\*PATCH\*\*, \*\*POST\*\* এর মধ্যে পার্থক্য কী?

9.1.3. \*\*JWT Authentication\*\* কিভাবে কাজ করে?

9.1.4. \*\*Refresh Tokens\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

9.1.5. \*\*JWT\*\* এবং \*\*OAuth2\*\* এর মধ্যে পার্থক্য কী?

9.1.6. \*\*OAuth2\*\* এ \*\*Scopes\*\* কী?

9.1.7. \*\*API Endpoints\*\* কিভাবে সুরক্ষিত করবেন?

9.1.8. \*\*API Rate Limiting\*\* কী এবং Laravel/NestJS এ কিভাবে এটি প্রয়োগ করবেন?

9.1.9. API তে \*\*Versioning\*\* কিভাবে হ্যান্ডেল করবেন?

9.1.10. \*\*CORS\*\* কী এবং কিভাবে এটি সমাধান করবেন?

**9.2. JWT (JSON Web Token):**

9.2.1. \*\*JWT\*\* কিভাবে কাজ করে? এর \*\*structure\*\* (Header, Payload, Signature) ব্যাখ্যা করুন।

9.2.2. \*\*JWT\*\* এর সুবিধা এবং অসুবিধা কী?

9.2.3. \*\*JWT\*\* দিয়ে \*\*authentication\*\* এবং \*\*authorization\*\* কিভাবে ইমপ্লিমেন্ট করবেন?

9.2.4. \*\*JWT security\*\* এর জন্য আপনি কী কী বিষয় বিবেচনা করবেন? (secret key management, token expiration)।

9.2.5. \*\*Refresh Tokens\*\* এর উদ্দেশ্য কী?

**9.3. OAuth2:**

9.3.1. \*\*OAuth2\*\* কী এবং এটি কিভাবে \*\*delegated authorization\*\* প্রদান করে?

9.3.2. \*\*OAuth2\*\* এর বিভিন্ন \*\*Grant Types\*\* (যেমন: Authorization Code, Client Credentials, Implicit) ব্যাখ্যা করুন এবং কখন কোনটি ব্যবহার করবেন?

9.3.3. \*\*OAuth2\*\* এবং \*\*OpenID Connect\*\* এর মধ্যে পার্থক্য কী?

9.3.4. আপনার \*\*Subscription Management System\*\* এ \*\*Stripe\*\* এর সাথে \*\*OAuth2\*\* বা অনুরূপ অথরাইজেশন কিভাবে ব্যবহার করেছেন?

---

**10. DevOps, GIT, CI/CD, Jira, Trello**

**10.1. GIT Concepts:**

10.1.1. আপনার \*\*Git Workflow\*\* (feature branch, merge) কী?

10.1.2. \*\*Merge Conflicts\*\* কিভাবে সমাধান করবেন?

10.1.3. একটি \*\*Git Commit\*\* কিভাবে revert করবেন?

10.1.4. \*\*CI/CD\*\* এর জন্য \*\*GitHub Actions\*\* কিভাবে ব্যবহার করবেন?

10.1.5. \*\*Jira\*\* ব্যবহার করে অগ্রগতি কিভাবে ট্র্যাক করবেন?

10.1.6. \*\*Jira\*\* এবং \*\*Trello\*\* এর মধ্যে পার্থক্য কী?

10.1.7. `.gitignore` কীসের জন্য ব্যবহৃত হয়?

10.1.8. \*\*Pull Request\*\* কী এবং আপনি কিভাবে কোড রিভিউ করেন?

10.1.9. \*\*Rebasing\*\* বনাম \*\*Merging\*\* কী?

10.1.10. \*\*Environment Variables\*\* dev/stage/prod জুড়ে কিভাবে ম্যানেজ করবেন?

**10.2. Platform Specifics:**

10.2.1. \*\*Git workflow\*\* এর বিভিন্ন মডেল (যেমন: Git Flow, GitHub Flow, Trunk-based development) আলোচনা করুন। আপনি কোনটি পছন্দ করেন এবং কেন?

10.2.2. \*\*Rebase\*\* এবং \*\*Merge\*\* এর মধ্যে পার্থক্য কী? কখন কোনটি ব্যবহার করবেন?

10.2.3. \*\*Merge Conflicts\*\* কিভাবে সমাধান করবেন?

10.2.4. \*\*Git revert\*\*, \*\*Git reset\*\*, \*\*Git cherry-pick\*\* এর ব্যবহার এবং তাদের মধ্যে পার্থক্য কী?

10.2.5. \*\*Git hooks\*\* কী এবং তাদের ব্যবহার কী?

10.2.6. \*\*Pull Requests\*\* / \*\*Merge Requests\*\* এর মাধ্যমে কোড রিভিউ প্রসেস কিভাবে ম্যানেজ করেন?

10.2.7. \*\*Github Actions\*\* বা \*\*Bitbucket Pipelines\*\* এর সাথে আপনার অভিজ্ঞতা আছে কি? যদি থাকে, কিভাবে ব্যবহার করেছেন?

10.2.8. \*\*Branching strategy\*\* এর গুরুত্ব কী এবং আপনার প্রজেক্টে কোন \*\*branching strategy\*\* ব্যবহার করেছেন?

---

**11. Project Management & Communication Tools (Jira, Trello)**

**11.1. Workflow & Management:**

11.1.1. \*\*Agile methodologies\*\* (যেমন: Scrum, Kanban) সম্পর্কে আপনার জ্ঞান কী? আপনি কোন মেথডলজি পছন্দ করেন এবং কেন?

11.1.2. \*\*Jira\*\* বা \*\*Trello\*\* ব্যবহার করে আপনি কিভাবে \*\*project backlog\*\* এবং \*\*sprint planning\*\* করেন?

11.1.3. \*\*Task estimation\*\* কিভাবে করেন? (যেমন: Story Points, T-shirt sizing)।

11.1.4. \*\*Jira\*\* বা \*\*Trello\*\* ব্যবহার করে \*\*team communication\*\* এবং \*\*collaboration\*\* কিভাবে উন্নত করেন?

11.1.5. \*\*Bug tracking\*\* এবং \*\*issue resolution\*\* এর জন্য এই টুলসগুলো কিভাবে ব্যবহার করেছেন?

---

**12. Third-Party Integrations (Mailchimp, Mailgun, Pusher, Twilio)**

**12.1. Email Services (Mailchimp, Mailgun):**

12.1.1. Laravel এ \*\*Mailgun\*\* দিয়ে কিভাবে ইমেল পাঠাবেন?

12.1.2. \*\*Transactional\*\* এবং \*\*Marketing Email\*\* এর মধ্যে পার্থক্য কী?

12.1.3. \*\*Mailgun\*\* এবং \*\*Mailchimp\*\* এর মধ্যে পার্থক্য কী?

12.1.4. \*\*Twilio\*\* ব্যবহার করে কিভাবে \*\*SMS\*\* বা \*\*OTP\*\* পাঠাবেন?

12.1.5. \*\*Twilio\*\* থেকে \*\*Delivery Failures\*\* কিভাবে হ্যান্ডেল করবেন?

12.1.6. \*\*WebSocket\*\* কী এবং \*\*Pusher\*\* কিভাবে এটি ব্যবহার করে?

12.1.7. \*\*Pusher\*\* দিয়ে কিভাবে \*\*Real-time Chat Messages\*\* পাঠাবেন?

12.1.8. \*\*Twilio Webhooks\*\* কিভাবে হ্যান্ডেল করবেন?

12.1.9. \*\*Queues\*\* ব্যবহার করে \*\*Email Sending\*\* কিভাবে থ্রোটল করবেন?

12.1.10. কোড থেকে \*\*SMS/Email\*\* পাঠানোর সময় কী কী নিরাপত্তা উদ্বেগ বিদ্যমান?

**12.2. Real-time & Messaging (Pusher, Twilio):**

12.2.1. \*\*Pusher\*\* ব্যবহার করে আপনার প্রোজেক্টে \*\*real-time notifications\*\* বা \*\*chat system\*\* কিভাবে ইমপ্লিমেন্ট করেছেন?

12.2.2. \*\*Twilio\*\* ব্যবহার করে \*\*SMS notifications\*\* বা \*\*voice calls\*\* কিভাবে ইমপ্লiment করেছেন?

12.2.3. \*\*Webhooks\*\* এর গুরুত্ব কী যখন \*\*third-party services\*\* (যেমন: Stripe, Twilio) এর সাথে ইন্টিগ্রেট করছেন?

---

**13. Real-World Problem-Solving / Scenario**

**13.1. Scenario-Based Questions:**

13.1.1. রেট লিমিটের কারণে \*\*Product Sync\*\* ব্যর্থ হলে আপনি কিভাবে এটি হ্যান্ডেল করবেন?

13.1.2. Redis এর মেমরি ব্যবহার বেশি হলে আপনি কী পরীক্ষা করবেন?

13.1.3. Stripe Webhook ব্যর্থ হলে – আপনি কিভাবে ডিবাগ এবং রিট্রাই করবেন?

13.1.4. BigBuy এর পণ্যের দাম হঠাৎ পরিবর্তিত হলে – আপনি কিভাবে নিরাপদে সিঙ্ক করবেন?

13.1.5. একটি Queue Job আটকে গেলে – আপনি কিভাবে এটি সনাক্ত এবং ঠিক করবেন?

13.1.6. Deployment এর পর গ্রাহকরা \*\*500 Errors\*\* পেলে – আপনার পদক্ষেপগুলো কী হবে?

13.1.7. Production এ একটি রিলিজ কিভাবে \*\*রোল ব্যাক\*\* করবেন?

13.1.8. Third-party services থেকে \*\*API ব্যর্থতা\*\* কিভাবে হ্যান্ডেল করবেন?

13.1.9. Subscription Status আপডেট না হলে – \*\*Webhook Chain\*\* কিভাবে ডিবাগ করবেন?

13.1.10. Production এ \*\*Slow SQL Queries\*\* কিভাবে ডিবাগ করবেন?

13.1.11. একটি রিয়েল-টাইম চ্যাট অ্যাপ্লিকেশনে মেসেজ ডেলিভারি নিশ্চিত করতে আপনি কী কৌশল অবলম্বন করবেন?

13.1.12. একটি ই-কমার্স সাইটে হঠাৎ করে প্রচুর ট্র্যাফিক এলে (যেমন, ব্ল্যাক ফ্রাইডে সেলে) আপনি পারফরম্যান্স বজায় রাখতে কী পদক্ষেপ নেবেন?

13.1.13. আপনার সিস্টেমে ডেটা কনসিস্টেন্সি (data consistency) নিশ্চিত করতে আপনি কী ধরনের ডিজাইন প্যাটার্ন বা কৌশল ব্যবহার করবেন?

---

**14. 1 Million User's Data Management Strategy**

**14.1. প্রশ্ন: আপনি ১ মিলিয়ন ব্যবহারকারীর ডেটা কিভাবে ম্যানেজ করবেন?**

Managing data for 1 million users is a significant challenge that requires a robust and scalable architecture. Here's how I would approach it, broken down into key areas, considering the technologies you've mentioned in your CV:

১ মিলিয়ন ব্যবহারকারীর ডেটা ম্যানেজ করা একটি বিশাল চ্যালেঞ্জ যার জন্য একটি শক্তিশালী এবং স্কেলেবল আর্কিটেকচার প্রয়োজন। আপনার CV তে উল্লিখিত প্রযুক্তিগুলো বিবেচনা করে, আমি কীভাবে এটি সামলাব তার একটি বিস্তারিত ব্যাখ্যা নিচে দেওয়া হলো:

**14.1. Database Strategy (Primary Data Store)**

14.1.1. ডেটাবেজ স্ট্র্যাটেজি (প্রাথমিক ডেটা স্টোর)

The choice between relational (MySQL, PostgreSQL) and NoSQL (MongoDB) depends on the data structure and access patterns. For a 1 million user base, a hybrid approach or a strong relational database with proper scaling is common.

রিলেশনাল (MySQL, PostgreSQL) এবং NoSQL (MongoDB) এর মধ্যে ডেটাবেজ নির্বাচন ডেটা স্ট্রাকচার এবং অ্যাক্সেস প্যাটার্নের উপর নির্ভর করে। ১ মিলিয়ন ব্যবহারকারীর জন্য, একটি হাইব্রিড অ্যাপ্রোচ বা সঠিক স্কেলিং সহ একটি শক্তিশালী রিলেশনাল ডেটাবেজ সাধারণত ব্যবহৃত হয়।

**14.1.1.1. Schema Design & Normalization/Denormalization**

English: For critical, transactional data (e.g., user profiles, orders, billing), ensure proper \*\*normalization\*\* to reduce redundancy and maintain data integrity. For frequently accessed, read-heavy data, consider strategic \*\*denormalization\*\* (e.g., embedding frequently accessed user details in a related table) to reduce JOINs and improve read performance. Implement comprehensive \*\*indexing\*\* on frequently queried columns (user IDs, timestamps, foreign keys). Use \*\*EXPLAIN\*\* (MySQL/PostgreSQL) to analyze query plans and optimize indexes.

বাংলা: গুরুত্বপূর্ণ, লেনদেন সম্পর্কিত ডেটার জন্য (যেমন: ব্যবহারকারীর প্রোফাইল, অর্ডার, বিলিং), ডেটা রিডান্ডেন্সি কমাতে এবং ডেটার অখণ্ডতা বজায় রাখতে সঠিক \*\*নর্মালাইজেশন\*\* নিশ্চিত করব। ঘন ঘন অ্যাক্সেস করা, রিড-হেভি ডেটার জন্য, কৌশলগত \*\*ডিনর্মালাইজেশন\*\* বিবেচনা করব (যেমন: ঘন ঘন অ্যাক্সেস করা ব্যবহারকারীর বিবরণ একটি সম্পর্কিত টেবিলে এমবেড করা) যাতে JOIN কমানো যায় এবং রিড পারফরম্যান্স উন্নত হয়। ঘন ঘন কোয়েরি করা কলামগুলিতে (ব্যবহারকারী আইডি, টাইমস্ট্যাম্প, ফরেন কি) ব্যাপক \*\*ইনডেক্সিং\*\* প্রয়োগ করব। কোয়েরি প্ল্যান বিশ্লেষণ এবং ইনডেক্স অপ্টিমাইজ করার জন্য \*\*EXPLAIN\*\* (MySQL/PostgreSQL) ব্যবহার করব।

**14.1.1.2. Database Scaling**

English:

\* \*\*Vertical Scaling:\*\* Initially, upgrading server resources (CPU, RAM, faster disks) can provide quick wins. However, this has limits.

\* \*\*Horizontal Scaling (Sharding/Partitioning):\*\* \*\*Sharding\*\* involves distributing data across multiple independent database servers (shards). Each shard holds a subset of the total data. Choosing an effective \*\*sharding key\*\* (e.g., `user\_id`, `company\_id` for SaaS) is crucial for even data distribution. \*\*Partitioning\*\* (within a single database) for very large tables can improve query performance.

বাংলা:

\* \*\*ভার্টিকাল স্কেলিং:\*\* প্রাথমিকভাবে, সার্ভারের রিসোর্স আপগ্রেড করা (CPU, RAM, দ্রুত ডিস্ক) দ্রুত সুবিধা দিতে পারে। তবে এর সীমাবদ্ধতা আছে।

\* \*\*হরাইজন্টাল স্কেলিং (শার্ডিং/পার্টিশনিং):\*\* \*\*শার্ডিং\*\* বলতে একাধিক স্বাধীন ডেটাবেজ সার্ভার (শার্ড) জুড়ে ডেটা বিতরণ করা বোঝায়। প্রতিটি শার্ড মোট ডেটার একটি উপসেট ধারণ করে। একটি কার্যকর \*\*শার্ডিং কী\*\* (যেমন: `user\_id`, SaaS এর জন্য `company\_id`) নির্বাচন করা ডেটা বিতরণের জন্য অত্যন্ত গুরুত্বপূর্ণ। খুব বড় টেবিলের জন্য \*\*পার্টিশনিং\*\* (একক ডেটাবেজের মধ্যে) কোয়েরি পারফরম্যান্স উন্নত করতে পারে।

**14.1.1.3. Replication & High Availability**

English: Implement \*\*Master-Slave/Primary-Replica Replication\*\* to ensure high availability and fault tolerance. This also allows directing \*\*read queries to replica databases\*\*, offloading the primary (read replicas). Implement automated failover mechanisms (e.g., PostgreSQL streaming replication with Patroni, MySQL with Orchestrator).

বাংলা: উচ্চ প্রাপ্যতা (high availability) এবং ফল্ট টলারেন্স (fault tolerance) নিশ্চিত করার জন্য \*\*মাস্টার-স্লেভ/প্রাইমারি-রেপ্লিকা রেপ্লিকেশন\*\* প্রয়োগ করব। এটি \*\*রিড কোয়েরিগুলি রেপ্লিকা ডেটাবেজে\*\* নির্দেশ করারও সুযোগ দেয়, যা প্রাইমারি ডেটাবেজের লোড কমায়। স্বয়ংক্রিয় ফেইলওভার প্রক্রিয়া (যেমন: Patroni সহ PostgreSQL স্ট্রিমিং রেপ্লিকেশন, Orchestrator সহ MySQL) প্রয়োগ করব।

**14.1.1.4. Choosing the Database**

English: \*\*PostgreSQL/MySQL\*\* would be chosen for complex relationships (e.g., AT SMS's job management, AT POS's inventory), strong transactional guarantees, and reporting needs. With proper sharding/replication, they can scale to millions of users. \*\*MongoDB\*\* would be chosen for flexible schema (e.g., Form Builder where fields can vary), large volumes of unstructured/semi-structured data, and high write throughput.

বাংলা: জটিল সম্পর্কের জন্য (যেমন: AT SMS-এর জব ম্যানেজমেন্ট, AT POS-এর ইনভেন্টরি), শক্তিশালী ট্রানজ্যাকশনাল গ্যারান্টি এবং রিপোর্টিং প্রয়োজনের জন্য \*\*PostgreSQL/MySQL\*\* বেছে নেব। সঠিক শার্ডিং/রেপ্লিকেশন সহ, এগুলো লক্ষ লক্ষ ব্যবহারকারীকে স্কেল করতে পারে। ফ্লেক্সিবল স্কিমার জন্য (যেমন: ফর্ম বিল্ডারে যেখানে ক্ষেত্রগুলি ভিন্ন হতে পারে), প্রচুর পরিমাণে আনস্ট্রাকচারড/সেমি-স্ট্রাকচারড ডেটা এবং উচ্চ রাইট থ্রুপুটের জন্য \*\*MongoDB\*\* বেছে নেব।

**14.2. Caching Strategy (Redis)**

14.2.1. ক্যাশিং স্ট্র্যাটেজি (রেডিস)

Redis is essential for dramatically reducing database load and improving response times for 1 million users.

১ মিলিয়ন ব্যবহারকারীর জন্য ডেটাবেজ লোড কমানো এবং রেসপন্স টাইম উন্নত করার জন্য রেডিস (Redis) অপরিহার্য।

14.2.1.1. Data Caching

English: Cache frequently accessed objects (e.g., user profiles, product details) and results of expensive database queries. Use a \*\*cache-aside\*\* pattern. Store user sessions in Redis for scalability, especially in a distributed application. Consider full-page caching for static pages.

বাংলা: ঘন ঘন অ্যাক্সেস করা অবজেক্ট (যেমন: ব্যবহারকারী প্রোফাইল, পণ্যের বিবরণ) এবং ব্যয়বহুল ডেটাবেজ কোয়েরির ফলাফল ক্যাশ করব। একটি \*\*cache-aside\*\* প্যাটার্ন ব্যবহার করব। ব্যবহারকারীর সেশনগুলি রেডিসে সংরক্ষণ করব স্কেলেবিলিটির জন্য, বিশেষ করে একটি ডিস্ট্রিবিউটেড অ্যাপ্লিকেশনে। স্ট্যাটিক পেজগুলির জন্য সম্পূর্ণ পেজ ক্যাশিং বিবেচনা করব।

14.2.1.2. Distributed Caching

English: Ensure Redis runs in a highly available setup (e.g., Redis Sentinel or Redis Cluster) to prevent a single point of failure.

বাংলা: একটি সিঙ্গেল পয়েন্ট অফ ফেইলর (single point of failure) রোধ করতে রেডিসকে উচ্চ প্রাপ্যতা সেটআপে (যেমন: রেডিস সেনটিনেল বা রেডিস ক্লাস্টার) চালানোর ব্যবস্থা করব।

**14.3. Application Architecture (NodeJS/NestJS, Laravel, Microservices)**

14.3.1. অ্যাপ্লিকেশন আর্কিটেকচার (NodeJS/NestJS, Laravel, Microservices)

Your existing microservices architecture is a strong foundation for managing 1 million users.

আপনার বিদ্যমান মাইক্রোসার্ভিসেস আর্কিটেকচার ১ মিলিয়ন ব্যবহারকারীকে পরিচালনা করার জন্য একটি শক্তিশালী ভিত্তি।

14.3.1.1. Microservices

English: Continue to \*\*decompose services\*\* based on business domains for independent scaling. For high-throughput internal communication, consider \*\*message queues\*\* (e.g., RabbitMQ, Kafka) over synchronous REST calls, especially for background tasks.

বাংলা: স্বাধীন স্কেলিংয়ের জন্য ব্যবসায়িক ডোমেনের উপর ভিত্তি করে \*\*সার্ভিসগুলিকে আরও বিভক্ত\*\* করব। উচ্চ থ্রুপুট অভ্যন্তরীণ যোগাযোগের জন্য, সিঙ্ক্রোনাস REST কলের পরিবর্তে \*\*মেসেজ কিউ\*\* (যেমন: RabbitMQ, Kafka) বিবেচনা করব, বিশেষ করে ব্যাকগ্রাউন্ড টাস্কের জন্য।

14.3.1.2. Load Balancing

English: Place a \*\*load balancer\*\* (e.g., Nginx, AWS ELB) in front of backend services to distribute incoming requests across multiple instances, ensuring high availability and load distribution.

বাংলা: ইনকামিং রিকোয়েস্ট একাধিক ইনস্ট্যান্সে বিতরণ করার জন্য ব্যাকএন্ড সার্ভিসগুলির সামনে একটি \*\*লোড ব্যালেন্সার\*\* (যেমন: Nginx, AWS ELB) স্থাপন করব, যা উচ্চ প্রাপ্যতা এবং লোড বিতরণ নিশ্চিত করবে।

14.3.1.3. Asynchronous Processing & Queues

English: Offload any task that doesn't require immediate user feedback (e.g., sending emails via Mailgun, bulk data syncs) to a \*\*queue\*\*. Leverage Laravel's robust queue system with Redis as the driver. For CPU-intensive Node.js tasks, use worker threads or separate worker processes.

বাংলা: যে কোনো কাজ যার জন্য তাৎক্ষণিক ব্যবহারকারীর প্রতিক্রিয়ার প্রয়োজন নেই (যেমন: Mailgun এর মাধ্যমে ইমেল পাঠানো, বাল্ক ডেটা সিঙ্ক) একটি \*\*কিউতে\*\* অফলোড করব। রেডিসকে ড্রাইভার হিসাবে ব্যবহার করে Laravel এর শক্তিশালী কিউ সিস্টেম ব্যবহার করব। CPU-নিবিড় Node.js কাজের জন্য, ওয়ার্কার থ্রেড বা আলাদা ওয়ার্কার প্রসেস ব্যবহার করব।

14.3.1.4. Stateless Backend Services

English: Ensure NestJS/Laravel services are \*\*stateless\*\*. User-specific state should be stored externally (e.g., Redis for sessions, database for user data).

বাংলা: NestJS/Laravel সার্ভিসগুলি \*\*স্টেটলেস\*\* নিশ্চিত করব। ব্যবহারকারী-নির্দিষ্ট স্টেট বাইরে সংরক্ষণ করা উচিত (যেমন: সেশনের জন্য রেডিস, ব্যবহারকারীর ডেটার জন্য ডেটাবেজ)।

14.3.1.5. API Gateway

English: A \*\*centralized API Gateway\*\* (e.g., Nginx, or a custom NestJS service) can handle authentication (JWT validation), rate limiting, request routing, and logging before requests hit individual microservices.

বাংলা: একটি \*\*কেন্দ্রীয় API গেটওয়ে\*\* (যেমন: Nginx, বা একটি কাস্টম NestJS সার্ভিস) প্রমাণীকরণ (JWT যাচাইকরণ), রেট লিমিটিং, রিকোয়েস্ট রাউটিং এবং লগিং হ্যান্ডেল করতে পারে রিকোয়েস্টগুলি পৃথক মাইক্রোসার্ভিসগুলিতে পৌঁছানোর আগে।

**14.4. Infrastructure & Monitoring**

14.4.1. অবকাঠামো ও মনিটরিং

14.4.1.1. Cloud Providers

English: Utilize \*\*cloud services\*\* (AWS, Google Cloud, Azure) for their scalability, managed database services (RDS, DocumentDB), load balancers, and monitoring tools.

বাংলা: স্কেলেবিলিটি, ম্যানেজড ডেটাবেজ সার্ভিস (RDS, DocumentDB), লোড ব্যালেন্সার এবং মনিটরিং টুলসের জন্য \*\*ক্লাউড সার্ভিস\*\* (AWS, Google Cloud, Azure) ব্যবহার করব।

14.4.1.2. Containerization (Docker) & Orchestration (Kubernetes - if needed)

English: \*\*Docker\*\* for containerizing microservices. \*\*Kubernetes (K8s)\*\* for very large-scale deployments to automate deployment, scaling, and management of containerized applications.

বাংলা: মাইক্রোসার্ভিসগুলি কন্টেইনারাইজ করার জন্য \*\*Docker\*\* ব্যবহার করব। খুব বড় আকারের স্থাপনার জন্য \*\*Kubernetes (K8s)\*\* ব্যবহার করব কন্টেইনারাইজড অ্যাপ্লিকেশনগুলির স্থাপনা, স্কেলিং এবং পরিচালনার জন্য।

14.4.1.3. Monitoring & Alerting

English: Implement robust \*\*monitoring\*\* for all layers: application performance (APM - e.g., New Relic), database metrics, server resources. Set up \*\*alerts\*\* for critical thresholds. Utilize \*\*Distributed Tracing\*\* (e.g., Jaeger) in microservices to debug performance.

বাংলা: সমস্ত স্তরের জন্য শক্তিশালী \*\*মনিটরিং\*\* প্রয়োগ করব: অ্যাপ্লিকেশন পারফরম্যান্স (APM - যেমন: New Relic), ডেটাবেজ মেট্রিক্স, সার্ভার রিসোর্স। গুরুতর থ্রেশহোল্ডের জন্য \*\*সতর্কতা\*\* সেটআপ করব। পারফরম্যান্স ডিবাগ করার জন্য মাইক্রোসার্ভিসগুলিতে \*\*ডিস্ট্রিবিউটেড ট্রেসিং\*\* (যেমন: Jaeger) ব্যবহার করব।

14.4.1.4. Logging

English: \*\*Centralized logging\*\* (e.g., ELK stack or a managed service) is essential for debugging and auditing in a distributed system.

বাংলা: একটি ডিস্ট্রিবিউটেড সিস্টেমে ডিবাগিং এবং নিরীক্ষণের জন্য \*\*কেন্দ্রীয় লগিং\*\* (যেমন: ELK স্ট্যাক বা একটি পরিচালিত পরিষেবা) অপরিহার্য।

**14.5. Security Considerations**

14.5.1. নিরাপত্তা বিবেচনা

14.5.1.1. JWT/OAuth2

English: Continue leveraging \*\*JWT\*\* for stateless authentication. Use \*\*OAuth2\*\* for third-party integrations.

বাংলা: স্টেটলেস প্রমাণীকরণের জন্য \*\*JWT\*\* ব্যবহার চালিয়ে যাব। তৃতীয় পক্ষের ইন্টিগ্রেশনের জন্য \*\*OAuth2\*\* ব্যবহার করব।

14.5.1.2. Input Validation

English: Strict \*\*input validation\*\* on all API endpoints to prevent injection attacks (SQL, NoSQL, XSS).

বাংলা: ইনজেকশন অ্যাটাক (SQL, NoSQL, XSS) রোধ করতে সমস্ত API এন্ডপয়েন্টে কঠোর \*\*ইনপুট ভ্যালিডেশন\*\* প্রয়োগ করব।

14.5.1.3. Rate Limiting

English: Implement \*\*API rate limiting\*\* (possibly at the API Gateway or using Redis) to protect against abuse and DDoS attacks.

বাংলা: অপব্যবহার এবং DDoS অ্যাটাক থেকে রক্ষা করতে \*\*API রেট লিমিটিং\*\* (সম্ভবত API গেটওয়েতে বা রেডিস ব্যবহার করে) প্রয়োগ করব।

14.5.1.4. Regular Security Audits

English: Conduct regular \*\*security audits\*\* and penetration testing.

বাংলা: নিয়মিত \*\*নিরাপত্তা নিরীক্ষা\*\* এবং অনুপ্রবেশ পরীক্ষা পরিচালনা করব।

---

**15. General & Behavioral Questions**

**15.1. Common Questions:**

15.1.1. আপনি কেন এই \*\*চাকরি পরিবর্তন\*\* করতে চাইছেন? (Why are you looking for a job change?)

15.1.2. আপনার \*\*শর্ট টার্ম\*\* এবং \*\*লং টার্ম ক্যারিয়ার গোল\*\* কী? (What are your short-term and long-term career goals?)

15.1.3. আপনার \*\*দুর্বলতা\*\* এবং \*\*শক্তি\*\* কী কী? (What are your strengths and weaknesses?)

15.1.4. আপনি \*\*চাপের মধ্যে\*\* কিভাবে কাজ করেন? (How do you handle pressure?)

15.1.5. আপনি \*\*দলগতভাবে\*\* কাজ করতে পছন্দ করেন নাকি \*\*একাকী\*\*? (Do you prefer working in a team or individually?)

15.1.6. আপনার \*\*বর্তমান বেতন\*\* কত এবং \*\*প্রত্যাশিত বেতন\*\* কত? (What is your current salary and what are your salary expectations?)

15.1.7. আমাদের কোম্পানি সম্পর্কে আপনার \*\*ধারণা\*\* কী? (What do you know about our company?)

15.1.8. আমাদের কাছে আপনার \*\*কোন প্রশ্ন\*\* আছে কি? (Do you have any questions for us?)

15.1.9. আপনার এমন একটি প্রকল্পের কথা বলুন যেখানে আপনি একটি বড় চ্যালেঞ্জের মুখোমুখি হয়েছিলেন এবং আপনি কিভাবে তা সমাধান করেছিলেন। (Tell me about a project where you faced a significant challenge and how you overcame it.)

15.1.10. আপনি কীভাবে নতুন প্রযুক্তি শেখেন এবং আপনার দক্ষতা আপগ্রেড করেন? (How do you learn new technologies and keep your skills updated?)

---

**16. Big Data & Mass Communication**

**16.1. Scenario: Sending 10 Million Emails/Notifications at a Specific Time (e.g., 7 AM)**

16.1.1. 10 মিলিয়ন ব্যবহারকারীকে একই সময়ে (যেমন, সকাল 7টায়) ইমেল/নোটিফিকেশন পাঠানোর জন্য আপনি কী আর্কিটেকচার বা কৌশল ব্যবহার করবেন?

16.1.2. এই ধরনের বৃহৎ স্কেলের মেসেজিংয়ের জন্য আপনার সিস্টেমের \*\*Reliability\*\* এবং \*\*Deliverability\*\* কিভাবে নিশ্চিত করবেন?

16.1.3. যদি আপনার সিস্টেমের \*\*Rate Limit\*\* থাকে (যেমন, প্রতি সেকেন্ডে X সংখ্যক ইমেল/নোটিফিকেশন পাঠানো যায়), তাহলে আপনি কিভাবে 10 মিলিয়ন মেসেজ দক্ষতার সাথে শিডিউল এবং বিতরণ করবেন?

16.1.4. এই প্রক্রিয়ায় সম্ভাব্য \*\* bottlenecks\*\* (যেমন: ডেটাবেজ, API কল, নেটওয়ার্ক) কী কী হতে পারে এবং কিভাবে সেগুলো মোকাবেলা করবেন?

16.1.5. মেসেজ পাঠানো সফল হয়েছে কিনা, বা কোন ত্রুটি হয়েছে কিনা তা ট্র্যাক করার জন্য আপনি কী \*\*monitoring\*\* এবং \*\*logging\*\* কৌশল ব্যবহার করবেন?

16.1.6. যদি একটি নির্দিষ্ট সময়ে (যেমন, সকাল 7টায়) মেসেজ পাঠানোর কথা থাকে, কিন্তু প্রক্রিয়াটি 7 AM-এর আগে বা পরে শুরু হয়ে যায়, তাহলে আপনি কিভাবে এটি নিশ্চিত করবেন যে সকল মেসেজ ঠিক সময়ে পৌঁছায়? (Hint: Consider distributed cron, queuing with delay/available\_at, dedicated message brokers).

16.1.7. 10 মিলিয়ন ব্যবহারকারীর জন্য \*\*Dynamic Content\*\* (ব্যক্তিগতকৃত মেসেজ) কিভাবে দক্ষতার সাথে জেনারেট এবং হ্যান্ডেল করবেন?

16.1.8. এই স্কেলে \*\*Subscription Management\*\* (কে ইমেল/নোটিফিকেশন পাবে বা পাবে না) এবং \*\*Opt-out\*\* প্রক্রিয়া কিভাবে ম্যানেজ করবেন?

16.1.9. \*\*Failed deliveries\*\* (যেমন: Invalid email, bounced, unreachable device) কিভাবে হ্যান্ডেল করবেন এবং রিট্রাই মেকানিজম কেমন হবে?

16.1.10. এই ধরনের মাস কমিউনিকেশনের জন্য আপনি কি থার্ড-পার্টি সার্ভিস (যেমন: Amazon SES, Twilio, Firebase Cloud Messaging) ব্যবহার করবেন নাকি নিজস্ব সমাধান তৈরি করবেন? আপনার সিদ্ধান্তের কারণ কী?

```