A. মূল জ্ঞান (মৌখিক/লিখিত প্রশ্ন)

1. React এর **Virtual DOM** কী এবং এটি কিভাবে কাজ করে?

   Virtual DOM হলো আসল DOM-এর একটি লাইটওয়েট কপি, যা React ব্যবহার করে ইউজার ইন্টারফেস দ্রুত আপডেট করার জন্য।
   এটা এক ধরনের **JavaScript** অবজেক্ট, যা DOM-এর কাঠামো মেমোরিতে ধরে রাখে।

   ### ⚡ ফলাফল (Benefits):

- অ্যাপ অনেক দ্রুত ও পারফরম্যান্ট হয়
- DOM manipulation কম হয় (যেটা সবচেয়ে ভারী কাজ)
- ইউজার এক্সপেরিয়েন্স উন্নত হয় কারণ আপডেট স্মুথ লাগে

2. React lifecycle (functional components এ) — `useEffect` কখন ও কীভাবে ব্যবহার করবে? dependency array ব্যাখ্যা কর।

   React এর Functional Component-এ কোনো **lifecycle method** (যেমন `componentDidMount`, `componentDidUpdate`, `componentWillUnmount`) সরাসরি থাকে না।
   এর পরিবর্তে React আমাদের দিয়েছে **`useEffect()` Hook**, যা lifecycle এর মতো কাজ করে।
   ⚙️ **useEffect** কী করে:
   `useEffect()` মূলত **side effects handle** করার জন্য ব্যবহার করা হয় —
   যেমনঃ

- API call করা
- DOM ম্যানিপুলেশন
- Event listener যোগ/মুছে ফেলা
- Timer (setTimeout / setInterval) ব্যবহার করা
- State বা props পরিবর্তনের পর কোনো কাজ করা

3. Explain **props vs state** — কখন কোনটা ব্যবহার করবে?

   Props = বাইরের থেকে পাঠানো ডেটা (read-only)
   State = কম্পোনেন্টের ভেতরের পরিবর্তনযোগ্য ডেটা
   Props দিয়ে কম্পোনেন্টকে dynamic করা হয়, আর **State** দিয়ে সেটাকে interactive করা হয়।

4. React এ **controlled vs uncontrolled components** কী? উদাহরণ দেখাও।

   Controlled Component: React state data control করে → predictable, powerful, dynamic.
   Uncontrolled Component: DOM data control করে → simple, fast, less flexible.

5. Explain Hooks: `useMemo`, `useCallback`, `useRef` — প্রতিটির ব্যবহার ও পারফরম্যান্স প্রভাব।

   `useMemo()` ব্যবহার করা হয় কোনো ব্যয়বহুল **(expensive)** বা বারবার গণনা হওয়া ফাংশনের ফলাফলকে মেমোরিতে সংরক্ষণ করার জন্য।
   `useCallback()` ব্যবহার করা হয় **function** মেমোরাইজ করার জন্য, যাতে একই function instance বারবার তৈরি না হয়।
   `useRef()` ব্যবহার করা হয় **DOM element** বা কোনো **mutable** মানের রেফারেন্স ধরে রাখতে, যা পরিবর্তন হলেও re-render করে না।

6. TypeScript: `interface` vs `type` — কখন কোনটা প্রাধান্য দিবে? Generic types কীভাবে লিখবে?

   `interface` ব্যবহার করা হয় **object-**এর গঠন **(structure)** নির্ধারণ করতে — অর্থাৎ কোন property কী টাইপের হবে সেটা নির্দিষ্ট করতে।

```ts
interface User {
  name: string;
  age: number;
  isAdmin?: boolean; // optional property
}
```

```ts
interface User {
  name: string;
  email: string;
}

interface Admin extends User {
  role: string;
}

const adminUser: Admin = {
  name: "Rifat",
  email: "rifat@example.com",
  role: "super-admin",
};
```

`type` ব্যবহার করা হয় কোনো **data type-এর নাম (alias)** তৈরি করতে।
এটি শুধু object নয়, বরং **union, intersection, function, tuple, primitive type** — সব কিছুর জন্য ব্যবহার করা যায়।

```ts
type ID = string | number;

type User = {
  name: string;
  age: number;
};
```

```ts
type Status = "success" | "error" | "loading";

type AddFunc = (a: number, b: number) => number;

const add: AddFunc = (a, b) => a + b;
```

সংক্ষেপে
🧩 **interface → Object structure define ও extend** করতে।
🧩 **type → Flexible alias (object, union, tuple, function) define** করতে।
🧩 **generic → Reusable, type-safe component/function** বানাতে।

7. Explain **strict typing of React props** with TypeScript — উদাহরণ দেখাও।

   React-এ TypeScript ব্যবহার করলে আমরা **strict typing** দিয়ে props-এর ধরন (type) ঠিক করি। এটার সুবিধা হলো:

   কম্পোনেন্টে যেই ডেটা যাবে, তার ধরন নিশ্চিত করা হয়।
   ভুল ডেটা টাইপ গেলে কম্পাইল টাইমে ভুল ধরিয়ে দেয়।
   Auto-completion এবং **editor warnings** পাওয়া যায়, ফলে runtime bug কমে।
   কোড **Maintainable** এবং **Reliable** হয়।
   সহজভাবে বলা যায়, strict typing মানে হলো "কম্পোনেন্ট ঠিক কোন props গ্রহণ করবে তা আগে থেকেই নির্দিষ্ট করা"।

8. Next.js: **SSR (getServerSideProps)** vs **SSG (getStaticProps)** vs **Client-side fetching** — কবে কোনটা বেছে নেবে?

   SSR (Server-Side Rendering) – `getServerSideProps`
   ব্যাখ্যা :

SSR মানে হলো প্রতিটি রিকোয়েস্টে সার্ভার **React** কম্পোনেন্টকে রেন্ডার করে **HTML** পাঠাবে।
প্রতিটি ভিজিটে ডেটা fresh থাকে।
SEO এবং dynamic content-এর জন্য উপযুক্ত।
কখন ব্যবহার করবেন:
যখন পেজের ডেটা প্রতি ভিজিটে পরিবর্তিত হয়।
ইউজার স্পেসিফিক বা authentication-required ডেটা দরকার।
উদাহরণ

```ts
export async function getServerSideProps(context) {
  const res = await fetch('https://api.example.com/products');
  const products = await res.json();

  return {
    props: { products }, // page component-এ props হিসাবে যাবে
  };
}
```

SSG (Static Site Generation) – `getStaticProps`

ব্যাখ্যা:

SSG মানে হলো বিল্ড টাইমে **HTML generate** করা, তারপর **static**ভাবে সার্ভ করা।
দ্রুত লোড হয়, CDN-এর মাধ্যমে সহজে cache করা যায়।
SEO-friendly।
কখন ব্যবহার করবেন:
ডেটা খুব কম পরিবর্তিত হয় বা predictable।
উদাহরণ:

```ts
export async function getStaticProps() {
  const res = await fetch('https://api.example.com/blogs');
  const blogs = await res.json();

  return {
    props: { blogs },
    revalidate: 60, // ISR: প্রতি 60 সেকেন্ডে পেজ regenerate হবে
  };
}
```

Client-side Fetching (CSR) – useEffect / SWR / React Query

ব্যাখ্যা:
CSR মানে হলো কম্পোনেন্ট mount হওয়ার পর ব্রাউজারে ডেটা fetch করা।
User interaction-এর উপর নির্ভর করে।
Initial HTML lightweight থাকে, কিন্তু SEO কম-friendly।
কখন ব্যবহার করবেন:
যখন ডেটা খুব dynamic এবং ইউজার ইন্টারঅ্যাকশন বা ফিল্টারিং এর উপর নির্ভর করে।
লেটেস্ট data চাই কিন্তু SEO প্রয়োজন নেই।

উদাহরণ:

```tsx
import { useEffect, useState } from 'react';

export default function ProductsPage() {
  const [products, setProducts] = useState([]);

  useEffect(() => {
    fetch('https://api.example.com/products')
      .then(res => res.json())
      .then(data => setProducts(data));
  }, []);

  return (
    <div>
      {products.map(p => (
        <div key={p.id}>{p.name}</div>
      ))}
    </div>
  );
}
```

Professional Summary
“Next.js-এ getServerSideProps প্রতিটি রিকোয়েস্টে fresh data দেয়, getStaticProps build-time বা ISR-এর মাধ্যমে static site generate করে, আর client-side fetching user interaction বা dynamic updates-এর জন্য। পেজের nature এবং SEO প্রয়োজন অনুসারে আমাদের যথাযথ approach নির্বাচন করতে হবে।”

9. Next.js App Router vs Pages Router এর মূল পার্থক্য কি? (সংক্ষিপ্ত)

Next.js: App Router vs Pages Router

বৈশিষ্ট্য	**Pages Router**	**App Router**
Routing structure	`pages/` ফোল্ডারের ফাইল অনুযায়ী স্বয়ংক্রিয়	`app/` ফোল্ডারের ফাইল ও ফোল্ডার ভিত্তিক, nested routes সহজ
Data fetching	`getStaticProps`, `getServerSideProps`, `getStaticPaths`	Server Components, `fetch`, `async/await`, এবং `loading UI` native support
Server Components	সমর্থন করে না	Native support, কম bundle size ও better performance
Layouts	সাধারণ component-based	Nested & shared layouts সহজভাবে define করা যায়

Streaming / Suspense	সমর্থন সীমিত	Full support, incremental loading সম্ভব
Flexibility	Traditional, simple	Modern, scalable, nested & dynamic route friendly

10. Tailwind CSS: utility-first approach কী? কিভাবে responsive ও state-based (hover, focus) styling করবে?

    Utility-first approach মানে হলো:

    Tailwind CSS মূলত ছোট ছোট **utility class** ব্যবহার করে styling করে।

    প্রতিটি class একটি নির্দিষ্ট CSS property represent করে।

    ফলে আমরা **custom CSS** লিখতে কম সময় লাগায়, এবং কোড **readable, maintainable** ও **scalable** হয়।

    ✅ **HR-friendly summary:**

    “Tailwind CSS utility-first approach ব্যবহার করে আমরা ছোট ছোট classes দিয়ে styling করি। Responsive design করতে screen-size prefix (`sm:`, `md:`) ব্যবহার করি, আর hover/focus/active state handle করতে pseudo-class variants (`hover:`, `focus:`) ব্যবহার করি, ফলে scalable ও maintainable UI তৈরি করা সহজ হয়।”

11. Explain CORS — browser-তে কেন CORS সমস্যা হয় এবং backend এ কীভাবে ঠিক করা যায়?
    Browser-এ কেন CORS সমস্যা হয়?
    # CORS হলো browser security feature।
    # যখন একটি ওয়েবসাইট (origin A) অন্য server/ API (origin B) থেকে ডেটা fetch করতে চায়, browser **same-origin policy** enforce করে।
    # যদি server B স্পষ্টভাবে origin A কে allow না করে, browser request **block** করে।

```js
// Frontend: http://localhost:3000
fetch('http://api.example.com/data')
  .then(res => res.json())
  .then(console.log)
  .catch(console.error);
```

Backend-এ কিভাবে ঠিক করা যায়?
`Access-Control-Allow-Origin` → কোন origin allow হবে
Optional: `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers`

```js
const express = require('express');
const cors = require('cors');
const app = express();

// সব origin allow
app.use(cors());

// অথবা specific origin allow
// app.use(cors({ origin: 'http://localhost:3000' }));

app.get('/data', (req, res) => {
  res.json({ message: 'CORS enabled!' });
});

app.listen(5000, () => console.log('Server running on port 5000'));
```

✅ **Summary (HR-friendly)**
"CORS হলো browser security mechanism যা এক origin থেকে অন্য origin-এ request block করে। এটা prevent করে malicious requests। Backend-এ Access-Control-Allow-Origin header বা middleware ব্যবহার করলে browser-কে অনুমতি দেওয়া যায়, ফলে frontend থেকে data access করা সম্ভব হয়।"

12. REST vs GraphQL — pros/cons, কখন কোনটা বেছে নেবে?

✅ HR-friendly Summary

“REST traditional API যেখানে fixed endpoints থাকে, simple এবং cache-friendly। GraphQL modern approach, যেখানে client ঠিক করে কি data চাইছে, single endpoint দিয়ে nested data fetch সম্ভব। যদি app simple CRUD এবং caching প্রয়োজন হয় REST বেছে নেওয়া যায়, আর complex, dynamic, nested data-এর জন্য GraphQL বেশি উপযুক্ত।”

13. Explain optimistic UI updates and when you'd use them.

Optimistic UI update হলো এমন একটি UI approach যেখানে **frontend** আগেভাগে **UI** পরিবর্তন করে দেখায়, মনে হয় action সফল হয়েছে, আর পরে **backend confirm** করে।

মূল উদ্দেশ্য: **better user experience**, latency বা network delay কম দেখাতে।
যদি backend operation fail হয়, তখন UI rollback করা হয়।

```tsx
const [likes, setLikes] = useState(10);

const handleLike = async () => {
  // 1. Optimistically update UI
  setLikes(likes + 1);

  try {
    // 2. Send request to backend
    const res = await fetch('/api/like', { method: 'POST' });
    if (!res.ok) throw new Error('Failed to like');
  } catch (error) {
    // 3. Rollback if request fails
    setLikes(likes);
    alert('Could not like the post. Try again.');
  }
};
```

14. How do you debug performance issues in the browser? (network, CPU, paint/layout, bundle size tools)

HR-friendly summary

“Browser-এ performance debug করতে আমি Chrome DevTools ব্যবহার করি network delay, JS execution (CPU), paint/layout সমস্যা, এবং bundle size পরীক্ষা করার জন্য। Lighthouse এবং Bundle Analyzer-এর মতো tools bottleneck শনাক্ত করতে সাহায্য করে, এবং lazy loading, caching, memoization, ও code-splitting ব্যবহার করে speed ও responsiveness উন্নত করি।”

15. Git: explain `rebase` vs `merge`. When would you prefer each?

✅ **HR-friendly Summary**

“Merge দুটি branch একত্রিত করে, history preserve করে এবং safer, কিন্তু messy হতে পারে। Rebase branch commits কে latest base branch-এর উপর পুনঃস্থাপন করে, linear history দেয়, কিন্তু public/shared branch-এ risky। সাধারণত feature branch update বা clean history চাইলে rebase, shared main/develop branch merge করার সময় merge ব্যবহার করি।”

B. Practical / Live-coding style প্রশ্ন (তুমি লিখে প্র্যাকটিস করো)

21. **Component task**: একটি `ProductCard` React component লিখো (TypeScript) যা receives `product: { id, name, price, image, rating }` — clickable, accessible, এবং responsive হবে। Include prop types, basic styling with Tailwind, এবং `onAddToCart` callback।

- Acceptance: TypeScript types present, keyboard-focusable button, image has `alt`, responsive layout.

22. **State management task**: ছোট একটা page তৈরি করো যেখানে 50+ আইটেম আছে। Implement client-side search (debounced), client-side filter (category), এবং pagination (virtualized list optional)। Explain why debouncing helps and how you'd avoid unnecessary re-renders.

- Acceptance: search is debounced, filters chainable, no heavy re-renders (useMemo/useCallback explained).

23. **API integration task**: Next.js page এ একটি server-side fetched list দেখাও (`getServerSideProps`) এবং client-side একটি filterable search যোগ করো। Explain how you would cache responses and reduce API calls.

- Acceptance: SSR fetch implemented, client filter works without additional server calls, caching strategy described.

24. **Bugfix task**: তোমাকে দেওয়া হবে (simulate) একটি React form যেখানে submit করলে state reset হয় না এবং form validation ঠিকমত কাজ করছে না — লিখে বলো common causes এবং কিভাবে step-by-step ডিবাগ করো। (উত্তরে specific console checks, breakpoint points, possible code fixes উল্লেখ কর)।

25. **Optimize bundle task**: Project এর bundle size বড় (>1.2MB). কী কী step নেবে shrink করার জন্য? (tree-shaking, dynamic import, analyze bundle, remove polyfills, replace heavy libs ইত্যাদি) — practical commands/tools উল্লেখ করো।

- Acceptance: tools: `source-map-explorer` / `webpack-bundle-analyzer` / `next build && next analyze` ইত্যাদি; concrete steps।

26. **Tailwind & Figma task**: Figma থেকে একটি simple card (title, subtitle, price, badge, CTA) pixel-perfect হিসেবে কনভার্ট করতে গেলে approach বলো — responsive ও utility classes দিলে কেমন structure থাকবে? (small code snippet)।

27. **Type challenge**: TypeScript এ একটা utility function লিখো `deepMerge<T, U>(a: T, b: U): T & U` — explain pitfalls and how to keep types safe.

28. **Testing task**: একটি small React component এর জন্য unit test লিখো (Jest + React Testing Library) — test cases কি রাখব, sample test code দেখাও।

29. **Vue task**: একটি ছোট Vue component লিখে দেখাও (Composition API) যা একটি input নেয় এবং live debounced search করে parent কে emit করে।

30. **AI tools question (practical)**: তুমি কিভাবে AI tools (GitHub Copilot, ChatGPT, Code-Formatter tools) use করে workflow speed-up করবে? Provide 2 concrete examples where AI helps and 2 cases where AI can mislead / be risky.

C. প্রায়োগিক/শার্ট-অন্সার (শতকরা দ্রুত জবাব আশা করা হবে)

31. `key` prop React এ কেন জরুরি?

    ✅ **HR-friendly Summary**

    "React-এ key prop হলো unique identifier যা list elements-এর জন্য ব্যবহার হয়। React এই key দিয়ে বুঝতে পারে কোন element update, insert, বা delete হয়েছে, ফলে DOM efficiently update হয় এবং unnecessary re-render বা UI glitches কমে।"

32. Explain `event delegation` in JS।

33. How to handle image optimization in Next.js? (built-in feature বলো)

    ✅ **HR-friendly Summary**
    "Next.js-এ image optimization built-in `next/image` component ব্যবহার করে করা হয়। এটি স্বয়ংক্রিয় lazy loading, responsive images, আধুনিক formats, এবং optimized performance দেয়, ফলে fast loading ও improved user experience পাওয়া যায়, কোনো extra configuration ছাড়া।"

34. How will you implement dark mode with Tailwind efficiently?

✅ **HR-friendly Summary**

“Tailwind CSS-এ dark mode কার্যকরভাবে implement করা যায় `darkMode: 'class'` কনফিগারেশনের মাধ্যমে, `dark:` variant classes ব্যবহার করে styling করতে, এবং root element-এ toggle যোগ করে। এই পদ্ধতি lightweight, maintainable, এবং কম কোডে dynamic switching সম্ভব করে।”

35. Loops in JavaScript

Loops হলো **repeated tasks automate** করার জন্য **structures**, যা একই কাজ একাধিকবার execute করে।

1️⃣ for loop

নির্দিষ্ট সংখ্যক বার **loop execute** হয়।

```js
for (let i = 0; i < 5; i++) {
  console.log(i);
}
// Output: 0 1 2 3 4
```

2️⃣ while loop

যতক্ষণ **condition true** থাকে, **loop** চলতে থাকে।

```js
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
// Output: 0 1 2 3 4
```

3️⃣ do…while loop

কমপক্ষে একবার **execute** হয়, তারপর **condition check** করা হয়।

```js
let i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
// Output: 0 1 2 3 4
```

4️⃣ for…of loop
Array, string বা **iterable objects iterate** করতে ব্যবহার হয়।

```js
const arr = [10, 20, 30];
for (const value of arr) {
  console.log(value);
}
// Output: 10 20 30
```

5️⃣ for…in loop
Object-এর **property iterate** করতে ব্যবহার হয়।

```js
const obj = { a: 1, b: 2 };
for (const key in obj) {
  console.log(key, obj[key]);
}
// Output: a 1
//         b 2
```

Practical test suggestions (যদি তুমি মক-টেস্ট দিতে চাও)

- **Take-home assignment (2–4ঘন্টা)**: Build a small product list app with React + TypeScript + Tailwind (or Next.js), includes: responsive product grid, server-simulated API (mock), product detail modal, add-to-cart (localStorage persistence), basic unit test, and README with run instructions + live demo link (Netlify/Vercel).
- **Onsite live task (30–60 মিনিট)**: Fix a broken component and implement one new feature (e.g., rating stars + saving to backend mock), and explain choices.

Evaluation критерিয়া / Checklist (হায়ারার দৃষ্টিকোণ থেকে)

- Code correctness & Type safety (TypeScript).
- Readability & component structure (reusability).
- Accessibility considerations (semantic tags, labels, keyboard).
- Responsive & pixel-accurate implementation.
- Git usage (clean commits, branch name, rebase/merge awareness).
- Problem-solving approach & communication (can explain trade-offs).
- Use of AI tools responsibly (acknowledge limitations).

HR-style introductory round প্রশ্নাবলী (Interviewer as Hiring Manager)

1. সংক্ষিপ্তভাবে নিজেকে পরিচয় দাও — বর্তমান স্ট্যাক, প্রজেক্ট ও তুমি যে ধরনের কাজ উপভোগ করো (১–২ মিনিট)।
2. কেন Softivus-এ কাজ করতে চাও? (আপনি কী কারণে অনসাইট Uttara টিমে যোগ দিতে ইচ্ছুক?)

3. তুমি 지난 ৬-১২ মাসে কোন সবচেয়ে চ্যালেঞ্জিং প্রজেক্টে কাজ করেছো? কোন সমস্যা ছিল এবং তুমি কীভাবে সমাধান করেছিলে? (Technical depth বলো)

4. তোমার strongest technical skill কী এবং weakest কী — তুমি weakest ঠিক করার জন্য কী করছো?

5. কিভাবে তুমি টিমে communicate করো যখন disagreement হয় architecture বা approach নিয়ে? উদাহরণ দাও।

6. তোমার পোর্টফোলিও/লাইভ লিংক যেগুলো দেখি? (prepare 2–3 highlights)

7. তুমি কিভাবে তোমার ডেভেলপমেন্ট কাজগুলো document করো (README, code comments, PR descriptions)? একটা উদাহরণ বলো।

8. তোমার প্রত্যাশিত salary range কি এবং কেন? (Be realistic; role salary up to 35,000 BDT)

9. তুমি কিভাবে সময় manage করো — multiple tasks দিলে priority কিভাবে ঠিক করবে?

10. কোন ৩টি বিষয় তুমি কাজের culture থেকে অবশ্যই চাও? (উদাহরণ: mentorship, code reviews, growth)

11. তুমি কি অন-সাইট কাজ করতে পারবে (Uttara, Dhaka) এবং কি notice period আছে?

12. শেষ প্রশ্ন: তুমি আমাদেরকে কেন নিও— ১ মিনিটে concise pitch দাও।

Answering tips (কয়েকটি ছোট টিপস)

- প্রতিটি practical টাস্কে **acceptance criteria** মাথায় রাখবে — interviewer দ্রুত দেখতে চাইবে কি কাজ করছে কি না।

- কথায় বলার সময় **structure** রাখো: Problem → Approach → Trade-offs → Result.

- কোড দেখালে ছোট, readable commits দেখাও; PR description এ key points লিখে দাও।

- যদি কোন প্রশ্নে জানো না, সৎ হও— “I don’t remember exact syntax, but here’s how I’d find/fix it” বলো এবং debugging steps দাও।

- AI ব্যবহার করলে বলবে “I used X to scaffold, then audited & fixed manually” — এটি প্রমাণ করে তুমি copy-paste না।