

FULL STACK

TypeScript · React · Next.js · Node.js

From how the web works to production: HTML, CSS, JavaScript, TypeScript, React, Next.js, Node APIs, databases, auth, caching, Docker, security, testing and 100+ interview questions.

```
localhost:3000/offer

export default async function Page({ params }: Props) {
  const { role } = await params // Next.js 16
  const skills = await db.skills.findMany() // Node + SQL
  return <Offer role={role} stack="Full" />
}
```

TypeScript

React 19

Next.js 16

Server Actions

Node.js

REST APIs

PostgreSQL

Prisma

Redis

Docker

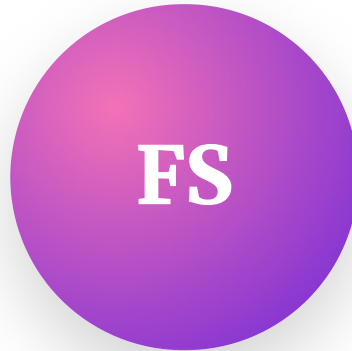
Auth

Core Web Vitals

WRITTEN BY

Shreyash Tripathi

JOB-READY SERIES • COMPACT EDITION



Full-Stack TypeScript and Next.js

Frontend and backend, from basics to production



WRITTEN BY

Shreyash Tripathi

Compact Edition, 2026

© Shreyash Tripathi. All rights reserved.

Preface

You already have one job. You write code, you push to production, and sometimes things break in ways nobody explained in college. Now you want the next job, and the interviews ask the same "basic" Full-Stack TypeScript and Next.js questions again, except now the interviewer also expects you to connect them to real work. This book is written for exactly that moment.

It follows the style of a school textbook on purpose: numbered chapters, simple language, worked examples step by step, a check with answers after every topic, and full exercises with an answer key at the end of every chapter. Every chapter also has an **Interview Corner** with questions interviewers really ask, and an **At Work** box that shows where the concept appears in a real job. There is humour too, because a joke is easier to remember than a definition.

Read one chapter a day, solve the questions before you look at the answers, and revise the Interview Corners the night before your interview. You will walk in prepared.

Shreyash Tripathi

Features of Every Chapter

DO YOU KNOW?

Real facts and history.

FUNNY SIDE

A joke that makes the idea stick.

KEY POINT

The line to remember.

INTERVIEW CORNER

Questions interviewers really ask, with model answers.

AT WORK

How the concept shows up in a real job.

COMMON MISTAKE

Traps that cost marks and cause bugs.

CHECK YOUR PROGRESS

Short questions with answers after every topic.

Each chapter ends with **What Have We Learnt?**, **Key Terms**, **Exercises** and a complete **Answer Key**. The last chapter is a full **Interview Question Bank**.

Contents

PART I: WEB FOUNDATIONS

1	How the Web Works for Developers	7
2	HTML and CSS Essentials	23
3	JavaScript Essentials Every Interviewer Asks	39

PART II: TYPESCRIPT

4	TypeScript Fundamentals	55
5	Advanced TypeScript	70
6	TypeScript in Real Projects	85

PART III: FRONTEND WITH REACT

7	React Essentials	100
8	State Management, Data Fetching and Forms	114

PART IV: NEXT.JS

9	Next.js Fundamentals and the App Router	130
10	Rendering in Next.js: Server and Client	146
11	Data Fetching, Server Actions, Route Handlers and Proxy	160
12	Next.js in Production	174

PART V: BACKEND WITH NODE.JS

13	Node.js Fundamentals	189
14	Building REST APIs with Express and NestJS	204
15	Databases for Web Developers	219
16	Authentication and Authorization on the Backend	234
17	Caching, Background Jobs and Real-Time	249
18	Docker, Deployment and DevOps Basics	265

PART VI: WEB DEV AT WORK

19	Web Performance and Core Web Vitals	282
20	Web Security for Frontend and Full-Stack Developers	297
21	Testing, Tooling and Team Workflow	312
22	Web Development Interview Question Bank	327

APPENDICES

A	TypeScript Cheat Sheet	347
B	Next.js and Node.js Cheat Sheet	350
C	Glossary	355

PART I

Web Foundations

How the web works, HTML and CSS, and the JavaScript every interviewer asks.

CHAPTER 1

How the Web Works for Developers

*"It works on my machine" is a statement about your machine, not about the web.***After studying this chapter, you will be able to**

- Explain what happens between typing a URL and seeing pixels on screen.
- Describe the critical rendering path and the difference between reflow and repaint.
- Choose correctly between `normal`, `async` and `defer` scripts.
- Use HTTP methods, status codes, headers and caching rules the way interviewers expect.
- Pick the right browser storage (cookies, `localStorage`, `sessionStorage`, `IndexedDB`).
- Explain CORS errors and the basic trade-offs of REST vs GraphQL.

Think of ordering food through a delivery app. You type the restaurant name, the app finds its address, a rider goes there, picks up the packed food, brings it home, and only then do you open the boxes and arrange them on the plate. A web page is the same: the browser finds the server (DNS), opens a road to it (TCP and TLS), asks for food (HTTP), and then unpacks and arranges everything on screen (rendering).

As a frontend or full-stack developer you do not need to be a networking expert. But when a page is slow, a cookie is missing, or the console screams "CORS", you must know which step broke. This chapter gives you that map.

1.1 Browser Architecture

A modern browser like Chrome is not one program. It is a group of cooperating processes. This design exists for safety and stability: if one tab crashes, the others should keep running.

Part	Job
Browser process	Address bar, tabs, bookmarks, permissions; coordinates everything.
Renderer process	One per site (roughly). Runs the rendering engine (Blink in Chrome, WebKit in Safari, Gecko in Firefox) and the JavaScript engine (V8, JavaScriptCore, SpiderMonkey).
Network process/service	DNS, connections, HTTP, cache.
GPU process	Draws layers on screen using the graphics card.

Inside a renderer, the **main thread** does almost everything: parse HTML, run JavaScript, calculate styles, do layout. This is the single most important fact for performance. If your JavaScript runs for 500 ms, the page cannot respond to clicks for 500 ms.

💡 DO YOU KNOW?

Chrome's "Site Isolation" puts different websites in different renderer processes. It was rolled out widely in 2018, partly as a defence against the Spectre CPU vulnerability, so that one site's process cannot read another site's memory.

🔑 KEY POINT

JavaScript, style calculation and layout all share one main thread. Long tasks block user input. Keep tasks short.

✅ CHECK YOUR PROGRESS 1.1

1. Why does a browser use multiple processes?

Answer: For stability and security. A crash or a malicious page in one renderer process cannot easily take down or read other tabs.

2. Which engine runs JavaScript in Chrome and Node.js?

Answer: V8, Google's open-source JavaScript engine.

1.2 From URL to Pixels

The classic interview question: "What happens when you type a URL and press Enter?" Here is a crisp version.

1. **Parse the input.** Is it a URL or a search term? Add `https://` if needed. Check HSTS list (forces HTTPS).
2. **DNS lookup.** Convert `www.example.com` to an IP address. Check browser cache, OS cache, then the resolver (ISP or 8.8.8.8 style public DNS).
3. **TCP connection.** Three-way handshake: SYN, SYN-ACK, ACK. (HTTP/3 uses QUIC over UDP instead.)
4. **TLS handshake.** Agree on keys, verify the server certificate. Now traffic is encrypted.
5. **HTTP request.** `GET / HTTP/1.1` plus headers like `Host`, `Cookie`, `Accept`.
6. **Server responds.** Status code, headers, and HTML body (often compressed with gzip or Brotli).
7. **Render.** Parse HTML, fetch CSS/JS/images, build DOM and CSSOM, layout, paint, composite.



Figure 1.1: The journey from a URL to a painted page

1.2.1 Worked example: why round trips matter

Suppose the round-trip time (RTT) from Chennai to a server in the US is about 200 ms. For a fresh HTTPS connection with HTTP/1.1 or HTTP/2 over TLS 1.3:

- DNS (uncached): about 1 RTT to resolver, say 50 ms (resolver is closer).
- TCP handshake: 1 RTT = 200 ms.
- TLS 1.3 handshake: 1 RTT = 200 ms (TLS 1.2 needs 2 RTT = 400 ms).
- HTTP request and first byte: 1 RTT = 200 ms (plus server think time).

Total $\approx 50 + 200 + 200 + 200 = 650$ ms before the first byte of HTML arrives. If the server is in Mumbai with 20 ms RTT, the same steps cost $\approx 50 + 20 + 20 + 20 = 110$ ms. This is why CDNs (Cloudflare, Vercel Edge, CloudFront) put servers close to users.

AT WORK

In Chrome DevTools, open the Network tab, click any request and open "Timing". You will see "DNS Lookup", "Initial connection", "SSL", "Waiting for server response (TTFB)" and "Content Download". When a manager says "the site is slow", this panel tells you if the problem is the network, the server, or the frontend.

✓ CHECK YOUR PROGRESS 1.2

1. Name the steps before the browser sends the HTTP request on a new HTTPS site.

Answer: DNS lookup, TCP handshake, then TLS handshake. Only after these can the encrypted HTTP request be sent.

2. How does a CDN reduce load time?

Answer: It serves content from a server near the user, so each round trip is shorter, and it can cache static files so the origin is not contacted at all.

1.3 The Critical Rendering Path

The **critical rendering path** (CRP) is the sequence of steps the browser takes to turn HTML, CSS and JS into pixels. Making it short is what "fast first paint" means.

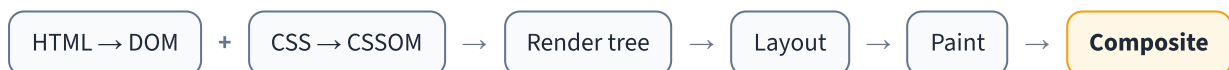


Figure 1.2: Critical rendering path

1. **HTML parsing → DOM.** The parser reads bytes, makes tokens, then builds the Document Object Model, a tree of nodes. It is incremental: the browser can start before all HTML arrives.
2. **CSS → CSSOM.** All CSS (files, `<style>`, inline) becomes the CSS Object Model. CSS is **render-blocking**: the browser will not paint until the CSSOM is ready, otherwise you would see a flash of unstyled content.

3. **Render tree.** DOM + CSSOM combined, keeping only visible nodes. `display: none` elements and `<head>` are excluded. (`visibility: hidden` elements stay; they take space.)
4. **Layout (reflow).** Compute exact position and size of every box, in pixels, for this viewport.
5. **Paint.** Fill in pixels: text, colours, borders, shadows, images, into layers.
6. **Composite.** The GPU stacks the layers in the right order and shows the final frame.

Analogy: a wedding hall. DOM is the guest list, CSSOM the dress code, render tree the guests who actually came, layout the seating plan, paint the decoration, composite the final group photo.

1.3.1 Reflow vs Repaint

	Reflow (layout)	Repaint
Trigger	Geometry changes: width, height, margin, font-size, adding DOM nodes, window resize	Visual-only changes: color, background, visibility, outline
Cost	High: may affect parents, children, siblings	Lower: no position recalculation
Then	Always followed by repaint	Not followed by reflow

Cheapest of all: changes to `transform` and `opacity`, which can often be done by the compositor alone, skipping layout and paint. That is why smooth animations use `transform: translateX()` instead of changing `left`.

1.3.2 Layout thrashing

```
// BAD: read, write, read, write... forces layout each loop
for (const box of boxes) {
  const h = box.offsetHeight;           // read (forces layout)
  box.style.height = h * 2 + "px";      // write (invalidates layout)
}

// GOOD: batch reads, then batch writes
const heights = boxes.map(b => b.offsetHeight);
boxes.forEach((b, i) => { b.style.height = heights[i] * 2 + "px"; });
```

Reading properties like `offsetHeight` or `getBoundingClientRect()` after a write forces the browser to do layout immediately (a "forced synchronous layout"). With 100 boxes, the bad version may do 100 layouts; the good one does about 1.

😄 FUNNY SIDE

Layout thrashing is like asking the tailor to measure you, then changing your shirt, then asking again, 100 times. The tailor is the browser. The tailor is tired.

🔧 ACTIVITY 1.1

Open DevTools → Performance, click Record, scroll a heavy page for 3 seconds, stop. Look for purple "Layout" and green "Paint" blocks, and red triangles marking long tasks over 50 ms.

✓ CHECK YOUR PROGRESS 1.3

1. Why is CSS called render-blocking?

Answer: The browser waits for the CSSOM before painting, because painting without styles would show unstyled content that then jumps.

2. Which is cheaper to animate: `left` or `transform`? Why?

Answer: `transform`. It can be handled at the composite step without triggering layout or paint, while `left` triggers reflow.

1.4 Script Loading: `async` vs `defer`

By default, when the HTML parser meets `<script src="app.js">`, it **stops parsing**, downloads the script, runs it, then continues. Why? Because the script might call `document.write` or change the DOM. This is parser-blocking.

Attribute	Download	Execution	Order kept?	Use for
none	Blocks parsing	Immediately	Yes	Rarely; legacy
<code>async</code>	In parallel	As soon as downloaded (pauses parser then)	No	Independent scripts: analytics, ads
<code>defer</code>	In parallel	After HTML parsing, before <code>DOMContentLoaded</code>	Yes	Your app code that needs the DOM
<code>type="module"</code>	In parallel	Deferred by default	Yes	Modern ES module code

```
<script src="analytics.js" async></script>
<script src="vendor.js" defer></script>
<script src="app.js" defer></script>  ←!— runs after vendor.js →
```

Analogy: at a chai stall, a normal script is the owner stopping all orders to go buy milk. `async` is sending a helper to buy milk and using it the moment it arrives, even mid-order. `defer` is sending the helper but using the milk only after current orders are done, in the order packets were bought.

🔗 INTERVIEW CORNER

Q: Difference between `async` and `defer`?

A: Both download in parallel. `async` runs as soon as it is ready, in any order. `defer` waits until HTML is parsed and runs in document order.

Q: What is the difference between `DOMContentLoaded` and `load`?

A: `DOMContentLoaded` fires when HTML is parsed and deferred scripts have run. `load` fires later, after images, stylesheets and iframes have also loaded.

✓ CHECK YOUR PROGRESS 1.4

1. Two `async` scripts `a.js` and `b.js`. Is `a.js` guaranteed to run first?

Answer: No. Whichever finishes downloading first runs first.

2. Do module scripts need `defer` ?

Answer: No. `type="module"` scripts are deferred by default. You can add `async` to run them as soon as ready.

1.5 HTTP for Web Developers

1.5.1 Methods

Method	Purpose	Safe?	Idempotent?
GET	Read a resource	Yes	Yes
POST	Create / trigger an action	No	No
PUT	Replace a resource fully	No	Yes
PATCH	Partially update	No	Not guaranteed
DELETE	Remove	No	Yes
OPTIONS	Ask what is allowed (used in CORS preflight)	Yes	Yes

Safe means no change on the server. **Idempotent** means doing it once or 5 times leaves the same final state. Deleting order 42 five times: still deleted. Posting "pay ₹500" five times: five payments. That is why payment APIs (Stripe, Razorpay) support an idempotency key.

1.5.2 Status codes

Code	Meaning	Typical use
200 OK	Success	GET returns data
201 Created	New resource made	POST /users
204 No Content	Success, empty body	DELETE
301 / 308	Moved permanently	Old URL to new URL (308 keeps method)
302 / 307	Temporary redirect	Login redirect (307 keeps method)
304 Not Modified	Use your cached copy	Conditional GET
400 Bad Request	Invalid input	Validation failed
401 Unauthorized	Not authenticated	Missing/expired token

Code	Meaning	Typical use
403 Forbidden	Authenticated but not allowed	Normal user hits admin API
404 Not Found	No such resource	Wrong ID
409 Conflict	State conflict	Email already registered
429 Too Many Requests	Rate limited	Tatkal-time traffic
500 / 502 / 503 / 504	Server error / bad gateway / unavailable / gateway timeout	Bugs, crashed upstream, overload

1.5.3 Important headers

- **Content-Type** : format of body, e.g. `application/json` .
- **Authorization** : e.g. `Bearer <token>` .
- **Cookie** / **Set-Cookie** : send / store cookies.
- **Cache-Control** , **ETag** , **Last-Modified** : caching.
- **Content-Encoding** : gzip or br (Brotli).
- **Location** : target URL for redirects.

1.5.4 HTTP caching

Caching saves round trips. Two kinds:

1. **Fresh (strong) caching**: `Cache-Control: max-age=31536000, immutable` means "use this for one year without asking". Perfect for hashed files like `app.3f9a1c.js` , because a new deploy gives a new file name.
2. **Revalidation**: server sends `ETag: "abc123"` . Next time the browser asks with `If-None-Match: "abc123"` . If unchanged, server replies **304** with no body. Saves bandwidth, still costs one round trip.

Directive	Meaning
<code>no-store</code>	Never store. Use for bank statements.
<code>no-cache</code>	Store, but revalidate before every use (misleading name!).
<code>private</code>	Only the browser may cache, not CDNs. User-specific pages.
<code>public</code> , <code>s-maxage</code>	Shared caches (CDN) may store; <code>s-maxage</code> sets CDN lifetime.
<code>stale-while-revalidate=60</code>	Serve stale copy for up to 60 s while fetching fresh one in background.

⚠ COMMON MISTAKE

Thinking `no-cache` means "do not cache". It means "always check with the server first". To truly forbid storage use `no-store`. Also: caching `index.html` for a year means users never see your new deploy.

👛 AT WORK

A classic incident: a user-specific dashboard returned `Cache-Control: public, max-age=300`. The CDN cached user A's page and served it to user B. Fix: `private, no-store` for personalised responses, and check the CDN's cache key. Always review cache headers in code review.

✅ CHECK YOUR PROGRESS 1.5**1. 401 vs 403?**

Answer: 401 means the server does not know who you are (log in). 403 means it knows you, but you are not allowed.

2. How should a file named `main.8c2f.js` be cached?

Answer: `Cache-Control: public, max-age=31536000, immutable`. The hash changes when content changes, so long caching is safe.

1.6 Browser Storage

	Cookies	localStorage	sessionStorage	IndexedDB
Size	About 4 KB each	About 5 MB per origin (varies)	About 5 MB per origin	Large (hundreds of MB, quota-based)
Sent to server?	Yes, automatically with every matching request	No	No	No
Lifetime	Until Expires/Max-Age, or session	Until cleared	Until tab closes	Until cleared
API	<code>document.cookie</code> / headers	Sync, strings only	Sync, strings only	Async, structured data, indexes
Readable by JS?	Not if <code>HttpOnly</code>	Yes	Yes	Yes
Use for	Session IDs, auth	Theme, small prefs	Multi-step form in one tab	Offline data, big caches

1.6.1 Cookie attributes

```
Set-Cookie: sid=9f8e7d; HttpOnly; Secure; SameSite=Lax; Path=/; Max-Age=86400
```

- `HttpOnly` : JavaScript cannot read it. Protects against token theft through XSS.

- `Secure` : only sent over HTTPS.
- `SameSite=Lax` (default in modern browsers): not sent on cross-site sub-requests, but sent on top-level navigation. Helps against CSRF. `Strict` is stricter; `None` requires `Secure` .

🔑 KEY POINT

Do not store auth tokens in `localStorage` if you can avoid it: any XSS script can read it. An `HttpOnly; Secure; SameSite` cookie is the safer default.

✅ CHECK YOUR PROGRESS 1.6

1. Which storage is cleared when the tab is closed?

Answer: `sessionStorage`. It is also separate per tab, even for the same site.

2. Why are big cookies bad for performance?

Answer: Cookies are sent with every matching request, including images and API calls, so they add bytes to every upload.

1.7 CORS Basics

The **same-origin policy** says a script from origin A cannot read responses from origin B. An **origin** = scheme + host + port. `https://app.com` and `https://api.app.com` are different origins; so are `http://localhost:3000` and `http://localhost:8080` .

CORS (Cross-Origin Resource Sharing) is how server B says "I allow origin A to read me".

1. Browser sends request with `Origin: https://app.com` .
2. For "non-simple" requests (e.g. `PUT` , JSON body, custom header like `Authorization`) the browser first sends a **preflight** `OPTIONS` request.
3. Server replies with `Access-Control-Allow-Origin: https://app.com` , plus `Allow-Methods` and `Allow-Headers` .
4. If headers match, the browser lets your JS read the response. Otherwise it blocks it and logs a CORS error.



Figure 1.3: CORS preflight flow

Important: CORS is enforced by the **browser**, not the server. Postman and curl ignore it. And CORS protects users from other sites reading their data, it is not your API's authentication.

⚠️ COMMON MISTAKE

"Fixing" CORS with `Access-Control-Allow-Origin: *` together with cookies. Browsers reject `*` when credentials are sent. Return the exact allowed origin and `Access-Control-Allow-Credentials: true`. In Next.js, calling your own Route Handler from the same origin avoids CORS completely.

✅ CHECK YOUR PROGRESS 1.7**1. Why does the request work in Postman but fail in the browser?**

Answer: CORS is a browser rule. Postman is not a browser, so it does not enforce the same-origin policy.

2. What triggers a preflight?

Answer: A non-simple request: methods like PUT/DELETE, a Content-Type like `application/json`, or custom headers such as `Authorization`.

1.8 REST vs GraphQL (Briefly)

REST models resources as URLs: `GET /users/42`, `GET /users/42/orders`. **GraphQL** exposes one endpoint (usually `POST /graphql`) where the client sends a query describing exactly the fields it wants.

```
query {
  user(id: 42) {
    name
    orders(last: 3) { id total }
  }
}
```

	REST	GraphQL
Endpoints	Many	One
Over/under-fetching	Common	Client picks fields
HTTP caching	Easy (GET + URL)	Harder (mostly POST); needs client caches
Schema/types	Optional (OpenAPI)	Built-in, strongly typed
Good for	Simple CRUD, public APIs	Many clients with different data needs

💡 DO YOU KNOW?

GraphQL was built inside Facebook in 2012 for its mobile apps and was open-sourced in 2015. GitHub's public API v4 is GraphQL.

 INTERVIEW CORNER

Q: What is the N+1 problem in GraphQL?

A: Fetching a list of 10 users and then each user's orders makes 1 + 10 database queries. Fix it with batching, for example the DataLoader pattern.

Q: Is PUT idempotent? Is POST?

A: PUT is idempotent (same final state on repeat). POST is not.

 CHECK YOUR PROGRESS 1.8

1. What is over-fetching?

Answer: Receiving more fields than the screen needs, e.g. a full user object when you only show the name.

2. Why is caching easier in REST?

Answer: GET requests with unique URLs fit naturally into browser and CDN caches; GraphQL usually sends POST to one URL.

 THINK AND DISCUSS

Your Next.js app uses Server Components that read the database directly. Do you still need a REST or GraphQL API? Who else (mobile app, partners) might need one?

What Have We Learnt?

- Browsers use separate processes; the renderer's main thread runs JS, style and layout.
- URL to pixels: DNS → TCP → TLS → HTTP → render. Each round trip costs latency.
- CRP: DOM + CSSOM → render tree → layout → paint → composite.
- CSS is render-blocking; normal scripts are parser-blocking.
- Reflow (geometry) is costlier than repaint (visuals); transform and opacity are cheapest.
- Batch DOM reads and writes to avoid layout thrashing.
- `async` : run when ready, any order. `defer` : run after parsing, in order.
- Know safe vs idempotent methods and key status codes (401 vs 403, 304, 429).
- `Cache-Control` , `ETag` and 304 drive HTTP caching; `no-cache` ≠ `no-store` .
- Cookies go to the server; Web Storage does not; IndexedDB is for large async data.
- Use `HttpOnly`; `Secure`; `SameSite` cookies for sessions.
- CORS is browser-enforced permission for cross-origin reads, with OPTIONS preflight.
- REST is simple and cache-friendly; GraphQL gives clients precise data.

Key terms: renderer process, main thread, DNS, TLS, TTFB, critical rendering path, DOM, CSSOM, render tree, layout, reflow, repaint, composite, layout thrashing, async, defer, idempotent, Cache-Control, ETag, HttpOnly, SameSite, same-origin policy, CORS, preflight, REST, GraphQL.

Exercises

A. Multiple Choice Questions

- Which change usually triggers only a repaint?
 - width
 - font-size
 - background-color
 - margin
- Which script attribute keeps execution order and waits for parsing?
 - async
 - defer
 - nomodule
 - crossorigin
- A server reply with no body meaning "your cached copy is still valid" is:
 - 200
 - 204
 - 304
 - 404
- Which storage is sent to the server automatically?
 - localStorage
 - Cookies
 - IndexedDB
 - sessionStorage
- Which pair is the same origin?
 - http://a.com and https://a.com
 - https://a.com and https://b.a.com
 - https://a.com/x and https://a.com/y
 - https://a.com:443 and https://a.com:8443

B. Fill in the Blanks

- DOM and CSSOM are combined to form the _____.
- The HTTP method used for a CORS preflight is _____.
- The cookie attribute that hides a cookie from JavaScript is _____.

C. True or False

- Cache-Control: no-cache forbids the browser from storing a response.
- Elements with `display: none` are not in the render tree.
- CORS errors can be fixed by changing frontend code only.

D. Short Answer Questions

- What is layout thrashing and how do you avoid it?
- Explain idempotency with an example.
- When would you choose IndexedDB over localStorage?

E. Long Answer Questions

- Explain in detail what happens when you type `https://www.irc.tc.co.in` in the browser and press Enter.

2. Design a caching strategy for a Next.js e-commerce site: hashed JS/CSS, product images, the HTML page, and the logged-in cart API.

F. Design Challenge

Your app runs at `https://shop.in` and the API at `https://api.shop.in`. Logged-in users call `PUT /cart` with a JSON body and session cookie. List the exact response headers the API must send for this to work, and the cookie attributes you would use.

Answer Key: Chapter 1

A. MCQ

1. (c) Colour change does not move boxes.
2. (b) defer runs in order after parsing.
3. (c) 304 Not Modified.
4. (b) Cookies travel with matching requests.
5. (c) Path does not affect origin.

B. Fill in the Blanks

1. render tree
2. OPTIONS
3. HttpOnly

C. True or False

1. False. It allows storage but requires revalidation; `no-store` forbids storage.
2. True. They take no space and are excluded.
3. False. The server must send the right `Access-Control-Allow-*` headers (or you proxy through your own origin).

D. Short Answers

1. Layout thrashing is repeatedly writing styles and then reading layout properties in a loop, forcing many synchronous layouts. Avoid it by batching all reads first, then all writes, or by using `requestAnimationFrame`.
2. An operation is idempotent if repeating it gives the same final state. `DELETE /orders/42` twice leaves the order deleted; `POST /payments` twice creates two payments, so it is not idempotent.
3. Use IndexedDB for large or structured data (thousands of records, blobs, offline mode) and when you need indexes or async access that does not block the main thread. `localStorage` is synchronous, string-only and small.

E. Long Answers

1. Model answer:
 - Browser checks input, HSTS, and its cache.
 - DNS resolves the host to an IP (browser, OS, resolver caches, then recursive lookup).
 - TCP three-way handshake (or QUIC for HTTP/3).
 - TLS handshake, certificate verification, session keys.
 - HTTP GET with Host, cookies, Accept headers.
 - Server (possibly via CDN and load balancer) returns HTML with status and headers.
 - Parser builds DOM; CSS builds CSSOM; scripts are fetched per async/defer rules.
 - Render tree, layout, paint, composite produce the first frame.
 - Further resources (images, fonts, API calls) load; `DOMContentLoaded` then `load` fire.

2. Model answer:

- Hashed JS/CSS: `public, max-age=31536000, immutable`.
- Product images: long CDN caching with versioned URLs, served via an image optimiser.
- HTML pages: short CDN lifetime with `s-maxage` and `stale-while-revalidate`, or revalidate on product updates.
- Cart API: `private, no-store`, since it is per-user.
- Use ETags for revalidation where data changes often.
- Never cache personalised HTML at the CDN; vary or bypass on session cookie.

F. Design Challenge: Model Solution

Preflight response to `OPTIONS /cart`: `Access-Control-Allow-Origin: https://shop.in`, `Access-Control-Allow-Methods: PUT`, `Access-Control-Allow-Headers: Content-Type`, `Access-Control-Allow-Credentials: true`, optionally `Access-Control-Max-Age: 600` to cache the preflight. The actual PUT response also needs `Access-Control-Allow-Origin: https://shop.in` and `Access-Control-Allow-Credentials: true`. The frontend uses `fetch(url, { credentials: "include" })`. `Cookie: HttpOnly; Secure; SameSite=Lax; Domain=shop.in; Path=/`. Since `shop.in` and `api.shop.in` are same-site, Lax works.

CHAPTER 2

HTML and CSS Essentials

CSS is easy. It is the other 10% that takes 90% of your Friday evening.

After studying this chapter, you will be able to

- Write semantic, accessible HTML with proper forms and validation.
- Add meta tags that help SEO and social sharing.
- Explain the box model, display, position, specificity and the cascade.
- Build layouts with Flexbox and Grid, and make them responsive.
- Choose the right CSS unit and use CSS variables for theming.
- Explain what Tailwind CSS is and why teams adopt it.

Building a web page is like building a house. HTML is the structure: walls, doors, rooms with names ("kitchen", not "room 3"). CSS is the paint, furniture placement and decoration. If the structure is wrong, no amount of paint will make the kitchen work as a bedroom.

Interviewers love HTML and CSS questions because they quickly reveal whether a developer understands the platform or just copies classes from Stack Overflow. Let us make sure you understand it.

2.1 Semantic HTML

Semantic HTML means using elements that describe meaning, not just appearance. A `<div>` says nothing. A `<nav>` says "this is navigation".

```
<header>
  <nav><a href="/">Home</a> <a href="/menu">Menu</a></nav>
</header>
<main>
  <article>
    <h1>Best Chai in Pune</h1>
    <section><h2>Cutting chai</h2><p>... </p></section>
  </article>
  <aside>Related posts</aside>
</main>
<footer>© 2026</footer>
```

Why it matters:

- **Accessibility:** screen readers let users jump between landmarks (`main` , `nav`) and headings.

- **SEO:** search engines understand structure better.
- **Free behaviour:** a `<button>` is focusable and works with Enter and Space. A `<div onClick>` does none of that.
- **Maintainability:** the next developer reads intent directly.

⚠ COMMON MISTAKE

"Div soup": `<div class="button" onClick="...">`. It cannot be reached by Tab, does not respond to Enter, and screen readers do not announce it as a button. Use `<button>` for actions and `<a href>` for navigation.

✓ CHECK YOUR PROGRESS 2.1

1. When do you use `<a>` vs `<button>`?

Answer: `<a href>` navigates to a URL. `<button>` performs an action on the page, like submitting or opening a modal.

2. How many `<main>` elements should a page have visible?

Answer: One. It marks the primary content of the page.

2.2 Forms and Validation

```
<form action="/signup" method="post">
  <label for="email">Email</label>
  <input id="email" name="email" type="email" required autocomplete="email">

  <label for="age">Age</label>
  <input id="age" name="age" type="number" min="18" max="120">

  <label for="pin">PIN code</label>
  <input id="pin" name="pin" pattern="[0-9]{6}" inputmode="numeric">

  <button type="submit">Sign up</button>
</form>
```

- Every input needs a `<label>` linked via `for / id`. Clicking the label focuses the input, and screen readers read it.
- The `name` attribute is what gets sent to the server (and what `FormData` reads in a Next.js Server Action).
- Built-in validation: `required`, `type="email"`, `min / max`, `minlength`, `pattern`. The CSS pseudo-classes `:invalid` and `:user-invalid` let you style errors.
- `inputmode="numeric"` shows a number keypad on phones.

KEY POINT

Client-side validation is for user experience. Server-side validation is for security. Anyone can open DevTools and remove `required`, or call your API with curl. Always validate again on the server (for example with Zod).

✓ CHECK YOUR PROGRESS 2.2

1. Which attribute decides the key name in submitted form data?

Answer: `name`. The `id` is for linking labels and scripts.

2. Is HTML validation enough to stop bad data?

Answer: No. It can be bypassed, so the server must validate too.

2.3 Accessibility (a11y) Basics

Accessibility means people with disabilities (blind, low vision, motor, hearing, cognitive) can use your site. In many countries it is also a legal requirement. The standard is **WCAG** (Web Content Accessibility Guidelines).

- **Alt text:** ``. Decorative images use `alt=""` so screen readers skip them.
- **Keyboard:** everything clickable must work with Tab, Enter, Space, Escape. Never remove the focus outline without a replacement (`:focus-visible`).
- **Contrast:** WCAG AA asks for at least 4.5:1 for normal text and 3:1 for large text. Light grey on white usually fails.
- **Headings in order:** `h1` → `h2` → `h3`, without skipping for style.

2.3.1 ARIA

ARIA (Accessible Rich Internet Applications) attributes add meaning when native HTML cannot. Examples: `aria-label`, `aria-expanded`, `aria-live`, `role="dialog"`.

```
<button aria-label="Close" aria-expanded="false">×</button>
<div aria-live="polite">3 items added to cart</div>
```

The first rule of ARIA: **do not use ARIA if a native element does the job**. ARIA changes what is announced, not behaviour. `role="button"` on a `div` still needs you to add keyboard handling and `tabindex`.

✏ ACTIVITY 2.1

Unplug your mouse (or just do not touch it) and try to sign up on a website you use. Then run Lighthouse (DevTools → Lighthouse → Accessibility) on your own project and fix the top three issues.

✓ CHECK YOUR PROGRESS 2.3

1. What alt text should a purely decorative divider image have?

Answer: An empty alt, `alt=""`, so assistive technology ignores it.

2. What is the minimum contrast ratio for normal text under WCAG AA?

Answer: 4.5:1.

2.4 Meta Tags and SEO Basics

```
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Masala Chai Recipe | ChaiTime</title>
  <meta name="description" content="Make strong masala chai in 10 minutes.">
  <link rel="canonical" href="https://chaitime.in/masala-chai">
  <meta property="og:title" content="Masala Chai Recipe">
  <meta property="og:image" content="https://chaitime.in/og/chai.png">
</head>
```

- **viewport:** without it, phones render at a fake desktop width and zoom out.
- **title and description:** shown in search results.
- **canonical:** tells search engines the preferred URL when duplicates exist (e.g. with `?utm_source=`).
- **Open Graph (`og:*`):** preview card on WhatsApp, LinkedIn, Slack.
- **Also:** `robots.txt`, `sitemap.xml`, real links (`<a href>`), fast load (Core Web Vitals).

In Next.js App Router you rarely write these by hand: you export a `metadata` object or a `generateMetadata` function from a page or layout.

```
export const metadata = {
  title: "Masala Chai Recipe",
  description: "Make strong masala chai in 10 minutes.",
};
```

✓ CHECK YOUR PROGRESS 2.4

1. Why is the viewport meta tag needed?

Answer: It tells mobile browsers to use the device width, so responsive CSS and media queries work properly.

2. What does a canonical link solve?

Answer: Duplicate content. It tells search engines which URL is the main one, so ranking signals are not split.

2.5 The Box Model, Display and Position

2.5.1 Box model

Every element is a box: **content** → **padding** → **border** → **margin**.

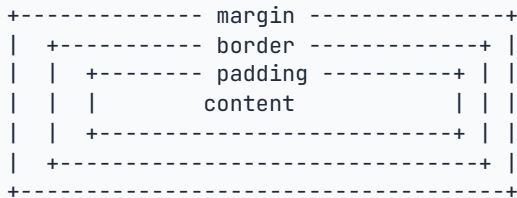


Figure 2.1: The CSS box model

Worked example. `width: 200px; padding: 20px; border: 5px solid; margin: 10px;`

- With `box-sizing: content-box` (default): visible width = $200 + 2 \times 20 + 2 \times 5 = \mathbf{250px}$. Space taken including margin = $250 + 2 \times 10 = 270px$.
- With `box-sizing: border-box`: visible width = **200px** (content shrinks to $200 - 40 - 10 = 150px$). Space taken = 220px.

That is why almost every project starts with `*, *::before, *::after { box-sizing: border-box; }`. Tailwind's base styles do this too.

Margin collapse: vertical margins of adjacent block elements combine into the larger one. A `margin-bottom: 20px` followed by `margin-top: 30px` gives a 30px gap, not 50px. It does not happen inside flex or grid containers.

2.5.2 Display

Value	Behaviour
<code>block</code>	New line, full width, width/height work (div, p, h1)
<code>inline</code>	Flows in text, width/height ignored (span, a)
<code>inline-block</code>	Flows in text but width/height work
<code>flex / grid</code>	Turns children into flex/grid items
<code>none</code>	Removed from layout and render tree

2.5.3 Position

Value	Relative to	In normal flow?
<code>static</code>	Default; top/left ignored	Yes
<code>relative</code>	Its own normal position	Yes (space kept)
<code>absolute</code>	Nearest positioned ancestor (non-static)	No

Value	Relative to	In normal flow?
fixed	Viewport (usually)	No
sticky	Acts relative until a scroll threshold, then sticks	Yes

⚠ COMMON MISTAKE

A `z-index: 9999` modal still appears behind the header. Reason: **stacking contexts**. A parent with `transform`, `opacity < 1`, or `position + z-index` creates a new context, and children cannot escape it. Render modals at the end of `body` (React portals) instead of adding more 9s.

✅ CHECK YOUR PROGRESS 2.5

1. A box has width 300px, padding 10px, border 2px, content-box. What is its visible width?

Answer: $300 + 20 + 4 = 324\text{px}$.

2. An absolutely positioned child jumps to the page corner. Why?

Answer: No ancestor is positioned, so it is placed relative to the initial containing block. Add `position: relative` to the intended parent.

2.6 Specificity and the Cascade

When two rules target the same element and property, the browser decides the winner by the **cascade**, in this order:

1. **Origin and importance:** `!important` declarations beat normal ones.
2. **Cascade layers** (`@layer`), if used.
3. **Specificity**.
4. **Source order:** later rule wins on a tie.

Specificity is counted as (IDs, classes/attributes/pseudo-classes, elements/pseudo-elements). Inline styles beat all selectors.

Selector	Specificity
<code>p</code>	(0,0,1)
<code>.card p</code>	(0,1,1)
<code>nav .link:hover</code>	(0,2,1)
<code>#main .card</code>	(1,1,0)
<code>:where(.card) p</code>	(0,0,1), since <code>:where</code> adds zero

Compare left to right like cricket scores where one century beats any number of fifties: (1,0,0) beats (0,15,0). **Inheritance** is separate: properties like `color` and `font-family` pass to children; `margin`

and `border` do not.

🔗 INTERVIEW CORNER

Q: Which wins: `#nav a` or `.menu .item a`?

A: `#nav a = (1,0,1)`; `.menu .item a = (0,2,1)`. The ID wins.

Q: Why avoid `!important`?

A: It breaks normal cascade, so the only way to override it is another `!important`. It leads to a specificity war. Fix the selector or use layers instead.

Q: Difference between `display: none` and `visibility: hidden`?

A: `none` removes the element from layout. `hidden` hides it but keeps its space.

✅ CHECK YOUR PROGRESS 2.6

1. Specificity of `ul li.active a`?

Answer: (0,1,3): one class, three elements.

2. Two rules have equal specificity. Which wins?

Answer: The one that appears later in source order.

2.7 Flexbox and Grid

2.7.1 Flexbox (one dimension)

Flexbox lays items out in a row or column. Like people in a Mumbai local queue: one line, you decide spacing and alignment.

```
.toolbar {
  display: flex;
  flex-direction: row;          /* main axis */
  justify-content: space-between; /* along main axis */
  align-items: center;          /* along cross axis */
  gap: 12px;
  flex-wrap: wrap;
}
.search { flex: 1; }           /* grow to fill remaining space */
```

`flex: 1` is shorthand for `flex-grow: 1; flex-shrink: 1; flex-basis: 0%`. **Worked example:** container 600px, two fixed items 100px each, gap 0, and two items with `flex: 1` and `flex: 2`. Free space = $600 - 200 = 400\text{px}$. Split 1:2 → 133.3px and 266.7px.

2.7.2 Grid (two dimensions)

Grid controls rows and columns together, like a cinema seat plan.

```
.products {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 16px;
}
.page {
  display: grid;
  grid-template-columns: 240px 1fr;
  grid-template-areas: "side main";
}
```

The first rule makes a responsive card grid without any media query: as many 200px+ columns as fit. `fr` means a fraction of free space.

	Flexbox	Grid
Dimensions	One (row or column)	Two (rows and columns)
Driven by	Content size	Layout definition
Best for	Navbars, button groups, aligning items	Page layouts, card galleries, dashboards

2.7.3 Centering a div (the joke)

😄 FUNNY SIDE

"How to center a div" is the most famous meme in web development. Senior engineers have shipped payment systems and still Googled it last Tuesday. Here is your permanent answer so you can retire the meme.

```
/* Flexbox */
.parent { display: flex; justify-content: center; align-items: center; }

/* Grid: shortest */
.parent { display: grid; place-items: center; }

/* Horizontal only, block with width */
.child { width: 300px; margin-inline: auto; }

/* Absolute (overlay) */
.child { position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); }
```

Remember: vertical centering needs the parent to have a height (e.g. `min-height: 100dvh`).

✅ CHECK YOUR PROGRESS 2.7

1. In a column flex container, which property aligns items horizontally?

Answer: `align-items` , because in a column the cross axis is horizontal.

2. Give a one-line way to center a child in both axes.

Answer: `display: grid; place-items: center;` on the parent.

2.8 Responsive Design, Units and Variables

2.8.1 Media queries and mobile-first

```
.grid { display: grid; grid-template-columns: 1fr; } /* phones */
@media (min-width: 768px) { .grid { grid-template-columns: 1fr 1fr; } }
@media (min-width: 1024px) { .grid { grid-template-columns: repeat(3, 1fr); } }
@media (prefers-color-scheme: dark) { body { background: #111; color: #eee; } }
```

Mobile-first means base styles target small screens and `min-width` queries add complexity. Most Indian users browse on phones, so this is the sensible default. Newer tools: **container queries** (`@container`) respond to the parent's width instead of the viewport, which is great for reusable components.

2.8.2 Units

Unit	Relative to	Use for
px	CSS pixel	Borders, fine details
rem	Root (<code>html</code>) font size, usually 16px	Font sizes, spacing (respects user settings)
em	Current element's font size	Padding that scales with button text
%	Parent's size (for width)	Fluid widths
vh / vw	1% of viewport height / width	Full-screen sections
dvh	Dynamic viewport height (changes as mobile browser bars show/hide)	Mobile full-height layouts

Worked example (em compounding): `html` = 16px. A `.card` has `font-size: 1.25em` = 20px. A nested `.title` with `font-size: 1.5em` = $1.5 \times 20 = 30\text{px}$. With `1.5rem` it would be $1.5 \times 16 = 24\text{px}$ regardless of nesting. That is why `rem` is safer for fonts.

Why dvh? On mobile, `100vh` is often taller than the visible area because of the address bar, so the bottom button hides. `100dvh` tracks the actual visible height. (`svh` and `lvh` are the smallest and largest versions.)

2.8.3 CSS variables (custom properties)

```
:root { --brand: #e85d04; --radius: 8px; }
[data-theme="dark"] { --brand: #ffba08; }
.btn { background: var(--brand); border-radius: var(--radius); }
```

Unlike Sass variables, CSS variables live at runtime: they cascade, inherit and can be changed with JavaScript (`el.style.setProperty("--brand", "red")`). That makes dark mode a matter of

switching one attribute.

✓ CHECK YOUR PROGRESS 2.8

1. Why prefer rem over px for font sizes?

Answer: rem scales with the user's browser font setting, which helps users with low vision, and it does not compound like em.

2. What is the advantage of CSS variables over Sass variables?

Answer: They exist at runtime, so they can change per theme, per component scope or via JavaScript without recompiling.

2.9 Tailwind CSS Basics

Tailwind CSS is a utility-first framework: instead of writing `.card { ... }` in a CSS file, you compose small classes in the markup.

```
<div className="flex items-center gap-4 rounded-lg bg-white p-4 shadow md:p-6 dark:bg-zinc-900">
  
  <p className="text-sm font-medium text-zinc-700 hover:text-orange-600">Asha</p>
</div>
```

- `md:` applies at the medium breakpoint and up (mobile-first). `hover:`, `dark:`, `focus-visible:` are variants.
- Only classes you actually use end up in the final CSS, so bundles stay small.
- Tailwind v4 moved configuration into CSS (`@import "tailwindcss";` and `@theme` for tokens) instead of a JS config file. Check the docs for your version.

Why teams use it	Common complaints
No naming fights (<code>.card-wrapper-inner-2</code>)	Long class strings look noisy
Design system built in (spacing, colours scale)	Learning the class names
No dead CSS; delete markup and styles go too	Repetition unless you extract components
Works well with React components and shadcn/ui	Conditional classes need helpers like <code>clsx</code> or <code>cn()</code>

💡 DO YOU KNOW?

Tailwind CSS was created by Adam Wathan and first released in 2017. It is now the default styling option offered by `create-next-app`.

 AT WORK

A common code-review bug: building class names dynamically like ``bg-${color}-500``. Tailwind scans source files for complete class strings, so this class never gets generated and the button appears unstyled in production. Use a lookup object with full names: `{ red: "bg-red-500", green: "bg-green-500" }`.

 CHECK YOUR PROGRESS 2.9

1. What does `md:flex` mean?

Answer: Apply `display: flex` from the medium breakpoint (768px by default) upward.

2. Why does ``text-${size}`` fail in Tailwind?

Answer: Tailwind generates CSS only for complete class names it finds in source files; it cannot evaluate template strings.

 THINK AND DISCUSS

Is Tailwind against "separation of concerns"? Or is the real concern the component, not the file type?

What Have We Learnt?

- Semantic elements give meaning, accessibility and free keyboard behaviour.
- Label every input; use the `name` attribute; validate again on the server.
- Accessibility: alt text, keyboard support, visible focus, 4.5:1 contrast, native elements before ARIA.
- Viewport, title, description, canonical and Open Graph tags are SEO basics; Next.js uses the metadata API.
- Box model: content, padding, border, margin; use `border-box`.
- Absolute positioning uses the nearest positioned ancestor; stacking contexts trap z-index.
- Cascade: importance, layers, specificity, then source order.
- Specificity is (IDs, classes, elements); inline beats selectors.
- Flexbox is one-dimensional; Grid is two-dimensional.
- Center anything with `display: grid; place-items: center`.
- Mobile-first with min-width queries; container queries for components.
- rem for fonts, em compounds, dvh fixes mobile full-height.
- CSS variables are runtime values ideal for theming.
- Tailwind is utility-first; never build class names dynamically.

Key terms: semantic HTML, landmark, label, constraint validation, WCAG, ARIA, alt text, contrast ratio, viewport meta, canonical, Open Graph, box model, box-sizing, margin collapse, stacking context, specificity, cascade, inheritance, flexbox, grid, fr, media query, container query, rem, em, dvh, custom property, utility-first, Tailwind.

Exercises

A. Multiple Choice Questions

- Which selector has the highest specificity?
 - `.a .b .c`
 - `#x`
 - `div p span`
 - `:where(#x)`
- With `border-box`, width 100px and padding 10px, the content width is:
 - 120px
 - 100px
 - 80px
 - 90px
- Best layout tool for a 2D dashboard of cards with fixed rows and columns:
 - float
 - Flexbox
 - Grid
 - inline-block
- Which unit tracks mobile browser bars appearing and disappearing?
 - vh
 - dvh
 - em
 - %
- Correct alt for a decorative background flourish image:
 - `alt="image"`
 - no alt
 - `alt=""`
 - `alt="decorative flourish image"`

B. Fill in the Blanks

- The first rule of ARIA is: use a native _____ element if one exists.
- An absolutely positioned element is placed relative to its nearest _____ ancestor.
- In Tailwind, the prefix _____ applies a style only at medium screens and above.

C. True or False

- Vertical margins collapse between children of a flex container.
- `em` values compound when elements are nested.
- `role="button"` on a div automatically makes it keyboard accessible.

D. Short Answer Questions

- Explain the cascade order in CSS.
- What is a stacking context, and how does it break z-index?
- Why do teams adopt Tailwind? Give two pros and two cons.

E. Long Answer Questions

- Compare Flexbox and Grid with examples, and show three ways to center an element.

2. You are asked to make an existing signup page accessible and SEO-friendly. List what you would check and change.

F. Design Challenge

Design the CSS for a product listing page: a sticky header, a sidebar of filters that becomes a top bar on phones, and a responsive grid of product cards. Use mobile-first media queries and CSS variables for a light/dark theme.

Answer Key: Chapter 2

A. MCQ

1. (b) One ID (1,0,0) beats any number of classes; `:where` is zero.
2. (c) $100 - 2 \times 10 = 80\text{px}$.
3. (c) Grid handles two dimensions.
4. (b) `dvh` is the dynamic viewport height.
5. (c) Empty `alt` hides it from screen readers.

B. Fill in the Blanks

1. HTML
2. positioned (non-static)
3. md:

C. True or False

1. False. Margin collapse does not happen for flex or grid items.
2. True. Each `em` is relative to the parent's computed font size.
3. False. You must add `tabindex="0"` and Enter/Space handlers; better use `<button>`.

D. Short Answers

1. The browser first considers origin and importance (`!important` wins over normal), then cascade layers, then specificity, then source order where the later rule wins. Inherited values apply only when no rule targets the element.
2. A stacking context is an isolated group in which children's z-index values are compared only with each other. It is created by things like `transform`, `opacity < 1`, or positioned elements with z-index. A child with z-index 9999 cannot rise above a sibling of its parent context; the fix is moving the element (portal) or adjusting the parent.
3. Pros: no class naming, consistent design tokens, no unused CSS, fast iteration in components.
Cons: noisy markup, learning curve, repetition without component extraction, dynamic class names do not work.

E. Long Answers

1. Model answer:
 - Flexbox handles one axis; content decides sizes; ideal for navbars and toolbars.
 - Grid defines rows and columns; ideal for page shells and galleries.
 - Flex properties: `justify-content` (main axis), `align-items` (cross axis), `gap`, `flex: 1`.
 - Grid properties: `grid-template-columns`, `repeat`, `minmax`, `fr`, `grid-template-areas`.
 - Card grid example: `repeat(auto-fill, minmax(200px, 1fr))`.
 - Center 1: flex with `justify-content` and `align-items center`.
 - Center 2: grid with `place-items center`.
 - Center 3: absolute top/left 50% with `translate(-50%, -50%)`.
 - Parent needs a height for vertical centering.

2. Model answer:

- Use semantic landmarks (header, main, form) and one h1.
- Labels for every input, correct type and autocomplete.
- Error messages linked with aria-describedby, announced via aria-live.
- Real <button type="submit"> ; full keyboard flow; visible focus.
- Contrast at least 4.5:1; meaningful alt text.
- Run Lighthouse and axe.
- SEO: title, description, canonical, Open Graph, viewport.
- Fast loading and server-side validation.

F. Design Challenge: Model Solution

Define `:root { --bg: #fff; --fg: #111; --accent: #e85d04; }` and override them under `[data-theme="dark"]`. Header: `position: sticky; top: 0; z-index: 10; background: var(--bg)`. Base layout (phones): `.page { display: grid; grid-template-columns: 1fr; }` with filters as a horizontal scrollable flex row (`display: flex; overflow-x: auto; gap: .5rem`). At `@media (min-width: 1024px)`: `.page { grid-template-columns: 16rem 1fr; }` and filters become a vertical list with `position: sticky; top: 4rem`. Products: `display: grid; grid-template-columns: repeat(auto-fill, minmax(12rem, 1fr)); gap: 1rem`. Use rem for spacing, border-box globally, and `min-height: 100dvh` for the page.