Next.js Notes

Chapter 1 —> Birth

JavaScript Evolution

- Created by Brendan Eich (1995) at Netscape.

- Next.js was created in **2016** by **Vercel** (led by Guillermo Rauch) to address React's limitations.

- Framework progression:　　jQuery → Angular → Node.js → React.js → Next.js

Hello World Example

- **Vanilla JS**: Verbose DOM manipulation.

- **jQuery**: Simplified syntax.

- **Angular/React**: More code for this example but scalable in "bigger picture" (component-based).

Why Modern Frameworks?

- **Component Architecture**: Reusable UI pieces (e.g., buttons).

- **Virtual DOM**: Efficient UI updates (only changes rendered).

- **Ecosystem**: Strong community, documentation, and tools.

- Modern frameworks improve efficiency, scalability, and performance.

Chapter 2 —> Introduction

Next.js is a **full-stack web framework** built on top of **React.js** or simply we can say it's a React framework. While React is a **UI library** that focuses on building components, Next.js extends it into a complete framework for building **production-grade web applications**.

What is a Framework?

- A framework serves as a tool equipped with predefined rules and conventions that offer a structured approach for building applications.

- Handles database integration, routing, authentication, etc.

- Helps developers focus on writing application logic rather than low-level setups.

Key features of Next.js:

1. Solves React limitations (SEO, routing, performance)

2. Built-in features:

   - File-based routing

   - Efficient code splitting

   - Hybrid rendering (SSR/SSG)

   - Built-in optimizations (images, fonts, SEO)

   - HMR (Hot Module Replacement)

   - API Routes (backend)

   - Built-in support for Sass

   - CSS modules

   - Data fetching choice (SSG, SSR, ISR)

   - Error handling

   - Metadata API (For SEO)

   - Internationalization(support for any spoken language), etc.

Why Use a React Framework like Next.js?

1. Less Tooling Time

   - No need to configure bundlers, compilers, formatters, etc.

   - Built-in support for routing, rendering, auth, and more.

   - Focus more on business logic and React code.

2. Easy Learning Curve

   - Easier to learn if you're already familiar with React.

   - Includes backend features but without complex setup (no routing config needed).

3. Improved Performance

   - Built-in SSR (Server-Side Rendering) & SSG (Static Site Generation).

   - Automatic code splitting for faster page loads and better UX.

   - React has introduced React Server Components for SSR, but Next.js automates the setup.

   > Follows "Convention over Configuration" = less boilerplate code.

4. SEO Advantage

   - React.js renders everything on the client side, sending a minimal initial HTML response from the server. The server sends a minimal HTML file code and a JavaScript file that the browser executes to generate the HTML —hard for search engines to crawl.

   - Next.js sends **full HTML file** and minimal JavaScript code to render only the content requiring client-side interaction.

   - This improves:

     - Visibility

     - Ranking

     - Traffic

     - User trust

When to Use Next.js over React

Choose **Next.js** when:

- You care about **SEO**

- You want **fast page loads** (via SSR/SSG)

- You don't want to configure everything yourself

- You want an all-in-one full-stack React framework

- You need **routing, data fetching**, and **backend API** in one codebase

Choose **React (only)** when:

- You're building a **simple SPA or PWA**

- You need complete control over the setup

- You're integrating into an existing app (e.g., with a non-React backend)

Chapter 3 —> Prerequisites

Web Development Fundamentals

1. HTML -

   a. Structure
      <!DOCTYPE>,<html>,<head>,<body>

   b. Elements
      headings, paragraph, lists, <a>, <img>, <input>, <textarea>, <button>, <div>

   c. Semantics
      header, nav, main, section, aside, footer

```html
<header>Site Logo/Navigation</header>
<nav>
 <a href="/">Home</a> | <a href="/about">About</a>
</nav>
<main>
 <section id="intro">
  <h2>Welcome</h2>
  <p>Introduction text...</p>
 </section>
 <aside>Related links (Content indirectly related to main content)</asi
```

```
de>
</main>
<footer>Copyright © 2024</footer>
```

d. Forms

handling user input, perform form validations by using form element and onSubmit event listener

```
<form onsubmit="validateForm()">
  <label for="name">Name:</label>
  <input type="text" id="name" required>

  <label for="email">Email:</label>
  <input type="email" id="email" required>

  <button type="submit">Submit</button>
</form>
```

2. CSS -

a. Structure

Box model - padding, margin, border
Selectors - type, class, id, child, sibling
Typography - font, size, weight, alignment
Colors & Background - colors, gradients, background images

```
/* Box model */
div {
  width: 300px;
  padding: 20px;    /* Inner space */
  border: 2px solid black;
  margin: 30px;     /* Outer space */
}

/* Type */ h1 { color: blue; }
/* Class */ .btn { background: red; }
```

```css
/* ID */ #header { height: 80px; }
/* Child */ ul > li { list-style: none; }
/* Sibling */ h2 + p { margin-top: 0; }

body {
  font-family: 'Arial', sans-serif;
  font-size: 16px;
  line-height: 1.5;
  font-weight: 400/bold;
  text-align: center;
}

.element {
  color: #ffffff; /* Text color */
  background-color: rgba(0,0,0,0.5);
  /* A gradient is like a smooth blend of two or more colors. Instead of
  one solid color, the colors gradually change. */
  background: linear-gradient(to right/135deg, red, yellow);
  background-image: url('image.jpg');
}
```

b. Layout and Positioning (Refer NotesFS)
Display - block, inline, inline-block
Position - relative, absolute, sticky, fixed
Flexbox & Grid

c. Effects
Transition - Learn to create smooth transitions using different CSS properties like delay, duration, property, timing-function
Think of a transition like a magic trick: when you change something—like the color or size of a box —the change doesn't happen instantly; it slides or fades smoothly. You control how long it takes, and how it moves.
Key properties:

- `transition-property` : what you want to change (e.g., `background-color` , `transform` , `width` , `opacity`)

- `transition-duration` : how long the change takes (e.g., `2s` for two seconds)

- `transition-delay` : wait this long before starting (e.g., `0.5s`)

- `transition-timing-function` : how the speed of the change feels like "slow at start," "fast in the middle".

`linear`	Same speed from start to finish
`ease`	Starts slow, speeds up, then slows down
`ease-in`	Starts slow, then speeds up
`ease-out`	Starts fast, then slows down
`ease-in-out`	Slow → Fast → Slow
`cubic-bezier(...)`	Custom timing with control points

Transformations - Explore 2D and 3D transformations like scaling, rotating, translating elements

Think of a piece of paper. You can **rotate it**, **scale it**, or **move it**. CSS lets you do this to elements on a web page.

Types of Transforms:

2D Transforms:

Transform	What it does
`translate(x, y)`	Moves element left/right (x) or up/down (y)
`rotate(deg)`	Rotates the element (like a clock hand)
`scale(x, y)`	Grows or shrinks the element
`skew(x, y)`	Tilts the element

3D Transforms:

Transform	What it does
`rotateX(deg)`	Rotates around X-axis (up/down flip)
`rotateY(deg)`	Rotates around Y-axis (sideways flip)
`translateZ(px)`	Moves closer/farther away (depth)

Animations - Learn how to create animations using keyframes

Think of a cartoon—it's made of **frames**. In CSS, **keyframes** tell the

browser how an element should change over time.

How It Works:

Define `@keyframes name { ... }` with percentages (from 0% to 100%).

Apply that animation with:

- `animation-name`

- `animation-duration`

- `animation-timing-function` , etc.

Shadows and Gradients - Explore with box shadows and linear or radial gradients

Shadows

- **Box-shadow**: gives an element a shadow, like a floating box.

  Syntax: `box-shadow: offsetX offsetY blur spread color;`

Gradients

- **Linear-gradient**: colors fade in a straight line.

- **Radial-gradient**: colors fade in a circle (like a spotlight).

```css
/* Transition */
.button {
  transition: <property> <duration> <timing-function> <delay>;
  transition: background-color 0.3s ease 2s;

  /* comma-separate transitions for multiple properties */
  transition: background-color 0.5s ease, transform 0.3s linear;
}
.button:hover {
  background-color: blue;
}


/* Transformation */
.element {
  transform: rotate(15deg) scale(1.1);
}
```

```css
/* Animation */
@keyframes slide {
  from { transform: translateX(-100%); }
  to { transform: translateX(0); }
}
.slide-in {
  animation: slide 0.5s forwards;
}

/* Shadows and Gradient */
.card {
  box-shadow: 2px 2px 10px rgba(0,0,0,0.1);
  background: linear-gradient(45deg, red, blue);
}
```

d. Advanced (Plus)

Learn how to use CSS processors like sass or frameworks like TailwindCSS for more powerful and efficient styling

What are they?

- **Sass**: a helpful tool that lets you write **variables**, **nest CSS rules**, and reuse code pieces. Then it *magically* turns into normal CSS.

- **Tailwind CSS**: a toolkit with lots of tiny building blocks (classes) you can combine quickly to style your page. No writing long CSS—just use class names!

Since these require setup and not pure HTML+CSS, here's a simple illustration to show how they make styling easier:

```scss
/* Imagine this is Sass — it doesn't work directly in HTML */
/* Pretend file: style.scss */
$main-color: tomato;

.nav {
  background: $main-color;
```

```
  ul {
    list-style: none;
    li {
      display: inline-block;
      margin-right: 10px;
    }
  }
}

/* This compiles to regular CSS like: */
.nav { background: tomato; }
.nav ul { list-style: none; }
.nav ul li { display: inline-block; margin-right: 10px; }
```

3. JS -

    a. Variables and Data Types

    b. Operators

    c. Control Flow

    d. Functions

    e. DOM Manipulation