TypeScript and Next.js Learning Guide: From Basics to Advanced

Table of Contents

Introduction

Why TypeScript and Next.js?

Both TypeScript and Next.js were created to solve specific problems in web development:

- **TypeScript** addresses JavaScript's dynamic typing limitations by adding a static type system, making code more predictable, easier to debug, and better suited for large-scale applications.

- **Next.js** extends React with server-side rendering, simplified routing, and an optimized developer experience to address challenges in building production-ready React applications.

Together, they form a powerful combination for building robust, maintainable, and high-performance web applications.

Evolution from Predecessors

TypeScript's Evolution from JavaScript:

- Released by Microsoft in 2012 as a superset of JavaScript
- Adds optional static typing to JavaScript's dynamic typing
- Compiles down to plain JavaScript that runs in any environment
- Maintains full compatibility with JavaScript while adding type safety

Next.js' Evolution from React:

- Created by Vercel (formerly Zeit) in 2016
- Builds upon React's component model
- Adds server-side rendering capabilities to React's client-side rendering
- Provides file-based routing instead of requiring manual route configuration
- Introduces simplified data fetching methods
- Offers built-in optimizations and developer experience improvements

TypeScript Fundamentals

The Philosophy Behind TypeScript

TypeScript follows several core principles:

1. **Progressive Enhancement**: TypeScript is a superset of JavaScript, meaning any valid JavaScript is also valid TypeScript.
2. **Type Inference**: TypeScript can often infer types without explicit declarations, reducing the need for verbose type annotations.
3. **Structural Typing**: TypeScript uses a structural type system (duck typing) rather than a nominal type system.
4. **Compile-Time Checking**: TypeScript catches errors at compile time rather than runtime.
5. **Erasable Types**: Type information exists only at compile time and is erased in the compiled JavaScript output.

Setting Up a TypeScript Development Environment

Prerequisites

- Node.js and npm installed
- A code editor (VS Code recommended for its excellent TypeScript support)

Basic Setup

1. **Install TypeScript globally:**

   ```
   npm install -g typescript
   ```

2. **Create a new project:**

   ```
   mkdir ts-project
   cd ts-project
   npm init -y
   ```

3. **Install TypeScript locally:**

   ```
   npm install typescript --save-dev
   ```

4. **Initialize TypeScript configuration:**

   ```
   npx tsc --init
   ```

5. **Create a simple TypeScript file (index.ts):**

```
function greet(name: string): string {
  return `Hello, ${name}!`;
}

console.log(greet('TypeScript'));
```

6. **Compile and run:**

```
npx tsc
node index.js
```

VS Code Setup

1. Install VS Code
2. Install the TypeScript and ESLint extensions
3. Configure settings.json for TypeScript:

```
{
  "typescript.updateImportsOnFileMove.enabled": "always",
  "typescript.suggestionActions.enabled": true,
  "typescript.preferences.quoteStyle": "single",
  "editor.formatOnSave": true
}
```

Basic Type System

Primitive Types

```
// Basic primitives
let isDone: boolean = false;
let decimal: number = 6;
let color: string = 'blue';

// Special primitives
let notFound: null = null;
let notDefined: undefined = undefined;
let penta: symbol = Symbol('star');
let bigInt: bigint = 100n;
```

Arrays and Tuples

```
// Arrays
let list: number[] = [1, 2, 3];
let names: Array<string> = ['Alice', 'Bob', 'Charlie']; // Generic array type

// Tuples (fixed-length arrays with ordered types)
let person: [string, number] = ['Alice', 30]; // Name and age
```

Objects

```
// Object with specified properties
let user: { name: string; age: number } = {
  name: 'Alice',
  age: 30,
};

// Object with optional properties
let profile: { name: string; age?: number } = {
  name: 'Bob',
  // age is optional
};
```

Functions

```
// Function with type annotations
function add(x: number, y: number): number {
  return x + y;
}

// Function type with arrow syntax
let multiply: (x: number, y: number) => number;
multiply = (x, y) => x * y;

// Optional and default parameters
function buildName(firstName: string, lastName?: string): string {
  return lastName ? `${firstName} ${lastName}` : firstName;
}

function greet(name: string, greeting = 'Hello'): string {
  return `${greeting}, ${name}!`;
}

// Rest parameters
function sum(...numbers: number[]): number {
  return numbers.reduce((total, n) => total + n, 0);
}
```

Enums

```
// Numeric enum
enum Direction {
  Up, // 0
  Down, // 1
  Left, // 2
  Right, // 3
}

// String enum
enum HttpStatus {
  OK = 'OK',
  NotFound = 'NOT_FOUND',
  ServerError = 'SERVER_ERROR',
}

// Usage
let status: HttpStatus = HttpStatus.OK;
```

Any, Unknown, and Never

```
// Any - opt out of type checking (use sparingly)
let notSure: any = 4;
notSure = 'maybe a string';
notSure = false;

// Unknown - safer alternative to any
let mystery: unknown = 4;
if (typeof mystery === 'string') {
  console.log(mystery.toUpperCase()); // Only allowed after type check
}

// Never - represents values that never occur
function throwError(message: string): never {
  throw new Error(message);
}

function infiniteLoop(): never {
  while (true) {}
}
```

Interfaces vs Types

Interfaces

```
// Basic interface
interface User {
  name: string;
  age: number;
  email?: string; // Optional property
  readonly id: number; // Read-only property
}

// Interface with methods
interface Greeter {
  greet(name: string): string;
}

// Implementing interfaces with classes
class EnglishGreeter implements Greeter {
  greet(name: string): string {
    return `Hello, ${name}!`;
  }
}

// Extending interfaces
interface Employee extends User {
  department: string;
  salary: number;
}

// Merging declarations (a unique feature of interfaces)
interface Animal {
  name: string;
}

interface Animal {
  age: number;
}

// The final Animal interface will have both name and age
```

Type Aliases

```
// Basic type alias
type Point = {
  x: number;
  y: number;
};

// Union types
type ID = string | number;

// Intersection types
type Employee = Person & {
  employeeId: number;
  department: string;
};

// Type aliases for functions
type GreetFunction = (name: string) => string;

// Type aliases with generics
type Result<T> = {
  data: T;
  error: Error | null;
};
```

When to Use Interface vs Type

Use Interfaces When:

- Defining object shapes that might be extended later
- Working with classes that need to implement a contract
- You want to leverage declaration merging
- Following object-oriented design patterns

Use Types When:

- Creating union or intersection types
- Defining complex types that aren't just shapes of objects
- Creating mapped or conditional types
- Working with primitives or tuples
- You need to use features like mapped types

Classes in TypeScript

```
class Person {
  // Properties with access modifiers
  private _name: string;
  protected age: number;
  readonly id: number;

  // Static property
  static species = 'Human';

  // Constructor
  constructor(name: string, age: number, id: number) {
    this._name = name;
    this.age = age;
    this.id = id;
  }

  // Getter
  get name(): string {
```

```typescript
      return this._name;
    }

    // Setter
    set name(value: string) {
      if (value.length > 0) {
        this._name = value;
      }
    }

    // Method
    greet(): string {
      return `Hello, I'm ${this._name} and I'm ${this.age} years old.`;
    }

    // Static method
    static createAnonymous(): Person {
      return new Person('Anonymous', 0, Math.floor(Math.random() * 1000));
    }
  }

  // Inheritance
  class Employee extends Person {
    department: string;

    constructor(name: string, age: number, id: number, department: string) {
      // Call parent constructor
      super(name, age, id);
      this.department = department;
    }

    // Override method
    greet(): string {
      return `${super.greet()} I work in the ${this.department} department.`;
    }
  }

  // Abstract classes
  abstract class Shape {
    color: string;

    constructor(color: string) {
      this.color = color;
    }

    abstract calculateArea(): number; // Must be implemented by subclasses
  }

  class Circle extends Shape {
    radius: number;

    constructor(color: string, radius: number) {
      super(color);
      this.radius = radius;
    }

    calculateArea(): number {
      return Math.PI * this.radius * this.radius;
    }
  }
```

Modules and Namespaces

ES Modules (Recommended)

math.ts

```
// Named exports
export function add(x: number, y: number): number {
  return x + y;
}

export function subtract(x: number, y: number): number {
  return x - y;
}

// Default export
export default class Calculator {
  multiply(x: number, y: number): number {
    return x * y;
  }

  divide(x: number, y: number): number {
    return x / y;
  }
}
```

app.ts

```
// Import named exports
import { add, subtract } from './math';

// Import default export
import Calculator from './math';

// Import everything as a namespace
import * as MathUtils from './math';

console.log(add(5, 3)); // 8
console.log(subtract(5, 3)); // 2

const calc = new Calculator();
console.log(calc.multiply(5, 3)); // 15
console.log(MathUtils.add(5, 3)); // 8
```

Namespaces (Legacy)

```
// Define a namespace
namespace Geometry {
  export interface Point {
    x: number;
    y: number;
  }

  export class Circle {
    constructor(public center: Point, public radius: number) {}

    area(): number {
      return Math.PI * this.radius * this.radius;
    }
  }

  // Nested namespace
  export namespace ThreeDimensional {
    export interface Point {
      x: number;
      y: number;
      z: number;
    }
  }
}

// Usage
const point: Geometry.Point = { x: 0, y: 0 };
const circle = new Geometry.Circle(point, 10);
console.log(circle.area());

const point3d: Geometry.ThreeDimensional.Point = { x: 0, y: 0, z: 0 };
```

Advanced TypeScript

Advanced Types

Union Types

```
// Union type
type ID = string | number;

function printId(id: ID) {
  console.log(`ID: ${id}`);

  // Narrowing
  if (typeof id === 'string') {
    console.log(id.toUpperCase());
  } else {
    console.log(id.toFixed(2));
  }
}
```

Intersection Types

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

type Employee = {
  employeeId: number;
  department: string;
};

// Combine the types
type EmployeePerson = Person & Employee;

const worker: EmployeePerson = {
  name: 'Alice',
  age: 30,
  employeeId: 123,
  department: 'Engineering',
};
```

Generics

```
// Generic function
function identity<T>(arg: T): T {
  return arg;
}

const num = identity<number>(42);
const str = identity('hello'); // Type inferred as string

// Generic interface
interface Box<T> {
  value: T;
}

const numberBox: Box<number> = { value: 42 };
const stringBox: Box<string> = { value: 'hello' };

// Generic classes
class Queue<T> {
  private items: T[] = [];

  enqueue(item: T): void {
    this.items.push(item);
  }

  dequeue(): T | undefined {
    return this.items.shift();
  }
}

const numberQueue = new Queue<number>();
numberQueue.enqueue(1);
numberQueue.enqueue(2);
console.log(numberQueue.dequeue()); // 1

// Generic constraints
interface Lengthwise {
  length: number;
}

function logLength<T extends Lengthwise>(arg: T): void {
  console.log(arg.length);
}

logLength('hello'); // 5
logLength([1, 2, 3]); // 3
// logLength(123); // Error: number doesn't have a length property
```

Type Narrowing and Type Guards

```
// Type narrowing with typeof
function padLeft(value: string, padding: string | number) {
  if (typeof padding === 'number') {
    // padding is narrowed to number
    return ' '.repeat(padding) + value;
  } else {
    // padding is narrowed to string
    return padding + value;
  }
}

// Type narrowing with instanceof
class Bird {
  fly() {
    console.log('Flying...');
  }
}

class Fish {
  swim() {
    console.log('Swimming...');
  }
}

function move(animal: Bird | Fish) {
  if (animal instanceof Bird) {
    animal.fly();
  } else {
    animal.swim();
  }
}

// Custom type guards with predicate functions
interface Car {
  make: string;
  model: string;
  year: number;
}

interface Motorcycle {
  make: string;
  model: string;
  year: number;
  type: 'sport' | 'cruiser' | 'touring';
}

// Type guard function
function isCar(vehicle: Car | Motorcycle): vehicle is Car {
  return (vehicle as Motorcycle).type === undefined;
}

function describeVehicle(vehicle: Car | Motorcycle) {
  if (isCar(vehicle)) {
    // vehicle is narrowed to Car
    console.log(`Car: ${vehicle.make} ${vehicle.model} (${vehicle.year})`);
  } else {
    // vehicle is narrowed to Motorcycle
    console.log(
      `Motorcycle: ${vehicle.make} ${vehicle.model} (${vehicle.year}) -
${vehicle.type}`
    );
  }
}
```

Utility Types

```
// Partial<T> - Makes all properties optional
interface User {
  name: string;
  age: number;
  email: string;
}

function updateUser(user: User, updates: Partial<User>): User {
  return { ...user, ...updates };
}

const user: User = {
  name: 'John',
  age: 30,
  email: 'john@example.com',
};

const updatedUser = updateUser(user, { age: 31 });

// Required<T> - Makes all properties required
interface Config {
  host?: string;
  port?: number;
  protocol?: string;
}

const completeConfig: Required<Config> = {
  host: 'localhost',
  port: 8080,
  protocol: 'https',
};

// Readonly<T> - Makes all properties readonly
const frozenUser: Readonly<User> = {
  name: 'Alice',
  age: 25,
  email: 'alice@example.com',
};

// frozenUser.age = 26; // Error: Cannot assign to 'age' because it is a read-only
property

// Record<K, T> - Creates a type with properties of K and values of type T
type UserRoles = Record<string, boolean>;

const permissions: UserRoles = {
  canEdit: true,
  canDelete: false,
  canCreate: true,
};

// Pick<T, K> - Picks a set of properties K from T
type UserBasicInfo = Pick<User, 'name' | 'email'>;
```

```
const basicInfo: UserBasicInfo = {
  name: 'Bob',
  email: 'bob@example.com',
};

// Omit<T, K> - Removes a set of properties K from T
type UserWithoutAge = Omit<User, 'age'>;

const noAgeUser: UserWithoutAge = {
  name: 'Charlie',
  email: 'charlie@example.com',
};

// Exclude<T, U> - Excludes types in U from T
type Numbers = 1 | 2 | 3 | 4 | 5;
type EvenNumbers = Exclude<Numbers, 1 | 3 | 5>;
// EvenNumbers = 2 | 4

// Extract<T, U> - Extracts types in U from T
type OddNumbers = Extract<Numbers, 1 | 3 | 5>;
// OddNumbers = 1 | 3 | 5

// NonNullable<T> - Removes null and undefined from T
type MaybeString = string | null | undefined;
type DefinitelyString = NonNullable<MaybeString>;
// DefinitelyString = string

// ReturnType<T> - Gets the return type of a function type
function createUser(name: string, age: number): User {
  return { name, age, email: `${name}@example.com` };
}

type CreateUserReturn = ReturnType<typeof createUser>;
// CreateUserReturn = User

// Parameters<T> - Gets the parameter types of a function type
type CreateUserParams = Parameters<typeof createUser>;
// CreateUserParams = [string, number]
```

Declaration Files and Working with Libraries

Creating Declaration Files

math.ts

```
export function add(x: number, y: number): number {
  return x + y;
}

export function multiply(x: number, y: number): number {
  return x * y;
}
```

math.d.ts (Declaration file)

```
export declare function add(x: number, y: number): number;
export declare function multiply(x: number, y: number): number;
```

Consuming Declaration Files

```
// For libraries with declarations
import * as React from 'react'; // TypeScript knows React's types

// For libraries without declarations, you can install them separately
// npm install --save-dev @types/lodash
import * as _ from 'lodash';

// Or create a module declaration file
// my-module.d.ts
declare module 'some-untyped-module' {
  export function doSomething(value: string): number;
  export function doSomethingElse(): void;
}
```

Ambient Declarations

```
// Declare global variables (e.g., from a script tag)
declare const API_URL: string;

// Extend existing interfaces
declare global {
  interface Window {
    myCustomProperty: string;
  }
}

// Use them
console.log(API_URL);
console.log(window.myCustomProperty);
```

TypeScript Configuration (tsconfig.json)

```
{
  "compilerOptions": {
    // Basic Options
    "target": "es2020", // ECMAScript target version
    "module": "esnext", // Module code generation
    "lib": ["dom", "es2020"], // Library files to include
    "jsx": "react", // JSX code generation
    "sourceMap": true, // Generate source maps
    "outDir": "./dist", // Output directory
    "rootDir": "./src", // Root directory
    "removeComments": true, // Remove comments in output

    // Strict Type-Checking Options
    "strict": true, // Enable all strict type-checking options
    "noImplicitAny": true, // Error on implied 'any' type
    "strictNullChecks": true, // Enable strict null checks
    "strictFunctionTypes": true, // Enable strict checking of function types
    "noImplicitThis": true, // Error on 'this' with implied 'any' type
    "alwaysStrict": true, // Parse in strict mode

    // Additional Checks
    "noUnusedLocals": true, // Report errors on unused locals
    "noUnusedParameters": true, // Report errors on unused parameters
    "noImplicitReturns": true, // Report error when not all code paths return
    "noFallthroughCasesInSwitch": true, // Report errors for fallthrough cases in
switch

    // Module Resolution Options
    "moduleResolution": "node", // Module resolution strategy
    "baseUrl": "./", // Base directory for resolving non-relative module names
    "paths": {
      // Path mapping for module resolution
      "@app/*": ["src/app/*"],
      "@components/*": ["src/components/*"]
    },
    "esModuleInterop": true, // Emit __importStar and __importDefault helpers
    "resolveJsonModule": true, // Include modules imported with .json extension

    // Advanced Options
    "forceConsistentCasingInFileNames": true, // Disallow inconsistently-cased
references to the same file
    "skipLibCheck": true // Skip type checking of declaration files
  },
  "include": [
    // Files to include
    "src/**/*"
  ],
  "exclude": [
    // Files to exclude
    "node_modules",
    "dist"
  ]
}
```

Advanced Patterns

Discriminated Unions

```typescript
// Define types with a common discriminant property
interface Circle {
  kind: 'circle';
  radius: number;
}

interface Square {
  kind: 'square';
  sideLength: number;
}

interface Rectangle {
  kind: 'rectangle';
  width: number;
  height: number;
}

// Union type
type Shape = Circle | Square | Rectangle;

// Function that uses the discriminant to determine the shape
function calculateArea(shape: Shape): number {
  switch (shape.kind) {
    case 'circle':
      return Math.PI * shape.radius ** 2;
    case 'square':
      return shape.sideLength ** 2;
    case 'rectangle':
      return shape.width * shape.height;
    default:
      // Exhaustiveness checking
      const _exhaustiveCheck: never = shape;
      throw new Error(`Unhandled shape kind: ${_exhaustiveCheck}`);
  }
}

const circle: Circle = { kind: 'circle', radius: 5 };
console.log(calculateArea(circle)); // 78.54...
```

Mapped Types

```typescript
// Basic mapped type
type Readonly<T> = {
  readonly [P in keyof T]: T[P];
};
```

```
interface User {
  name: string;
  age: number;
}

const readonlyUser: Readonly<User> = {
  name: 'Alice',
  age: 30,
};

// readonlyUser.name = "Bob"; // Error: Cannot assign to 'name' because it is a
read-only property

// Mapped type with modifiers
type Optional<T> = {
  [P in keyof T]?: T[P];
};

// Mapped type that changes property types
type Stringify<T> = {
  [P in keyof T]: string;
};

const stringifiedUser: Stringify<User> = {
  name: 'Bob',
  age: '30', // Now a string instead of a number
};

// Mapped type with filtering
type PickByType<T, U> = {
  [P in keyof T as T[P] extends U ? P : never]: T[P];
};

interface Person {
  name: string;
  age: number;
  isActive: boolean;
}

type StringProperties = PickByType<Person, string>;
// Result: { name: string }
```

Conditional Types

```
// Basic conditional type
type TypeName<T> = T extends string
  ? 'string'
  : T extends number
  ? 'number'
  : T extends boolean
```

```
    ? 'boolean'
    : T extends undefined
    ? 'undefined'
    : T extends Function
    ? 'function'
    : 'object';

type T0 = TypeName<string>; // "string"
type T1 = TypeName<number>; // "number"
type T2 = TypeName<boolean>; // "boolean"
type T3 = TypeName<() => void>; // "function"
type T4 = TypeName<{}>; // "object"

// Conditional types with inference
type ReturnType<T> = T extends (...args: any[]) => infer R ? R : any;

function add(a: number, b: number): number {
  return a + b;
}

type AddReturnType = ReturnType<typeof add>; // number

// Distributed conditional types
type ToArray<T> = T extends any ? T[] : never;

type StrArrOrNumArr = ToArray<string | number>;
// StrArrOrNumArr = string[] | number[]

// Conditional type constraints
type NonNullable<T> = T extends null | undefined ? never : T;

type T5 = NonNullable<string | null | undefined>; // string
```

Type-Level Programming

```
// Recursive types
type JSONValue =
  | string
  | number
  | boolean
  | null
  | { [key: string]: JSONValue }
  | JSONValue[];

// Deep partial type
type DeepPartial<T> = T extends object
  ? {
      [P in keyof T]?: DeepPartial<T[P]>;
    }
  : T;
```

```
interface DeepObject {
  foo: {
    bar: {
      baz: string;
    };
  };
}

const partial: DeepPartial<DeepObject> = {
  foo: {
    bar: {}, // baz is optional
  },
};

// String manipulation types
type CamelCase<S extends string> = S extends `${infer P}_${infer Q}${infer R}`
  ? `${P}${Uppercase<Q>}${CamelCase<R>}`
  : S;

type T6 = CamelCase<'hello_world'>; // "helloWorld"

// Tuple operations
type Tail<T extends any[]> = T extends [any, ...infer R] ? R : never;

type T7 = Tail<[1, 2, 3]>; // [2, 3]

type Prepend<E, T extends any[]> = [E, ...T];

type T8 = Prepend<0, [1, 2, 3]>; // [0, 1, 2, 3]
```

Migrating from JavaScript to TypeScript

Gradual Migration Strategy

1. **Setup TypeScript in your project:**

   ```
   npm install --save-dev typescript
   npx tsc --init
   ```

2. **Configure tsconfig.json for migration:**

   ```
   {
     "compilerOptions": {
       "target": "es2020",
       "module": "esnext",
       "allowJs": true, // Allow JavaScript files
       "checkJs": false, // Don't type-check JavaScript files initially
       "jsx": "react",
       "outDir": "./dist",
   ```

```
      "strict": false, // Start with loose typing
      "noImplicitAny": false, // Don't require explicit any types
      "moduleResolution": "node",
      "esModuleInterop": true,
      "skipLibCheck": true,
      "forceConsistentCasingInFileNames": true
    },
    "include": ["src/**/*"]
  }
```

3. **Rename .js files to .ts/.tsx one by one:**
    - Start with simpler, standalone files
    - Fix any errors as they appear
    - Add type annotations gradually
    - Use `// @ts-ignore` or `any` for complex cases initially
4. **Gradually tighten TypeScript configuration:**
    - Enable `"noImplicitAny": true`
    - Enable `"strict": true`
    - Remove `// @ts-ignore` and `any` types

JavaScript vs. TypeScript Examples

JavaScript:

```
function calculateTotal(items, tax) {
  let total = 0;
  for (const item of items) {
    if (item.price) {
      total += item.price;
    }
  }
  return total * (1 + tax);
}

const items = [
  { name: 'Book', price: 10.99 },
  { name: 'Pen', price: 1.99 },
  { name: 'Coffee', price: 3.99 },
];

const orderTotal = calculateTotal(items, 0.07);
console.log(`Order total: ${orderTotal.toFixed(2)}`);
```

TypeScript:

```
interface Item {
  name: string;
  price: number;
}

function calculateTotal(items: Item[], tax: number): number {
  let total = 0;
  for (const item of items) {
    total += item.price;
  }
  return total * (1 + tax);
}

const items: Item[] = [
  { name: 'Book', price: 10.99 },
  { name: 'Pen', price: 1.99 },
  { name: 'Coffee', price: 3.99 },
];

const orderTotal = calculateTotal(items, 0.07);
console.log(`Order total: $${orderTotal.toFixed(2)}`);
```

Common TypeScript Pitfalls and How to Avoid Them

Any Type Overuse

Problem:

```
function processData(data: any) {
  return data.length; // No type safety
}
```

Solution:

```
function processData<T extends { length: number }>(data: T) {
  return data.length; // Type-safe
}
```

Type Assertions vs. Type Casting

Problem:

```
const value: any = 'hello';
const length: number = <number>value; // Type assertion, but no runtime conversion
```

Solution:

```
const value: any = 'hello';
if (typeof value === 'number') {
  const length: number = value; // Type-safe after runtime check
} else {
  const length: number = Number(value); // Explicit conversion
}
```

Non-null Assertion Operator Misuse

Problem:

```
function getUser(id: string): User | null {
  // Return user or null
}

const user = getUser('123')!; // Assumes user is not null
console.log(user.name); // May cause runtime error
```

Solution:

```
function getUser(id: string): User | null {
  // Return user or null
}

const user = getUser('123');
if (user) {
  console.log(user.name); // Safe
}
```

Object Literal Type Widening

Problem:

```
const config = {
  apiUrl: 'https://api.example.com',
  timeout: 3000,
};

// Later
config.apiUrl = 123; // TypeScript allows this even though it should be a string
```

Solution:

```
const config = {
  apiUrl: 'https://api.example.com',
  timeout: 3000,
} as const; // Makes object readonly and types exact

// Now this is an error
// config.apiUrl = 123; // Error: Cannot assign to 'apiUrl' because it is a read-
only property
```

Function Parameter Bivariance

Problem:

```
interface Person {
  name: string;
  age: number;
}

interface User extends Person {
  email: string;
}

let processPerson = (person: Person) => {};
let processUser: (user: User) => void;

processUser = processPerson; // TypeScript allows this by default
// This could lead to runtime errors if processPerson tries to access user.email
```

Solution:

```
// Enable strictFunctionTypes in tsconfig.json
{
  "compilerOptions": {
    "strictFunctionTypes": true
  }
}
```

Next.js Fundamentals

The Philosophy Behind Next.js

Next.js was built on several core principles:

1. **Developer Experience**: Making React development more efficient and enjoyable.
2. **Performance by Default**: Automatic code splitting, optimized images, and smart bundling.

3. **Zero Configuration**: Works out of the box while being highly customizable.
4. **Hybrid Rendering**: Supports multiple rendering strategies (SSR, SSG, CSR, ISR).
5. **Platform Agnostic**: Can be deployed anywhere (Vercel, AWS, self-hosted, etc.).

Setting Up a Next.js Development Environment

Basic Setup

1. **Create a new Next.js project:**

   ```
   npx create-next-app my-next-app
   # or with TypeScript
   npx create-next-app my-next-app --typescript
   ```

2. **Navigate to the project directory:**

   ```
   cd my-next-app
   ```

3. **Start the development server:**

   ```
   npm run dev
   ```

4. **Open your browser and navigate to http://localhost:3000**

Project Structure

```
my-next-app/
├── app/                    # App Router (Next.js 13+)
│   ├── layout.tsx          # Root layout
│   ├── page.tsx            # Home page
│   └── ...                 # Other routes
├── pages/                  # Pages Router (legacy but still supported)
│   ├── _app.tsx            # Custom App component
│   ├── _document.tsx       # Custom Document component
│   ├── index.tsx           # Home page
│   ├── about.tsx           # About page
│   └── api/                # API routes
│       └── hello.ts        # API endpoint
├── public/                 # Static files
│   ├── favicon.ico
│   └── images/
├── components/             # Reusable components
│   ├── Header.tsx
│   └── Footer.tsx
├── styles/                 # CSS styles
```

```
│   ├── globals.css
│   └── Home.module.css
├── lib/                    # Utility functions
├── types/                  # TypeScript type definitions
├── next.config.js          # Next.js configuration
├── tsconfig.json           # TypeScript configuration
├── package.json            # Project dependencies
└── README.md               # Project documentation
```

Pages and Routing

Pages Router (Legacy but Still Supported)

```
// pages/index.tsx - Home page (/), index routes
export default function Home() {
  return <h1>Welcome to Next.js!</h1>;
}

// pages/about.tsx - About page (/about)
export default function About() {
  return <h1>About Us</h1>;
}

// pages/blog/index.tsx - Blog index page (/blog)
export default function Blog() {
  return <h1>Blog Posts</h1>;
}

// pages/blog/[slug].tsx - Dynamic route (/blog/post-1, /blog/post-2, etc.)
import { useRouter } from 'next/router';

export default function BlogPost() {
  const router = useRouter();
  const { slug } = router.query;

  return <h1>Blog Post: {slug}</h1>;
}

// pages/posts/[year]/[month]/[day]/[slug].tsx - Nested dynamic routes
import { useRouter } from 'next/router';

export default function Post() {
  const router = useRouter();
  const { year, month, day, slug } = router.query;

  return (
    <h1>
      Post from {year}/{month}/{day}: {slug}
    </h1>
  );
}
```

```
// pages/dashboard/[[...params]].tsx - Catch-all routes
import { useRouter } from 'next/router';

export default function Dashboard() {
  const router = useRouter();
  const { params } = router.query;

  return (
    <div>
      <h1>Dashboard</h1>
      <p>Path segments: {params ? params.join('/') : 'None'}</p>
    </div>
  );
}
```

App Router (Next.js 13+)

```
// app/page.tsx - Home page (/)
export default function Home() {
  return <h1>Welcome to Next.js!</h1>;
}

// app/about/page.tsx - About page (/about)
export default function About() {
  return <h1>About Us</h1>;
}

// app/blog/page.tsx - Blog index page (/blog)
export default function Blog() {
  return <h1>Blog Posts</h1>;
}

// app/blog/[slug]/page.tsx - Dynamic route (/blog/post-1, /blog/post-2, etc.)
export default function BlogPost({ params }: { params: { slug: string } }) {
  return <h1>Blog Post: {params.slug}</h1>;
}

// app/posts/[year]/[month]/[day]/[slug]/page.tsx - Nested dynamic routes
export default function Post({
  params,
}: {
  params: {
    year: string;
    month: string;
    day: string;
    slug: string;
  };
}) {
  return (
    <h1>
```

```
      Post from {params.year}/{params.month}/{params.day}: {params.slug}
    </h1>
  );
}

// app/dashboard/[...params]/page.tsx - Catch-all routes
export default function Dashboard({
  params,
}: {
  params: { params: string[] };
}) {
  return (
    <div>
      <h1>Dashboard</h1>
      <p>Path segments: {params.params.join('/')}</p>
    </div>
  );
}
```

Navigation

```
// Import the Link component
import Link from 'next/link';
import { useRouter } from 'next/router'; // Pages Router only

export default function Navigation() {
  const router = useRouter(); // Pages Router only

  return (
    <nav>
      <ul>
        {/* Static routes */}
        <li>
          <Link href="/">Home</Link>
        </li>
        <li>
          <Link href="/about">About</Link>
        </li>

        {/* Dynamic routes */}
        <li>
          <Link href="/blog/hello-world">Hello World Blog Post</Link>
        </li>

        {/* With route object */}
        <li>
          <Link
            href={{
              pathname: '/blog/[slug]',
              query: { slug: 'hello-world' },
            }}
          >
            Hello World (Object Syntax)
          </Link>
        </li>

        {/* With hash */}
        <li>
          <Link href="/about#team">About Our Team</Link>
        </li>

        {/* Imperative navigation (Pages Router) */}
        <li>
          <button onClick={() => router.push('/contact')}>
            Contact Us (Imperative)
          </button>
        </li>
      </ul>
    </nav>
  );
}
```

Data Fetching Methods

Pages Router Data Fetching

```
// pages/posts/index.tsx
import { GetStaticProps } from 'next';

interface Post {
  id: number;
  title: string;
}

interface PostsProps {
  posts: Post[];
}

// Static Site Generation (SSG)
export const getStaticProps: GetStaticProps<PostsProps> = async () => {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts');
  const posts = await res.json();

  return {
    props: {
      posts: posts.slice(0, 10),
    },
    // Revalidate every 60 seconds (ISR - Incremental Static Regeneration)
    revalidate: 60,
  };
};
```

```
export default function Posts({ posts }: PostsProps) {
  return (
    <div>
      <h1>Posts</h1>
      <ul>
        {posts.map((post) => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}
```

```
// pages/users/[id].tsx
import { GetServerSideProps } from 'next';

interface User {
  id: number;
  name: string;
  email: string;
}

interface UserProps {
  user: User;
}

// Server-Side Rendering (SSR)
export const getServerSideProps: GetServerSideProps<UserProps> = async (
  context
) => {
  const { id } = context.params as { id: string };
  const res = await fetch(`https://jsonplaceholder.typicode.com/users/${id}`);
  const user = await res.json();

  if (!user.id) {
    return {
      notFound: true, // Returns 404 page
    };
  }

  return {
    props: {
      user,
    },
  };
};

export default function User({ user }: UserProps) {
  return (
    <div>
      <h1>{user.name}</h1>
      <p>Email: {user.email}</p>
    </div>
  );
}
```

```
// pages/posts/[id].tsx - With getStaticPaths
import { GetStaticProps, GetStaticPaths } from 'next';

interface Post {
  id: number;
  title: string;
  body: string;
}

interface PostProps {
  post: Post;
}

// Define which posts to pre-render at build time
export const getStaticPaths: GetStaticPaths = async () => {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts');
  const posts = await res.json();

  // Get the paths we want to pre-render
  const paths = posts.slice(0, 10).map((post: Post) => ({
    params: { id: post.id.toString() },
  }));

  return {
    paths,
    fallback: 'blocking', // 'blocking', true, or false
    // false: 404 for any paths not returned by getStaticPaths
    // true: generates page on request, shows fallback UI during generation
    // 'blocking': generates page on request, shows nothing until complete
  };
};

// Get the data for each post
export const getStaticProps: GetStaticProps<PostProps> = async (context) => {
  const { id } = context.params as { id: string };
  const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`);
  const post = await res.json();

  if (!post.id) {
    return {
      notFound: true,
    };
  }

  return {
    props: {
      post,
    },
    revalidate: 60, // ISR: regenerate after 60 seconds
  };
};

export default function Post({ post }: PostProps) {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.body}</p>
    </div>
  );
}
```

App Router Data Fetching

```
// app/posts/page.tsx - Server Component with fetch
async function getPosts() {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts', {
    next: {
      revalidate: 60, // Revalidate every 60 seconds (ISR)
      // Or: cache: 'no-store' // Always fetch fresh data (SSR)
    },
  });

  if (!res.ok) {
    throw new Error('Failed to fetch posts');
  }

  return res.json();
}

export default async function Posts() {
  const posts = await getPosts();

  return (
    <div>
      <h1>Posts</h1>
      <ul>
        {posts.slice(0, 10).map((post: any) => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}
```

```
// app/users/[id]/page.tsx - Dynamic route with fetch
async function getUser(id: string) {
  const res = await fetch(`https://jsonplaceholder.typicode.com/users/${id}`, {
    next: {
      revalidate: 3600, // Revalidate every hour
    },
  });

  if (!res.ok) {
    throw new Error('Failed to fetch user');
  }

  return res.json();
}

export default async function User({ params }: { params: { id: string } }) {
  const user = await getUser(params.id);

  return (
    <div>
      <h1>{user.name}</h1>
      <p>Email: {user.email}</p>
    </div>
  );
}
```

```
// app/posts/[id]/page.tsx - With generateStaticParams (similar to getStaticPaths)
async function getPosts() {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts');
  return res.json();
}

async function getPost(id: string) {
  const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`, {
    next: {
      revalidate: 60,
    },
  });

  if (!res.ok) {
    throw new Error('Failed to fetch post');
  }

  return res.json();
}

// Generate static params at build time
export async function generateStaticParams() {
  const posts = await getPosts();

  return posts.slice(0, 10).map((post: any) => ({
    id: post.id.toString(),
  }));
}

export default async function Post({ params }: { params: { id: string } }) {
  const post = await getPost(params.id);

  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.body}</p>
    </div>
  );
}
```

API Routes

Pages Router API Routes

```
// pages/api/hello.ts
import type { NextApiRequest, NextApiResponse } from 'next';

type Data = {
  name: string;
};

export default function handler(
  req: NextApiRequest,
  res: NextApiResponse<Data>
) {
  res.status(200).json({ name: 'John Doe' });
}
```

```
// pages/api/users/[id].ts - Dynamic API route
import type { NextApiRequest, NextApiResponse } from 'next';

type User = {
  id: string;
  name: string;
  email: string;
};

type ErrorResponse = {
  message: string;
};

export default function handler(
  req: NextApiRequest,
  res: NextApiResponse<User | ErrorResponse>
) {
  const { id } = req.query;

  // Handle different HTTP methods
  switch (req.method) {
    case 'GET':
      // Get user by ID
      return res.status(200).json({
        id: id as string,
        name: 'John Doe',
        email: 'john@example.com',
      });

    case 'PUT':
      // Update user
      return res.status(200).json({
        id: id as string,
        name: req.body.name || 'John Doe',
        email: req.body.email || 'john@example.com',
      });

    case 'DELETE':
      // Delete user
      return res.status(200).json({
        id: id as string,
        name: 'John Doe',
        email: 'john@example.com',
      });

    default:
      return res.status(405).json({ message: 'Method not allowed' });
  }
}
```

App Router Route Handlers

```
// app/api/hello/route.ts
import { NextResponse } from 'next/server';

export async function GET() {
  return NextResponse.json({ name: 'John Doe' });
}
```

```
// app/api/users/[id]/route.ts - Dynamic API route
import { NextRequest, NextResponse } from 'next/server';

export async function GET(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  const id = params.id;

  // Fetch user from database or API
  const user = {
    id,
    name: 'John Doe',
    email: 'john@example.com',
  };

  return NextResponse.json(user);
}

export async function PUT(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  const id = params.id;
  const data = await request.json();

  // Update user in database
  const updatedUser = {
    id,
    name: data.name || 'John Doe',
    email: data.email || 'john@example.com',
  };

  return NextResponse.json(updatedUser);
}

export async function DELETE(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  const id = params.id;

  // Delete user from database

  return NextResponse.json({ message: `User ${id} deleted` }, { status: 200 });
}
```

CSS and Styling

CSS Modules

```
// styles/Button.module.css
.button {
  padding: 8px 16px;
  background-color: #0070f3;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.button:hover {
  background-color: #0051a2;
}

.large {
  font-size: 18px;
  padding: 12px 24px;
}
```

```
// components/Button.tsx
import styles from '../styles/Button.module.css';

interface ButtonProps {
  children: React.ReactNode;
  size?: 'default' | 'large';
  onClick?: () => void;
}

export default function Button({
  children,
  size = 'default',
  onClick,
}: ButtonProps) {
  return (
    <button
      className={`${styles.button} ${size === 'large' ? styles.large : ''}`}
      onClick={onClick}
    >
      {children}
    </button>
  );
}
```

Global Styles

```
/* styles/globals.css */
html,
body {
  padding: 0;
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen,
    Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
}

a {
  color: #0070f3;
  text-decoration: none;
}

* {
  box-sizing: border-box;
}
```

```
// pages/_app.tsx (Pages Router)
import '../styles/globals.css';
import type { AppProps } from 'next/app';

export default function MyApp({ Component, pageProps }: AppProps) {
  return <Component {...pageProps} />;
}
```

```
// app/layout.tsx (App Router)
import '../styles/globals.css';

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}
```

CSS-in-JS (Styled Components)

```
// First, install styled-components:
// npm install styled-components
// npm install --save-dev @types/styled-components

// components/StyledButton.tsx
import styled from 'styled-components';

interface ButtonProps {
  primary?: boolean;
  size?: 'small' | 'medium' | 'large';
}

const StyledButton = styled.button<ButtonProps>`
```

```
  padding: ${(props) =>
    props.size === 'small'
      ? '8px 16px'
      : props.size === 'large'
      ? '16px 32px'
      : '12px 24px'};
  background-color: ${(props) => (props.primary ? '#0070f3' : 'white')};
  color: ${(props) => (props.primary ? 'white' : '#0070f3')};
  border: 2px solid #0070f3;
  border-radius: 4px;
  cursor: pointer;
  font-size: ${(props) =>
    props.size === 'small' ? '14px' : props.size === 'large' ? '18px' : '16px'};
  transition: all 0.3s ease;

  &:hover {
    background-color: ${(props) => (props.primary ? '#0051a2' : '#f8f9fa')};
  }
`;

export default StyledButton;
```

```
// For Pages Router, add the following to _document.tsx:
import Document, {
  Html,
  Head,
  Main,
  NextScript,
  DocumentContext,
} from 'next/document';
import { ServerStyleSheet } from 'styled-components';

export default class MyDocument extends Document {
  static async getInitialProps(ctx: DocumentContext) {
    const sheet = new ServerStyleSheet();
    const originalRenderPage = ctx.renderPage;

    try {
      ctx.renderPage = () =>
        originalRenderPage({
          enhanceApp: (App) => (props) =>
            sheet.collectStyles(<App {...props} />),
        });

      const initialProps = await Document.getInitialProps(ctx);
      return {
        ...initialProps,
        styles: (
          <>
            {initialProps.styles}
            {sheet.getStyleElement()}
          </>
        ),
      };
    } finally {
      sheet.seal();
    }
  }

  render() {
    return (
      <Html lang="en">
        <Head />
        <body>
          <Main />
          <NextScript />
        </body>
      </Html>
    );
  }
}
```

Tailwind CSS

```
// Install Tailwind CSS:
// npm install tailwindcss postcss autoprefixer
// npx tailwindcss init -p

// tailwind.config.js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    './app/**/*.{js,ts,jsx,tsx,mdx}',
    './pages/**/*.{js,ts,jsx,tsx,mdx}',
    './components/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

```css
/* styles/globals.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
// components/TailwindButton.tsx
interface ButtonProps {
  children: React.ReactNode;
  variant?: 'primary' | 'secondary';
  size?: 'sm' | 'md' | 'lg';
  onClick?: () => void;
}

export default function TailwindButton({
  children,
  variant = 'primary',
  size = 'md',
  onClick,
}: ButtonProps) {
  const baseClasses =
    'rounded-md font-medium transition-colors focus:outline-none focus:ring-2
focus:ring-blue-500 focus:ring-offset-2';

  const variantClasses = {
    primary: 'bg-blue-600 text-white hover:bg-blue-700',
    secondary: 'bg-white text-gray-700 border border-gray-300 hover:bg-gray-50',
  };

  const sizeClasses = {
    sm: 'py-1 px-3 text-sm',
    md: 'py-2 px-4 text-base',
    lg: 'py-3 px-6 text-lg',
  };

  const classes = `${baseClasses} ${variantClasses[variant]}
${sizeClasses[size]}`;

  return (
    <button className={classes} onClick={onClick}>
      {children}
    </button>
  );
}
```

Advanced Next.js

Rendering Strategies

Server-Side Rendering (SSR)

SSR generates the HTML for each request on the server.

Benefits:

- SEO-friendly

- Fast initial page load
- Always fresh data

Drawbacks:

- Slower Time to First Byte (TTFB)
- Higher server load
- Full page reload on navigation

When to use:

- Pages that need SEO
- Pages with frequently changing data
- Pages with user-specific content

Pages Router Implementation:

```
// pages/products.tsx
import { GetServerSideProps } from 'next';

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

  return {
    props: {
      products,
      timestamp: new Date().toISOString(),
    },
  };
};

export default function Products({ products, timestamp }) {
  return (
    <div>
      <h1>Products</h1>
      <p>Data fetched at: {timestamp}</p>
      <ul>
        {products.map((product) => (
          <li key={product.id}>{product.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

App Router Implementation:

```
// app/products/page.tsx
async function getProducts() {
  const res = await fetch('https://api.example.com/products', {
    cache: 'no-store', // SSR - fetch fresh data every request
  });
  return res.json();
}

export default async function Products() {
  const products = await getProducts();
  const timestamp = new Date().toISOString();

  return (
    <div>
      <h1>Products</h1>
      <p>Data fetched at: {timestamp}</p>
      <ul>
        {products.map((product) => (
          <li key={product.id}>{product.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

Static Site Generation (SSG)

SSG generates HTML at build time and reuses it for each request.

Benefits:

- Very fast performance
- Reduced server load
- Can be deployed to CDNs

Drawbacks:

- Data may become stale
- Build time increases with more pages
- Not suitable for user-specific content

When to use:

- Marketing pages
- Blog posts
- Documentation
- Any page with infrequently changing data

Pages Router Implementation:

```
// pages/posts/[slug].tsx
import { GetStaticProps, GetStaticPaths } from 'next';
```

```
export const getStaticPaths: GetStaticPaths = async () => {
  const res = await fetch('https://api.example.com/posts');
  const posts = await res.json();

  const paths = posts.map((post) => ({
    params: { slug: post.slug },
  }));

  return {
    paths,
    fallback: false, // Show 404 for paths not returned by getStaticPaths
  };
};

export const getStaticProps: GetStaticProps = async ({ params }) => {
  const res = await fetch(`https://api.example.com/posts/${params.slug}`);
  const post = await res.json();

  return {
    props: {
      post,
      generatedAt: new Date().toISOString(),
    },
  };
};

export default function Post({ post, generatedAt }) {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>Generated at: {generatedAt}</p>
      <div dangerouslySetInnerHTML={{ __html: post.content }} />
    </div>
  );
}
```

App Router Implementation:

```
// app/posts/[slug]/page.tsx
async function getPosts() {
  const res = await fetch('https://api.example.com/posts');
  return res.json();
}

async function getPost(slug: string) {
  const res = await fetch(`https://api.example.com/posts/${slug}`);
  return res.json();
}

export async function generateStaticParams() {
  const posts = await getPosts();
  return posts.map((post) => ({
    slug: post.slug,
  }));
}

export default async function Post({ params }: { params: { slug: string } }) {
  const post = await getPost(params.slug);
  const generatedAt = new Date().toISOString();

  return (
    <div>
      <h1>{post.title}</h1>
      <p>Generated at: {generatedAt}</p>
      <div dangerouslySetInnerHTML={{ __html: post.content }} />
    </div>
  );
}
```

Incremental Static Regeneration (ISR)

ISR allows you to update static pages after they've been built without rebuilding the entire site.

Benefits:

- Combines benefits of SSG and SSR
- Fast initial load like SSG
- Data stays relatively fresh
- Reduced server load compared to SSR

Drawbacks:

- More complex to understand
- First user after revalidation gets stale data
- Not suitable for real-time data

When to use:

- E-commerce product pages
- Content that changes periodically
- Pages that need good performance but also fresh data

Pages Router Implementation:

```
// pages/products/[id].tsx
import { GetStaticProps, GetStaticPaths } from 'next';

export const getStaticPaths: GetStaticPaths = async () => {
  const res = await fetch('https://api.example.com/popular-products');
  const popularProducts = await res.json();

  const paths = popularProducts.map((product) => ({
    params: { id: product.id.toString() },
  }));

  return {
    paths,
    fallback: 'blocking', // Generate missing pages on demand
  };
};

export const getStaticProps: GetStaticProps = async ({ params }) => {
  const res = await fetch(`https://api.example.com/products/${params.id}`);
  const product = await res.json();

  return {
    props: {
      product,
      generatedAt: new Date().toISOString(),
    },
    revalidate: 60, // Regenerate page after 60 seconds
  };
};

export default function Product({ product, generatedAt }) {
  return (
    <div>
      <h1>{product.name}</h1>
      <p>${product.price}</p>
      <p>Last updated: {generatedAt}</p>
    </div>
  );
}
```

App Router Implementation:

```
// app/products/[id]/page.tsx
async function getProduct(id: string) {
  const res = await fetch(`https://api.example.com/products/${id}`, {
    next: { revalidate: 60 }, // ISR - revalidate every 60 seconds
  });
  return res.json();
}

export default async function Product({ params }: { params: { id: string } }) {
  const product = await getProduct(params.id);
  const generatedAt = new Date().toISOString();

  return (
    <div>
      <h1>{product.name}</h1>
      <p>${product.price}</p>
      <p>Last updated: {generatedAt}</p>
    </div>
  );
}
```

Client-Side Rendering (CSR)

With CSR, the initial HTML is minimal and JavaScript runs in the browser to populate the content.

Benefits:

- Rich user interactions
- Reduces server load
- Good for dashboards and private pages

Drawbacks:

- Slower initial load
- Poor SEO for public content
- May need extra loading states

When to use:

- Dashboards
- User accounts
- Highly interactive applications
- Private pages that don't need SEO

Implementation with SWR:

```
// Install SWR: npm install swr

// components/Dashboard.tsx
import { useState } from 'react';
import useSWR from 'swr';

const fetcher = (url: string) => fetch(url).then((res) => res.json());

export default function Dashboard() {
  const { data, error, isLoading } = useSWR('/api/dashboard-data', fetcher);

  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error loading data</div>;

  return (
    <div>
      <h1>Dashboard</h1>
      <p>Welcome back, {data.user.name}</p>
      <div>
        {data.items.map((item) => (
          <div key={item.id}>{item.name}</div>
        ))}
      </div>
    </div>
  );
}
```

Image Optimization

Next.js provides the `Image` component for automatic image optimization:

```
// components/OptimizedImage.tsx
import Image from 'next/image';

export default function OptimizedImage() {
  return (
    <div>
      {/* Basic usage */}
      <Image
        src="/images/profile.jpg"
        alt="Profile Picture"
        width={300}
        height={200}
        priority // Load this image immediately (LCP)
      />

      {/* Remote image */}
      <Image
        src="https://example.com/photo.jpg"
        alt="Remote Photo"
        width={400}
        height={300}
        // Remote images need to be configured in next.config.js
      />

      {/* Responsive image */}
      <div style={{ position: 'relative', width: '100%', height: '40vh' }}>
        <Image
          src="/images/banner.jpg"
          alt="Banner"
          fill
          style={{ objectFit: 'cover' }}
          sizes="(max-width: 768px) 100vw, (max-width: 1200px) 50vw, 33vw"
        />
      </div>

      {/* Blur-up placeholder */}
      <Image
        src="/images/large-photo.jpg"
        alt="Large Photo"
        width={800}
        height={600}
        placeholder="blur"
blurDataURL="data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABsSFBcUERsXFh
ceHBsgKEIrKCUlKFE6PTBCYFVlZF9VXVtqeJmBanGQc1tdhbWGkJ6jq62rZ4C8ybqmx5moq6T/2wBDARwe
HigjKE4rK06kbl1upKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpKSkpK
T/wAARCAAGAAoDASIAAhEBAxEB/8QAFQABAQAAAAAAAAAAAAAAAAAAAAf/xAAbEAADAAMBAQAAAAAAAAAA
AAAAAQIDEQQFIf/EABQBAQAAAAAAAAAAAAAAAAAAAAAD/xAAUEQEAAAAAAAAAAAAAAAAAAAAAAA/9oADAMBAA
IRAxEAPwDTQAf/2Q=="
      />
    </div>
  );
}
```

Configuration in next.config.js

```
// next.config.js
module.exports = {
  images: {
    domains: ['example.com', 'images.unsplash.com'],
    formats: ['image/avif', 'image/webp'],
    deviceSizes: [640, 750, 828, 1080, 1200, 1920, 2048, 3840],
    imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],
    minimumCacheTTL: 60,
  },
};
```

Authentication Strategies

Using NextAuth.js

```
// Install NextAuth.js: npm install next-auth

// pages/api/auth/[...nextauth].ts
import NextAuth from 'next-auth';
import GoogleProvider from 'next-auth/providers/google';
import CredentialsProvider from 'next-auth/providers/credentials';

export default NextAuth({
  providers: [
    // OAuth authentication
    GoogleProvider({
      clientId: process.env.GOOGLE_ID,
      clientSecret: process.env.GOOGLE_SECRET,
    }),

    // Email/Password authentication
    CredentialsProvider({
      name: 'Credentials',
      credentials: {
        email: { label: 'Email', type: 'email' },
        password: { label: 'Password', type: 'password' },
      },
      async authorize(credentials) {
        // Add your own authentication logic here
        const user = await loginUser(credentials.email, credentials.password);

        if (user) {
          return user;
        } else {
          return null;
        }
      },
    }),
  ],
  session: {
    strategy: 'jwt',
    maxAge: 30 * 24 * 60 * 60, // 30 days
  },
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id;
        token.role = user.role;
      }
      return token;
    },
    async session({ session, token }) {
      session.user.id = token.id;
      session.user.role = token.role;
      return session;
    },
  },
  pages: {
    signIn: '/auth/signin',
    signOut: '/auth/signout',
    error: '/auth/error',
    verifyRequest: '/auth/verify-request',
  },
});

// pages/_app.tsx
import { SessionProvider } from 'next-auth/react';
import type { AppProps } from 'next/app';

export default function MyApp({ Component, pageProps }: AppProps) {
  return (
    <SessionProvider session={pageProps.session}>
      <Component {...pageProps} />
    </SessionProvider>
  );
}

// components/LoginButton.tsx
import { signIn, signOut, useSession } from 'next-auth/react';
```

```
export default function LoginButton() {
  const { data: session, status } = useSession();
  const loading = status === 'loading';

  if (loading) {
    return <div>Loading...</div>;
  }

  if (session) {
    return (
      <div>
        <p>Signed in as {session.user.email}</p>
        <button onClick={() => signOut()}>Sign out</button>
      </div>
    );
  }

  return <button onClick={() => signIn()}>Sign in</button>;
}

// middleware.ts (Route protection)
import { getToken } from 'next-auth/jwt';
import { NextRequest, NextResponse } from 'next/server';

export async function middleware(req: NextRequest) {
  const path = req.nextUrl.pathname;

  // Paths that are always accessible
  const publicPaths = ['/login', '/register', '/api/auth'];
  if (publicPaths.some((publicPath) => path.startsWith(publicPath))) {
    return NextResponse.next();
  }

  // Check if the user is authenticated
  const token = await getToken({ req, secret: process.env.NEXTAUTH_SECRET });

  // Redirect to login if not authenticated
  if (!token) {
    return NextResponse.redirect(new URL('/login', req.url));
  }

  // Admin route protection
  if (path.startsWith('/admin') && token.role !== 'admin') {
    return NextResponse.redirect(new URL('/unauthorized', req.url));
  }

  return NextResponse.next();
}

export const config = {
  matcher: ['/((?!_next/static|_next/image|favicon.ico).*)'],
};
```

Deployment Options

Vercel (Optimal for Next.js)

```
# Install Vercel CLI
npm install -g vercel

# Login to Vercel
vercel login

# Deploy to production
vercel --prod
```

Custom Server Setup

```
// server.js
const { createServer } = require('http');
const { parse } = require('url');
const next = require('next');

const dev = process.env.NODE_ENV !== 'production';
const app = next({ dev });
const handle = app.getRequestHandler();

app.prepare().then(() => {
  createServer((req, res) => {
    const parsedUrl = parse(req.url, true);
    handle(req, res, parsedUrl);
  }).listen(3000, (err) => {
    if (err) throw err;
    console.log('> Ready on http://localhost:3000');
  });
});
```

```
// package.json
{
  "scripts": {
    "dev": "node server.js",
    "build": "next build",
    "start": "NODE_ENV=production node server.js"
  }
}
```

Docker Deployment

```
# Dockerfile
FROM node:18-alpine AS base

# Install dependencies only when needed
FROM base AS deps
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN npm run build

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV production

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs

EXPOSE 3000

ENV PORT 3000

CMD ["node", "server.js"]
```

```
# docker-compose.yml
version: '3'

services:
  nextjs:
    build: .
    ports:
      - '3000:3000'
    environment:
      - DATABASE_URL=postgresql://postgres:password@postgres:5432/myapp
      - NEXTAUTH_SECRET=your-secret-here
      - NEXTAUTH_URL=http://localhost:3000
    depends_on:
```

```yaml
      - postgres

  postgres:
    image: postgres:14
    ports:
      - '5432:5432'
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=myapp
    volumes:
      - postgres-data:/var/lib/postgresql/data

volumes:
  postgres-data:
```

Performance Optimization

Core Web Vitals Optimization

```tsx
// pages/_document.tsx or app/layout.tsx
import { Html, Head, Main, NextScript } from 'next/document';

export default function Document() {
  return (
    <Html lang="en">
      <Head>
        {/* Preload critical fonts */}
        <link
          rel="preload"
          href="/fonts/inter-var.woff2"
          as="font"
          type="font/woff2"
          crossOrigin="anonymous"
        />
        {/* Preconnect to external domains */}
        <link rel="preconnect" href="https://fonts.googleapis.com" />
        <link
          rel="preconnect"
          href="https://fonts.gstatic.com"
          crossOrigin="anonymous"
        />
      </Head>
      <body>
        <Main />
        <NextScript />
      </body>
    </Html>
  );
}
```

Dynamic Imports

```
// components/DynamicComponent.tsx
import dynamic from 'next/dynamic';
import { Suspense } from 'react';

// Dynamic import with loading component
const HeavyComponent = dynamic(() => import('./HeavyComponent'), {
  loading: () => <div>Loading...</div>,
  ssr: false, // Disable Server-Side Rendering for this component
});

// Dynamic import based on condition
const AdminPanel = dynamic(() =>
  user.isAdmin ? import('./AdminPanel') : import('./UnauthorizedPanel')
);

export default function DynamicComponentDemo() {
  return (
    <div>
      <h1>Dynamic Components</h1>

      {/* Using Suspense for code-split components */}
      <Suspense fallback={<div>Loading...</div>}>
        <HeavyComponent />
      </Suspense>

      {/* Conditionally loaded component */}
      <AdminPanel />
    </div>
  );
}
```

Script Optimization

```
// components/OptimizedScripts.tsx
import Script from 'next/script';

export default function OptimizedScripts() {
  return (
    <>
      {/* Load script after page load (low priority) */}
      <Script
        src="https://example.com/analytics.js"
        strategy="afterInteractive"
        onLoad={() => console.log('Analytics loaded')}
      />

      {/* Load script during idle time */}
      <Script src="https://example.com/chat-widget.js" strategy="lazyOnload" />
      {/* High priority script (blocks rendering) */}
      <Script
        src="https://example.com/critical.js"
        strategy="beforeInteractive"
      />

      {/* Inline script */}
      <Script id="show-banner">
        {`document.addEventListener('DOMContentLoaded', function() {
          document.getElementById('banner').classList.remove('hidden');
        });`}
      </Script>
    </>
  );
}
```

App Router vs. Pages Router

Pages Router (Original)

Structure:

```
pages/
├── index.tsx           # Route: /
├── about.tsx           # Route: /about
├── blog/
│   ├── index.tsx       # Route: /blog
│   └── [slug].tsx      # Route: /blog/:slug
└── api/
    └── hello.ts        # API Route: /api/hello
```

Data Fetching:

- `getStaticProps` and `getStaticPaths` for Static Generation
- `getServerSideProps` for Server-Side Rendering
- Client-side with hooks like `useEffect` or libraries like SWR/React Query

Examples:

```
// pages/posts/[id].tsx
import { GetStaticPaths, GetStaticProps } from 'next';

export const getStaticPaths: GetStaticPaths = async () => {
  // ...fetch paths
  return { paths, fallback: false };
};

export const getStaticProps: GetStaticProps = async ({ params }) => {
  // ...fetch data
  return { props: { post } };
};

export default function Post({ post }) {
  return <div>...</div>;
}
```

App Router (Next.js 13+)

Structure:

```
app/
├── page.tsx            # Route: /
├── layout.tsx          # Root layout
├── about/
│   └── page.tsx        # Route: /about
├── blog/
│   ├── page.tsx        # Route: /blog
│   └── [slug]/
│       └── page.tsx    # Route: /blog/:slug
└── api/
    └── hello/
        └── route.ts    # API Route: /api/hello
```

Data Fetching:

- Server Components fetch data directly with `async/await`
- `fetch` with built-in caching/revalidation options
- Route Handlers for API routes

Examples:

```
// app/posts/[id]/page.tsx
async function getPost(id: string) {
  const res = await fetch(`https://api.example.com/posts/${id}`, {
    next: { revalidate: 60 },
  });
  if (!res.ok) throw new Error('Failed to fetch post');
  return res.json();
}

export async function generateStaticParams() {
  // ...fetch paths
  return paths;
}

export default async function Post({ params }: { params: { id: string } }) {
  const post = await getPost(params.id);
  return <div>...</div>;
}
```

Key Differences

Feature	Pages Router	App Router
Default Component Type	Client Components	Server Components
Data Fetching	Special functions (getStaticProps, etc.)	Direct async/await in components
Layouts	Custom _app.tsx	Nested layouts with layout.tsx
API Routes	pages/api/*.ts files	app/api/*/route.ts files
Loading States	Manual implementation	Built-in loading.tsx
Error Handling	Custom error pages	Nested error.tsx
Metadata	Head component	Metadata API
Routing Concepts	File-based routing	Nested folder-based routing

Server Components vs. Client Components

Server Components

Server Components render on the server and send HTML to the client.

Benefits:

- Reduced JavaScript bundle size
- Direct access to backend resources
- Secure data fetching
- Improved performance for data-heavy pages

Limitations:

- Cannot use hooks (`useState`, `useEffect`, etc.)
- Cannot attach event listeners (`onClick`, etc.)
- Cannot use browser-only APIs

Example:

```
// app/users/page.tsx (Server Component by default)
async function getUsers() {
  // Direct database query or API call
  const res = await fetch('https://api.example.com/users');
  return res.json();
}

export default async function UsersPage() {
  const users = await getUsers();

  return (
    <div>
      <h1>Users</h1>
      <ul>
        {users.map((user) => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

Client Components

Client Components render on the client and enable interactivity.

Benefits:

- Can use React hooks
- Can attach event handlers
- Can access browser APIs
- Enable interactive UI elements

How to Use: Add the `"use client"` directive at the top of your file.

Example:

```
'use client';

import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>Counter: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

Combining Server and Client Components

```
// app/products/page.tsx (Server Component)
import ProductSearch from './ProductSearch';
```

```
async function getProducts() {
  const res = await fetch('https://api.example.com/products');
  return res.json();
}

export default async function ProductsPage() {
  const products = await getProducts();

  return (
    <div>
      <h1>Products</h1>
      {/* Client Component passed server data */}
      <ProductSearch initialProducts={products} />
    </div>
  );
}
```

```
// app/products/ProductSearch.tsx (Client Component)
'use client';

import { useState, useEffect } from 'react';

export default function ProductSearch({ initialProducts }) {
  const [products, setProducts] = useState(initialProducts);
  const [searchTerm, setSearchTerm] = useState('');

  useEffect(() => {
    if (searchTerm) {
      const filtered = initialProducts.filter((product) =>
        product.name.toLowerCase().includes(searchTerm.toLowerCase())
      );
      setProducts(filtered);
    } else {
      setProducts(initialProducts);
    }
  }, [searchTerm, initialProducts]);

  return (
    <div>
      <input
        type="text"
        placeholder="Search products..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
      />

      <ul>
        {products.map((product) => (
          <li key={product.id}>
            {product.name} - ${product.price}
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Data Fetching Patterns

SWR (Stale-While-Revalidate)

```
// Install SWR: npm install swr

// hooks/usePosts.ts
import useSWR from 'swr';

const fetcher = (url: string) => fetch(url).then((res) => res.json());

export function usePosts() {
  const { data, error, isLoading, mutate } = useSWR('/api/posts', fetcher, {
    revalidateOnFocus: true,
    revalidateOnReconnect: true,
    dedupingInterval: 5000,
  });

  return {
    posts: data,
    isLoading,
    isError: error,
    mutate, // Function to revalidate data
  };
}

// components/Posts.tsx
('use client');

import { usePosts } from '../hooks/usePosts';

export default function Posts() {
  const { posts, isLoading, isError, mutate } = usePosts();

  if (isLoading) return <div>Loading...</div>;
  if (isError) return <div>Error loading posts</div>;

  return (
    <div>
      <button onClick={() => mutate()}>Refresh Posts</button>
      <ul>
        {posts.map((post) => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}
```

React Query

```
// Install React Query: npm install @tanstack/react-query

// lib/queryClient.ts
import { QueryClient } from '@tanstack/react-query';

export const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 60 * 1000, // 1 minute
      cacheTime: 5 * 60 * 1000, // 5 minutes
      retry: 1,
    },
  },
});

// app/providers.tsx
('use client');

import { QueryClientProvider } from '@tanstack/react-query';
import { ReactQueryDevtools } from '@tanstack/react-query-devtools';
import { queryClient } from '../lib/queryClient';

export function Providers({ children }: { children: React.ReactNode }) {
  return (
    <QueryClientProvider client={queryClient}>
      {children}
      <ReactQueryDevtools initialIsOpen={false} />
    </QueryClientProvider>
  );
}

// app/layout.tsx
import { Providers } from './providers';

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>
        <Providers>{children}</Providers>
      </body>
    </html>
  );
}

// hooks/useProducts.ts
('use client');

import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';

const API_URL = '/api/products';

export function useProducts() {
  const queryClient = useQueryClient();

  // Get products
  const {
    data: products,
    isLoading,
    error,
  } = useQuery({
    queryKey: ['products'],
    queryFn: () => fetch(API_URL).then((res) => res.json()),
  });

  // Add product
  const addProductMutation = useMutation({
    mutationFn: (newProduct) => {
      return fetch(API_URL, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(newProduct),
      }).then((res) => res.json());
    },
    onSuccess: () => {
      // Invalidate and refetch
      queryClient.invalidateQueries({ queryKey: ['products'] });
    },
  });

  return {
    products,
    isLoading,
    error,
    addProduct: addProductMutation.mutate,
  };
}

// components/ProductsList.tsx
('use client');

import { useState } from 'react';
import { useProducts } from '../hooks/useProducts';

export default function ProductsList() {
  const { products, isLoading, error, addProduct } = useProducts();
  const [newProductName, setNewProductName] = useState('');

  const handleAddProduct = () => {
    if (newProductName) {
      addProduct({ name: newProductName, price: 9.99 });
      setNewProductName('');
    }
  };

  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error loading products</div>;

  return (
    <div>
      <h1>Products</h1>

      <div>
        <input
          type="text"
          value={newProductName}
          onChange={(e) => setNewProductName(e.target.value)}
          placeholder="New product name"
        />
        <button onClick={handleAddProduct}>Add Product</button>
      </div>

      <ul>
        {products.map((product) => (
          <li key={product.id}>
            {product.name} - ${product.price}
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Server Actions (App Router)

```
// app/actions.ts (Server Action)
'use server';

import { revalidatePath } from 'next/cache';

export async function addTodo(formData: FormData) {
  const title = formData.get('title') as string;

  // Add to database
  await db.todo.create({
    data: {
      title,
      completed: false,
    },
  });

  // Revalidate the todos page
  revalidatePath('/todos');
}

// app/todos/page.tsx
import { addTodo } from '../actions';

async function getTodos() {
  const todos = await db.todo.findMany({
    orderBy: { createdAt: 'desc' },
  });
  return todos;
}

export default async function TodosPage() {
  const todos = await getTodos();

  return (
    <div>
      <h1>Todos</h1>

      <form action={addTodo}>
        <input type="text" name="title" placeholder="New todo..." required />
        <button type="submit">Add Todo</button>
      </form>

      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>{todo.title}</li>
        ))}
      </ul>
    </div>
  );
}
```

```
// app/todos/[id]/actions.ts (Server Action with parameters)
'use server';

import { revalidatePath } from 'next/cache';

export async function toggleTodoComplete(id: string, completed: boolean) {
  // Update in database
  await db.todo.update({
    where: { id },
    data: { completed },
  });
  // Revalidate the todos paths
  revalidatePath('/todos');
  revalidatePath(`/todos/${id}`);
}

// app/todos/[id]/page.tsx
import { toggleTodoComplete } from './actions';

async function getTodo(id: string) {
  const todo = await db.todo.findUnique({
    where: { id },
  });
  return todo;
}

export default async function TodoPage({ params }: { params: { id: string } }) {
  const todo = await getTodo(params.id);

  if (!todo) {
    return <div>Todo not found</div>;
  }

  return (
    <div>
      <h1>{todo.title}</h1>
      <form
        action={async () => {
          'use server';
          await toggleTodoComplete(todo.id, !todo.completed);
        }}
      >
        <button type="submit">
          Mark as {todo.completed ? 'incomplete' : 'complete'}
        </button>
      </form>
    </div>
  );
}
```

State Management Approaches

React Context with TypeScript

```
// contexts/ThemeContext.tsx
'use client';

import { createContext, useContext, useState, ReactNode } from 'react';

type Theme = 'light' | 'dark';
```

```
interface ThemeContextType {
  theme: Theme;
  toggleTheme: () => void;
}

const ThemeContext = createContext<ThemeContextType | undefined>(undefined);

export function ThemeProvider({ children }: { children: ReactNode }) {
  const [theme, setTheme] = useState<Theme>('light');

  const toggleTheme = () => {
    setTheme((prevTheme) => (prevTheme === 'light' ? 'dark' : 'light'));
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

export function useTheme() {
  const context = useContext(ThemeContext);
  if (context === undefined) {
    throw new Error('useTheme must be used within a ThemeProvider');
  }
  return context;
}

// app/layout.tsx or pages/_app.tsx
import { ThemeProvider } from '../contexts/ThemeContext';

export default function RootLayout({ children }) {
  return <ThemeProvider>{children}</ThemeProvider>;
}

// components/ThemeToggle.tsx
('use client');

import { useTheme } from '../contexts/ThemeContext';

export default function ThemeToggle() {
  const { theme, toggleTheme } = useTheme();

  return <button onClick={toggleTheme}>Current theme: {theme}</button>;
}
```

Zustand State Management

```
// Install Zustand: npm install zustand
// stores/useCounterStore.ts
import { create } from 'zustand';
import { persist } from 'zustand/middleware';

interface CounterState {
  count: number;
  increment: () => void;
  decrement: () => void;
  reset: () => void;
}

export const useCounterStore = create<CounterState>()(
  persist(
    (set) => ({
      count: 0,
      increment: () => set((state) => ({ count: state.count + 1 })),
      decrement: () => set((state) => ({ count: state.count - 1 })),
      reset: () => set({ count: 0 }),
    }),
    {
      name: 'counter-storage', // unique name for localStorage
    }
  )
);

// components/Counter.tsx
('use client');

import { useCounterStore } from '../stores/useCounterStore';

export default function Counter() {
  const { count, increment, decrement, reset } = useCounterStore();

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}
```

Redux Toolkit

```
// Install Redux: npm install @reduxjs/toolkit react-redux

// features/counter/counterSlice.ts
import { createSlice, PayloadAction } from '@reduxjs/toolkit';

interface CounterState {
  value: number;
}

const initialState: CounterState = {
  value: 0,
};

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
    incrementByAmount: (state, action: PayloadAction<number>) => {
      state.value += action.payload;
    },
  },
});

export const { increment, decrement, incrementByAmount } = counterSlice.actions;
export default counterSlice.reducer;

// store.ts
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './features/counter/counterSlice';

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

// hooks.ts
import { TypedUseSelectorHook, useDispatch, useSelector } from 'react-redux';
import type { RootState, AppDispatch } from './store';

export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;

// app/providers.tsx
('use client');

import { Provider } from 'react-redux';
import { store } from '../store';

export function Providers({ children }: { children: React.ReactNode }) {
  return <Provider store={store}>{children}</Provider>;
```

```
}

// components/ReduxCounter.tsx
('use client');

import { useAppSelector, useAppDispatch } from '../hooks';
import {
  increment,
  decrement,
  incrementByAmount,
} from '../features/counter/counterSlice';

export default function ReduxCounter() {
  const count = useAppSelector((state) => state.counter.value);
  const dispatch = useAppDispatch();

  return (
    <div>
      <h2>Redux Counter: {count}</h2>
      <button onClick={() => dispatch(increment())}>Increment</button>
      <button onClick={() => dispatch(decrement())}>Decrement</button>
      <button onClick={() => dispatch(incrementByAmount(5))}>Add 5</button>
    </div>
  );
}
```

Middleware and Edge Functions

Custom Middleware

```
// middleware.ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

export function middleware(request: NextRequest) {
  // Get the pathname
  const pathname = request.nextUrl.pathname;

  // Clone the request URL
  const url = request.nextUrl.clone();

  // Redirect /home to /
  if (pathname === '/home') {
    url.pathname = '/';
    return NextResponse.redirect(url);
  }

  // Rewrite /blog to /posts
  if (pathname.startsWith('/blog')) {
    url.pathname = pathname.replace(/^\/blog/, '/posts');
    return NextResponse.rewrite(url);
  }

  // Add custom headers to all responses
  const response = NextResponse.next();
  response.headers.set('x-custom-header', 'my-custom-value');

  return response;
}

// Only run middleware on specific paths
export const config = {
  matcher: [
    '/home',
    '/blog/:path*',
    '/((?!api|_next/static|_next/image|favicon.ico).*)',
  ],
};
```

Authentication Middleware

```
// middleware.ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';
import { verifyAuth } from './lib/auth';

export async function middleware(request: NextRequest) {
  // Get the pathname
  const pathname = request.nextUrl.pathname;

  // Public paths that don't require authentication
  const publicPaths = ['/', '/login', '/register', '/api/auth'];

  if (
    publicPaths.some((path) => pathname === path || pathname.startsWith(path))
  ) {
    return NextResponse.next();
  }

  // Verify authentication
  const token = request.cookies.get('token')?.value;
  const verifiedToken = token && (await verifyAuth(token));

  if (!verifiedToken) {
    // Create a login URL with a redirect back to the current page
    const loginUrl = new URL('/login', request.url);
    loginUrl.searchParams.set('callbackUrl', pathname);
    return NextResponse.redirect(loginUrl);
  }

  // Check for role-based permissions
  if (pathname.startsWith('/admin') && verifiedToken.role !== 'admin') {
    return NextResponse.redirect(new URL('/unauthorized', request.url));
  }

  return NextResponse.next();
}

export const config = {
  matcher: ['/((?!_next/static|_next/image|favicon.ico).*)'],
};
```

Edge API Route

```
// app/api/edge/hello/route.ts
import { NextResponse } from 'next/server';

export const runtime = 'edge'; // Specify Edge runtime

export async function GET(request: Request) {
  const { searchParams } = new URL(request.url);
  const name = searchParams.get('name') || 'World';

  return NextResponse.json({ message: `Hello, ${name}!` });
}
```

Geolocation with Edge Functions

```
// app/api/geolocation/route.ts
import { NextRequest, NextResponse } from 'next/server';

export const runtime = 'edge';

export async function GET(request: NextRequest) {
  // Get geolocation data from request
  const country = request.geo?.country || 'Unknown';
  const city = request.geo?.city || 'Unknown';
  const region = request.geo?.region || 'Unknown';

  return NextResponse.json({
    country,
    city,
    region,
    ip: request.ip || 'Unknown',
    timestamp: new Date().toISOString(),
  });
}
```

Internationalization (i18n)

Basic Next.js Internationalization

```
// Install next-i18next: npm install next-i18next

// next-i18next.config.js
module.exports = {
  i18n: {
    defaultLocale: 'en',
    locales: ['en', 'fr', 'de', 'es'],
  },
};

// next.config.js
const { i18n } = require('./next-i18next.config');

module.exports = {
  i18n,
};

// public/locales/en/common.json
{
  "greeting": "Hello",
  "welcome": "Welcome to our website",
  "description": "This is an internationalized website"
}

// public/locales/fr/common.json
{
  "greeting": "Bonjour",
  "welcome": "Bienvenue sur notre site",
  "description": "Ceci est un site internationalisé"
}

// pages/_app.tsx
import { appWithTranslation } from 'next-i18next';
import type { AppProps } from 'next/app';

function MyApp({ Component, pageProps }: AppProps) {
  return <Component {...pageProps} />;
}

export default appWithTranslation(MyApp);

// pages/index.tsx
import { GetStaticProps } from 'next';
import { useTranslation } from 'next-i18next';
import { serverSideTranslations } from 'next-i18next/serverSideTranslations';
import Link from 'next/link';
import { useRouter } from 'next/router';

export default function Home() {
  const { t } = useTranslation('common');
  const router = useRouter();

  const changeLanguage = (locale: string) => {
    router.push(router.pathname, router.asPath, { locale });
  };

  return (
    <div>
      <h1>{t('greeting')}</h1>
      <p>{t('welcome')}</p>
      <p>{t('description')}</p>

      <div>
        <button onClick={() => changeLanguage('en')}>English</button>
        <button onClick={() => changeLanguage('fr')}>Français</button>
        <button onClick={() => changeLanguage('de')}>Deutsch</button>
        <button onClick={() => changeLanguage('es')}>Español</button>
      </div>
    </div>
  );
}

export const getStaticProps: GetStaticProps = async ({ locale }) => {
  return {
    props: {
      ...(await serverSideTranslations(locale || 'en', ['common'])),
    },
  };
};
```

App Router Internationalization

```
// Install next-intl: npm install next-intl

// middleware.ts
import createMiddleware from 'next-intl/middleware';

export default createMiddleware({
  // A list of all locales that are supported
  locales: ['en', 'fr', 'de', 'es'],

  // The default locale to use when a non-locale prefixed
  // path is visited
  defaultLocale: 'en',

  // Optional: Specify a path for the language switcher to be excluded
  // pathnames: {
  //   '/api': false,
  // },
});
```

```
export const config = {
  // Match all paths except for
  // - API routes
  // - Static files
  // - Images (in the `/` namespace)
  matcher: ['/((?!api|_next|.*\\..*).*)']
};

// messages/en.json
{
  "Index": {
    "title": "Hello world!",
    "description": "This is an internationalized app."
  }
}

// messages/fr.json
{
  "Index": {
    "title": "Bonjour le monde !",
    "description": "Ceci est une application internationalisée."
  }
}

// app/[locale]/layout.tsx
import { NextIntlClientProvider } from 'next-intl';
import { notFound } from 'next/navigation';

export function generateStaticParams() {
  return [{ locale: 'en' }, { locale: 'fr' }];
}

export default async function LocaleLayout({
  children,
  params: { locale }
}: {
  children: React.ReactNode;
  params: { locale: string };
}) {
  let messages;
  try {
    messages = (await import(`../../messages/${locale}.json`)).default;
  } catch (error) {
    notFound();
  }

  return (
    <html lang={locale}>
      <body>
        <NextIntlClientProvider locale={locale} messages={messages}>
          {children}
        </NextIntlClientProvider>
      </body>
    </html>
  );
}

// app/[locale]/page.tsx
import { useTranslations } from 'next-intl';
import LocaleSwitcher from './LocaleSwitcher';

export default function Index() {
  const t = useTranslations('Index');

  return (
    <div>
      <h1>{t('title')}</h1>
      <p>{t('description')}</p>
      <LocaleSwitcher />
    </div>
  );
}

// app/[locale]/LocaleSwitcher.tsx
'use client';

import { useLocale } from 'next-intl';
import { usePathname, useRouter } from 'next-intl/client';

export default function LocaleSwitcher() {
  const locale = useLocale();
  const router = useRouter();
  const pathname = usePathname();

  const switchLocale = (newLocale: string) => {
    router.replace(pathname, { locale: newLocale });
  };

  return (
    <div>
      <button
        onClick={() => switchLocale('en')}
        disabled={locale === 'en'}
      >
        English
      </button>
      <button
        onClick={() => switchLocale('fr')}
        disabled={locale === 'fr'}
      >
        Français
      </button>
    </div>
  );
}
```

Testing Strategies

Unit Testing with Jest and React Testing Library

```
// Install testing libraries:
// npm install --save-dev jest @testing-library/react @testing-library/jest-dom
jest-environment-jsdom @types/jest

// jest.config.js
const nextJest = require('next/jest');

const createJestConfig = nextJest({
  // Provide the path to your Next.js app to load next.config.js and .env files
  dir: './',
});

const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  testEnvironment: 'jest-environment-jsdom',
};

module.exports = createJestConfig(customJestConfig);

// jest.setup.js
import '@testing-library/jest-dom';

// components/Counter.tsx
import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h2 data-testid="count">Count: {count}</h2>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}

// __tests__/components/Counter.test.tsx
import { render, screen, fireEvent } from '@testing-library/react';
import Counter from '../../components/Counter';

describe('Counter component', () => {
  it('renders the initial count', () => {
    render(<Counter />);

    expect(screen.getByTestId('count')).toHaveTextContent('Count: 0');
  });
  it('increments the count when increment button is clicked', () => {
    render(<Counter />);

    fireEvent.click(screen.getByText('Increment'));

    expect(screen.getByTestId('count')).toHaveTextContent('Count: 1');
  });

  it('decrements the count when decrement button is clicked', () => {
    render(<Counter />);

    fireEvent.click(screen.getByText('Decrement'));

    expect(screen.getByTestId('count')).toHaveTextContent('Count: -1');
  });
});
```

Testing API Routes

```
// pages/api/users.ts
import { NextApiRequest, NextApiResponse } from 'next';

type User = {
  id: number;
  name: string;
};

export default function handler(
  req: NextApiRequest,
  res: NextApiResponse<User[] | { error: string }>
) {
  if (req.method === 'GET') {
    res.status(200).json([
      { id: 1, name: 'John Doe' },
      { id: 2, name: 'Jane Smith' },
    ]);
  } else {
    res.status(405).json({ error: 'Method not allowed' });
  }
}

// __tests__/api/users.test.ts
import { createMocks } from 'node-mocks-http';
import handler from '../../pages/api/users';

describe('/api/users', () => {
  it('returns a list of users for GET request', async () => {
    const { req, res } = createMocks({
      method: 'GET',
    });
    await handler(req, res);

    expect(res._getStatusCode()).toBe(200);

    const data = JSON.parse(res._getData());
    expect(data).toEqual([
      { id: 1, name: 'John Doe' },
      { id: 2, name: 'Jane Smith' },
    ]);
  });

  it('returns 405 for non-GET requests', async () => {
    const { req, res } = createMocks({
      method: 'POST',
    });

    await handler(req, res);

    expect(res._getStatusCode()).toBe(405);

    const data = JSON.parse(res._getData());
    expect(data).toEqual({ error: 'Method not allowed' });
  });
});
```

End-to-End Testing with Cypress

```
// Install Cypress:
// npm install --save-dev cypress

// cypress/integration/home.spec.ts
describe('Home Page', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('should display the home page', () => {
    cy.get('h1').contains('Welcome');
  });

  it('should navigate to about page when clicking the About link', () => {
    cy.get('a').contains('About').click();
    cy.url().should('include', '/about');
    cy.get('h1').contains('About');
  });

  it('should increment counter when clicking the button', () => {
    cy.get('[data-testid="count"]').contains('Count: 0');
    cy.get('button').contains('Increment').click();
    cy.get('[data-testid="count"]').contains('Count: 1');
  });
});
```

Migration Strategies Between Next.js Versions

From Pages Router to App Router

Step 1: Update Dependencies

```
npm install next@latest react@latest react-dom@latest
```

Step 2: Create Minimal App Router Files

```
// app/layout.tsx
export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}

// app/page.tsx
export default function Home() {
  return (
    <div>
      <h1>App Router Home Page</h1>
      <p>Gradually migrating from Pages Router</p>
    </div>
  );
}
```

Step 3: Configure next.config.js for Dual Router Setup

```
// next.config.js
module.exports = {
  // Allow both /app and /pages directories to work
  experimental: {
    appDir: true,
  },
};
```

Step 4: Migrate Pages One by One

From:

```
// pages/about.tsx
export default function About() {
  return <h1>About Page</h1>;
}
```

To:

```
// app/about/page.tsx
export default function About() {
  return <h1>About Page</h1>;
}
```

Step 5: Migrate API Routes

From:

```
// pages/api/hello.ts
import { NextApiRequest, NextApiResponse } from 'next';

export default function handler(req: NextApiRequest, res: NextApiResponse) {
  res.status(200).json({ message: 'Hello World' });
}
```

To:

```
// app/api/hello/route.ts
import { NextResponse } from 'next/server';

export async function GET() {
  return NextResponse.json({ message: 'Hello World' });
}
```

Step 6: Migrate Data Fetching

From:

```
// pages/posts/[id].tsx
export async function getServerSideProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.id}`);
  const post = await res.json();
  return {
    props: { post },
  };
}

export default function Post({ post }) {
  return <h1>{post.title}</h1>;
}
```

To:

```
// app/posts/[id]/page.tsx
async function getPost(id: string) {
  const res = await fetch(`https://api.example.com/posts/${id}`);
  return res.json();
}

export default async function Post({ params }: { params: { id: string } }) {
  const post = await getPost(params.id);

  return <h1>{post.title}</h1>;
}
```

Step 7: Migrate from `_app.tsx` to App Router's Context Providers

From:

```
// pages/_app.tsx
import type { AppProps } from 'next/app';
import { ThemeProvider } from '../components/ThemeProvider';
import '../styles/globals.css';

export default function MyApp({ Component, pageProps }: AppProps) {
  return (
    <ThemeProvider>
      <Component {...pageProps} />
    </ThemeProvider>
  );
}
```

To:

```
// app/providers.tsx
'use client';

import { ThemeProvider } from '../components/ThemeProvider';
```

```
export function Providers({ children }: { children: React.ReactNode }) {
  return <ThemeProvider>{children}</ThemeProvider>;
}

// app/layout.tsx
import { Providers } from './providers';
import '../styles/globals.css';

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>
        <Providers>{children}</Providers>
      </body>
    </html>
  );
}
```

Migrating from Next.js 12 to Next.js 13+

1. **Update dependencies:**

   ```
   npm install next@latest react@latest react-dom@latest
   ```

2. **Update layout components:**

   Before:

   ```
   // components/Layout.tsx
   import Head from 'next/head';
   import Header from './Header';
   import Footer from './Footer';

   export default function Layout({ children }) {
     return (
       <>
         <Head>
           <title>My Website</title>
         </Head>
         <Header />
         <main>{children}</main>
         <Footer />
       </>
     );
   ```

```
}

// pages/index.tsx
import Layout from '../components/Layout';

export default function Home() {
  return (
    <Layout>
      <h1>Home Page</h1>
    </Layout>
  );
}
```

After:

```
// app/layout.tsx
import Header from '../components/Header';
import Footer from '../components/Footer';

export const metadata = {
  title: 'My Website',
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>
        <Header />
        <main>{children}</main>
        <Footer />
      </body>
    </html>
  );
}

// app/page.tsx
export default function Home() {
  return <h1>Home Page</h1>;
}
```

3. **Migrate data fetching:**

Before:

```
// pages/products/[id].tsx
export async function getStaticPaths() {
  // ...
  return { paths, fallback: false };
}

export async function getStaticProps({ params }) {
  // ...
  return { props: { product } };
}

export default function Product({ product }) {
  // ...
}
```

After:

```
// app/products/[id]/page.tsx
export async function generateStaticParams() {
  // ...
  return paths;
}

async function getProduct(id) {
  // ...
  return product;
}

export default async function Product({ params }) {
  const product = await getProduct(params.id);
  // ...
}
```

4. **Migrate image component:**

Before:

```
import Image from 'next/image';

export default function Avatar() {
  return (
    <Image
      src="/avatar.png"
      alt="User avatar"
      width={64}
      height={64}
      layout="fixed"
    />
  );
}
```

After:

```
import Image from 'next/image';

export default function Avatar() {
  return (
    <Image
      src="/avatar.png"
      alt="User avatar"
      width={64}
      height={64}
      // layout="fixed" removed
    />
  );
}
```

5. **Migrate navigation:**

   Before:

```
import { useRouter } from 'next/router';

export default function Navigation() {
  const router = useRouter();
  return (
    <button onClick={() => router.push('/dashboard')}>
      Go to Dashboard
    </button>
  );
}
```

   After:

```
'use client';

import { useRouter } from 'next/navigation';

export default function Navigation() {
  const router = useRouter();
  return (
    <button onClick={() => router.push('/dashboard')}>
      Go to Dashboard
    </button>
  );
}
```

Integrating TypeScript with Next.js

Setting Up a TypeScript Next.js Project

Creating a New Project

```
# Create a Next.js project with TypeScript
npx create-next-app@latest my-app --typescript

# Or add TypeScript to an existing Next.js project
npm install --save-dev typescript @types/react @types/node
# Then create a tsconfig.json file
touch tsconfig.json
# Run next dev to auto-populate the tsconfig.json
```

Next.js-specific TypeScript Types

```
// Page component props types
import { NextPage } from 'next';
import { AppProps } from 'next/app';
import { GetStaticProps, GetStaticPaths, GetServerSideProps } from 'next';

// For Pages Router
type NextPageWithLayout = NextPage & {
  getLayout?: (page: React.ReactElement) => React.ReactNode;
};

type AppPropsWithLayout = AppProps & {
  Component: NextPageWithLayout;
};

// _app.tsx with layout support
function MyApp({ Component, pageProps }: AppPropsWithLayout) {
  // Use the layout defined at the page level, if available
  const getLayout = Component.getLayout ?? ((page) => page);

  return getLayout(<Component {...pageProps} />);
}

// Example page with custom layout
const Page: NextPageWithLayout = () => {
  return <div>Page content</div>;
};
```

```
Page.getLayout = function getLayout(page: React.ReactElement) {
  return (
    <Layout>
      <SideNav />
      {page}
    </Layout>
  );
};

// For data fetching
interface Post {
  id: number;
  title: string;
  content: string;
}

interface PostPageProps {
  post: Post;
}

export const getStaticProps: GetStaticProps<PostPageProps> = async (
  context
) => {
  // Fetch post data
  const post: Post = await fetchPost(context.params?.id as string);

  return {
    props: {
      post,
    },
  };
};

export const getStaticPaths: GetStaticPaths = async () => {
  // Get list of all post IDs
  const posts: Post[] = await fetchPosts();
  const paths = posts.map((post) => ({
    params: { id: post.id.toString() },
  }));

  return {
    paths,
    fallback: false,
  };
};

// For App Router
interface PageProps {
  params: {
    id: string;
  };
  searchParams: {
    [key: string]: string | string[] | undefined;
  };
}

// App Router page component
export default function Page({ params, searchParams }: PageProps) {
  return <div>Page {params.id}</div>;
}
```

Type-Safe API Routes

```
// pages/api/users/[id].ts
import { NextApiRequest, NextApiResponse } from 'next';

interface User {
  id: string;
  name: string;
  email: string;
}

interface Error {
  message: string;
}

export default function handler(
  req: NextApiRequest,
  res: NextApiResponse<User | Error>
) {
  const { id } = req.query;

  if (!id || Array.isArray(id)) {
    return res.status(400).json({ message: 'Invalid user ID' });
  }

  // Fetch user data
  const user: User = {
    id,
    name: 'John Doe',
    email: 'john@example.com',
  };

  res.status(200).json(user);
}
```

Type-Safe Route Handlers (App Router)

```
// app/api/users/[id]/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { z } from 'zod';

// Define the user schema
```

```
const userSchema = z.object({
  id: z.string(),
  name: z.string(),
  email: z.string().email(),
});

type User = z.infer<typeof userSchema>;

export async function GET(
  request: NextRequest,
  { params }: { params: { id: string } }
) {
  try {
    // Validate the id parameter
    const id = z.string().parse(params.id);

    // Fetch user data (example)
    const userData = {
      id,
      name: 'John Doe',
      email: 'john@example.com',
    };

    // Validate the user data
    const user = userSchema.parse(userData);

    return NextResponse.json(user);
  } catch (error) {
    if (error instanceof z.ZodError) {
      return NextResponse.json({ error: error.errors }, { status: 400 });
    }

    return NextResponse.json(
      { error: 'Internal Server Error' },
      { status: 500 }
    );
  }
}
```

Type-Safe Forms

```
// app/contact/ContactForm.tsx
'use client';

import { useState, FormEvent } from 'react';
import { z } from 'zod';

// Define the form schema
const contactFormSchema = z.object({
  name: z.string().min(2, 'Name must be at least 2 characters'),
  email: z.string().email('Invalid email address'),
  message: z.string().min(10, 'Message must be at least 10 characters'),
});

type ContactForm = z.infer<typeof contactFormSchema>;

type FieldErrors = {
  [K in keyof ContactForm]?: string;
};

export default function ContactForm() {
  const [form, setForm] = useState<ContactForm>({
    name: '',
    email: '',
    message: '',
  });

  const [errors, setErrors] = useState<FieldErrors>({});
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [isSuccess, setIsSuccess] = useState(false);

  const updateField = <K extends keyof ContactForm>(
    field: K,
    value: ContactForm[K]
  ) => {
    setForm((prev) => ({ ...prev, [field]: value }));

    // Clear the error for this field when user starts typing again
    if (errors[field]) {
      setErrors((prev) => {
        const newErrors = { ...prev };
        delete newErrors[field];
        return newErrors;
      });
    }
  };

  const handleSubmit = async (e: FormEvent) => {
    e.preventDefault();
    setIsSubmitting(true);
    setErrors({});

    // Validate the form
    const result = contactFormSchema.safeParse(form);

    if (!result.success) {
      // Convert Zod errors to our format
      const fieldErrors: FieldErrors = {};
      result.error.errors.forEach((error) => {
        const path = error.path[0] as keyof ContactForm;
        fieldErrors[path] = error.message;
      });

      setErrors(fieldErrors);
      setIsSubmitting(false);
```

```
        return;
      }

      try {
        // Submit the form data
        const response = await fetch('/api/contact', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(form),
        });

        if (!response.ok) {
          throw new Error('Failed to submit the form');
        }

        // Reset the form on success
        setForm({ name: '', email: '', message: '' });
        setIsSuccess(true);
      } catch (error) {
        setErrors({ message: 'Failed to submit the form. Please try again.' });
      } finally {
        setIsSubmitting(false);
      }
    };

    return (
      <form onSubmit={handleSubmit} className="space-y-4">
        {isSuccess && (
          <div className="bg-green-100 p-4 rounded">
            Thank you for your message! We'll get back to you soon.
          </div>
        )}

        <div>
          <label htmlFor="name" className="block mb-1">
            Name
          </label>
          <input
            id="name"
            type="text"
            value={form.name}
            onChange={(e) => updateField('name', e.target.value)}
            className={`w-full p-2 border rounded ${
              errors.name ? 'border-red-500' : 'border-gray-300'
            }`}
          />
          {errors.name && (
            <p className="text-red-500 text-sm mt-1">{errors.name}</p>
          )}
        </div>

        <div>
```

```
          <label htmlFor="email" className="block mb-1">
            Email
          </label>
          <input
            id="email"
            type="email"
            value={form.email}
            onChange={(e) => updateField('email', e.target.value)}
            className={`w-full p-2 border rounded ${
              errors.email ? 'border-red-500' : 'border-gray-300'
            }`}
          />
          {errors.email && (
            <p className="text-red-500 text-sm mt-1">{errors.email}</p>
          )}
        </div>

        <div>
          <label htmlFor="message" className="block mb-1">
            Message
          </label>
          <textarea
            id="message"
            value={form.message}
            onChange={(e) => updateField('message', e.target.value)}
            rows={4}
            className={`w-full p-2 border rounded ${
              errors.message ? 'border-red-500' : 'border-gray-300'
            }`}
          />
          {errors.message && (
            <p className="text-red-500 text-sm mt-1">{errors.message}</p>
          )}
        </div>

        <button
          type="submit"
          disabled={isSubmitting}
          className="bg-blue-600 text-white py-2 px-4 rounded hover:bg-blue-700
disabled:opacity-50"
        >
          {isSubmitting ? 'Submitting...' : 'Send Message'}
        </button>
      </form>
    );
}
```

Type-Safe Database Access

```
// lib/db.ts
import { PrismaClient } from '@prisma/client';
```

```
// Prevent multiple instances of Prisma Client in development
declare global {
  var prisma: PrismaClient | undefined;
}

export const prisma = global.prisma || new PrismaClient();

if (process.env.NODE_ENV !== 'production') global.prisma = prisma;

// Using Prisma in an API route
// pages/api/posts/[id].ts
import { NextApiRequest, NextApiResponse } from 'next';
import { prisma } from '../../../lib/db';

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  const postId = req.query.id as string;

  if (req.method === 'GET') {
    try {
      const post = await prisma.post.findUnique({
        where: { id: postId },
        include: {
          author: {
            select: { name: true, email: true },
          },
          comments: true,
        },
      });

      if (!post) {
        return res.status(404).json({ message: 'Post not found' });
      }

      return res.status(200).json(post);
    } catch (error) {
      return res.status(500).json({ message: 'Error fetching post' });
    }
  }

  return res.status(405).json({ message: 'Method not allowed' });
}
```

Learning Path

Beginner Level: Getting Started

Week 1-2: TypeScript Foundations

- **Goals:**
  - Understand basic types and type annotations
  - Learn how to configure TypeScript
  - Create and run a simple TypeScript project
- **Learning Activities:**
  1. Set up TypeScript development environment
  2. Study primitive types, arrays, objects, and functions
  3. Understand interfaces vs. types
  4. Practice type checking and inference
- **Project:** Create a simple to-do list application with TypeScript

```typescript
interface Todo {
  id: number;
  title: string;
  completed: boolean;
}

class TodoList {
  private todos: Todo[] = [];

  addTodo(title: string): Todo {
    const todo: Todo = {
      id: Date.now(),
      title,
      completed: false,
    };
    this.todos.push(todo);
    return todo;
  }

  toggleTodo(id: number): Todo | undefined {
    const todo = this.todos.find((t) => t.id === id);
    if (todo) {
      todo.completed = !todo.completed;
    }
    return todo;
  }

  getTodos(): Todo[] {
    return this.todos;
  }
}
```

Week 3-4: Next.js Foundations

- **Goals:**

- Understand Next.js page-based routing
- Learn about server-side rendering basics
- Create a multi-page Next.js application

- **Learning Activities:**

  1. Set up a Next.js project
  2. Create basic pages with file-based routing
  3. Study how to use Link component for navigation
  4. Understand page layouts and styling

- **Project:** Build a personal blog with multiple pages

Intermediate Level: Strengthening Skills

Week 5-6: Advanced TypeScript

- **Goals:**

  - Master generics and utility types
  - Understand type narrowing and guards
  - Work with declaration files

- **Learning Activities:**

  1. Study advanced types (unions, intersections)
  2. Practice with generics in various contexts
  3. Create custom type guards
  4. Learn how to work with third-party libraries

- **Project:** Create a type-safe API client for a public API

Week 7-8: Next.js Data Fetching

- **Goals:**

  - Master different data fetching strategies
  - Understand the trade-offs between SSR, SSG, and ISR
  - Implement API routes

- **Learning Activities:**

  1. Implement getStaticProps, getStaticPaths, and getServerSideProps
  2. Create API routes with Next.js
  3. Study incremental static regeneration
  4. Implement client-side data fetching with SWR

- **Project:** Build a product catalog with data fetching from an API

Advanced Level: Deep Integration

Week 9-10: Integrating TypeScript with Next.js

- **Goals:**
  - Create fully type-safe Next.js applications
  - Understand Next.js-specific types
  - Implement type-safe API routes
- **Learning Activities:**
  1. Set up a TypeScript Next.js project
  2. Practice with Next.js-specific types
  3. Create type-safe data fetching functions
  4. Implement type-safe forms and API routes
- **Project:** Convert an existing JavaScript Next.js project to TypeScript

Week 11-12: Advanced Next.js Features

- **Goals:**
  - Master the App Router
  - Understand Server Components vs. Client Components
  - Implement authentication, middleware, and more
- **Learning Activities:**
  1. Migrate a project from Pages Router to App Router
  2. Study server components and when to use client components
  3. Implement authentication with NextAuth.js
  4. Create custom middleware
- **Project:** Build an e-commerce website with user authentication

Expert Level: Building Production Applications

Week 13-14: Performance Optimization and Testing

- **Goals:**
  - Optimize Next.js applications for performance
  - Implement comprehensive testing strategies
  - Understand image optimization and other performance features
- **Learning Activities:**
  1. Study Next.js image and font optimization
  2. Implement dynamic imports and code splitting
  3. Set up Jest, React Testing Library, and Cypress
  4. Optimize Core Web Vitals
- **Project:** Add comprehensive testing and performance optimization to previous projects

Week 15-16: Full Stack TypeScript with Next.js

- **Goals:**
  - Create a full-stack application with TypeScript
  - Implement type-safe database access
  - Deploy to production
- **Learning Activities:**
  1. Set up Prisma or other type-safe ORM
  2. Create a shared types package for frontend and backend
  3. Implement advanced state management
  4. Deploy to Vercel or other platforms
- **Project:** Build a full-stack SaaS application with TypeScript and Next.js

Recommended Project Progression

1. **Simple To-Do App**
   - TypeScript, basic state management
   - Learn core TypeScript concepts
2. **Personal Blog**
   - Next.js Pages Router, basic SSG
   - Learn file-based routing and basic data fetching
3. **Type-Safe API Client**
   - Advanced TypeScript, generics, utility types
   - Practice creating reusable, type-safe code
4. **Product Catalog**
   - Next.js data fetching strategies
   - Compare different rendering methods
5. **Convert JS Next.js to TypeScript**
   - Full TypeScript integration
   - Handle migration challenges
6. **E-commerce with Authentication**
   - App Router, authentication
   - Advanced routing and user management
7. **Optimized SaaS Application**
   - Full-stack TypeScript
   - Production-ready, performance-optimized

Combining TypeScript and Next.js Learning

1. **Start with TypeScript Basics**
    - Learn the type system first
    - Understand how TypeScript helps catch errors
2. **Add React with TypeScript**
    - Type your components and props
    - Use generics with hooks
3. **Introduce Next.js Fundamentals**
    - Start with the Pages Router for simplicity
    - Gradually adopt App Router
4. **Implement Full-Stack Features**
    - Create type-safe API routes
    - Add type-safe database access
5. **Adopt Advanced Patterns**
    - Server Components vs. Client Components
    - Type-level programming for complex scenarios

Additional Resources

Official Documentation

- TypeScript Handbook
- Next.js Documentation
- React TypeScript Cheatsheet

Books

- "Programming TypeScript" by Boris Cherny
- "Next.js in Action" by Adam Boduch
- "Fullstack React with TypeScript" by Khalil Stemmler

Courses and Tutorials

- TypeScript Deep Dive
- Next.js App Router Course
- Total TypeScript by Matt Pocock

GitHub Repositories

- TypeScript Starter
- Next.js GitHub Examples
- TypeScript-React-Starter

Community and Forums

- TypeScript Discord
- Next.js GitHub Discussions
- Stack Overflow - TypeScript
- Stack Overflow - Next.js

Tools and Extensions

- ESLint with TypeScript plugins
- VS Code with TypeScript and Next.js extensions
- TypeScript ESLint
- Prettier for code formatting

Type Definition Resources

- DefinitelyTyped
- TypeScript Types Search