Complete React + TypeScript Interview Guide

TypeScript + React Setup (CRA, Vite)

React with TypeScript can be set up using CRA or Vite. CRA provides configuration out of the box, while Vite is faster and lightweight.

Example: npm create vite@latest app -- --template react-ts

Project Structure & TypeScript Configuration

A proper structure separates components, hooks, services, and types. tsconfig.json controls strictness, paths, and compilation.

Example: src/components, src/hooks, src/api

JSX vs TSX

JSX is JavaScript syntax for UI. TSX is JSX with TypeScript support, allowing type safety.

Example: const App: React.FC = () => Hello

Typing Props

Props are typed using interfaces or type aliases to ensure correct data is passed.

Example: type Props = { name: string }

Types in Props

Props can include primitive types, objects, arrays, unions, and functions.

Example: onClick: () => void

defaultProps

defaultProps provide fallback values for props when none are passed.

Example: Component.defaultProps = { size: 'md' }

React.FC vs Direct Typing

React.FC provides implicit children typing but is less preferred now. Direct typing is explicit and clearer.

Example: function Comp({title}: Props)

Destructuring Props Safely

Props are destructured with default values to avoid undefined errors.

Example: function Button({label = 'OK'}: Props)

useState & useEffect (Typing)

useState manages local state, useEffect handles side effects. Types ensure correct state updates.

Example: useState(0)

useRef, useMemo, useCallback

useRef accesses DOM or mutable values. useMemo memoizes values. useCallback memoizes functions.

Example: const ref = useRef(null)

DOM Event Typing

Events like MouseEvent, ChangeEvent, and FormEvent are typed for safety.

Example: React.ChangeEvent

Controlled vs Uncontrolled Components

Controlled components use state, uncontrolled rely on refs.

Example: value vs ref

React Hook Form

React Hook Form handles forms efficiently with less re-rendering.

Example: useForm()

Schema Validation

Zod/Yup schemas centralize validation rules.

Example: z.string().email()

Custom Validation Messages

Custom messages improve UX and clarity.

Example: z.string().min(6, 'Too short')

Routing (Dynamic, Nested, Guards)

React Router enables dynamic params, nested layouts, and protected routes.

Example: /user/:id

Context API & useContext

Context avoids prop drilling and shares global state.

Example: AuthContext.Provider

Avoiding Prop Drilling

Context or state managers are used instead of passing props deeply.

Reusable Custom Hooks

Custom hooks extract reusable logic.

Example: useFetch, useCounter

Redux Toolkit Basics

Redux Toolkit simplifies Redux using store, slice, and reducers.

Typing Redux State & Actions

State and actions are typed using interfaces and PayloadAction.

Async Thunks

createAsyncThunk manages async logic.

Example: fetchUsers

React Query

useQuery fetches data, useMutation modifies data declaratively.

Query Keys & Caching

Query keys identify cached data and control refetching.

Optimistic UI (useOptimistic)

UI updates before server response for better UX.

Error Boundaries & Fallback UI

Error boundaries catch runtime errors and show fallback UI.

API Layer Abstraction

API logic is separated using service files and Axios instances.

Typing API Responses & Errors

API data and errors are typed using interfaces.

Payment Mock (Stripe/PayPal)

Mock payment simulates payment flow without real transactions.

Testing with Jest & RTL

Jest runs tests, RTL tests UI behavior.

Testing Hooks & Async Flows

renderHook and findBy are used for hooks and async testing.

Typing Mocks & Sample Data

Mocks and test data are typed for reliability.