Learning Path for React & Next - Part 2

As you already have a **pre-joining learning path** that covers fundamentals of React, Next.js, state management, and TypeScript. Now this **learning path**, the focus should shift from "getting started" to **real-world development practices & scalability.**

🔹 *Post-Joining Learning Path (REACT + NEXT JS)*

Estimated Duration: 10–15 days

1. Advanced React Patterns & Best Practices (2–3 days)

- Context API with custom hooks
- Compound components pattern
- Controlled vs uncontrolled components
- Render props & Higher Order Components (HOCs) (for legacy understanding)
- Performance optimization: `React. memo`, `useCallback`, `useMemo`
- Error boundaries & fallback UIs

📖 Resources:

- https://react.dev/learn/escape-hatches

- Kent C. Dodds Blog (React Patterns)

---

2. Advanced Next.js Features (2-3 days)

- Middleware & authentication (JWT / NextAuth.js basics)
- API route enhancements & middleware chaining
- Server Actions (new App Router feature)
- Streaming and Suspense with React Server Components
- Incremental Static Regeneration (ISR)
- Optimizing fonts & images
- Deployment strategies (Vercel, custom server, Dockerized Next.js)

📖 Resources:

- Next.js official advanced docs
- Vercel blog on App Router patterns

---

3. Scalable State Management (1.5 days)

- When to use Redux vs Zustand vs TanStack Query
- Query invalidation, optimistic updates with TanStack Query
- Zustand middleware (persist, devtools)
- Combining client state + server state effectively

📖 Resources:

- TanStack Query Docs (Mutations & Invalidation)
- Zustand Advanced Recipes

4. Performance & Optimization (2 days)

- Bundle analysis with `next-bundle-analyzer`
- Dynamic imports & code splitting
- Prefetching and caching strategies
- Lighthouse + Web Vitals optimization
- Using `React Profiler`

📖 Resources:

- Next.js Performance Docs
- Web Vitals by Google

5. Component libraries (2 days)

- MUI
- ShadCN
- Storybook

6. Form and Validation (1 days)

- React Hook Form
- Form Validation (ZOD Integration)
- Controlled and uncontrolled Form
- File upload
- Multi step forms

7. General Theme Structure understanding (1 days)

- Some of the admin panel we are directly using theme rather than creating UI from scratch like Vuexy, Sneat, Materio, Matronic etc…

8. Typescript (Depth dive) (1.5 days)

- https://www.typescriptlang.org/docs/handbook/intro.html
- https://basarat.gitbook.io/typescript/

9. Firebase

- https://www.youtube.com/watch?v=lQftwBTCejE

🔹 Assignments

Assignment 1 (React + NEXT JS + Firebase + Redux Toollik Advanced):

Build a full-stack Task Manager Application using Next.js (API + React frontend) with Firebase (Auth + Firestore), Redux Toolkit, React Hook Form, Tailwind CSS + MUI components, and role-based access (Admin & User).

This assignment should demonstrate frontend + NEXT JS (backend skills, database modeling, secure APIs), form validation, state management, and responsive UI design.

1. Authentication (Firebase Auth)

- User Registration & Login with email/password.
- Store user profile in Firestore with fields:
  - uid, email, role ("user" by default, "admin" manually assigned in Firestore).
- Implement protected routes (only logged-in users can access dashboard).

2. Role-Based Access

- User role:
  - Can create, view, update, delete only their own tasks.
  - Can assign tasks to other users.
  - Can see tasks assigned to them.

- Admin role:

  - Can view/edit/delete all users' tasks.
  - Can view list of all users.
  - Can see task analytics (counts, per user stats).
  - (Bonus) Can export tasks list to CSV/Excel.

3. Tasks (CRUD with Firestore)

- Task fields:

- title (string, min 3 chars, required)
- description (string, required)
- status (todo, in-progress, done)
- dueDate (must be a future date)
- ownerId (task creator)
- assignedTo (uid of assigned user, optional)
- createdAt timestamp

- APIs:

  - POST /api/tasks → create task
  - GET /api/tasks → fetch tasks (role-based)
  - PUT /api/tasks/:id → update task
  - DELETE /api/tasks/:id → delete task

4. Real-Time Updates

- Use Firestore onSnapshot listener.
- Tasks update live across dashboards (no refresh needed).

5. Forms (React Hook Form + Validation)

- Use React Hook Form with MUI inputs.
- Validations:
  - Title: required, min 3 chars.
  - Description: required.
  - Due date: required & must be in the future.

6. State Management (Redux Toolkit)

- authSlice → store uid, email, role.
- tasksSlice → store tasks (use createAsyncThunk for API calls).
- Fetch logic:
  - If User → get tasks where ownerId == uid OR assignedTo == uid.
  - If Admin → get all tasks.

7. UI & Styling

- Use Tailwind CSS for layout & responsiveness.
- Use MUI Components (Buttons, Dialogs, Tables, TextFields, Drawer).
- Mobile-first responsive design.

8. Search, Filter & Pagination

- Search tasks by title/description.
- Filter by status or due date.
- Paginate (10 tasks per page) OR implement infinite scroll.

Assignment 2 (Next.js Advanced):

Build a **Blog Application** with:

- App Router
- SSR + ISR pages
- API routes for CRUD posts
- Authentication with NextAuth.js
- Dark mode toggle with Zustand

👉 This way, **pre-joining** path ensures **fundamentals**, and **post-joining** ensures **real-world production readiness** with performance & scalability