React Stacks One-Pager

A concise, print-friendly guide to assembling React stacks with modern tooling. Copy into Notion as a table or keep as a reference sheet.

Overview

- **React + Hooks** for UI
- **Server state** → **TanStack Query**
- **Client/UI state** → **Zustand** (or **Redux Toolkit** at scale)
- **Forms** → **React Hook Form (RHF)** + **Zod** (TS-first); keep **Formik + Yup** if already adopted
- **Framework** → **Next.js** for SSR/SSG/SEO; **Vite** for SPA/dev speed
- **Tests** → **Vitest/Jest + React Testing Library (RTL)**; **Playwright** for critical E2E

Comparison Table

Tool	Role	Best when	Pairs well with	Notes
React (Hooks)	Component UI + local state	Always	TS, Query, Zustand/Redux	Extract custom hooks; keep pure
Next.js	Routing, SSR/SSG, server actions, APIs	SEO, content, hybrid apps	TanStack Query, Zod, RHF	Learn App Router, caching
Vite	Dev server & bundler	SPA, libraries	Vitest, RTL	Configure aliases/env early
TanStack Query	Fetch/caching of server data	Data-heavy UIs, pagination	Fetch/Axios, Zod	Don’t put server data in Redux/Zustand
Zustand	Lightweight client state	Modals, wizards, UI prefs	React, Query	Minimal boilerplate; simple selectors
Redux Toolkit	Structured global state mgmt	Large teams, cross-feature flows	RTK Query or TanStack Query	Prefer RTK over hand-rolled Redux
Zod	Runtime validation + TS inference	Full TS, shared schemas	Next APIs, RHF (zodResolver)	Generate types from schemas
Yup	Validation (no TS inference)	Legacy / Formik apps	Formik	Keep if already invested

Tool	Role	Best when	Pairs well with	Notes
RHF	Performant forms	Big/dynamic forms	Zod via zodResolver	Minimal rerenders; controlled inputs via Controller
Formik	Form helpers/ state	Simple forms or legacy	Yup	More rerenders vs RHF
Jest	Test runner	Node libs/legacy	RTL	Mature ecosystem; slower startup
Vitest	Vite-native test runner	Vite SPAs	RTL	Use jsdom for DOM tests
RTL	DOM testing utilities	Component/ integration tests	Jest/Vitest	Test behavior, not internals
Playwright	E2E browser automation	Full user flows	Next/Vite apps	Keep E2E thin and high-value

Stack Recipes

1) SPA Dashboard (no SSR)

- **Vite + React + TypeScript**
- **TanStack Query** for server state
- **Zustand** for client/UI state
- **RHF + Zod** for forms/validation
- **Vitest + RTL**; **Playwright** for a few flows **Why:** Fast DX; clear split between server/client state.

Quick start

```
npm create vite@latest my-spa -- --template react-ts
cd my-spa && npm i @tanstack/react-query zustand react-hook-form zod @hookform/
resolvers
npm i -D vitest @testing-library/react @testing-library/user-event jsdom
playwright
```

App wiring (snippet)

```
// main.tsx
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
const qc = new QueryClient();
root.render(
  <QueryClientProvider client={qc}>
```

```
    <App />
  </QueryClientProvider>
);
```

2) SEO/SaaS Marketing + App Shell

- **Next.js (App Router) + TypeScript**
- **TanStack Query** (client) + **Next fetch/cache** (server)
- **RHF + Zod**
- **RTL + Vitest/Jest**, **Playwright Why:** SSR/SSG for SEO; client interactivity with Query.

Quick start

```
npx create-next-app@latest my-app --ts
cd my-app && npm i @tanstack/react-query zod react-hook-form @hookform/resolvers
npm i -D @testing-library/react @testing-library/user-event vitest jsdom
playwright
```

Provider (snippet)

```
// app/providers.tsx
'use client';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
export default function Providers({ children }: { children: React.ReactNode }) {
  const qc = new QueryClient();
  return <QueryClientProvider client={qc}>{children}</QueryClientProvider>;
}
```

3) Forms-Heavy Back Office

- **Next.js or Vite + React + TS**
- **RHF + Zod** (conditional/dynamic forms)
- **TanStack Query** for CRUD & optimistic updates
- **Zustand** for wizard progress; **Redux Toolkit** if complexity grows
- **RTL + Vitest/Jest**; **Playwright** for critical paths

RHF + Zod (snippet)

```
import { z } from 'zod';
import { zodResolver } from '@hookform/resolvers/zod';
import { useForm } from 'react-hook-form';

const schema = z.object({ email: z.string().email(), age: z.number().min(18) });
export function MyForm() {
```

```
  const { register, handleSubmit, formState: { errors } } = useForm({ resolver:
zodResolver(schema) });
  return (
    <form onSubmit={handleSubmit(console.log)}>
      <input {...register('email')} /> {errors.email?.message}
      <input type="number" {...register('age', { valueAsNumber: true })} />
      <button>Submit</button>
    </form>
  );
}
```

4) Enterprise Web App (multi-team)

- **Next.js + TS**
- **Redux Toolkit** for cross-feature client workflows
- **RTK Query *or* TanStack Query** (choose one per feature area)
- **RHF + Zod** for forms; shared Zod schemas across client/server
- **Jest + RTL**, **Playwright** in CI

Redux Toolkit (snippet)

```
// store.ts
import { configureStore } from '@reduxjs/toolkit';
import userSlice from './userSlice';
export const store = configureStore({ reducer: { user: userSlice } });
export type RootState = ReturnType<typeof store.getState>;
```

5) MVP with Hosted Backend

- **Next.js or Vite + TS**
- **TanStack Query** + **Supabase/Firebase**
- **RHF + Zod**
- **Vitest + RTL**, **Playwright** optional

Testing Matrix

- **Units (logic/hooks/utils)** → Vitest *or* Jest
- **Components/Integration (DOM)** → RTL under Vitest/Jest (with **jsdom**)
- **E2E (real browser)** → Playwright (thin, high-value scenarios)

Vitest DOM setup

```
// vitest.config.ts
import { defineConfig } from 'vitest/config';
export default defineConfig({ test: { environment: 'jsdom', setupFiles: ['./
test/setup.ts'] } });
```

Validation & Forms

- **TS everywhere?** → **RHF + Zod** (zodResolver)
- **Existing Formik/Yup?** → Stick with it until refactor
- **Shared contracts?** → Centralize **Zod** schemas and `z.infer` types

---

State Strategy

- **Server state** (cached, refetchable) → **TanStack Query**
- **Client/UI state** (ephemeral/UI) → **Zustand** or Context
- **Complex cross-feature flows** → **Redux Toolkit**

Copy-Ready Checklists

New Vite + React TS SPA 1. Create app (Vite) 2. Install Query, Zustand, RHF, Zod 3. Add `QueryClientProvider` 4. Add app-level `useStore` (Zustand) 5. Add example form with Zod 6. Set up Vitest + RTL + jsdom

New Next.js App Router project 1. Create app (CNA) 2. Add `app/providers.tsx` with QueryClientProvider 3. RHF + Zod for forms 4. Co-locate Zod schemas with API routes 5. Configure Vitest/Jest + RTL; add Playwright in CI

Tip: Don't duplicate server data in client stores. Let Query own it; subscribe via selectors when you must derive UI state.