

===== INSTRUCTIONS =====

- 1) Prerequisites
 - Node.js 18+
 - npm
- 2) Run Backend

```
cd backend
npm install
npm run dev
Backend URL: http://localhost:4000
```
- 3) Run Frontend

```
cd frontend
npm install
npm run dev
Frontend URL: http://localhost:3000
```
- 4) Environment (frontend/.env.local)
NEXT_PUBLIC_API_BASE=http://localhost:4000
- 5) Quick Verification
 - Open http://localhost:4000/health
 - Open http://localhost:3000/posts
 - Create/Edit a post from the UI
- 6) API Endpoints (Backend)

```
GET    /api/posts
GET    /api/posts/:id
GET    /api/posts/:id/comments
POST   /api/posts
PATCH /api/posts/:id
GET    /api/posts-n-plus-1 (if implemented)
```
- 7) Notes
 - JSONPlaceholder is a mock API; POST/PATCH success is not persisted.
 - Frontend uses Next.js + React + TypeScript.
 - Backend uses Express + TypeScript + Zod.

===== CODE =====

```
=====
FILE: README.md
=====
# Qualitara Staff Fullstack Interview Project (Next.js + Node/Express)

This repo contains:
- `frontend/` - Next.js 14 (React + TypeScript + Tailwind)
- `backend/` - Node.js (Express + TypeScript) API that proxies JSONPlaceholder

## Prereqs
- Node 18+ recommended

## Run (2 terminals)

### 1) Backend
```bash
cd backend
npm install
```

```
npm run dev
```

Backend runs on: <http://localhost:4000>

### 2) Frontend

```
```bash
```

```
cd frontend
```

```
npm install
```

```
npm run dev
```

Frontend runs on: <http://localhost:3000>

Endpoints

- GET <http://localhost:4000/api/posts>
- GET <http://localhost:4000/api/posts/:id>
- GET <http://localhost:4000/api/posts/:id/comments>

Notes

qualitar

```
=====
FILE: backend/src/app.ts
=====
```

```
import express from "express";
import cors from "cors";
import { postsRouter } from "../routes/posts";

const app = express();

app.use(cors());
app.use(express.json());

app.get("/health", (_req, res) => res.status(200).json({ ok: true }));

app.use("/api", postsRouter);

const port = process.env.PORT ? Number(process.env.PORT) : 4000;

app.listen(port, () => {
  console.log(`Backend listening on http://localhost:${port}`);
});
```

```
=====
FILE: backend/src/routes/posts.ts
=====
```

```
import { Router } from "express";
import { z } from "zod";
import { fetchPost, fetchPostComments, fetchPosts, createPost, patchPost } from
"../services/jsonplaceholder";

const CreatePostSchema = z.object({
  title: z.string(),
  body: z.string(),
  userId: z.number(),
});

const PatchPostSchema = CreatePostSchema.partial();

export const postsRouter = Router();
```

```
// Get all posts
```

```
postsRouter.get("/posts", async (_req, res) => {  
  try {  
    const posts = await fetchPosts();  
    return res.status(200).json({ posts });  
  } catch {  
    return res.status(500).json({ error: "Failed to fetch posts" });  
  }  
});
```

```
//Get post by id
```

```
postsRouter.get("/posts/:id", async (req, res) => {  
  const id = Number(req.params.id);  
  if (!Number.isFinite(id) || id <= 0) return res.status(400).json({ error: "Invalid post id" });  
  
  try {  
    const post = await fetchPost(id);  
    return res.status(200).json({ post });  
  } catch {  
    return res.status(500).json({ error: "Failed to fetch post" });  
  }  
});
```

```
//get comments by post id
```

```
postsRouter.get("/posts/:id/comments", async (req, res) => {  
  const id = Number(req.params.id);  
  if (!Number.isFinite(id) || id <= 0) return res.status(400).json({ error: "Invalid post id" });  
  
  try {  
    const comments = await fetchPostComments(id);  
    return res.status(200).json({ comments });  
  } catch {  
    return res.status(500).json({ error: "Failed to fetch comments" });  
  }  
});
```

```
postsRouter.post("/posts", async (req, res) => {  
  const parsed = CreatePostSchema.safeParse(req.body);  
  if (!parsed.success) {  
    return res.status(400).json({ error: "Invalid body", details: parsed.error.flatten() });  
  }  
  
  try {  
    const post = await createPost(parsed.data);  
    return res.status(201).json({ post });  
  } catch {  
    return res.status(500).json({ error: "Failed to create post" });  
  }  
});
```

```
// Update post by id ( TODO: implementar connection con BD)
```

```
postsRouter.patch("/posts/:id", async (req, res) => {  
  const id = Number(req.params.id);  
  if (!Number.isFinite(id) || id <= 0) return res.status(400).json({ error: "Invalid post id" });  
});
```

```

    valid post id" });

    const parsed = PatchPostSchema.safeParse(req.body);
    if (!parsed.success) {
        return res.status(400).json({ error: "Invalid body", details: parsed.error.f
        latten() });
    }

    try {
        const post = await patchPost(id, parsed.data);
        return res.status(200).json({ post });
    } catch {
        return res.status(500).json({ error: "Failed to update post" });
    }
});

```

```

postsRouter.get("/posts-n1", async (req, res) => {
    try{
        const posts = await fetchPosts();
        const get10 = posts.slice(0,10).map(async (post) => {
            const comments = await fetchPostComments(post.id);
            return { ...post, comments };
        });
        const postsWithComments = await Promise.all(get10);
        return res.status(200).json({ posts: postsWithComments });

    } catch (error) {
        return res.status(500).json({ error: "Failed to fetch posts-n+1" });
    }
})

```

```

=====
FILE: backend/src/services/jsonplaceholder.ts
=====

```

```

import axios from "axios";
import { z } from "zod";

const BASE_URL = "https://jsonplaceholder.typicode.com";

const PostSchema = z.object({
    userId: z.number(),
    id: z.number(),
    title: z.string(),
    body: z.string()
});

const CommentSchema = z.object({
    postId: z.number(),
    id: z.number(),
    name: z.string(),
    email: z.string(),
    body: z.string()
});

const CreatePostSchema = z.object({
    title: z.string(),
    body: z.string(),
    userId: z.number(),
});

const PatchPostSchema = CreatePostSchema.partial();

```

```

export type Post = z.infer<typeof PostSchema>;
export type Comment = z.infer<typeof CommentSchema>;
export type CreatePostInput = z.infer<typeof CreatePostSchema>;
export type PatchPostInput = z.infer<typeof PatchPostSchema>;

export async function fetchPosts(): Promise<Post[]> {
  const { data } = await axios.get(`${BASE_URL}/posts`, { timeout: 10_000 });
  return z.array(PostSchema).parse(data);
}

export async function fetchPost(id: number): Promise<Post> {
  const { data } = await axios.get(`${BASE_URL}/posts/${id}`, { timeout: 10_000 });
  return PostSchema.parse(data);
}

export async function fetchPostComments(id: number): Promise<Comment[]> {
  const { data } = await axios.get(`${BASE_URL}/posts/${id}/comments`, { timeout: 10_000 });
  return z.array(CommentSchema).parse(data);
}

export async function createPost(input: CreatePostInput): Promise<Post> {
  const { data } = await axios.post(`${BASE_URL}/posts`, input, { timeout: 10_000 });
  return PostSchema.parse(data);
}

export async function patchPost(id: number, input: PatchPostInput): Promise<Post> {
  const { data } = await axios.patch(`${BASE_URL}/posts/${id}`, input, { timeout: 10_000 });
  return PostSchema.parse(data);
}

```

```

=====
FILE: frontend/src/lib/api.ts
=====

```

```

import type { Comment, Post } from "@types/jsonplaceholder";

// Server-side: call the upstream directly (backend or JSONPlaceholder).
// Client-side: call the Next.js API routes (relative URLs).
const UPSTREAM =
  process.env.BACKEND_API_BASE ||
  process.env.NEXT_PUBLIC_API_BASE ||
  "https://jsonplaceholder.typicode.com";

function serverUrl(path: string): string {
  // path is like "/posts" or "/posts/1/comments"
  if (UPSTREAM.includes("jsonplaceholder")) {
    return `${UPSTREAM}${path}`;
  }
  return `${UPSTREAM}/api${path}`;
}

async function parseServerPost(res: Response): Promise<Post> {
  const raw = await res.json();
  // Backend wraps in { post }, JSONPlaceholder returns the object directly
  return (raw as { post?: Post }).post ?? (raw as Post);
}

async function parseServerPosts(res: Response): Promise<Post[]> {

```

```

    const raw = await res.json();
    return (raw as { posts?: Post[] }).posts ?? (raw as Post[]);
}

async function parseServerComments(res: Response): Promise<Comment[]> {
    const raw = await res.json();
    return (raw as { comments?: Comment[] }).comments ?? (raw as Comment[]);
}

export async function getPosts(): Promise<Post[]> {
    if (typeof window !== "undefined") {
        const res = await fetch("/api/posts", { cache: "no-store" });
        if (!res.ok) throw new Error("Failed to load posts");
        return ((await res.json()) as { posts: Post[] }).posts;
    }
    const res = await fetch(serverUrl("/posts"), { cache: "no-store" });
    if (!res.ok) throw new Error("Failed to load posts");
    return parseServerPosts(res);
}

export async function getPosts_n1(): Promise<Post[]> {
    if (typeof window !== "undefined") {
        const res = await fetch("/api/posts", { cache: "no-store" });
        if (!res.ok) throw new Error("Failed to load posts");
        return ((await res.json()) as { posts: Post[] }).posts;
    }
    const res = await fetch(serverUrl("/posts-n1"), { cache: "no-store" });
    if (!res.ok) throw new Error("Failed to load posts");
    return parseServerPosts(res);
}

export async function getPost(id: number): Promise<Post> {
    if (typeof window !== "undefined") {
        const res = await fetch(`/api/posts/${id}`, { cache: "no-store" });
        if (!res.ok) throw new Error("Failed to load post");
        return ((await res.json()) as { post: Post }).post;
    }
    const res = await fetch(serverUrl(`/posts/${id}`), { cache: "no-store" });
    if (!res.ok) throw new Error("Failed to load post");
    return parseServerPost(res);
}

export async function getPostComments(id: number): Promise<Comment[]> {
    if (typeof window !== "undefined") {
        const res = await fetch(`/api/posts/${id}/comments`, { cache: "no-store" });
        if (!res.ok) throw new Error("Failed to load comments");
        return ((await res.json()) as { comments: Comment[] }).comments;
    }
    const res = await fetch(serverUrl(`/posts/${id}/comments`), { cache: "no-store" });
    if (!res.ok) throw new Error("Failed to load comments");
    return parseServerComments(res);
}

export async function createPost(input: {
    title: string;
    body: string;
    userId: number;
}): Promise<Post> {
    const res = await fetch("/api/posts", {
        method: "POST",
    });

```

```

    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(input),
  });
  if (!res.ok) throw new Error("Failed to create post");
  return ((await res.json()) as { post: Post }).post;
}

export async function patchPost(
  id: number,
  input: Partial<{ title: string; body: string; userId: number }>
): Promise<Post> {
  const res = await fetch(`/api/posts/${id}`, {
    method: "PATCH",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(input),
  });
  if (!res.ok) throw new Error("Failed to update post");
  return ((await res.json()) as { post: Post }).post;
}

```

```

=====
FILE: frontend/src/app/posts/page.tsx
=====

```

```

import { getPosts, getPosts_n1 } from "@lib/api";
import { PostsList } from "@components/PostsList";
import type { Post } from "@types/jsonplaceholder";

export const dynamic = "force-dynamic";

export default async function PostsPage() {
  let posts: Post[] = [];
  let loadError = false;

  try {
    posts = await getPosts_n1();
  } catch {
    loadError = true;
  }

  return (
    <main className="mx-auto max-w-5xl p-6">
      <h1 className="text-2xl font-bold">Posts</h1>
      <p className="mt-1 text-sm text-gray-600">
        Server-rendered list from the Node backend.
      </p>
      {loadError ? (
        <div className="mt-6 rounded-xl border border-red-200 bg-red-50 p-4 text-sm text-red-700">
          Failed to load posts. Verify backend is running on <code>http://localhost:4000</code> and refresh.
        </div>
      ) : (
        <PostsList initialPosts={posts} />
      )}
    </main>
  );
}

```

```

=====
FILE: frontend/src/app/posts/[id]/page.tsx
=====

```

```

import Link from "next/link";

```

```

import { getPost, getPostComments } from "@lib/api";
import { PostDetail } from "@components/PostDetail";
import type { Comment, Post } from "@types/jsonplaceholder";

export const dynamic = "force-dynamic";

export default async function PostDetailPage({ params }: { params: { id: string } }) {
  const postId = Number(params.id);
  let post: Post | null = null;
  let comments: Comment[] = [];
  let loadError = false;

  try {
    [post, comments] = await Promise.all([getPost(postId), getPostComments(postId)]);
  } catch {
    loadError = true;
  }

  return (
    <main className="mx-auto max-w-3xl p-6">
      <Link href="/posts" className="text-sm underline">
        ← Back
      </Link>
      {loadError || !post ? (
        <div className="mt-6 rounded-xl border border-red-200 bg-red-50 p-4 text-sm text-red-700">
          Failed to load post details. Verify backend is running on <code>http://localhost:4000</code> and refresh.
        </div>
      ) : (
        <PostDetail initialPost={post} comments={comments} />
      )}
    </main>
  );
}

```

```

=====
FILE: frontend/src/components/PostsList.tsx
=====
"use client";

```

```

import { useState } from "react";
import { PostCard } from "@components/PostCard";
import { NewPostForm } from "@components/NewPostForm";
import type { Post } from "@types/jsonplaceholder";
import styles from "@styles/components/PostsList.module.scss";

export function PostsList({ initialPosts }: { initialPosts: Post[] }) {
  const [posts, setPosts] = useState(initialPosts);

  return (
    <div className={styles["posts-list"]} >
      <NewPostForm onCreate={
        (post) => setPosts((prev) => [post, ...prev])
      } />
      <div className={styles["posts-list__grid"]} >
        {posts.slice(0, 30).map((p) => (
          <PostCard key={p.id} post={p} />
        ))}
      </div>
    </div>
  );
}

```



```
}
```

```
=====
FILE: frontend/src/components/PostCard.tsx
=====
```

```
import Link from "next/link";
import type { Post } from "@types/jsonplaceholder";
import styles from "@styles/components/PostCard.module.scss";

export function PostCard({ post }: { post: Post }) {
  return (
    <Link
      href={`\posts/${post.id}`}
      className={styles["post-card"]}
    >
      <h3 className={styles["post-card__title"]} >{post.title}</h3>
      <p className={styles["post-card__body"]} >{post.body}</p>
      <p className={styles["post-card__meta"]} >Post #{post.id}</p>
    </Link>
  );
}
```

```
=====
FILE: frontend/src/styles/components/PostsList.module.scss
=====
```

```
.posts-list {
  @apply mt-4;
}

.posts-list__grid {
  @apply mt-6 grid grid-cols-1 gap-4 sm:grid-cols-2 lg:grid-cols-3;
}
```

```
=====
FILE: frontend/src/styles/components/PostCard.module.scss
=====
```

```
.post-card {
  @apply block rounded-xl border border-brand-200 bg-surface-50 p-4 transition h
over:shadow-card;
}

.post-card__title {
  @apply line-clamp-1 text-lg font-semibold text-brand-800;
}

.post-card__body {
  @apply mt-2 line-clamp-2 text-sm text-gray-600;
}

.post-card__meta {
  @apply mt-3 text-xs text-gray-500;
}
```

```
=====
FILE: frontend/tailwind.config.js
=====
```

```
module.exports = {
  content: ["./src/**/*.{js,ts,jsx,tsx}"],
  theme: {
    extend: {
      colors: {
        brand: {
```

```

        50: "#ecfeff",
        200: "#052e0f",
        800: "#509872",
    },
    surface: {
        50: "#f8fafc",
        100: "#f1f5f9",
    },
},
boxShadow: {
    card: "0 8px 24px -12px rgba(8, 47, 73, 0.35)",
},
},
},
plugins: [],
};

```

```

=====
FILE: frontend/package.json
=====

```

```

{
  "name": "staff-frontend",
  "private": true,
  "version": "1.0.0",
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "next": "14.2.3",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "zod": "^3.22.4"
  },
  "devDependencies": {
    "@types/node": "^20.0.0",
    "@types/react": "^18.2.0",
    "@types/react-dom": "^18.2.0",
    "autoprefixer": "^10.4.0",
    "eslint": "^8.0.0",
    "eslint-config-next": "14.2.3",
    "postcss": "^8.4.0",
    "sass": "^1.97.3",
    "tailwindcss": "^3.4.0",
    "typescript": "^5.0.0"
  }
}

```

```

=====
FILE: backend/package.json
=====

```

```

{
  "name": "staff-backend",
  "private": true,
  "version": "1.0.0",
  "type": "commonjs",
  "main": "dist/app.js",
  "scripts": {
    "dev": "ts-node-dev --respawn --transpile-only src/app.ts",
    "build": "tsc -p tsconfig.json",
  }
}

```

```
    "start": "node dist/app.js"
  },
  "dependencies": {
    "axios": "^1.6.0",
    "cors": "^2.8.5",
    "express": "^4.19.2",
    "zod": "^3.22.4"
  },
  "devDependencies": {
    "@types/cors": "^2.8.17",
    "@types/express": "^4.17.21",
    "@types/node": "^20.0.0",
    "ts-node-dev": "^2.0.0",
    "typescript": "^5.0.0"
  }
}
```