Task A — Mini Full-Stack Search: Technical Project Reportt

Executive summary

> This report documents the design and implementation of a compact full-stack search application built with Next.js (App Router) and TypeScript. The app exposes a POST /api/search endpoint that scores and returns the top 3 matches from a local dataset and a single-page UI that submits queries, shows loading and empty states, and renders titled snippets. The system meets the acceptance criteria, including error handling, ordering, and an optional summary and sources field, and can be run locally with npm i && npm run dev.

Table of Contents

1. Project Title

Mini Full-Stack Search (Next.js + TypeScript)

2. Project Overview / Background

- Problem: Provide a minimal, local, dependency-light search UI backed by a server route over a small JSON dataset.
- Scope: One frontend page, one POST API endpoint, and an in-repo data file. No databases or external APIs.
- Purpose: Demonstrate correct retrieval, ordering, error handling, and a clean UX with loading and empty states, suitable for interview evaluation.

3. Objectives & Goals

- Return relevant results for keyword queries using simple case-insensitive scoring.
- Limit to top 3 matches sorted by relevance.
- Handle empty queries (400) and no-match cases (friendly message) gracefully.
- Show a single-page UI with input, Search button, loading, empty, and error states.
- Bonus: Return a short combined summary and sources: [ids].

Acceptance criteria

- Query “trust badges” returns id "1" first.

- Max 3 items, ordered by relevance.
- Empty query → HTTP 400.
- No matches → empty array + friendly message.

---

4. Prerequisites

- OS: Windows 10/11 (tested), macOS or Linux also fine.
- Runtimes: Node.js v18+ (tested with v22.19.0), npm ≥ 9 (tested with 10.9.3).
- Packages: Installed by create-next-app (next, react, react-dom, typescript, tailwindcss, postcss, autoprefixer).
- Environment variables: None required for Task A.
- Ports: 3000 (Next.js dev server).
- Hardware: Any modern laptop/desktop that runs Node 18+.

Screenshots (evidence)

- Screenshot 1: Node.js and npm versions validated.
- Screenshot 2: Project scaffold created successfully.
- Screenshot 3: Dev server ready with localhost and network URLs.

Screenshots note: Please go to the project's GitHub repository and open the /screenshots folder to view and download all referenced screenshots.

---

5. Tech Stack & Architecture

- Languages: TypeScript (preferred). JavaScript acceptable.
- Frameworks: Next.js (App Router), React.
- Styling: Tailwind CSS.
- Backend: Next.js route handler (no external DB).
- Data: Local file (data/data.ts).
- Dev tools: VS Code, npm, Git.

High-level architecture (ASCII)

```
[UI: app/page.tsx]
   |
   | fetch("/api/search", { method: "POST", body: { query } })
   v
[API: app/api/search/route.ts]
   │ loads → [data/data.ts]
   │ scoring, sort, slice → top 3
   v
 JSON { results, summary?, sources? }
```

6. Folder Structure

```
mini-search/
├─ app/
│   ├─ api/
│   │   └─ search/
│   │       └─ route.ts          # POST /api/search
│   └─ page.tsx                # Single-page UI
├─ data/
│   └─ data.ts                 # Local dataset
├─ public/                     # Static assets
├─ package.json                # Scripts and deps
├─ tsconfig.json               # TS config
├─ tailwind.config.ts          # Tailwind config
├─ postcss.config.mjs          # PostCSS config
└─ README.md                   # Setup and usage
```

7. Workflow & Implementation

7.1 Create project

```
npx create-next-app@latest mini-search --typescript
cd mini-search
npm i
```

7.2 Run dev server

```
npm run dev
# Visit http://localhost:3000
```

7.3 Add dataset (data/data.ts)

```
// data/data.ts
export const ITEMS = [
  { id: "1", title: "Trust badges near CTA", body: "Adding trust badges near th
e primary CTA increased signups by 12%." },
  { id: "2", title: "Above-the-fold form", body: "Visible form above the fold lift
ed lead submissions by 9%." },
  { id: "3", title: "Qualifying question", body: "A single qualifying question im
proved demo attendance by 6%." },
  { id: "4", title: "Test ID visibility", body: "Showing test IDs on hover sped up
analyst workflows." },
  { id: "5", title: "Down-funnel icon", body: "Green funnel means >10% down
-funnel uplift; red means <=0%." },
];
```

7.4 Implement API (app/api/search/route.ts)

```
// app/api/search/route.ts
import { NextResponse } from "next/server";
import type { NextRequest } from "next/server";
import { ITEMS } from "../../../data/data";

function normalize(s: string) {
  return (s || "").trim().toLowerCase();
}

export async function POST(req: NextRequest) {
  try {
    const body = await req.json().catch(() ⇒ ({}));
    const query = typeof body?.query === "string" ? body.query.trim() : "";

    // Empty query → 400
    if (!query) {
      return NextResponse.json(
        { error: "Empty query", message: "Please provide a non-empty search
term." },
        { status: 400 }
      );
    }

    const q = normalize(query);

    // Score: title has weight 2, body weight 1
    const scored = ITEMS.map((it) => {
      const title = normalize(it.title);
      const body = normalize(it.body);
      let score = 0;
      if (title.includes(q)) score += 2;
      if (body.includes(q)) score += 1;
      return { ...it, score };
    });

    // Filter, sort desc, top 3
    const matches = scored
      .filter((it) => it.score > 0)
      .sort((a, b) => b.score - a.score)
      .slice(0, 3);

    if (matches.length === 0) {
      return NextResponse.json({ results: [], message: "No matches found."
});
    }

    const sources = matches.map((m) => m.id);
    const summary = matches.map((m) => m.title).join(" • ");

    const results = matches.map((m) => ({
      id: m.id,
      title: m.title,
      snippet: m.body.length > 140 ? m.body.slice(0, 140).trim() + "..." : m.bod
y,
    }));
    return NextResponse.json({ results, sources, summary });
  } catch (err) {
    console.error("Search POST error:", err);
    return NextResponse.json({ error: "Server error" }, { status: 500 });
  }
}
```

7.5 Frontend UI (app/page.tsx)

```
"use client";
import { useState } from "react";

type Item = { id: string; title: string; snippet?: string; body?: string };

export default function Home() {
  const [query, setQuery] = useState("");
  const [results, setResults] = useState<Item[]>([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [message, setMessage] = useState<string | null>(null);
  const [summary, setSummary] = useState<string | null>(null);

  async function doSearch() {
    setError(null);
    setMessage(null);
    setSummary(null);
    const q = query.trim();
    if (!q) {
      setError("Please enter a search term.");
      setResults([]);
      return;
    }
    setLoading(true);
    try {
      const res = await fetch("/api/search", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ query: q }),
      });
      const data = await res.json();
      if (res.status === 400) {
        setError(data?.error || "Empty query");
        setResults([]);
        return;
      }
      if (!res.ok) throw new Error(`HTTP ${res.status}`);
      setResults(data.results ?? []);
      if (data.message) setMessage(data.message);
      if (data.summary) setSummary(data.summary);
    } catch (err: any) {
      setError(err?.message ?? "Network error");
      setResults([]);
    } finally {
      setLoading(false);
    }
  }

  return (
    <div className="min-h-screen bg-gray-50 dark:bg-black text-gray-90
0 dark:text-white p-8 flex flex-col items-center">
      <h1 className="text-3xl font-bold mb-6">Mini Search App</h1>
      <div className="w-full max-w-3xl flex gap-2">
        <input
          value={query}
          onChange={(e) => setQuery(e.target.value)}
          placeholder="Type your query..."
          className="flex-1 px-4 py-2 border rounded-lg bg-white dark:bg-gr
ay-900 text-black dark:text-white placeholder-gray-500"
        />
        <button onClick={doSearch} className="px-4 py-2 bg-indigo-600 h
over:bg-indigo-500 text-white rounded-lg">
          Search
        </button>
      </div>
```

```
      <div className="w-full max-w-3xl mt-6">
        {loading && <p className="text-sm text-gray-500">Searching...</p>}
        {error && <p className="text-sm text-red-500">Error: {error}</p>}
        {message && <p className="text-sm text-gray-400">{message}</p>}
        {summary && (
          <div className="mb-4 p-3 bg-gray-100 dark:bg-gray-800 rounded">
            <strong className="block mb-1">Summary</strong>
            <p className="text-sm text-gray-700 dark:text-gray-300">{summary}</p>
          </div>
        )}
        <ul className="space-y-3">
          {results.map((r) => (
            <li key={r.id} className="p-4 border border-gray-300 dark:border-gray-700 rounded bg-white dark:bg-gray-900">
              <h2 className="text-lg font-semibold">{r.title}</h2>
              <p className="text-sm text-gray-700 dark:text-gray-300 mt-1">{r.snippet ?? r.body}</p>
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
}
```

Common pitfalls and remediation

- 405 on GET /api/search in browser: Correct. Use POST with a JSON body.
- Import path errors for data/data.ts: Use relative path ../../../data/data from route.ts.
- CORS in dev: Not applicable with same-origin Next.js API.

8. API Description

- Method: POST
- Path: /api/search
- Request body (JSON): `{ "query": string }`

Responses

- 200 OK

```
{
  "results": [
    { "id": "1", "title": "Trust badges near CTA", "snippet": "Adding trust badges..." },
    { "id": "...", "title": "...", "snippet": "..." }
  ],
  "sources": ["1", "..."],
  "summary": "Trust badges near CTA • ..."
}
```

- 200 OK (no matches)

```
{ "results": [], "message": "No matches found." }
```

- 400 Bad Request

```
{ "error": "Empty query", "message": "Please provide a non-empty search term." }
```

- 500 Internal Server Error

```
{ "error": "Server error" }
```

Examples

- curl

```
curl -X POST http://localhost:3000/api/search \
  -H "Content-Type: application/json" \
  -d '{"query":"trust badges"}'
```

- fetch (browser)

```
await fetch("/api/search", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ query: "trust badges" })
});
```

9. UI / Frontend

- Component: app/page.tsx
- State: query, results, loading, error, message, summary.
- Network: fetch POST /api/search in doSearch().
- Styling: Tailwind utility classes in JSX.
- Customization: Adjust typography and colors in Tailwind config or inline classNames.

10. Backend

- Route handler: app/api/search/route.ts
- Data source: data/data.ts (local array).
- Algorithm: Case-insensitive substring search; score = 2 if title includes query, +1 if body includes query; sort desc; slice(0,3).
- Complexity: O(N) to score all N items per request, O(N log N) for sorting worst-case (N small = 5), then O(1) slice. For larger N, use partial-selection or heap for O(N log K).

11. Testing & Validation

Manual tests (representative)

- Empty query POST → 400 with error message.
- "trust badges" → id 1 ranked first.
- "form" or "test" → up to 3 results, ordered by relevance.

- "xyzabc" → results [], message "No matches found.".
- Case insensitivity: "Funnel" matches id 5.

Command examples

```
# Server
npm run dev

# Valid searches
curl -X POST http://localhost:3000/api/search -H "Content-Type: applicatio
n/json" -d '{"query":"trust badges"}'

# Empty query (expect 400)
curl -i -X POST http://localhost:3000/api/search -H "Content-Type: applicat
ion/json" -d '{}'
```

Screenshots (embed and caption in this section)

Screenshots note: Please go to the project's GitHub repository and open the /screenshots folder to view and download all referenced screenshots.

- Screenshot: Dev server ready at localhost:3000 — verifies server execution.
- Screenshot: Home page UI with search bar — verifies UI rendering.
- Screenshot: Network tab showing POST /api/search — verifies backend integration.

---

12. Logs, Metrics & Debugging

- Logs: Server errors printed by route.ts via console.error("Search POST error", err).
- Useful checks
  - Verify request body in Network tab Payload.
  - Inspect response JSON and status code.
  - Add temporary logs around scoring if needed.
- Common errors

    - 400 Empty query: ensure UI sends a non-empty string.
    - 500 Server error: check import path and data shape.

13. Security & Secrets

- No secrets required for Task A; no .env used.
- If adding APIs later: store secrets in .env (local) and provider-specific secret stores (e.g., Vercel project environment variables). Never expose via client bundles.

14. Deployment

Vercel (simple path)

1) Push repo to GitHub.

2) Import in Vercel, select framework = Next.js.

3) Build & deploy with defaults. No env vars required.

4) Validate POST /api/search on production URL.

Docker (optional)

```
# minimal Dockerfile example
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
EXPOSE 3000
CMD ["npm","start"]
```

15. Performance & Scaling Notes

- Bottleneck: O(N log N) sort per request. For small N this is trivial.
- Scaling:
    - Pre-tokenize and store lowercased fields.

    - Use K-selection or a heap for top-K (O(N log K)).
    - Cache frequent queries in memory with an LRU.
    - Move to an index (e.g., elasticlunr, MiniSearch) if dataset grows.

---

16. Key Achievements & Notes

- **Meets all acceptance criteria** including 400 on empty, friendly no-match, top-3 ordering.
- **Clean UX** with loading, error, and empty states.
- **Bonus fields** summary and sources implemented.
- **Zero external dependencies** for data and persistence.

---

17. Maintenance & Future Work

- Add unit tests for scoring and ordering.
- Add debounce on client input and auto-search on Enter.
- Highlight matched terms in snippet.
- Pagination and result count.
- Swap local data file for a persistent store when needed.

---

18. Appendix

18.1 Full dataset (data/data.ts)

```
export const ITEMS = [
  { id: "1", title: "Trust badges near CTA", body: "Adding trust badges near th
e primary CTA increased signups by 12%." },
  { id: "2", title: "Above-the-fold form", body: "Visible form above the fold lift
ed lead submissions by 9%." },
  { id: "3", title: "Qualifying question", body: "A single qualifying question im
proved demo attendance by 6%." },
  { id: "4", title: "Test ID visibility", body: "Showing test IDs on hover sped up
analyst workflows." },
  { id: "5", title: "Down-funnel icon", body: "Green funnel means >10% down
-funnel uplift; red means <=0%." },
];
```

18.2 Terminal logs (representative)

```
▲ Next.js 16.x (Turbopack)
- Local:   http://localhost:3000
- Network: http://192.168.x.x:3000
✓ Starting...
✓ Ready in 1.5s
```

How to Demonstrate This Project Live

1) Clone and install

```
git clone <your-repo-url> mini-search
cd mini-search
npm i
```

2) Run locally

```
npm run dev
# then open http://localhost:3000
```

3) Validate API

```
curl -X POST http://localhost:3000/api/search \
  -H "Content-Type: application/json" \
  -d '{"query":"trust badges"}'
```

Expected: id 1 first, with summary and sources.

4) Validate error handling

```
curl -i -X POST http://localhost:3000/api/search -H "Content-Type: application/json" -d '{}'
```

Expected: HTTP/1.1 400 with error message.

5) UI verification

- Type queries in the input and press Search.
- Observe loading, error, empty, and rendered results (title + snippet).

Printable Quick-Start Cheat Sheet

- Install: `npm i`
- Dev server: `npm run dev` then open `http://localhost:3000`
- Dataset: `data/data.ts`
- API: `POST /api/search` with `{ "query": "<text>" }`
- Success (200): `{ results: [...], summary?, sources? }`
- Empty query: 400 `{ error: "Empty query" }`
- No matches: 200 `{ results: [], message: "No matches found." }`
- Key files: `app/page.tsx`, `app/api/search/route.ts`, `data/data.ts`
- Deploy: push to GitHub → Vercel import (no env vars required)