Search Bar Enhancement Implementation Report

Date: November 30, 2025
Task: Implement READ-ONLY search suggestion endpoint `/search/v1/suggest`

1. Executive Summary

This report details the implementation of the new search suggestion endpoint. The feature provides real-time product and category suggestions with fuzzy search capabilities, typo correction, and rate limiting, without relying on external search engines.

2. File Changes

2.1 New Files

`src/controllers/search/searchSuggest.js`

Purpose: Handles the search logic, including validation, typo correction, database querying, and response formatting.

```
 import Product from '../../models/products.js';
import Category from '../../models/category.js';

// ... (Rate limiter and Typo map definitions) ...

export const searchSuggest = async (req, reply) => {
    // ... (Logic implementation) ...
};
```

`src/routes/search.js`

Purpose: Defines the Fastify route plugin for the search endpoint.

```
import { searchSuggest } from
'../controllers/search/searchSuggest.js';

export const searchRoutes = async (fastify, options) => {
    fastify.get('/search/v1/suggest', searchSuggest);
};
```

2.2 Modified Files

`src/models/products.js`

Change: Added an index on the `name` field to optimize text search performance.

```
// ... existing indexes ...
productSchema.index({ category: 1, status: 1 });
productSchema.index({ name: 1 }); // [NEW] Index for text-like search
```

`src/models/category.js`

Change: Added an index on the `name` field.

```
// ... schema definition ...
// [NEW] Index for search
categoryScehma.index({ name: 1 });
```

`src/routes/index.js`

Change: Registered the new search route plugin with an empty prefix to ensure the endpoint is accessible at `/search/v1/suggest` (not `/api/search...`).

```
import { searchRoutes } from './search.js'; // [NEW] Import

// ... inside registerRoutes ...
console.log('Registering search routes...');
await fastify.register(searchRoutes, { prefix: '' }); // [NEW] Register
console.log('Search routes registered');
```

3. Logic Explanation

3.1 Request Processing Flow

1. **Input Validation:** The controller checks if the query parameter `q` exists and is a string.
2. **Normalization:** The query is trimmed and multiple spaces are replaced with a single space.
3. **Short Query Check:** If the query length is less than 2 characters, an empty result set is returned immediately (Option A).
4. **Rate Limiting:** A simple in-memory counter tracks requests per IP. If an IP exceeds 10 requests in 5 seconds, a 429 error is returned.

3.2 Typo Correction

A dictionary-based approach is used for common typos.

- **Map:** `{'maggie': 'maggi', 'ata': 'atta', ...}`
- **Logic:** If the normalized query matches a key in the map, it is replaced with the correct value, and the `typoCorrected` flag is set to `true`.

3.3 Database Querying

- **Products:** Searches the `Product` collection for items where `status: 'approved'` and `isActive: true`.
- **Categories:** Searches the `Category` collection.
- **Method:** Uses MongoDB `RegExp` for case-insensitive fuzzy matching (`new RegExp(query, 'i')`).

3.4 Response Formatting

Results are mapped to a standardized JSON structure:

- `type` : "product" or "category"
- `id` : Unique identifier
- `name` : Item name

- `image` : Image URL
- `soldCount` : Sales count (defaults to 0 if missing)
- `typoCorrected` : Boolean flag

4. Error Codes & Handling

HTTP Status	Error Code	Description
200 OK	N/A	Successful request. Returns JSON with `results` array.
429 Too Many Requests	`RATE_LIMITED`	The client IP has exceeded the allowed request rate (10 req/5s).
500 Internal Server Error	`TEMPORARY_ERROR`	An unexpected server-side error occurred (e.g., database connection failure).

Error Response Shape:

```
{
  "results": [],
  "error": "ERROR_CODE",
  "typoCorrected": false,
  "originalQuery": "...",
  "correctedQuery": "..."
}
```

5. How it Works (End-to-End)

1. **User types "maggie"** in the search bar.
2. **Frontend** sends `GET /search/v1/suggest?q=maggie` .
3. **Server** receives request.
4. **Rate Limiter** checks IP. (Allowed).
5. **Typo Corrector** sees "maggie", changes it to "maggi", sets `typoCorrected=true` .

6. **Database** is queried for products/categories matching "maggi" (case-insensitive).
7. **Server** formats the found documents into the result list.
8. **Server** responds with JSON containing the corrected results and the `typoCorrected: true` flag.
9. **Frontend** displays "Showing results for **maggi**" and lists the items.