Overview

This documentation describes the structure and collection methodology for two structured data files which are used in this project.

- **`knowledge_base.json`**: A deeply structured JSON representing detailed metadata for each restaurant and its menu.
- **`optimized_corpus.json`**: A flattened, context-rich version of the same data, optimized for embedding and retrieval.

It also outlines the structured data creation pipeline , describing the full journey from raw website scraping to generating retrieval-optimized corpus files. Different code logics are used for different steps.

Stage	Output	Code file	Description
1. Crawling	`swiggy_restaurants_kanpur.csv`	`crawler.py`	Extracts basic metadata and links to restaurant pages
2. Scraping	`*_dishes.csv`, `complete_kanpur_restaurants_dishes.csv`	`scraper.py`	Extracts detailed dish data per restaurant
3. Knowledge Base	`knowledge_base.json`	`knowledge_base.py`	A hierarchical, structured JSON database of restaurants and menus
4. Optimized Corpus	`optimized_corpus.json`	`optimised_corpus.py`	Flattened and rich text-format for embedding into vector databases

Data Schema

In this section we will be explaining the internal structure of structured data.

1. knowledge_base.json (Structured Restaurant Knowledge)

Each entry in this JSON corresponds to a single restaurant object with the following fields:

Field	Description
`restaurant_name`	Name of the restaurant
`available_cuisine`	Comma-separated string of cuisine types served
`delivery_time`	Approximate delivery duration
`restaurant_rating`	Overall customer rating of the restaurant
`city`	The city in which the restaurant operates
`restaurant_location`	Detailed address
`restaurant_menu`	List of dictionaries, each representing a dish served

Nested structure of `restaurant_menu`:

Field	Description
`dish_name`	Name of the dish
`description`	Description of the dish (can be `null`)
`price`	Price (as string)
`rating`	Dish rating
`num_reviews`	Number of user reviews
`dish_type`	"Veg Item" or other classification
`tags`	Optional tags like "Bestseller" (list)
`dish_tags`	Additional categorization (e.g. "Combos", "Paneer", "Thali")

This schema enables fine-grained reasoning, filtering, and classification of restaurant offerings.

2. `optimized_corpus.json` (RAG-Optimized Flat Corpus)

This file transforms the deeply nested `knowledge_base.json` into a flattened, NLP-ready text format.

This file contains:`{`

```
  "documents": [...],
  "metadata": [...]
}
```

`documents`: Rich NLP-readable chunks

Each string is a concatenated, context-rich document combining dish + restaurant info.

Sample format of each document:

```
Dish: Paneer do Pyaza
Description: Paneer Do Pyaza Is Prepared With Julian Onion And
Capsicum In Brown Gravy
Price: 169
Type: Veg Item
Restaurant: Anandeshwar dhaba
Location: 117/n/306 raniganj kakadeo Kanpur nagar ,208025, kanpur
Dish rating: 4.2 based on Number of reviews: (192)
dish_tags: Recommended
Additional Tags: Restaurant Rating: 4.4 ((192) reviews)
Cuisine: Thalis, Indian, Chinese, Fast Food
Delivery Time:  20-25 mins
```

This format is designed to:

- Be easily searchable and embeddable via models like `BAAI/bge-base-en-v1.5`
- Contain context-rich information to improve retrieval quality

`metadata`: Corresponding structured fields

Each dictionary object has the following structure:

```
{
  "restaurant_name": "Chinese Wok",
  "location": "Moti Jheel",
  "city": "Kanpur",
  "dish_name": "Chilli Paneer",
  "dish_type": "Veg Item",
  "tags": "Bestseller",
  "dish_rating": "4.5",
  "restaurant_rating": "4.3",
  "price": "299"
}
```

This format is ideal for semantic embedding with models like `BAAI/bge-base-en-v1.5`, and powers the FAISS index for vector search in your RAG chatbot.

Collection Methodology

In this section we will describe the key components of all four code logic which were employed to scrape data from Swiggy website and convert it into structured data. (Swiggy website has permission for automated data scraping in its robots.txt)

We used python's **SELENIUM** library to employ web scraping from dynamic website like Swiggy's website.

`1.crawler.py` — *Restaurant Discovery via City Page Crawling*

Objective:

To scrape a **list of restaurants** (with name, cuisine, rating, and detail page link) from a Swiggy city page (like `swiggy.com/city/kanpur/order-online`).

Key Components:

Section	Purpose
`get_driver()`	Launches headless Chrome with safe scraping flags.
`click_show_more(driver)`	Simulates repeated clicking on "Show More" to load all restaurant cards. Uses **JS fallback** to handle click interception.
`scrape_restaurants(driver, city)`	Extracts: restaurant name, cuisines, Swiggy link, and rating.
`scrape_multiple_cities_to_csv()`	Iterates through city slugs and saves all scraped restaurant metadata into a CSV file (`swiggy_restaurants_kanpur.csv`).

Outcome:

We get a flat CSV list of all restaurants and links to their individual menu pages — critical for downstream scraping.

`2.scraper.py` — *Full Menu Scraping per Restaurant*

Objective:

To visit each restaurant link from `crawler.py` and scrape all available **dish-level details**.

Key Components:

Section	Purpose
`click_show_more()`	Not essential here, but present for consistency.
`get_driver()`	Loads the Selenium driver again with JS rendering support.

`scrape_restaurants(driver)`	• Extracts:
	  ◦ Restaurant name and address
	  ◦ Section titles (e.g., "Chinese (6)", "Combos")
	  ◦ Dish items per section
	  ◦ Cleans and structures:
	  ▪ Dish name
	  ▪ Description
	  ▪ Price
	  ▪ Rating
	  ▪ Section tag (e.g., "Chinese")
	• Stores each dish as a dictionary

`scrape_multiple_cities_to_csv()`	• Reads links from `swiggy_restaurants_kanpur.csv`
	• Iterates each link, scrapes dishes, saves as:
	  ◦ `*_dishes.csv` per restaurant
	  ◦ Combined as `complete_kanpur_restaurants_dishes.csv`

Outcome:

We get **detailed menus** for every restaurant in Kanpur, complete with dish-level granularity.

3. `knowledge_base.py` — *Structured JSON Construction*

Objective:

To transform multiple per-restaurant CSVs into a **clean, nested JSON knowledge base**.

Key Components:

Section	Purpose
`restaurant_data(df)`	Extracts and cleans delivery time, rating, and city from restaurant info.

`Data_Cleaning(csv_file)`	• Parses `Complete Info` block from dish
	• Splits into:
	  ○ Dish name
	  ○ Description
	  ○ Price
	  ○ Rating
	  ○ Tags like "Bestseller"
	• Extracts structured fields like dish type and review count

Iteration through all CSV files	• Extracts `restaurant_name` from file name • Retrieves the matching restaurant row from the CSV • Embeds menu in a `restaurant_menu` list Assembles each restaurant as: `{ "restaurant_name": "...", "city": "...", "restaurant_location": "...", "restaurant_menu": [...] }`
Save as `knowledge_base.json`	Full nested restaurant–menu data gets serialized.

Outcome:

We get a **structured, relational-style dataset** supporting hierarchical queries.

4. `optimized_corpus.py` — *RAG-Optimized Corpus Builder*

Objective:

To flatten structured data into a **textual corpus** that works well with semantic search (e.g., FAISS).

Key Components:

Section	Purpose
Load `knowledge_base.json`	Load structured restaurant–dish objects.

Iterate through entries	For every `restaurant_menu` item, build:
	• `text` string with rich context:
	  ○ Dish + Description + Restaurant + Location + Cuisine + Ratings
	• `metadata` dictionary:
	  ○ Dish name, tags, type, city, price, etc

Save to `optimized_corpus.json`	A dict with: `{ "documents": [...], # For embedding "metadata": [...] # For filtering }`

Outcome:

After this step our scraped data is now ready for **FAISS indexing** using an embedding model (`BAAI/bge-base-en-v1.5`), which powers the chatbot's RAG system.

Summary of Flow

```
crawler.py
  → swiggy_restaurants_kanpur.csv
      ↓
scraper.py
  → *_dishes.csv + combined CSV
      ↓
knowledge_base.py
  → knowledge_base.json
      ↓
optimized_corpus.py
  → optimized_corpus.json  → FAISS Indexing (RAG)
```