Zongrong Li
DSCI 510 ChatDB 82

Yelp Restaurant Finder ChatDB – Final Report

Repository / Code Link: https://github.com/Jasper0122/yelp-chatdb-demo

1. Introduction

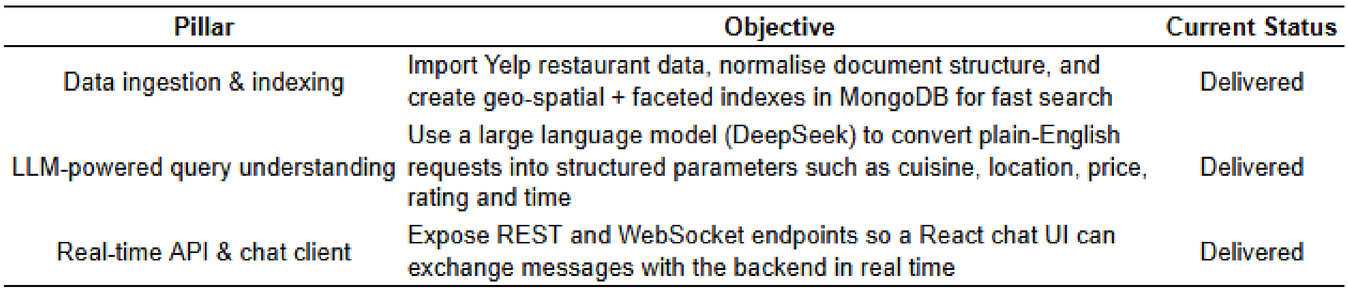
The Yelp Restaurant Finder ChatDB project offers a conversational interface that lets users discover restaurants through plain-English requests. A user can type, for example, “Show me a sushi place in San Francisco that is open now and rated at least four stars,” and immediately receive a curated list that meets the criteria. The system combines a large-language-model-driven natural-language understanding layer with a schema-aware query generator and a high-performance backend. Together they translate free-form utterances into MongoDB queries, execute those queries against a restaurant collection populated from the Yelp Fusion API, and return structured results in a chat-friendly format. The project was conceived as a solo capstone for DSCI 551 Spring 2025 and has progressed from a proof-of-concept to a fully functional prototype.

2. Planned Implementation

The system is built upon three fundamental components: data storage, natural language query processing, and backend API development.

First, restaurant data is gathered from the Yelp API and stored in MongoDB. To enable fast and accurate searches, the data is normalized and indexed on frequently queried fields such as location, cuisine type, and rating. This ensures responsive query performance across a large dataset. Next, the query processing pipeline utilizes the DeepSeek API, a large language model (LLM), to extract user intent and relevant parameters such as cuisine, location, rating, and price range. These extracted parameters are converted into structured MongoDB queries. The system also applies techniques such as entity recognition and query expansion to enhance interpretation accuracy.

Finally, the backend is developed using FastAPI, which handles user queries and communicates with the MongoDB database. The API returns structured responses and supports real-time interaction through both REST and WebSocket endpoints. This enables seamless integration with a front-end React-based chat interface. Additionally, enhancements like support for vegetarian or pet-friendly filters and sentiment-aware review summaries are in development. A breakdown of each implementation pillar, its objective, and completion status is shown in Figure 1 below.



Pillar	Objective	Current Status
Data ingestion & indexing	Import Yelp restaurant data, normalise document structure, and create geo-spatial + faceted indexes in MongoDB for fast search	Delivered
LLM-powered query understanding	Use a large language model (DeepSeek) to convert plain-English requests into structured parameters such as cuisine, location, price, rating and time	Delivered
Real-time API & chat client	Expose REST and WebSocket endpoints so a React chat UI can exchange messages with the backend in real time	Delivered

Figure 1. Implementation Pillars, Objectives, and Status

3. Architecture Design

The system architecture of Yelp Restaurant Finder ChatDB adopts a modular, top-down processing flow that integrates language modeling, query planning, and multi-source data retrieval. The complete architecture is illustrated in Figure 2.

Zongrong Li
DSCI 510 ChatDB 82

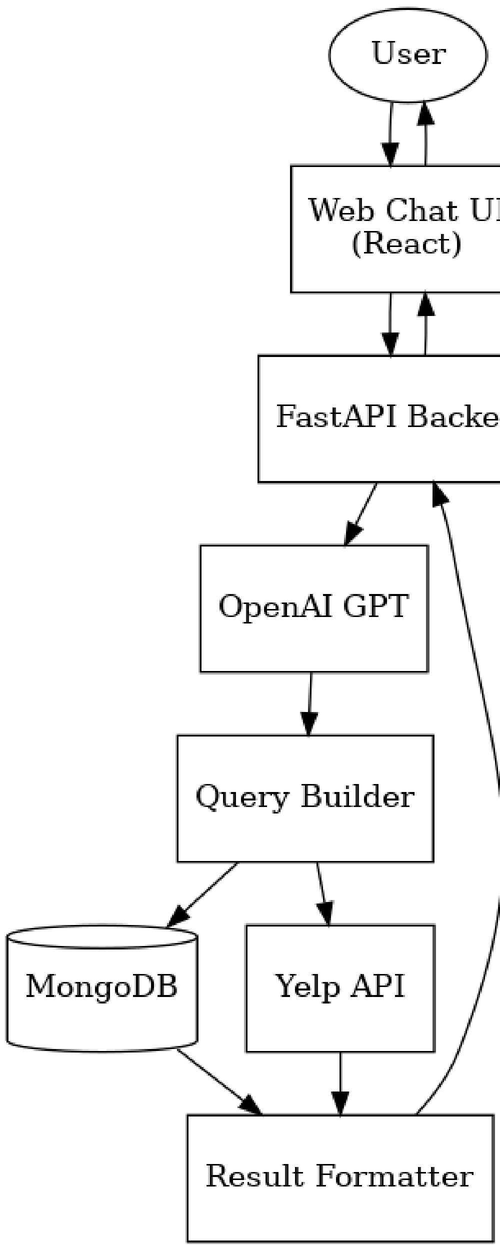


Figure 2. Diagram of Yelp ChatDB Architecture

The interaction begins with the user engaging the system through a browser-based chat interface, developed using React. This front-end component handles all user interactions, including capturing free-form natural language input and rendering system responses in real time. Once the user submits a query such as “Find me a 4-star sushi restaurant in San Francisco,” the message is transmitted to the FastAPI backend via an HTTP request. This backend serves as the core orchestrator, coordinating all downstream services.

Upon receiving the input, the FastAPI backend formulates a prompt and passes it to the OpenAI GPT model, which acts as the system’s natural language understanding engine. The GPT model parses the intent embedded in the user query and returns a structured JSON representation. This JSON contains semantic elements such as the desired location (“San Francisco”), cuisine type (“sushi”), minimum rating (≥4), and potentially constraints on price, distance, or open hours.

The structured representation is then handed over to a Query Builder module. This module has dual responsibilities: it generates an aggregation query for MongoDB, which acts as a local restaurant database, and it constructs a dynamic API request to the Yelp Fusion API for retrieving up-to-date listings. The MongoDB query serves as a fast-access option and fallback mechanism, particularly useful during testing or API quota constraints, while the Yelp API provides authoritative, real-time results from Yelp’s public data service. Both MongoDB and Yelp API return a list of candidate restaurant records. These results are then passed to the Result Formatter module. This component is responsible for merging the two sources, eliminating duplicates, normalizing field structures, and selecting the most relevant entries. It also formats the results into clean, concise summaries suitable for presentation in the chat UI, including restaurant names, ratings, addresses, and snippets of reviews.

The finalized output is returned to the FastAPI backend, which streams the response back to the React front-end. The UI renders the message as a conversational reply, completing one full interaction cycle. If the user sends a follow-up query, the process loops with full contextual continuity.

4. Implementation

4.1 Functionalities

The implemented system enables end-users to perform natural language-based restaurant discovery through a web-based chat interface. Users can submit open-ended queries such as “Find me cheap Mexican food near Santa Monica” or “Show 5-star places open now in San Jose.” The system handles not only keyword matching but semantic interpretation. Behind the scenes, the input is passed through an OpenAI GPT model, which identifies parameters such as location (“Santa Monica”), cuisine (“Mexican”), price filter (“cheap”), rating threshold, and service availability (e.g., “open now”).

The interface supports multi-turn conversations where users can refine results using follow-up prompts like “only the ones with takeout” or “exclude sushi.” The GPT model maintains context, enabling natural follow-ups. This conversational continuity enhances usability and gives the impression of interacting with an intelligent assistant rather than a static search box. Functionally, the system retrieves matching results from two sources: MongoDB, for internal testing and validation, and the Yelp Fusion API, which provides accurate, live data. The backend logic merges these sources, removes duplicates, and ranks results based on relevance, then streams them back in a clean message format that includes name, rating, address, and review snippet.

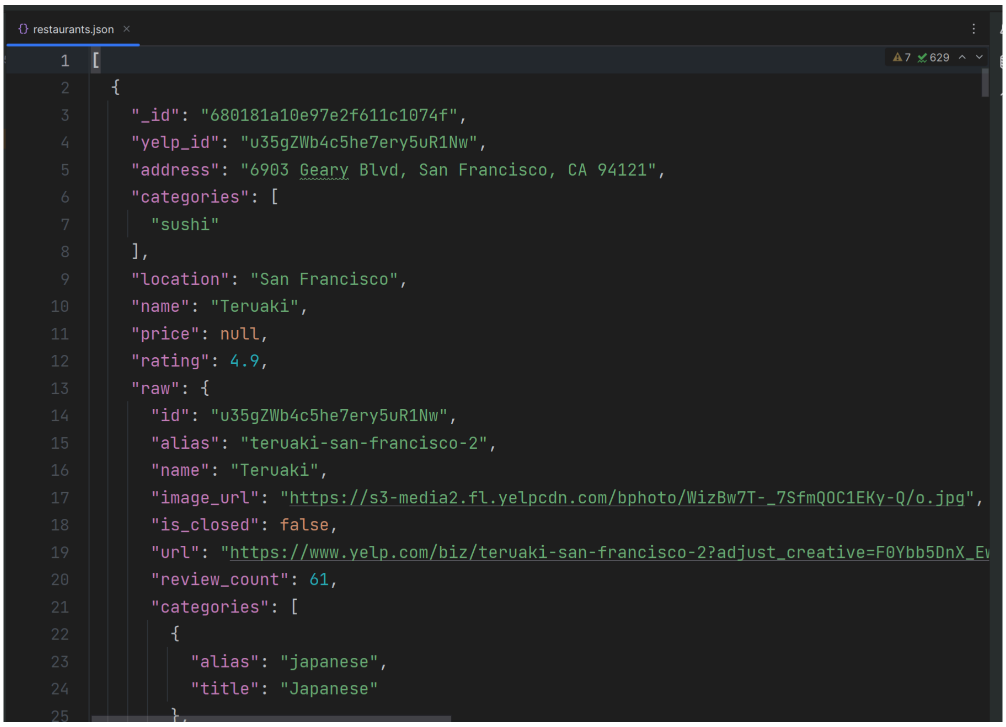
4.2 Tech Stack

The frontend is built using React, chosen for its component-based structure and virtual DOM performance. Tailwind CSS is used for utility-first styling, enabling a responsive and minimalist chat layout. The front end acts as the messaging layer between the user and the backend API, sending user queries and rendering system responses. The backend is implemented using FastAPI, a Python-based web framework selected for its asynchronous support and rapid development capabilities. It handles RESTful endpoints and controls the interaction with external services and internal modules. Natural language interpretation is powered by OpenAI GPT-3.5, accessed via API calls, which transforms input into structured semantic representations.

Data retrieval is conducted through two primary channels: MongoDB, used for local caching and schema debugging, and the Yelp Fusion API, which provides real-time restaurant information. A custom-designed Query Builder module converts the GPT-generated JSON into executable MongoDB aggregation pipelines or Yelp API query strings. Results from both are processed through a Result Formatter, which unifies the format, handles deduplication, and enforces presentation rules (e.g., top N results, field selection). This architecture separates concerns cleanly and promotes maintainability.

4.3 Implementation Screenshots

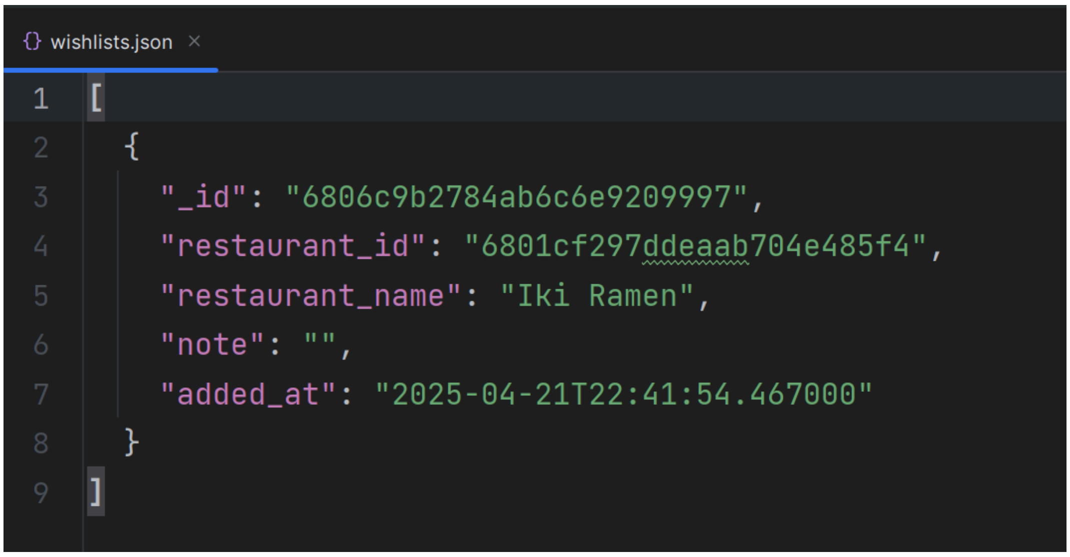
To demonstrate the system’s functionality and technical completeness, a series of screenshots are included below. These illustrate how various components of the Yelp ChatDB pipeline work together—from user interaction and GPT parsing, to MongoDB caching, Yelp API fallback, and final rendering of results in the chat interface.



```
[
  {
    "_id": "680181a10e97e2f611c1074f",
    "yelp_id": "u35gZWb4c5he7ery5uR1Nw",
    "address": "6903 Geary Blvd, San Francisco, CA 94121",
    "categories": [
      "sushi"
    ],
    "location": "San Francisco",
    "name": "Teruaki",
    "price": null,
    "rating": 4.9,
    "raw": {
      "id": "u35gZWb4c5he7ery5uR1Nw",
      "alias": "teruaki-san-francisco-2",
      "name": "Teruaki",
      "image_url": "https://s3-media2.fl.yelpcdn.com/bphoto/WizBw7T-_7SfmQOC1EKy-Q/o.jpg",
      "is_closed": false,
      "url": "https://www.yelp.com/biz/teruaki-san-francisco-2?adjust_creative=F0Ybb5DnX_Ew
      "review_count": 61,
      "categories": [
        {
          "alias": "japanese",
          "title": "Japanese"
        },
```

Figure 3. Restaurants JSON Structure

This screenshot shows a sample document stored in restaurants.json, which is exported from MongoDB. It includes essential metadata retrieved from Yelp, such as business name, address, rating, and raw fields. This structure mirrors what the backend uses to generate final chat responses.



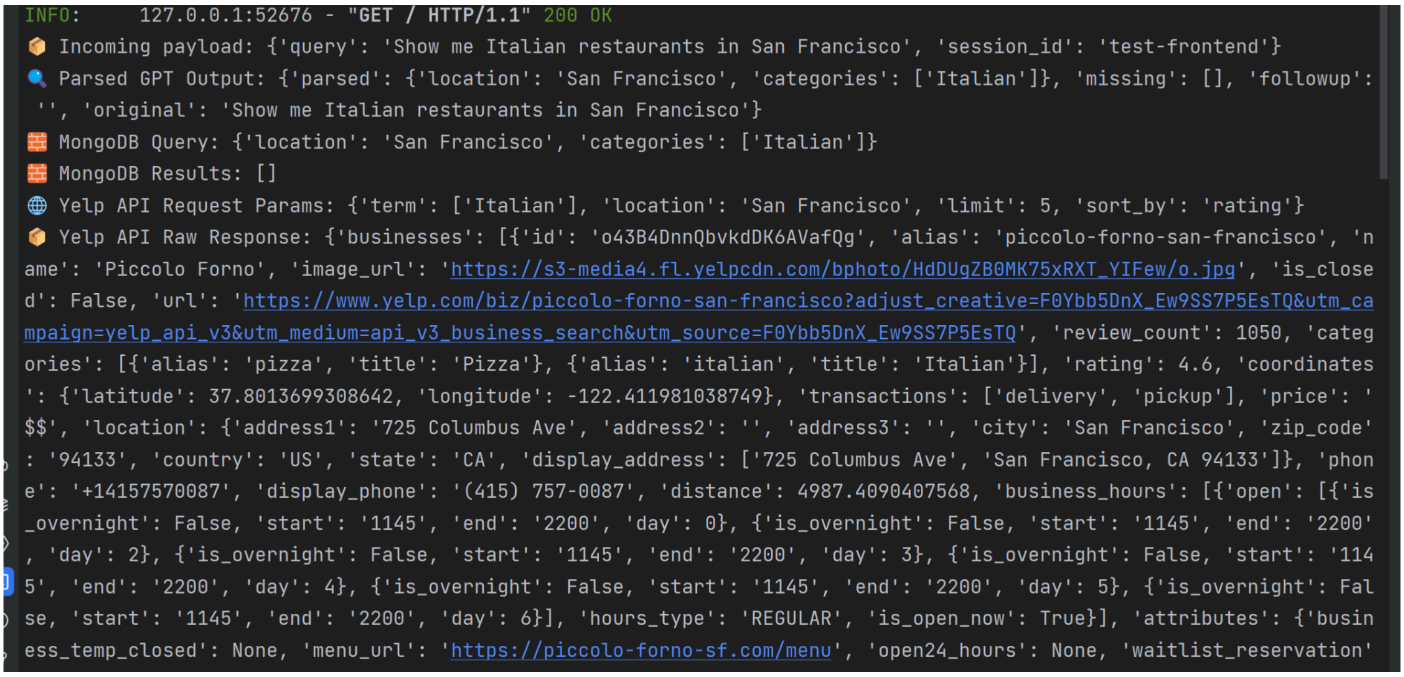
```
[
  {
    "_id": "6806c9b2784ab6c6e9209997",
    "restaurant_id": "6801cf297ddeaab704e485f4",
    "restaurant_name": "Iki Ramen",
    "note": "",
    "added_at": "2025-04-21T22:41:54.467000"
  }
]
```

Figure 4. Wishlist JSON Structure

This figure presents a single entry in the wishlists.json file. It shows a user's saved restaurant with fields including the restaurant ID, name, optional note, and timestamp. This data is used to reconstruct personal recommendations and support persistent user sessions.

Figure 5. React-based Chat UI Interface

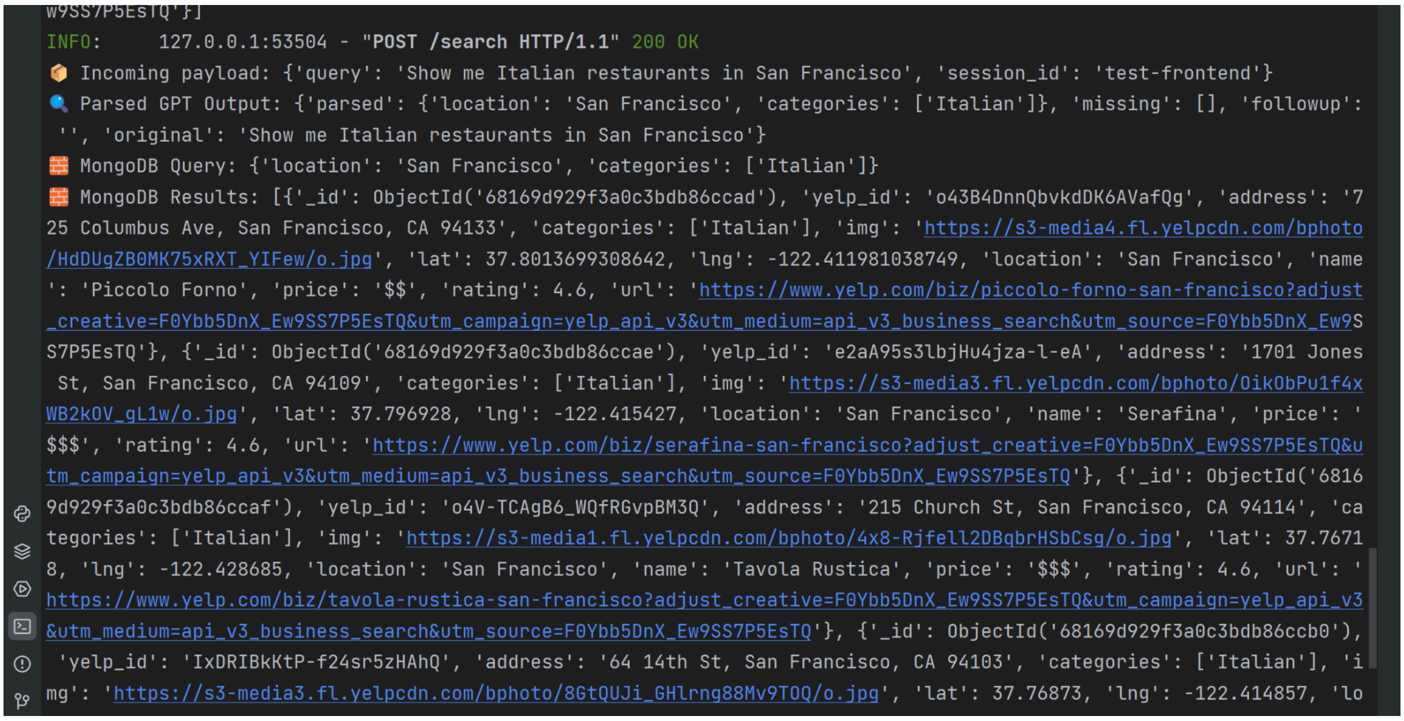
The main web interface allows users to interact with the assistant in natural language. It includes prompt suggestions, a text input field, and buttons for accessing saved wishlists and chat history. This React frontend is connected to the FastAPI backend through asynchronous HTTP requests.



```
INFO:     127.0.0.1:52676 - "GET / HTTP/1.1" 200 OK
📦 Incoming payload: {'query': 'Show me Italian restaurants in San Francisco', 'session_id': 'test-frontend'}
🔍 Parsed GPT Output: {'parsed': {'location': 'San Francisco', 'categories': ['Italian']}, 'missing': [], 'followup':
 '', 'original': 'Show me Italian restaurants in San Francisco'}
🧱 MongoDB Query: {'location': 'San Francisco', 'categories': ['Italian']}
🧱 MongoDB Results: []
🌐 Yelp API Request Params: {'term': ['Italian'], 'location': 'San Francisco', 'limit': 5, 'sort_by': 'rating'}
📦 Yelp API Raw Response: {'businesses': [{'id': 'o43B4DnnQbvkdDK6AVafQg', 'alias': 'piccolo-forno-san-francisco', 'n
ame': 'Piccolo Forno', 'image_url': 'https://s3-media4.fl.yelpcdn.com/bphoto/HdDUgZB0MK75xRXT_YIFew/o.jpg', 'is_close
d': False, 'url': 'https://www.yelp.com/biz/piccolo-forno-san-francisco?adjust_creative=F0Ybb5DnX_Ew9SS7P5EsTQ&utm_ca
mpaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=F0Ybb5DnX_Ew9SS7P5EsTQ', 'review_count': 1050, 'categ
ories': [{'alias': 'pizza', 'title': 'Pizza'}, {'alias': 'italian', 'title': 'Italian'}], 'rating': 4.6, 'coordinates
': {'latitude': 37.8013699308642, 'longitude': -122.411981038749}, 'transactions': ['delivery', 'pickup'], 'price': '
$$', 'location': {'address1': '725 Columbus Ave', 'address2': '', 'address3': '', 'city': 'San Francisco', 'zip_code'
: '94133', 'country': 'US', 'state': 'CA', 'display_address': ['725 Columbus Ave', 'San Francisco, CA 94133']}, 'phon
e': '+14157570087', 'display_phone': '(415) 757-0087', 'distance': 4987.4090407568, 'business_hours': [{'open': [{'is
_overnight': False, 'start': '1145', 'end': '2200', 'day': 0}, {'is_overnight': False, 'start': '1145', 'end': '2200'
, 'day': 2}, {'is_overnight': False, 'start': '1145', 'end': '2200', 'day': 3}, {'is_overnight': False, 'start': '114
5', 'end': '2200', 'day': 4}, {'is_overnight': False, 'start': '1145', 'end': '2200', 'day': 5}, {'is_overnight': Fal
se, 'start': '1145', 'end': '2200', 'day': 6}], 'hours_type': 'REGULAR', 'is_open_now': True}], 'attributes': {'busin
ess_temp_closed': None, 'menu_url': 'https://piccolo-forno-sf.com/menu', 'open24_hours': None, 'waitlist_reservation'
```

Figure 6. Terminal Log – Yelp API Requested on Cache Miss

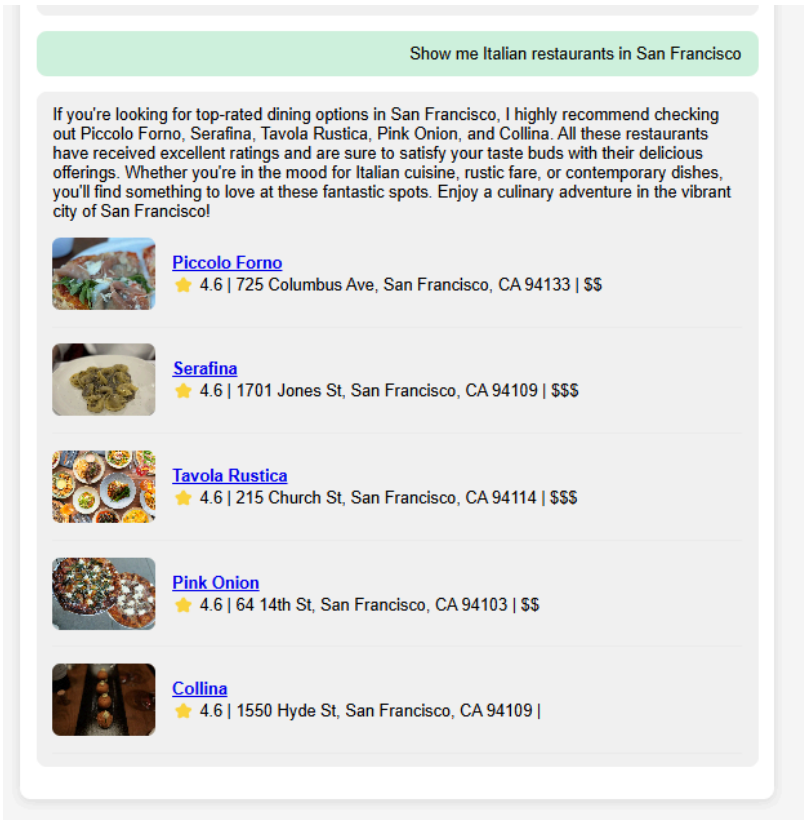
This terminal output shows how the backend handles a cache miss. When the user asks for Italian restaurants in San Francisco and no relevant documents are found in the MongoDB cache, the system automatically constructs a parameterized query to the Yelp API, receives the raw JSON response, and stores the data for future use.



```
w9SS7P5EsTQ'}]
INFO:     127.0.0.1:53504 - "POST /search HTTP/1.1" 200 OK
📦 Incoming payload: {'query': 'Show me Italian restaurants in San Francisco', 'session_id': 'test-frontend'}
🔍 Parsed GPT Output: {'parsed': {'location': 'San Francisco', 'categories': ['Italian']}, 'missing': [], 'followup':
 '', 'original': 'Show me Italian restaurants in San Francisco'}
🧱 MongoDB Query: {'location': 'San Francisco', 'categories': ['Italian']}
🧱 MongoDB Results: [{'_id': ObjectId('68169d929f3a0c3bdb86ccad'), 'yelp_id': 'o43B4DnnQbvkdDK6AVafQg', 'address': '7
25 Columbus Ave, San Francisco, CA 94133', 'categories': ['Italian'], 'img': 'https://s3-media4.fl.yelpcdn.com/bphoto
/HdDUgZB0MK75xRXT_YIFew/o.jpg', 'lat': 37.8013699308642, 'lng': -122.411981038749, 'location': 'San Francisco', 'name
': 'Piccolo Forno', 'price': '$$', 'rating': 4.6, 'url': 'https://www.yelp.com/biz/piccolo-forno-san-francisco?adjust
_creative=F0Ybb5DnX_Ew9SS7P5EsTQ&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=F0Ybb5DnX_Ew9S
S7P5EsTQ'}, {'_id': ObjectId('68169d929f3a0c3bdb86ccae'), 'yelp_id': 'e2aA95s3lbjHu4jza-l-eA', 'address': '1701 Jones
 St, San Francisco, CA 94109', 'categories': ['Italian'], 'img': 'https://s3-media3.fl.yelpcdn.com/bphoto/OikObPu1f4x
WB2kOV_gL1w/o.jpg', 'lat': 37.796928, 'lng': -122.415427, 'location': 'San Francisco', 'name': 'Serafina', 'price': '
$$$', 'rating': 4.6, 'url': 'https://www.yelp.com/biz/serafina-san-francisco?adjust_creative=F0Ybb5DnX_Ew9SS7P5EsTQ&u
tm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=F0Ybb5DnX_Ew9SS7P5EsTQ'}, {'_id': ObjectId('6816
9d929f3a0c3bdb86ccaf'), 'yelp_id': 'o4V-TCAgB6_WQfRGvpBM3Q', 'address': '215 Church St, San Francisco, CA 94114', 'ca
tegories': ['Italian'], 'img': 'https://s3-media1.fl.yelpcdn.com/bphoto/4x8-Rjfell2DBqbrHSbCsg/o.jpg', 'lat': 37.7671
8, 'lng': -122.428685, 'location': 'San Francisco', 'name': 'Tavola Rustica', 'price': '$$$', 'rating': 4.6, 'url': '
https://www.yelp.com/biz/tavola-rustica-san-francisco?adjust_creative=F0Ybb5DnX_Ew9SS7P5EsTQ&utm_campaign=yelp_api_v3
&utm_medium=api_v3_business_search&utm_source=F0Ybb5DnX_Ew9SS7P5EsTQ'}, {'_id': ObjectId('68169d929f3a0c3bdb86ccb0'),
 'yelp_id': 'IxDRIBkKtP-f24sr5zHAhQ', 'address': '64 14th St, San Francisco, CA 94103', 'categories': ['Italian'], 'i
mg': 'https://s3-media3.fl.yelpcdn.com/bphoto/8GtQUJi_GHlrng88Mv9TOQ/o.jpg', 'lat': 37.76873, 'lng': -122.414857, 'lo
```

Figure 7. Terminal Log – Cache Hit with MongoDB Result

This screenshot demonstrates a subsequent request for the same information. The system finds a cache hit in MongoDB and returns the full list of matching restaurants without calling Yelp again, improving performance and reducing API usage.



Show me Italian restaurants in San Francisco

If you're looking for top-rated dining options in San Francisco, I highly recommend checking out Piccolo Forno, Serafina, Tavola Rustica, Pink Onion, and Collina. All these restaurants have received excellent ratings and are sure to satisfy your taste buds with their delicious offerings. Whether you're in the mood for Italian cuisine, rustic fare, or contemporary dishes, you'll find something to love at these fantastic spots. Enjoy a culinary adventure in the vibrant city of San Francisco!

Piccolo Forno
⭐ 4.6 | 725 Columbus Ave, San Francisco, CA 94133 | $$

Serafina
⭐ 4.6 | 1701 Jones St, San Francisco, CA 94109 | $$$

Tavola Rustica
⭐ 4.6 | 215 Church St, San Francisco, CA 94114 | $$$

Pink Onion
⭐ 4.6 | 64 14th St, San Francisco, CA 94103 | $$

Collina
⭐ 4.6 | 1550 Hyde St, San Francisco, CA 94109 |

Figure 8. Rendered Restaurant Cards in Chat UI

The final user-facing result is a cleanly formatted list of restaurant cards. Each card includes the name, rating, address, pricing tier, and an image. The system also generates a human-readable summary recommendation at the top using GPT, creating a friendly and helpful chat experience.

5. Learning Outcomes

Through the development of the Yelp ChatDB system, I gained significant hands-on experience in integrating modern web development, database design, and natural language processing. From the frontend perspective, I deepened my understanding of how to build interactive, state-driven user interfaces using React and Tailwind CSS. I also learned how to manage asynchronous user input and effectively communicate with a Python-based backend using API calls.

On the backend side, I strengthened my knowledge of FastAPI and asynchronous request handling in Python. I became proficient in constructing RESTful endpoints that connect to both external APIs (Yelp Fusion API) and internal logic, such as GPT-based query parsing. I also gained practical experience in managing and querying a NoSQL database (MongoDB), including designing index strategies for performance and implementing cache logic to balance speed and freshness.

Furthermore, working with OpenAI's GPT model allowed me to explore how language models can be used to extract semantic meaning from unstructured queries and translate them into structured search filters. I learned how to design prompts and post-process model outputs to ensure alignment with backend query requirements.

This project also enhanced my ability to debug across a full-stack architecture, trace data flows end-to-end, and validate correctness at each stage—from raw user input, to GPT output, to query construction, to final presentation. Overall, I developed a clearer understanding of how real-world intelligent systems are engineered and deployed using modular and scalable components.

6. Challenges Faced

One of the primary challenges encountered during this project was achieving robust and consistent parsing of natural language queries using GPT. While large language models are powerful, they can produce outputs with slight variations or inconsistent field formatting, especially when user inputs are vague or contain compound conditions. To address this, I had to experiment with prompt engineering strategies, enforce strict output formats, and implement fallback mechanisms in the backend to detect missing fields or malformed JSON.

Another significant challenge was integrating and synchronizing multiple data sources—specifically, the Yelp Fusion API and MongoDB. Since Yelp responses are dynamic and MongoDB is used as a local cache, I needed to carefully design the logic that determines when to fetch fresh data from Yelp and when to return cached results. Ensuring data freshness without making redundant API calls required the implementation of smart query conditions and index optimization. The project also required handling asynchronous communication across the stack. Managing request timing, especially when calling the GPT and Yelp API in succession, occasionally led to race conditions or slow responses. To solve this, I leveraged FastAPI's native async capabilities and added timeout control and structured logging to identify bottlenecks in the pipeline.

On the frontend side, implementing a clean and responsive chat interface introduced challenges in state management. I had to design a system that supports multi-turn interactions while preserving context, user session data, and UI consistency across reloads. Integrating state logic with API response rendering required thoughtful component design in React and debugging tools like Redux DevTools.

Finally, ensuring data integrity during MongoDB exports and JSON serialization revealed edge cases—such as non-serializable ObjectId and datetime types—which I had to recursively sanitize before dumping collections to disk. These challenges pushed me to adopt more defensive programming practices and think end-to-end about data lifecycle and system robustness.

7. Conclusion

The Yelp Restaurant Finder ChatDB project successfully demonstrates how large language models can be integrated into a full-stack application to support natural language restaurant search. By combining a React-based chat interface, a FastAPI backend, a MongoDB cache, and GPT-powered intent parsing, the system allows users to interact conversationally while receiving accurate, real-time recommendations. Over the course of development, the prototype evolved into a robust pipeline capable of handling ambiguous queries, managing multi-source data retrieval, and presenting clean results in a user-friendly format. Building this system required solving practical challenges across NLP, API design, database structuring, and UI responsiveness. Ultimately, the project provided a valuable opportunity to apply modular system thinking and demonstrated the viability of using conversational agents to solve real-world information retrieval problems.

8. Future Scope

The Yelp ChatDB system offers a strong foundation for future enhancements in both functionality and scalability. One major area of improvement lies in the natural language understanding module. While the current implementation handles basic query patterns effectively, future versions could support more complex, multi-intent queries such as "Show me a vegan place in LA that's open late and good for groups." Fine-tuning the GPT model on restaurant-specific query logs or integrating a rule-based fallback system could improve output reliability. Additionally, adding support for temporal expressions, comparative queries, and preference learning would make the assistant more versatile and human-like.

Another promising direction is personalization and domain transfer. By introducing user authentication, the system could maintain individual profiles, enabling features like favorite cuisines, saved filters, or session memory. Enhanced caching strategies—such as TTL (time-to-live) indexing in MongoDB—would ensure Yelp data remains fresh without overloading the API. Furthermore, the existing architecture could be generalized for other domains like hotel bookings, movie recommendations, or travel itineraries. With its modular design and conversational interface, the ChatDB framework is well-positioned to evolve into a multi-purpose, intelligent assistant across verticals.