Case Study

Building a Nationwide Restaurant Menu Database and API

Introduction

When I started this project, I had exactly **zero budget** to work with. The only existing API that provided similar restaurant menu data was prohibitively expensive—charging **$0.02 per API call**. To put that in perspective, retrieving menu data for all **800,000 restaurants** individually would have cost **$16,000**—far beyond my resources.

I needed a way to **gather, structure, and serve** this massive dataset without relying on expensive third-party services. What followed was a deep dive into **scalable data collection, real-time API processing, and cost-effective infrastructure**, resulting in a fully functional, self-sustaining solution that could provide the same data at a fraction of the cost.

This case study outlines the **technical challenges, solutions, and optimizations** that made it possible to build a **nationwide restaurant menu database and API from scratch**.

Problem Statement

Despite the vast number of restaurants and menu items across the country, no centralized, **searchable database** for restaurant menus existed. Users—including businesses, developers, and consumers—had no simple way to look up detailed menu information at scale.

The primary challenges included:

- **Data Volume**: Managing and processing information from **800,000+ restaurants** and **60,000,000+ individual menu items**.
- **Real-Time Updates**: Keeping the database fresh by continuously checking for **new entries and updates**.
- **Efficient Querying**: Delivering rapid search results despite the sheer scale of the dataset.
- **Cost Efficiency**: Avoiding high infrastructure expenses that would make the project unsustainable.

Technical Implementation

Back End & Data Collection

The back end was built using **Node.js** and leveraged a **distributed network of multiple servers**. Instead of relying on a single expensive API, I developed a **custom web scraping and data aggregation system** that could efficiently extract and structure restaurant menu data.

Each server was responsible for scraping a portion of the data, enabling the system to **scale efficiently**. Over time, I optimized the process to reduce the **data collection cycle** to just **6 hours** for all **800,000 restaurants**, ensuring that the dataset remained fresh by continuously checking for new and updated entries.

API & Integrations

At the heart of this project was a **high-performance REST API**, designed to:

- **Handle Large-Scale Data**: Processing and serving menu data efficiently to thousands of potential users.
- **Manage Access & Monetization**: Integrating with **Stripe** to handle subscriptions and API quotas.
- **Optimize Performance**: Maintaining **sub-150ms response times** for most requests, even when handling **massive amounts of data**.

Key API Features:

- **Proprietary Request Tracking**: A custom system allowed for **tiered API requests**, ensuring fair usage across different subscription levels.
- **Stripe Integration**: Users could **upgrade, downgrade, or cancel** their subscriptions seamlessly through Stripe.
- **Performance Optimization**: Even for **data-intensive queries**, response times remained **under 400ms**.

Front End & User Interface

The front end was built with **React**, providing a seamless dashboard where users could:

- **Manage their subscriptions** (upgrade, downgrade, or cancel).
- **Track every API request** for transparency and usage insights.
- **Search and filter menu data in real-time**, leveraging the speed of Elasticsearch.

Data Storage & Search Optimization

The data was stored in **Elasticsearch**, which enabled:

- **Ultra-fast search capabilities**, even with **tens of millions** of records.
- **Real-time indexing**, ensuring new data was instantly searchable.
- **Optimized querying**, reducing infrastructure load while maintaining high-speed results.

Challenges & Innovative Solutions

Handling such a **massive dataset** posed unique challenges:

Challenge	Solution
Scalability – Managing **millions of records** across multiple servers.	**Implemented a distributed data scraping approach with workload allocation.**
Data Freshness – Keeping menus **up to date** in near real-time.	**Built a continuous refresh cycle, ensuring data was always current.**
Query Efficiency – Ensuring **fast search results** despite large data volumes.	**Leveraged Elasticsearch indexing for rapid retrieval speeds.**
Cost Constraints – Competing with expensive third-party APIs.	**Developed an independent data collection pipeline, avoiding external API costs.**

API Process & Monetization Strategy

The API was the **core** of this project, acting as the gateway to the data. Given the project's initial **zero-budget** nature, monetization was key to sustaining long-term viability.

Subscription & Access Management

- **Stripe Integration** ensured seamless **subscription handling** and access control.
- **Tiered Pricing** allowed different levels of data access, balancing affordability with sustainability.
- **Request Tracking System** enabled efficient quota management without overloading infrastructure.

Performance Optimization

Even with **millions of menu items**, the API consistently delivered:

- **Sub-150ms response times** for standard queries.

- **Sub-400ms response times** for high-load, complex queries.
- **Minimal downtime**, thanks to smart caching and load balancing.

Performance Metrics & Key Outcomes

The project achieved significant milestones in both **technical efficiency** and **cost savings**:

Metric	Outcome
Restaurants Processed	**800,000+**
Menu Items Indexed	**60,000,000+**
Data Refresh Rate	**Every 6 hours**
API Performance	**<150ms for most queries, <400ms for complex queries**
Infrastructure Costs	**90% lower than third-party API alternatives**

Despite the **technical success**, the project faced **financial sustainability challenges**—maintaining such a **large-scale infrastructure** required **significant resources**. To address this, I **scaled down the refresh cycle** and optimized costs, ensuring a **leaner, more sustainable model** for long-term use.

Conclusion & Future Directions

This project was both a **technical challenge** and a **lesson in cost-effective data management**. It reinforced my ability to:

- **Develop scalable, high-performance APIs** for large datasets.
- **Optimize data collection and storage** while minimizing costs.
- **Manage real-world constraints** by balancing infrastructure, speed, and affordability.

Next Steps:

- **Refining data refresh cycles** to further **reduce costs** while maintaining accuracy.
- **Exploring alternative cloud solutions** to optimize **infrastructure spending**.
- **Opening the API to new business models**, including **enterprise licensing and integrations**.

By combining **technical efficiency with business practicality**, this project showcases my ability to **build, scale, and sustain** data-driven solutions in the real world.

Final Thoughts

Starting with no budget and no feasible alternative, this project proved that **innovation and efficiency** can overcome even the most daunting challenges. It’s a testament to the power of **technical problem-solving** and the **importance of cost-conscious development** in the modern data economy.