CS 490 - P1 Report

GrubFindr

Date: February 6, 2026

Team:
Abyan Haq - Project Manager - abyan.haq@uwaterloo.ca
Andre Slavescu - System Architect - aslavesc@uwaterloo.ca
Bilal Nasar - Communication Officer - bnasar@uwaterloo.ca
Jeffrey Qingjie Xuzhang - Information Officer - jqxuzhang@uwaterloo.ca
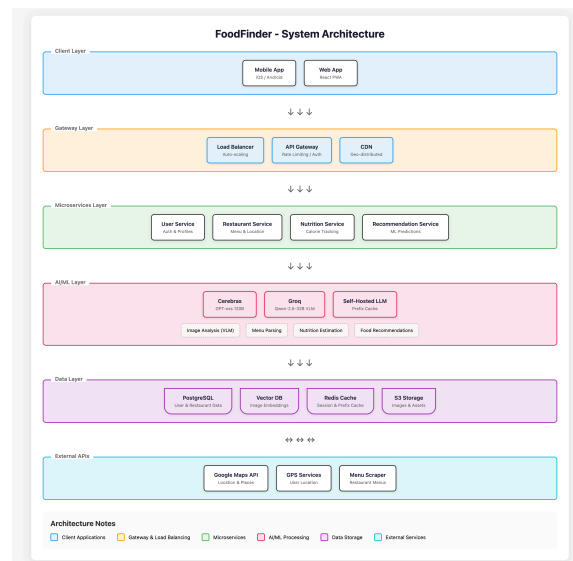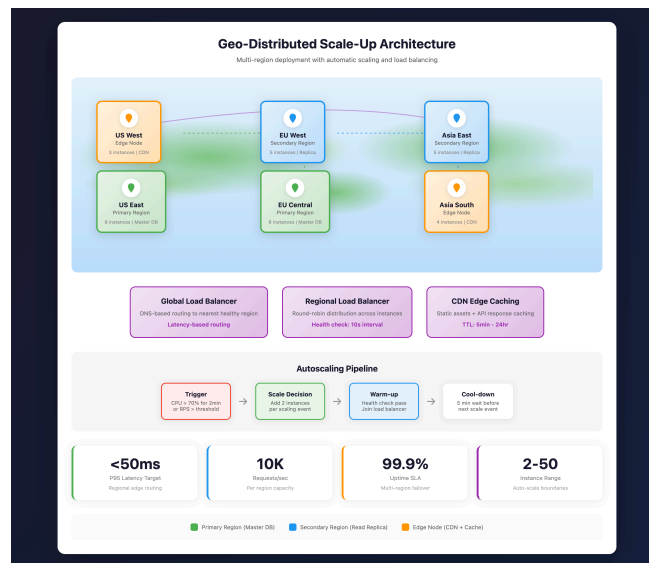
Introduction

Modern meal hunting has become more difficult than ever, due to a frustrating paradox: on one hand, people now have have access to more options and details than ever before, but on the other hand, it has now become harder than ever to choose what to eat. The majority of diners now find themselves trapped in a cycle of 'analysis, paralysis,' going through google maps, yelp, tiktok and every other app that recommends food in any capacity, yet still not being able to make a decision. This decision, which should have been simple and pleasant, is now becoming a task that people would rather avoid, as there is tons of fragmented data, making it seem impossible to get a straightforward and reliable answer.

GrubFindr came into being to clarify this confusion in the digital world. Instead of making the market even more complicated by launching yet another platform, the solution is like a clever filter which brings rationale to the discovery process once again. Thanks to the real-time data consolidation into a single, easy-to-use interface, GrubFindr makes the search a less troublesome task so that users don't have to spend so much time scrolling and can enjoy their meal at a restaurant. Users enter their location, budget, dietary requirements, and nutritional goals, which is then used by Large Language Models and Vision Language Models to provide a single, context based dish that users will undoubtedly enjoy. This report presents the architectural design of GrubFindr, covering the software architecture, data model, business model, and market analysis.

Software Architecture

GrubFindr follows a layered microservices architecture deployed across multiple geographic regions. The system is composed of six layers, each with distinct responsibilities. The full architecture and scale-up diagrams are shown below.

Geo-Distributed Scale-Up Architecture
Multi-region deployment with automatic scaling and load balancing

Client Layer: Users interact with GrubFindr through a native mobile app (iOS/Android) and a React-based progressive web app (PWA). Both clients communicate with the backend exclusively through the gateway layer.

Gateway Layer: A global load balancer routes traffic to the nearest healthy region using latency-based DNS routing. An API gateway handles authentication, rate limiting, and request validation. A geo-distributed CDN caches static assets and frequently accessed API responses with TTLs ranging from 5 minutes to 24 hours.

Microservices Layer: Four core services handle domain logic. The User Service manages authentication, profiles, and dietary preferences. The Restaurant Service manages restaurant metadata, menus, and geospatial location queries. The Nutrition Service tracks calorie and macronutrient data for menu items, sourcing data from VLM analysis and manual entry. The Recommendation Service orchestrates ML predictions to generate personalized dish-level suggestions based on user context.

AI/ML Layer: This layer powers the core intelligence of GrubFindr. Cerebras (Llama 3.3 70B) handles fast LLM inference for natural language queries and conversational recommendations. Groq (Llama 3.2 90B Vision) serves as the Vision Language Model for analyzing food images, extracting menu items from photos, and estimating nutritional content. A self-hosted LLM with per-user prefix caching maintains conversational context across sessions, enabling personalized recommendations that improve over time. An image analysis pipeline uses cosine similarity over 512-dimensional embeddings to match user-uploaded food photos against known menu items.

Data Layer: PostgreSQL serves as the primary relational database for users, restaurants, menu items, nutrition data, and meal logs, with PostGIS enabled for geospatial queries. A Vector Database (HNSW-indexed) stores 512-dimensional image embeddings for fast similarity search. Redis provides in-memory caching for user sessions, LLM prefix caches, API responses, and rate limiting counters. S3 object storage holds user-uploaded food images and restaurant menu photos with CDN integration.

External Services and Integrations: GrubFindr depends on the Google Maps API for location and place data, GPS services for real-time user positioning, and a menu scraper for ingesting restaurant menu data. GrubFindr provides services to external systems through referral and commission integrations: when a user decides on a dish, they can be redirected to partner platforms (e.g., Uber Eats, OpenTable) for ordering or reservations, with GrubFindr earning a commission on each conversion.

Scalability: The system is deployed across six regions (US West, US East, EU West, EU Central, Asia South, Asia East) with primary regions handling write operations and secondary regions serving read replicas. Autoscaling triggers when CPU exceeds 70% for 2 minutes or requests per second exceed the regional threshold, adding 2 instances per scaling event with a 5-minute cooldown. Target metrics include sub-50ms P95 latency, 10K requests/sec per region, and 99.9% uptime SLA.

Business Model

GrubFindr monetizes decision-making at the dish level by inserting itself at the exact moment a user chooses what to eat. Revenue comes from users who want better recommendations and restaurants that want higher-intent visibility.

Revenue Streams

1) Freemium subscription (B2C): Users access basic dish recommendations for free. A paid tier unlocks unlimited queries, detailed nutrition breakdowns, and meal personalization over time. This creates recurring, high-margin revenue with low incremental cost.

2) Sponsored placement / ranked results (B2B): Restaurants pay to promote specific dishes so they appear earlier in recommendation results, similar to paid search ranking. Placement is constrained by relevance, distance, and dietary fit, preserving user trust while allowing restaurants to capture high-intent demand.

3) Transaction and referral commissions (B2B2C): When a user clicks through to book a table or place an order via an integrated partner (delivery or reservations), the platform earns a commission. This ties monetization directly to conversion rather than impressions.

TAM

The addressable TAM is a rather conservative $1B dollars, this is based on a total potential of 100M users across Canada and the US, as it's reported that about 1 in 4 people from these countries use delivery services. Now the ARPU is lower as the service is an impression based one as opposed to something like uber eats where you actually place through the app.

Cost Basis vs. Margins

Our cost base is front-loaded and dominated by fixed engineering and infrastructure spend, including core mobile and backend development, data ingestion, and AI integration. Early on, this results in a steady-state run rate of roughly $1.0 to $1.5 million per year, driven primarily by engineering talent, and then with variable cloud, inference, and API costs scaling alongside usage. Importantly, while these variable costs grow with users, the cost per user declines over time due to batching, caching, and model efficiency, allowing margins to expand materially as the platform reaches scale.

Market Liquidity & 5 Year Scalability

The market is both highly liquid and structurally scalable, with over 60% of restaurant discovery controlled by the top 5 global players such as Uber Eats, Zomato, Doorsash, etc, which historically drives acquisition-led expansion rather than greenfield builds. Recent large transactions, such as the 2.9 billion pound acquisition of deliveroo suggest an acquisition driven market, and underlying demand is supported by strong secular growth, including a 28% CAGR in AI-driven meal planning. From a market scalability perspective, the platform expands horizontally across geographies with minimal incremental cost, allowing revenue growth over a five-year horizon to be driven primarily by user adoption and margin expansion rather than proportional increases in operating expense.

Competitor Landscape

Currently, the food discovery and recommendation space is currently dominated by platforms like Google Maps, Yelp and a newcomer, Beli. These apps all serve a different decision making criteria. Google Maps focuses on restaurant discovery filtered by distance and popularity resulting in similar recommendations for all users. Yelp focuses on restaurant discovery on review and rating from the generalization and finally Beli targets niche users from your social circle. The issues with these three competitors is that they have low personalization depth and lack awareness of intent. This is where GrubFindr differentiates itself by enabling dish-level decision-making with personalized recommendations based on dietary preferences, budget, time and location. This positions GrubFindr as an individualized dish-level decision engine ready for every meal.

Target Demographic

GrubFindr is designed for mobile-savvy urban consumers aged 18-35, who want to find good food fast, without overthinking. They value convenient meals that align with their health goals that include calories, macros and diet balance. Their primary pain points include decision fatigue due to option overload, lack of nutritional transparency on current menus and overly generic filters. GrubFindr solves these issues by recommending on the dish-level and utilizing personalized suggestions and filters to provide nutritional information from real up-to-date menus.

Market Opportunities

We want GrubFindr to be the decision-first entry point in the meal discovery process, encouraging our users to use GrubFindr before using navigation or delivery applications. By integrating APIs from other services like Google Maps or Uber Eats, GrubFindr can be the point-of-entry and guide users after a decision has been made. Outside of just recommendations, we want GrubFindr to expand into a daily decision tool. This can help us increase engagement by supporting users before, during and after meals. This also opens our horizons to adjacent markets such as meal planning, nutrition and wellness recommendations. Finally, we will enable continuous improvement over time with a 2-user system through data collection and user feedback to refine recommendations and data over time.

Data Model

GrubFindr's data model spans four main storage systems, each chosen for it's particular strengths. The full ER diagram is shown below.
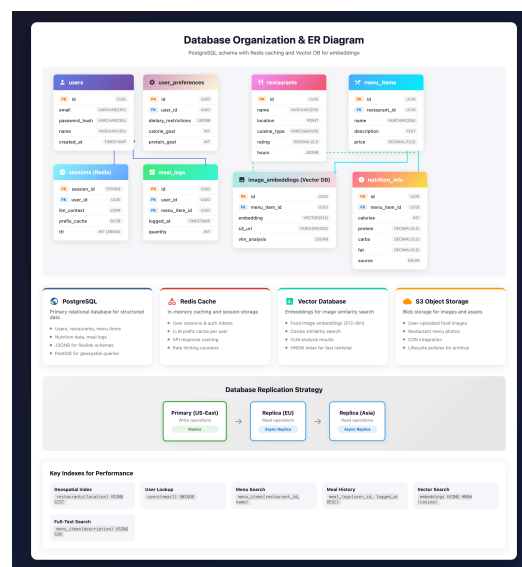
The PostgreSQL schema contains the core relational entities. The Users table stores account credentials and profile information, linked one-to-one with a User Preferences table that holds

dietary restrictions (stored as JSONB for flexibility across vegan, halal, gluten-free, etc.) along with daily calorie and protein goals. The Restaurants table stores name, cuisine type, operating hours (JSONB), rating, and geographic location as a PostGIS POINT for efficient proximity queries. Each restaurant has many Menu Items, which contain name, description, and price. Each menu item links one-to-one with a Nutrition Info record storing calories, protein, carbs, fat, and the data source (VLM-estimated or manually verified). A Meal Logs table tracks which users consumed which menu items and when, supporting dietary history and personalization.

Sessions are stored in Redis rather than PostgreSQL for low-latency access. Each session record includes the user ID, LLM conversational context, a prefix cache blob for the self-hosted model, and a TTL of 3600 seconds. Redis also serves as the caching layer for API responses and rate limiting counters.

Image Embeddings are stored in a dedicated Vector Database. Each record contains a 512-dimensional embedding vector generated by the VLM, a reference to the source image in S3, and the VLM's structured analysis of the food item. The vector index uses HNSW for fast cosine similarity search, enabling users to photograph a dish and find matching menu items across restaurants.

Key indexes include a GIST spatial index on restaurant locations, a unique index on user emails, a composite index on menu items by restaurant and name, a descending time index on meal logs for fast history retrieval, a GIN index for full-text search on menu item descriptions, and the HNSW vector index on image embeddings.

Database replication follows a primary-replica model: a primary instance in US-East handles all write operations, with asynchronous read replicas in EU and Asia to minimize latency for global users.

Conclusion

GrubFindr addresses the growing problem of meal decision fatigue by consolidating fragmented restaurant and dish data into a single, personalized recommendation engine powered by LLMs and VLMs. The platform monetizes through a freemium subscription model, sponsored dish placements, and referral commissions. By focusing on dish-level personalization rather than generic restaurant discovery, GrubFindr differentiates itself from incumbents like Google Maps, Yelp, and Beli, positioning itself as the decision-first entry point in the meal discovery process. The platform is well-suited to capture high-intent demand in a market increasingly driven by AI-powered personalization. Architecturally, the system is built on a layered microservices design with dedicated AI/ML inference providers (Cerebras for LLM, Groq for VLM), a polyglot data layer spanning PostgreSQL, Redis, a Vector Database, and S3, and a geo-distributed multi-region deployment with autoscaling to meet sub-50ms latency and 99.9% uptime targets at scale.

Appendix A: Project Diary

Feb 2nd – Team Meeting
Attendees: Abyan Haq, Andre Slavescu, Bilal Nasar, Jeffrey Qingjie Xuzhang

Discussion:

- Discussed project scope and brainstormed app concepts
- Decided on GrubFindr as the project direction
- Assigned initial research areas

Contributions:

- Bilal: Created group chat, set up shared Google Drive, drafted project timeline
- Andre: Researched existing food recommendation APIs and LLM integration feasibility
- Jeffrey: Conducted initial competitor analysis (Google Maps, Yelp, Beli)
- Abyan: Narrowed down problem scope and crafted a story explaining why GrubFindr

To-Dos (due Feb 5th):

- Bilal: Research findings and define business model structure
- Andre: Create system architecture diagram and ER model
- Abyan: Start working on slides and set up the problem statements
- Jeffrey: Conduct market analysis and find differentiation from competitors

To-Dos for Later Phases (due Feb 20th):

- Bilal: Dive deeper into system costs and how costs will scale with more usage
- Andre: Build mock prototypes with digital tools such as Figma / Claude
- Abyan: Work on creating a live product demo to show others how to use the app
- Jeffrey: Start working on the 5-page TCO report and structure with initial findings and research