Team Information

Instructions: Fill in basic information about your team. This helps us track submissions.

Team Name: Local Bites

Team Members (Name, UCINetID):

• Linyu Chen, linyuic

• Diego Nunez, diegon1

• Alex Gonzalez, alexg1

PROGRESS REPORT CONTENT

1. Executive Summary

Instructions: In 2–4 sentences, summarize the problem, the target user/context, and what your system outputs. Example: [In one paragraph, what your system does and why it is useful.]

Choosing a place to eat lunch is a simple yet frustrating task for many individuals with hectic schedules, particularly college students and working professionals. Finding an affordable, convenient, and satisfying lunch option often requires significant prior knowledge of the surrounding area or time-consuming research. Our system takes in users' real-time information, such as time and location, to recommend various lunch spots that are nearby, open, and match their personal preferences. The system removes the stress of deciding on lunch from people who already have a lot on their plates.

2. System Overview (what you built)

2.1 Project Title

Instructions: Provide your project title (8–12 words).

Smart Lunch Restaurant Search Using Location and Time

2.2 Target Users and Scenario

Instructions: Describe who the user is and one concrete scenario (context) where the system is used. (~80–120 words)

The target users of this system are busy college students and working professionals who have limited time to decide where to eat lunch. For example, a college student finishing a lecture at noon only has 30 minutes before their next class and needs to find a nearby place to eat quickly. Instead of searching manually through many options, the student opens the app, which uses their current location and time to generate a ranked list of nearby restaurants that are open, affordable, and reachable within their available time. The system helps the user quickly select a practical lunch option, reducing decision time and improving convenience.

2.3 User Need → System Response

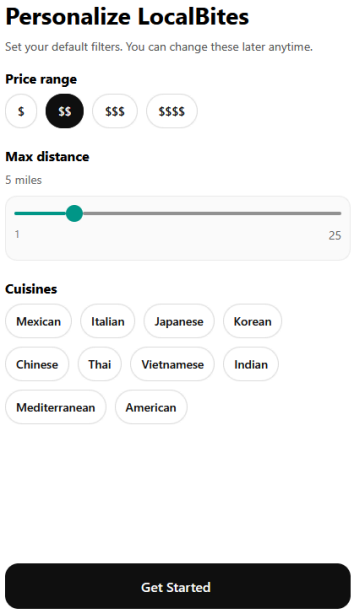
Instructions: Describe what triggers a recommendation/search and what the system returns (top-k) plus a short 'why'. (~80–120 words)

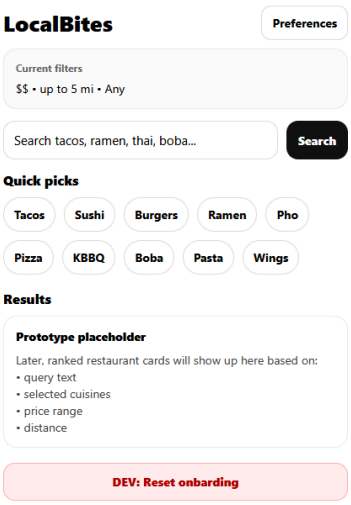
When a user needs to quickly find a place to eat lunch, they open the app and allow access to their location or enter a search query or cuisine preference. This action triggers the recommendation system to evaluate nearby restaurants using the user's current location, query text, and optional cuisine filter. The system computes the distance between the user and each restaurant, applies any preference or intent signals, and returns a ranked top-k list of restaurants. Each result includes the restaurant name, distance, and relevant metadata. The recommendations prioritize nearby and relevant options, helping the user quickly identify realistic lunch choices without manually searching through many possibilities.

2.4 Current UI/UX (status)

Instructions: Briefly describe your current user-facing UI. Include screenshots in the report if possible.

The current LocalBites prototype includes two main user-facing screens: a Personalization screen and a Search screen. When users first open the app, they are shown the Personalization screen, where they can configure their default preferences, including price range, maximum distance, and preferred cuisines. These options are presented using selectable buttons, a distance slider, and cuisine selection chips. After selecting their preferences users press a "Get Started" button, which saves their settings and transitions them to the Search screen.





The Search screen serves as the primary user interface for interacting with the system. Users can enter food-related queries using a search bar, view their current preferences, and quickly initiate searches using suggestion chips. Users can also return to the Personalization screen to update their preferences. While restaurant results are current placeholders, the interface fully supports onboarding, navigation, and preference storage.

3. Data & Ingestion

3.1 Data Sources

Instructions: List your data sources and what each provides (APIs, datasets, user input, logs).

The primary data source for the system is OpenStreetMap. We queried for restaurants within Orange County. The dataset provides structured geographic and metadata information including name, latitude and longitude coordinates, address fields, cuisine, and hours of operation.

Besides restaurant data, the system collects real-time user input through a mobile interface, including the user's GPS location, current time, and optional cuisine preference. The inputs as contextual information that influence ranking.

3.2 Collection / Ingestion Pipeline

Instructions: Explain how data is obtained and stored. Include the corpus size (# items) and the update strategy (one-time, periodic, or on-demand).

Restaurant data is downloaded from OpenStreetMap in GeoJson format and processed through a Python script into a CSV that keeps restaurants that provide all the required fields to rank, filtering out incomplete entries. The CSV is structured for quick loading and ranking and contains only relevant information for our ranking. The current corpus size is approximately 500 restaurants entries throughout all of Orange County. The ingestion process is currently a one-time process as a dataset stored locally.

3.3 Representation

Instructions: Describe what you extract (fields, text, metadata, features) and any preprocessing.

Each restaurant is represented as a structured record containing name, latitude and longitude coordinates, cuisine type, address, opening hours, phone number, and website. For ranking and retrieval purposes, all entries must have at least a name, coordinates, and cuisine type. Restaurants without these required fields are removed during preprocessing through our GEOJson to CSV Python script.

4. Personal Model & Context

4.1 Personal Model (implemented)

Instructions: Describe what you store about the user (goals, preferences, history, feedback) and how it affects results.

Currently, we store search interaction history rather than a full authenticated user profile. Each time a user sends a request to /recommend, we log a structured search event (JSONL format). Each record includes timestamp, location, cuisine(if provided), q(text search query), and result metadata (number of results returned, preview of top results). This allows us to build a behavioral profile over time without requiring login or identity management.

Currently, personalization affects the results in two direct ways. First, the current ranking system supports cuisine filtering in rank_restaurants. If a cuisine is provided, only restaurants matching that cuisine are returned. Second, the search query affects ranking by boosting restaurants whose name matches query tokens or the cuisine matches the query tokens.

4.2 Context (implemented and planned)

Instructions: Identify contextual signals that affect rankings and recommendations. Explicit-only is acceptable, but stronger projects include at least one implicit signal.

The system currently implements location, cuisine, and a search explicitly. Location is first computed using geopy in a function. This ensures that results are locally relevant and the search automatically adapts to the user position. The current system also allows users to freely enter queries that reflect user intent. For example, if the user searches for "ramen", the system prioritizes ramen restaurants. The intent scoring shifts ranking before distance sorting. Additionally, the cuisine filter filters the ranking based on the user's preference. This is an explicit preference.

Additionally, we plan to implement a time of day that automatically fetches the current time and recommends restaurants that are open currently. This can be further enhanced by prioritizing different restaurants based on the time and day. For example, for lunch (11am - 2pm) the system can prioritize quick service restaurants while weekends can prioritize sit-down restaurants. Additionally, we would also implement a way to learn the user's preferences based on stored history of what they search for and their preferred cuisine. For example if a user repeatedly searches for "boba", the system can pre-weight Asian cafes even without an explicit query search.

5. Search/Recommendation & Ranking

5.1 Items or Actions Being Ranked

Instructions: Define what the system ranks (documents/resources/actions/options).

The current system ranks restaurants across Orange County as possible lunch options. Each restaurant in the dataset is treated as a candidate and evaluated based on distance, cuisine, and a query. The system produces a ranked list of the top lunch locations based on these factors. The final output is a top-k list of restaurant recommendations.

5.2 Baseline Ranking

Instructions: Describe your current baseline approach (e.g., TF-IDF, BM25, embeddings, rules, hybrid). Keep it high-level but specific.

The current baseline ranking model is a rule-based geographic proximity approach. Restaurants are ranked using the Haversine distance between the user's latitude and longitude and each restaurant's coordinates. Candidates are sorted in ascending order of distance, prioritizing the closest options.

After sorting by proximity, the system applies a cuisine filter when a user specifies a preferred cuisine. In this baseline version, cuisine preference alongside a optional query is given a relevance score. This relevance score with the distance determines final ranking.

Beyond the baseline, we plan to incorporate additional contextual signals. Restaurants will be filtered based on real-time open status using opening hours data. Price range will become considered as weighted ranking signal alongside proximity, cuisine and query.

5.3 How Personal Model + Context Affect Ranking

Instructions: Explain how you combine relevance with user/context signals (e.g., reranking, filtering, weighting).

The system incorporates contextual and personal signals to refine ranking. Proximity ensures the recommendation is a realistically reachable option. Time of day will hard filtered to ensure every recommendation is a valid choice. Cuisine preferences and a query search provides users the ability to customize their results through weighting scoring. The system combines these signals to produce practical lunch recommendations.

6. Evaluation & Results (so far)

Instructions: Provide evidence that the system is working. Include 2–3 example queries/contexts with the top results and brief explanations.

Optional: Include lightweight metrics (precision@k on a small labeled set, latency, or user feedback).

Example 1: Nearby search (no cuisine, no query). Context, the user wants "whatever is closest"

Payload: { "lat": 33.64931, "lon": -117.84638 }

Top Results (sample):

1) **Blaze Pizza — ~0.42 mi**
2) **Eureka! — ~0.42 mi**
3) **Northern Cafe — ~0.44 mi**
4) **Gogi — ~0.45 mi**
5) **Fast Hotpot — ~0.45 mi**

Why: with no preference signals provided, the system defaults to distance-based ranking, so the closest restaurants appear first.

Example 2: Explicit cuisine preference (Mexican). Context: user wants Mexican food nearby.

Payload: { "lat": 33.64931, "lon": -117.84638, "cuisine": "mexican" }

Top results (sample):

1) **Javi's — ~2.17 mi**
2) **Sharky's Woodfired Mexican Grill — ~3.08 mi**
3) **Casa Del Sol Cucina Mexicana — ~3.55 mi**
4) **Sharky's Woodfired Mexican Grill (Irvine) — ~3.97 mi**
5) **Amorelia Mexican Cafe — ~4.10 mi**

Why: the cuisine filter removes non-Mexican restaurants first, then the remaining set is ranked by distance.

Example 3: Intent-based search using free-text query (q = "ramen"). Context: The user wants ramen, not just nearby restaurants.

Payload: { "lat": 33.64931, "lon": -117.84638, "q": "ramen" }

Top results (sample):

1) **HiroNori Craft Ramen — ~1.95 mi**
2) **Kitakata Ramen Ban Nai — ~3.43 mi**
3) **Rakkan Ramen — ~3.51 mi**
4) **Kashiwa Ramen — ~4.59 mi**
5) **Silverlake Ramen — ~5.79 mi**

Why: the query signal ("ramen") increases a restaurant's relevance score when the token appears in the restaurant name or cuisine field, causing ramen-related restaurants to rise above closer but irrelevant places. Distance is still used as a tie-breaker within similarly relevant matches.

7. Demo & Video Presentation

Requirement: Submit a link to a ~3-minute progress presentation video (unlisted YouTube or shared Drive link). Video should cover: (1) problem & user, (2) approach, (3) current status, (4) next steps.

Demo: Optional but strongly encouraged if you have a running UI. Show at least one end-to-end interaction. Demo script example (fill in):

- Step 1: [Open app / set user & context]
- Step 2: [Scenario A → show top-k + explanation]
- Step 3: [Change context/user preference → show ranking change]
- Step 4: [Scenario B → show second example]

Demo/Presentation Link:

video1412661073.mp4

8. Team Contributions & Plan

8.1 Roles and Responsibilities

Instructions: Briefly list what each team member contributed since the proposal.

Linyu Chen: Keyword Based Search, Cuisine Preference Filtering, Relevance Scoring

Diego Nunez: Data ingestion, Baseline ranking on distance, API

Alex Gonzalez: UI creation and development

8.2 Remaining Work and Milestones

Instructions: List remaining tasks to reach the final demo. Include owner + target date.

Task	Description	Owner	Target Date
Connect frontend to backend ranking	**Connect the React Native frontend search screen to the backend API or ranking module so that real restaurant results are displayed instead of placeholders**	**All group members**	**Feb 28, 2026**
Enable preference editing and persistence	**Ensure preferences update correctly and affect future rankings without requiring app restart. Validate AsyncStorage persistence and reload behavior**	**Alex Gonzalez**	**Feb 28, 2026**
Display ranked results in UI	**Implement a results list component showing restaurant name, cuisine, price, rating, and distance in ranked order**	**Alex Gonzalez**	**Feb 28, 2026**
UI polish and usability improvements	**Improve layout, spacing, and visual clarity. Add loading states, empty states, and improve overall usability**	**Alex Gonzalez**	**March 4, 2026**
Improve relevance scoring	**Transition relevance scoring to include distance and price range and compute a comprehensive "match" score. This score will become the source of sorting the final rankings.**	**Diego Nunez**	**Mar 5, 2026**
Include a max distance parameter to ranking	**Provide a parameter for max distance in ranking algorithm to filter out restaurants are too far for user to reach**	**Diego Nunez**	**Feb 22, 2026**
Parse opening hours from dataset	**Convert opening hours from dataset to Python datetime to be implemented into ranking**	**Diego Nunez**	**Feb 28, 2026**
Improve query tokenization	**Still show results whenever the user mistypes their intended search. For example, when the user types "taacos" the**	**Linyu Chen**	**Feb 20th, 2026**

	system will still return results with "tacos"		
User history	**Builds a detailed user history and then recommends restaurants based on what they have searched in the past.**	**Linyu Chen**	**Mar 5th, 2026**
Time based recommendations	**Implement a system feature where it will recommend different results depending on what the time of day that the user is searching. For example, if the user is searching during a weekday, they will more likely get results for casual and quick takeout restaurants while weekends will give more results to sitdown restaurants.**	**Linyu Chen**	**Feb 26th, 2026**
Improve dataset to include more details	**Add additional info(opening times, price, restaurant type) to our current dataset. This is for a more robust ranking system so it allows us to better filter the search results**	**Linyu Chen, Diego Nunez**	**Feb 23th, 2026**

9. Appendix: Repo / How to Run

Instructions: Provide the repository link, commit hash/tag, and the minimum steps to run the system. Example: requirements, setup commands, how to run ingestion (if any), and how to start UI/backend.

Repository Link:

https://github.com/diegon14/LocalBites.git

Requirements:

- **Python 3.10 or newer**
- **pip package manager**

Install dependencies using:

pip install -r requirements.txt

Running the Backend:

1. **Navigate to backend folder**
2. **Run api.py to start api server**

3. **Run test_api.py to view sample response**

Running the UI:

1. **Navigate to LocalBites folder**
2. **Install dependencies (requires Node.js)**

```
npm install
```

3. **Start the app**

```
npx expo start
```

In the output, you'll find options to open the app in

- **Development build**
- **Android emulator**
- **iOS simulator**
- **Expo go**
- **Web**

Select your desired preference

(Note: UI and Backend are not connected yet)