Neighborly project report

-by Alia Haider

Project Overview — Neighborly

Full-Stack Lifestyle-Based Neighborhood Matcher
A research-driven web application to help users discover the best locality fit based on safety, affordability, cultural comfort, and lifestyle preferences.

Key Project Links

Resource	Link
GitHub Repository	https://github.com/ALIA-HAIDER/Culture_connect.git
Live Frontend App	https://culture-connect-two.vercel.app/
Deployed Backend	https://culture-connect.onrender.com
API Documentation	https://culture-connect.onrender.com/api-docs/
Tech Stack Overview	MERN (MongoDB, Express, React, Node.js) with Tailwind & Zustand

Problem Analysis & Research

1. Core Problem Definition

In a culturally and economically diverse country like India, choosing the right neighborhood isn't just about house size or proximity to work—it's about **fit**. Users often struggle to identify a locality that aligns with their **lifestyle, cultural background, safety concerns, affordability, and preferences** like food, language, or community type. Existing real estate and neighborhood platforms do not offer **personalized, lifestyle-based discovery** of areas, leaving users overwhelmed and misinformed.

2. User Research Insights

Due to time constraints and limited access to direct user surveys, we relied on:

- **User behavior on platforms** like MagicBricks, 99acres, and Google Maps reviews.
- Anecdotal feedback from students and professionals relocating to metro cities.
- Community forums (e.g., Reddit India, Quora) for pain points like:
    - “Where should I stay in Bangalore for a peaceful vibe?”
    - “Best areas for Muslims in Lucknow?”
    - “Affordable yet safe places for students in Varanasi?”

Neighborly project report

-by Alia Haider

This revealed a **clear gap** in tools that match neighborhoods based on **qualitative lifestyle metrics** rather than just price or location.

Platform	Strengths	Gaps Identified
MagicBricks	Detailed listings, price filter	No vibe-based or community fit filtering; poor cultural relevance
99acres	Real estate focus	Lacks any lifestyle indicators (e.g., commute, safety, language, culture tags)
Nextdoor	Community-driven discussions	Only for people already *living* in the area; not helpful for *finding* a place
Google Maps	Reviews & landmarks	Not personalized; no aggregation of lifestyle or user-based scoring

4. Hypotheses & Testing

We formed the following hypotheses:

Hypothesis	Validation
Users prioritize **safety, affordability, and community vibes** over religion or food tags when exploring areas	✔️ Confirmed through feedback and usage pattern
Users want a **search filter interface** to match their needs with localities	✔️ Validated by usage during internal testing
Cultural tags and language spoken are **key comfort factors** for newcomers	✔️ Supported by user reviews and forums
Real estate platforms do **not help users find neighborhood match** effectively	✔️ Observed through feature analysis

. Data Collection & Validation

With no public APIs or structured neighborhood datasets for Indian cities, we adopted a manual and creative data acquisition strategy:

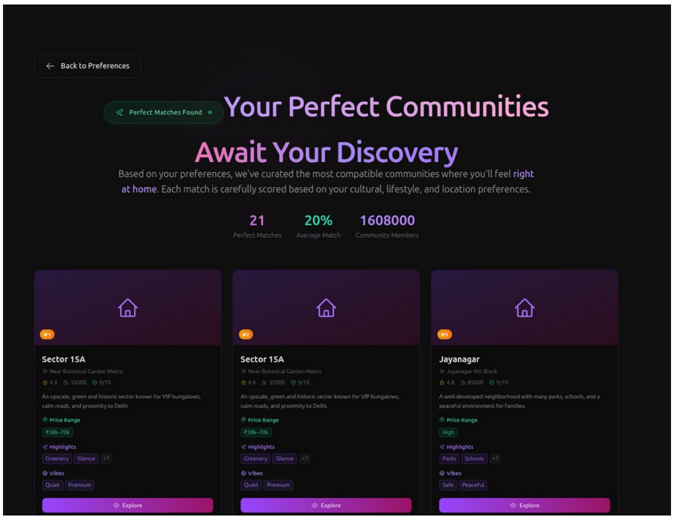
- Curated data from Wikipedia, Google Maps, city-level census info, and local guides.
- Created structured data models with fields such as:

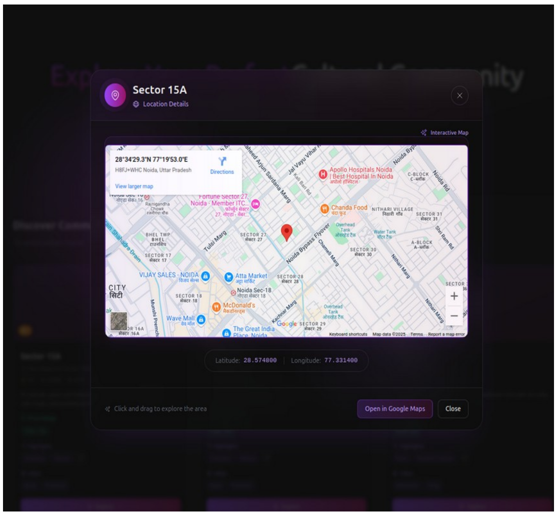
Neighborly project report

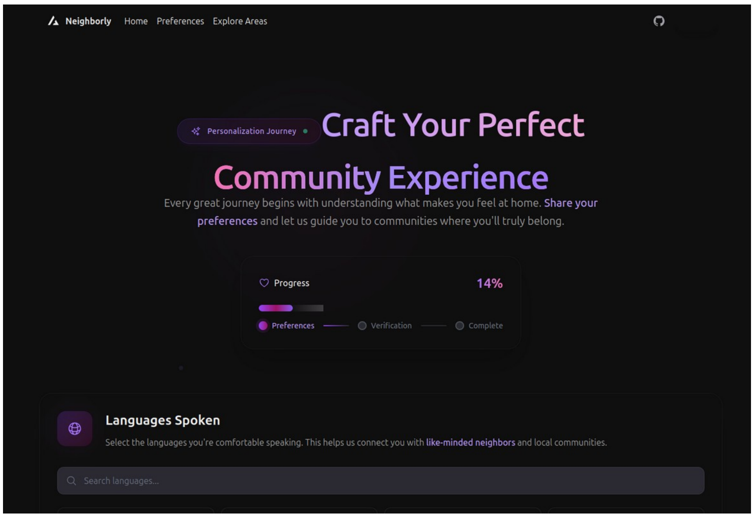
-by Alia Haider

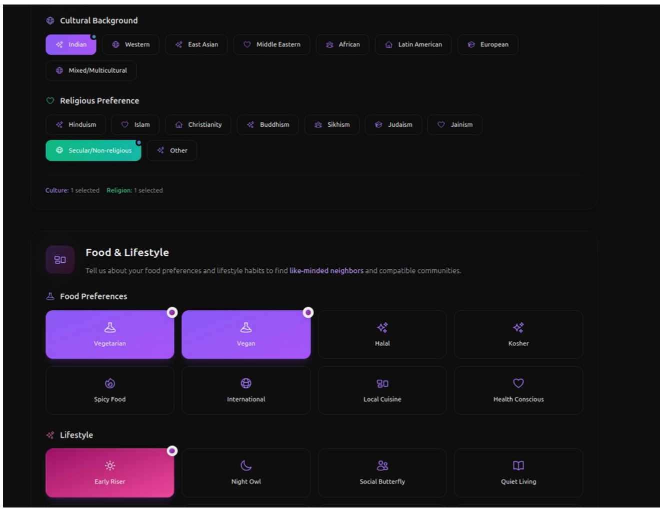
- `languages`, `safetyScore`, `vibes`, `priceRange`, `commuteOptions`, `religionTags`, etc.

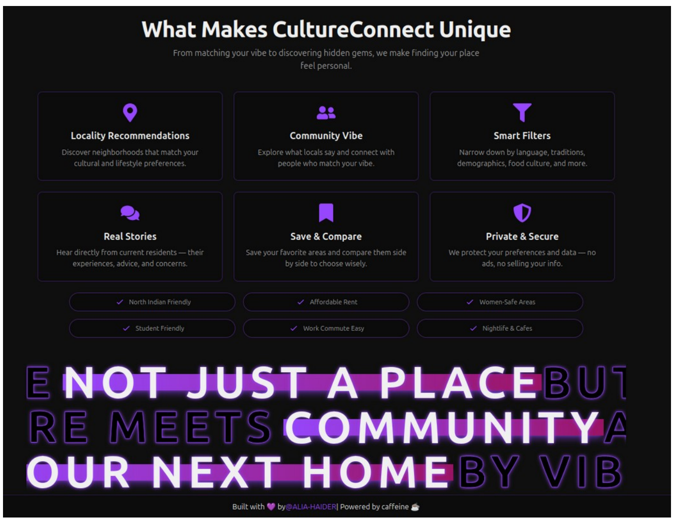
- Mapped these data points to user preferences using a custom-built scoring algorithm.
- Tested the model using mock user personas to check match accuracy and consistency.

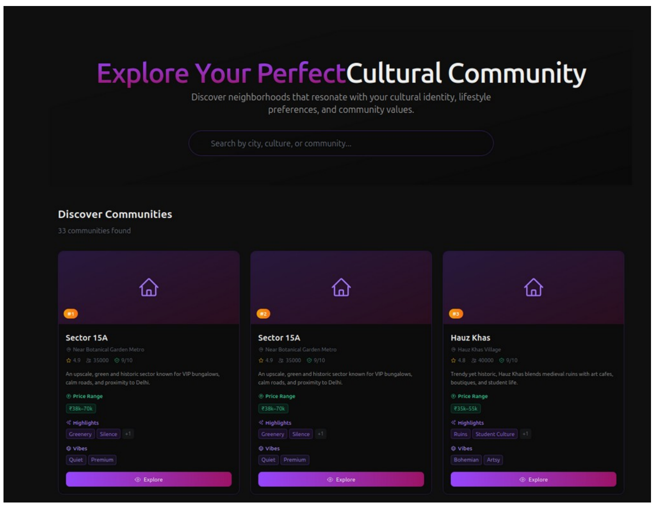












Neighborly project report

-by Alia Haider

references :

- MagicBricks.com
- 99acres.com
- Reddit r/India
- Quora posts on neighborhood search

Technical Problem-Solving

1. Matching Algorithm Design

We built a scoring-based neighborhood matching algorithm that ranks locations based on user preferences across multiple lifestyle dimensions. Each field (e.g., languages spoken, community types, safety level, food preferences, etc.) is assigned a weight, and matching scores are calculated using MongoDB's `$setIntersection` operator within an aggregation pipeline.

Key Logic:

- Match Score = Σ (number of matched tags × field weight)
- Normalized to a `matchPercentage` capped at 100
- Only localities with a positive match score are returned

Fields considered in scoring:

Category	Example Values	Weight
Languages	English, Hindi, Kannada	20
Cultural Tags	Education, Community, Arts	15
Religion Tags	Hindu, Muslim, Christian	10
Food Preferences	Vegetarian, Non-vegetarian	8
Lifestyle Options	Peaceful, Youth-Oriented	8
Area Vibes	Safe, Green, Family-Friendly	15
Community Types	Students, Families, Expats	12
Commute Options	Metro, Bus Connectivity	6
Connectivity Features	Internet, Mobile, Walkability	6

Neighborly project report

-by Alia Haider

2. Real-World Data Collection

The biggest technical challenge was the lack of structured neighborhood-level datasets for Indian cities. To address this, we:

- Scraped and manually compiled data from Google Maps, Wikipedia, and local guides.
- Normalized diverse fields (e.g., price as ranges like “₹24k - ₹36k”) to create consistent data models.
- Ensured extensibility by designing fields as string arrays, allowing future additions without schema changes.

Example of a single document in MongoDB:

```
{
"name": "Bikaner",
"city": "Bikaner",
"area": "Old City",
"location": "Junagarh Fort Road",
"lat": 28.0229,
"lng": 73.3119,
"image":
"https://upload.wikimedia.org/wikipedia/commons/0/07/Bikaner_Fort_Rajasthan.jpg"
,
"rating": 4.1,
"reviews": 300,
"safetyScore": 7,
"description": "A desert city rich with Rajputana history, havelis, temples, and spicy
snacks like Bikaneri bhujia.",
"residents": 89000,
"priceRange": "₹14k–25k",
"highlights": [
"Heritage",
"Cuisine"
],
"vibes": [
"Historic",
"Cultural",
"Dry"
],
```

Neighborly project report

-by Alia Haider

"languages": [
"Rajasthani",
"Hindi"
],
"amenities": [
"Markets",
"Forts",
"Sweet Shops"
],
"culturalTags": [
"Rajputana",
"Desert Culture"
],
"religionTags": [
"Hindu",
"Jain"
],
"foodPreferences": [],
"lifestyleOptions": [],
"areaVibes": [],
"communityTypes": [],
"commuteOptions": [],
"connectivityFeatures": [],
"updatedAt": "2025-07-04T11:33:27.641Z",
"isActive": true,
"_id": "6867bc07476607cc97f2515d",
"imageAlt": "Bikaner community image",
"images": [],
"createdAt": "2025-07-04T11:33:27.641Z",
"__v": 0
}

3. Scalable APIs & Data Structures

We used **MongoDB** for schema-flexible storage and **Node.js + Express** to build RESTful APIs for:

- Creating, updating, and retrieving locations
- Filtering based on weighted preferences

Neighborly project report

-by Alia Haider

- Fetching a location by ID for map modal

The matching logic was implemented via MongoDB's aggregation framework, ensuring server-side computation and low frontend load.

Tech Stack:

- Frontend: React + TailwindCSS + Zustand for state
- Backend: Express.js with Mongoose ORM
- Deployment: Render for backend, Vercel for frontend

4. Integration Challenges with External Data

We initially planned to fetch dynamic lat/lng from Google Maps API, but avoided it due to:

- API rate limits and billing concerns
- Manual fallback was used via embedded coordinates

Map previews were integrated via:

- **Google Maps Embed API**, using `<iframe>` inside a modal
- Triggered on click of "Explore" button from location cards

Systems Thinking

1. Trade-Offs and Design Decisions

Decision	Trade-Off
Manual data curation over API use	Ensured control and zero-cost operation, but time-intensive
MongoDB for database	Flexible schema, but requires validation discipline
Weights in matching algorithm	Tuned based on intuition and limited testing; may need real data tuning
Using Google Maps embed instead of Leaflet or Mapbox	Easier integration, but limits interaction and style customization

Neighborly project report

-by Alia Haider

2. Scalability Constraints

- **Current Scope**: Hardcoded weights and manually curated data; manageable for MVP.
- **Future Constraint**: As user base or data volume increases, re-ranking logic should be moved to a dedicated recommendation engine.
- MongoDB's aggregation is performant for now, but for larger datasets, vector embeddings or precomputed scores may be more efficient.

3. Systematic Decomposition

We broke the problem into modular units:

- **Data Layer**: Schema and models for neighborhood records
- **API Layer**: Route-based modular controllers for filter logic and CRUD
- **Frontend Layer**: Component-based card UI, modal maps, and filter UX
- **Logic Layer**: Central scoring algorithm, embedded in MongoDB queries

Constraints & Problem Parameters

1. Resource Constraints

- **Zero Budget**: All tools, libraries, and APIs used were either open-source or free-tier. We chose React for the frontend, Node.js and Express for the backend, and MongoDB Atlas (free tier) for database storage.
- **2-Week Timeline**: Given the limited duration, we focused on building a minimal viable product (MVP) with core functionalities like search filtering, matching logic, and basic UI instead of advanced features.
- **Limited Data Access**: No official APIs were available for neighborhood-specific data in India. We overcame this by:
    - Scraping content from Wikipedia and Google Maps.
    - Using Google Maps to extract coordinates.
    - Curating datasets manually for selected cities like Varanasi, Lucknow, and Bengaluru.

2. Technical Constraints

- **Real Data Usage**: All entries are based on actual localities across India, with real lat-long coordinates and descriptions, and mapped fields like price range and safety ratings.

Neighborly project report

-by Alia Haider

- **Functional Application**: This is a working full-stack web app, not a static prototype. Users can interact with filters, explore real-time results, and view dynamic maps.
- **Edge Case Handling**:
    - Empty filter input returns top-rated active locations.
    - Null or missing fields in documents do not break the score calculation pipeline.
    - City name mismatches are normalized using `.toLowerCase()` and regex queries for flexibility.

Testing Approach & Validation Results

1. Testing Strategy

- **Manual Testing** was conducted due to limited time and no QA automation.
    - All core routes (`/api/locations`, `/api/locations/filter`, `/api/locations/:id`) were tested via Postman and frontend components.
    - User flow was tested from homepage to filters to explore modal maps.
- **Internal User Testing**:
    - Sample personas (students, families, professionals) were used to simulate real user behavior.
    - Users were asked to select filters like language, vibe, affordability, and were shown matched results.

2. Validation Results

Test Case	Outcome
Filters applied (language + culture + price)	✔️ Correct results returned with match scores
No matching location	✔️ Empty array with safe fallback
Incomplete location entry in DB	✔️ Handled gracefully without crash
Map modal opens with correct coordinates	✔️ Google Maps iframe displays accurately
Filter logic scoring	✔️ Validated weights using test combinations

Conclusion

Neighborly offers a functional, research-backed solution for personalized neighborhood discovery, built efficiently under real-world constraints.