Nazdeeq

Project Team

Muhammad Ejaz	21I-0499
Hayat Sikandar Dehraj	21I-0564
Jeremy Sigamony	21I-0814

Session 2021-2025

Supervised by

Mr. Aqib Rehman

Co-Supervised by

Dr. Hina Ayaz

Department of Computer Science

National University of Computer and Emerging Sciences
Islamabad, Pakistan

May, 2025

Contents

1 Introduction **1**
1.1 Problem Statement 1
1.2 Background 1
1.3 Existing Solutions 2
1.4 Problem Statement 2
1.5 Scope 3
1.6 Modules 3
1.6.1 Module 1: NeighborCommute 3
1.6.2 Module 2: Neighborworks 4
1.6.3 Module 3: VibeTribe 4
1.6.4 Module 4: CommunityPulse 5
1.6.5 Module 5: Raabta 5
1.7 Work Division 5

2 Project Requirements **7**
2.1 Use Case Diagrams 7
2.2 Functional Requirements 14
2.2.0.1 NeighborCommute 14
2.2.0.2 Neighborworks 15
2.2.0.3 VibeTribe 16
2.2.0.4 CommunityPulse 17
2.2.0.5 Raabta 18
2.3 Non-Functional Requirements 19
2.3.1 Reliability 19
2.3.2 Usability 20
2.3.3 Performance 20
2.3.4 Security 20

3 System Overview **23**
3.1 Architectural Design 23
3.2 Data Design 24
3.3 Domain Model 24

3.4 Design Models 26
3.4.1 Activity Diagrams 27
3.4.2 Data Flow Diagrams 28
3.4.3 Data Flow Diagrams 28
3.4.4 Data Flow Diagrams 29

4 Implementation and Testing **31**
4.1 Context and Design: 31
4.2 Algorithm Design 31
4.3 Unit Testing 33

References **35**

List of Figures

2.1 NeighborCommute High-Level Use Case Diagram 11
2.2 NeighborCommute High-Level Use Case Diagram 11
2.3 NeighborCommute High-Level Use Case Diagram 11
2.4 Neighborworks High-Level Use Case Diagram 12
2.5 Neighborworks High-Level Use Case Diagram 12
2.6 Neighborworks High-Level Use Case Diagram 12
2.7 VibeTribe High-Level Use Case Diagram 12
2.8 VibeTribe High-Level Use Case Diagram 13
2.9 CommunityPulse High-Level Use Case Diagram 13
2.10 CommunityPulse High-Level Use Case Diagram 13
2.11 RaabtaAI High-Level Use Case Diagram 13
2.12 RaabtaAI High-Level Use Case Diagram 13
2.13 RaabtaAI High-Level Use Case Diagram 14
2.14 SSD Diagram for NeighborCommute 15
2.15 SSD Diagram for Neighborworks 16
2.16 SSD Diagram for VibeTribe 17
2.17 SSD Diagram for Raabta 19

3.1 Nazdeeq Architecture Diagram 24
3.2 ERD for Nazdeeq entities and relationships 26
3.3 Activity Diagram: NeighborCommute Booking Flow 27
3.4 Level 0 Data Flow Diagram for Nazdeeq 28
3.5 Level 1 Data Flow Diagram for Nazdeeq 28
3.6 Level 2 Data Flow Diagram for Nazdeeq 29

4.1 NeighborCommute Algorithm Design Workflow 32
4.2 NeighborWorks Algorithm Design Workflow 32

List of Tables

1.1 Comparison of Representative Community and Service Platforms 2
1.2 Work Division and Feature Responsibilities 6

4.1 Unit Test Cases for NeighborCommute and Neighborworks Modules . . . 34

Chapter 1

Introduction

In modern urban life, community engagement has significantly declined, leading to weakened neighborhood connections. People often live next to one another without meaningful interactions, missing opportunities for collaboration, support, and shared experiences. Our proposed mobile application aims to bridge this gap by fostering stronger neighborhood relationships and offering a unified platform for various community-driven activities.

1.1 Problem Statement

Despite technological advancements, local communities are becoming increasingly disconnected. Neighbors rarely interact, leading to missed opportunities for mutual assistance, resource sharing, and social engagement. Existing social platforms focus on global connections rather than hyperlocal interactions, leaving a void in community-based networking. There is a need for a dedicated platform that facilitates local collaboration, support, and engagement within neighborhoods.

1.2 Background

In recent years, the rapid urbanization and proliferation of digital platforms have transformed how individuals interact with their local communities. However, most social networks and service-marketplace apps focus on either global connections (e.g., Facebook, Twitter) or specific verticals (e.g., Uber for transportation, TaskRabbit for odd jobs), leaving a gap in truly hyper-local, trust-based community engagement. Studies in humancomputer interaction and urban informatics have highlighted the importance of *social capital* and *localized trust* in fostering resilient neighborhoods [2, 5].

Existing platforms often suffer from:

- **Fragmentation:** Users must install and learn multiple apps to carpool, hire technicians, join hobby groups, or raise community concerns.
- **Trust Deficits:** Reviews and ratings are frequently gamed or insufficiently verified, leading to unreliable service experiences [6].
- **Lack of Integration:** No single system provides a unified interface for *rides*, *home services*, *interest groups*, *local discussions*, and *fundraising*.

These gaps represent both a challenge and an opportunity: by unifying multiple community-centric functions under a robust trust framework, one can enhance convenience, increase civic participation, and strengthen neighborhood bonds.

1.3 Existing Solutions

Several commercial and academic systems have attempted to address parts of this problem. Below is a representative comparison:

System Name	**System Overview**	**System Limitations**
TaskRabbit	On-demand task marketplace connecting users with local freelancers for chores, repairs, and errands [3]. Features: service listings, ratings, in-app payments.	- Focused solely on paid services; no free community exchange - Reviews can be sparse or biased; limited verification - Lacks integrated ride-sharing or social group features
Thumbtack	Two-sided platform for hiring local professionals (e.g., plumbers, electricians) with quote requests [4]. Features: professional profiles, instant quotes, customer messaging.	Professional-only marketplace; no volunteer or barter options hyper-local discussion forums or interest groups Does not offer real-time matching for rides or events
Nextdoor	Private social network for neighborhoods enabling announcements, classifieds, and safety alerts [1]. Features: localized feed, crime and safety posts, event listings.	Lacks integrated service-booking or ride-sharing modules No structured skills exchange or AI-driven matchmaking

Table 1.1: Comparison of Representative Community and Service Platforms

1.4 Problem Statement

Despite technological advancements, local communities are becoming increasingly disconnected. Neighbors rarely interact, leading to missed opportunities for mutual assistance, resource sharing, and social engagement. Existing social platforms focus on global connections rather than hyperlocal interactions, leaving a void in community-based networking. There is a need for a dedicated platform that facilitates local collaboration, support, and engagement within neighborhoods.

1.5 Scope

Modern communities are becoming increasingly disconnected, with minimal interaction between neighbors. Our project aims to rebuild neighborhood connections by providing a dedicated mobile platform for local engagement. By facilitating resource-sharing, collaboration, and social interaction, the app will help foster a stronger sense of community.

The platform will feature four core modules: NeighborCommute, TalentTrade, VibeTribe, and CommunityPulse. These modules will enable neighbors to share rides, exchange skills and services, connect over common interests, and discuss local issues, ultimately making daily life more convenient and strengthening community bonds.

1.6 Modules

Nazdeeq is a mobile application that enables seamless community interactions through six key modules:

- **NeighborCommute:** Allows neighbors to share rides, reducing transportation costs and environmental impact.
- **Neighborworks:** Connects individuals with skills to those in need of voluntary assistance.
- **VibeTribe:** Helps users find like-minded individuals for shared activities.
- **CommunityPulse:** Facilitates discussions on local issues, ranked by sentiment analysis for relevance.
- **Raabta:** A centralized AI-chatbot for every type of query related to neighbourhood.

1.6.1 Module 1: NeighborCommute

NeighborCommute is a trusted ride-sharing platform that connects neighbors traveling in the same direction. It reduces transportation costs, lowers environmental impact, and fosters community bonds through shared journeys. The module leverages credibility scoring and real-time tracking to ensure safe and reliable commutes.

1. **Credibility-Based Matching:** Pairs neighbors for rides using verified user ratings, endorsements, and past ride history.

2. **Real-Time Tracking:** Provides live route updates, estimated arrival times, and emergency alerts for enhanced commuter safety.

1.6.2 Module 2: Neighborworks

Neighborworks is a community-driven service hub that connects individuals with specialized skills to those in need of assistance. It promotes volunteerism and local support, ensuring reliable help is always available close to home. This module emphasizes transparency and trust by verifying service providers through user feedback, background checks, and multimedia reviews.

1. **Verified Service Directory:** Lists local technicians and helpers with credentials, certifications, and background verifications.

2. **Transparent Reviews:** Features authentic user reviews, ratings, and proof-of-work images or videos.

3. **Instant Booking:** Enables quick scheduling for services rendered.

4. **Service History:** Maintains a record of past service interactions to build long-term trust.

1.6.3 Module 3: VibeTribe

VibeTribe helps users discover and connect with like-minded individuals for shared activities and interests. It fosters community engagement by creating groups around hobbies and passions, enhancing social bonds among neighbors. The module uses intuitive matching algorithms to bring together users with similar interests and facilitates event planning.

1. **Interest Matching:** Connects users based on shared hobbies, interests, and activity preferences using personalized algorithms.

2. **Event Coordination:** Facilitates the organization of group activities, meet-ups, and local events.

3. **Community Forums:** Provides dedicated spaces for discussions on specific interests and activities.

1.6.4 Module 4: CommunityPulse

CommunityPulse is a dynamic forum that facilitates discussions on local issues and community initiatives. It empowers residents to voice their opinions, participate in decision-making, and contribute to local development. The module employs sentiment analysis to rank and highlight the most relevant and urgent topics.

1. **Interactive Forums:** Provides an easy-to-use platform for residents to discuss local issues and share ideas.
2. **Sentiment Analysis:** Uses AI to analyze discussion sentiment, highlighting urgent concerns and trending topics.
3. **Polls and Surveys:** Enables users to create and participate in polls to gauge community opinion.
4. **Issue Tracking:** Allows residents to follow and receive updates on ongoing community initiatives.

1.6.5 Module 5: Raabta

Raabta serves as a centralized AI-chatbot that handles every type of query related to neighborhood services. Acting as a digital concierge, Raabta seamlessly guides users across all modules, ensuring that residents receive prompt, accurate information and support for all community-related needs.

1. **Unified Assistance:** Provides one-stop support for queries spanning all modules, from ride-sharing to local services and events.
2. **Smart Navigation:** Uses advanced NLP to understand user requests and direct them to the appropriate services or information.
3. **Personalized Recommendations:** Offers tailored suggestions based on user behavior and preferences.
4. **24/7 Availability:** Ensures round-the-clock support with instant responses to critical inquiries.

1.7 Work Division

Each team member is responsible for specific modules and detailed features to ensure efficient, parallel development.

Name	Registration	Assigned Modules & Features
Jeremy Sigamony	21I-0814	**NeighborCommute:** 1. Ride Offer Creation (Geo-based matching) 2. Ride Request & Booking Workflow 3. Real-Time Tracking & ETA Updates 4. Emergency Alert Button & Notifications **Raabta (AI Chatbot):** 1. Intent Recognition & NLP Integration 2. Personalized Answer Routing (All Modules) 3. Query History Logging 4. Suggestive Prompts & Fallback Handling
Muhammad Ejaz Alvi	21I-0499	**VibeTribe:** 1. Interest Profile Setup & Tagging 2. Group Matching Algorithm 3. Event Scheduling & RSVP System 4. Push Notifications & Calendar Integration **UI/UX Design:** 1. High-Fidelity Figma Prototypes 2. Responsive Layout for Mobile 3. Onboarding Flow & Transitions
Hayat Sikandar Dehraj	21I-0564	**Neighborworks:** 1. Service Listing & Booking 2. Provider Profile Verification 3. Ratings, Reviews & Payment Flow 4. Booking History & Status Updates **CommunityPulse:** 1. Discussion Forum CRUD 2. Poll Creation & Voting 3. Sentiment Analysis & Ranking 4. Community Flag/Report System

Table 1.2: Work Division and Feature Responsibilities

Chapter 2

Project Requirements

This section outlines the necessary requirements for the successful completion of the project. Project requirements can be divided into two main categories: functional requirements, which describe the systems core operations, and non-functional requirements, which specify performance, security, and usability standards.

2.1 Use Case Diagrams

This section contains visual representations of the system's use cases using the Unified Modeling Language (UML). Use case diagrams illustrate the relationships between actors (users or external systems) and the use cases they interact with.

NeighborCommute

High Level Use Cases

Use Case	User Story
Find Available Rides	As a resident, I want to view available carpool options so I can travel with my neighbors and save money.
Offer a Ride	As a resident, I want to post a ride offer so others can join and we can commute together.
Schedule Recurring Rides	As a daily commuter, I want to schedule my rides for the week so I dont have to repeat the process every day.

Extended Use Cases

Use Case	User Story
Filter Rides by Time and Location	As a user, I want to filter ride offers based on departure time and location so I can find the most suitable option.
Rate Ride Experience	As a passenger, I want to rate my ride so other users can benefit from my feedback.
Receive Ride Alerts	As a frequent rider, I want to receive notifications when a ride matches my preferences.

Neighborworks

High Level Use Cases

Use Case	User Story
Browse Local Service Providers	As a resident, I want to browse through verified service providers so I can choose someone trustworthy.
Book a Service Appointment	As a user, I want to book a service provider for a task at my home.
View Provider Profiles	As a user, I want to view provider ratings and reviews before booking.

Extended Use Cases

Use Case	User Story
Schedule Services with Time Slot	As a customer, I want to pick a time for the service so it fits into my day and be able to book the service provider on a specific slot
Leave Reviews After Service	As a resident, I want to share my feedback after a service is completed.
Report a Problem with a Provider	As a user, I want to be able to report service providers for bad behavior or incomplete work.

VibeTribe

High Level Use Cases

Use Case	User Story
Find Interest-Based Partners	As a resident, I want to find like minded individuals for doing an activity together.

Extended Use Cases

Use Case	User Story
Interest Partner Information	For safety purposes, I want to be shown information about my interest partner.
Chat with Interest Partner	When I find an interest partner, I should be able to chat with them for discussion.

CommunityPulse

High Level Use Cases

Use Case	User Story
Post a New Discussion	As a resident, I want to create discussion threads to talk about local topics.
Comment on Threads	As a user, I want to participate in neighborhood discussions by replying to threads.
Report Inappropriate Content	As a responsible user, I want to report harmful or offensive posts.

Extended Use Cases

Use Case	User Story
Sort Threads by Sentiment	As a resident, I want to browse positive or constructive discussions easily, i.e critical posts should be shown on priority.
Post Anonymously	As a concerned neighbor, I want to share sensitive issues without revealing my identity.

RaabtaAI

High Level Use Cases

Use Case	User Story
Chat with Raabta	As a user, I want to be able to chat about general topics with Raabta.
Information about App	As a user, I want Raabta AI to answer queries regarding the application.

Extended Use Cases

Use Case	User Story
Chat Memory	As a user, when I chat with Raabta , I want it to have memory about past conversations and chats.
Display Results from Application	For easy accessibility, I want to be able to perform many of the actions through the Raabta AI (finding interest partners, displaying communitypulse posts etc)
Information about App	As a user, when I chat with Raabta I want it to have information about the app so I can get any help regarding the application itself.

Diagrams

Below are the Use Diagrams designed for each of the 5 modules.

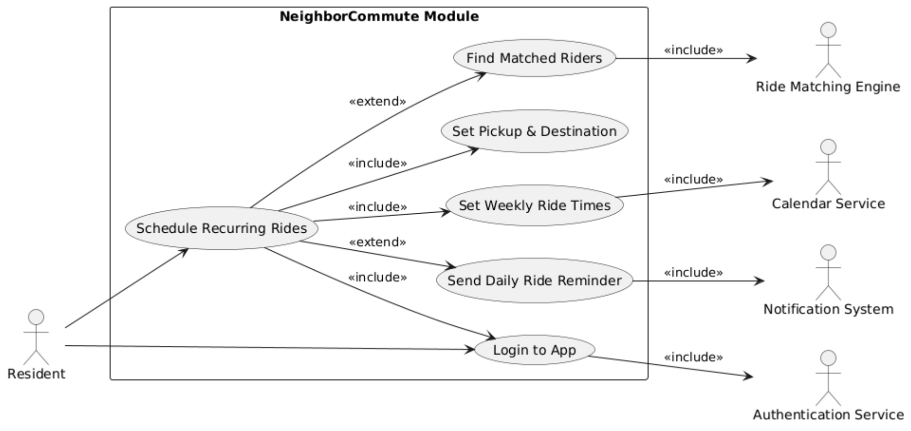


Figure 2.1: NeighborCommute High-Level Use Case Diagram

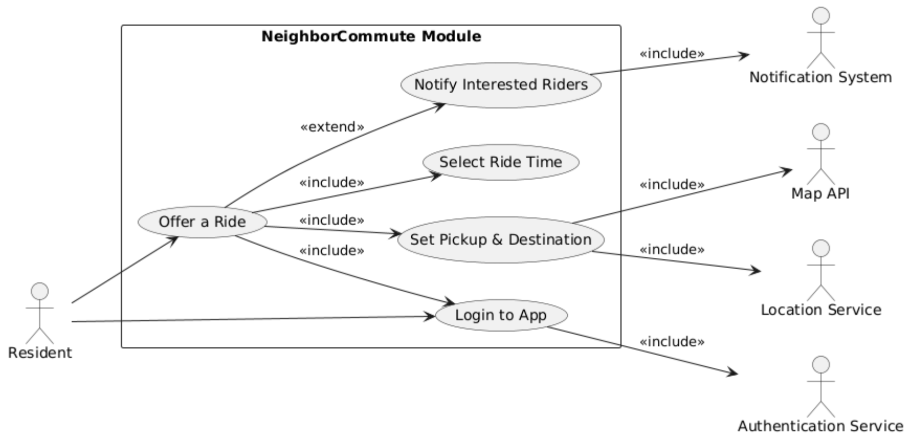


Figure 2.2: NeighborCommute High-Level Use Case Diagram

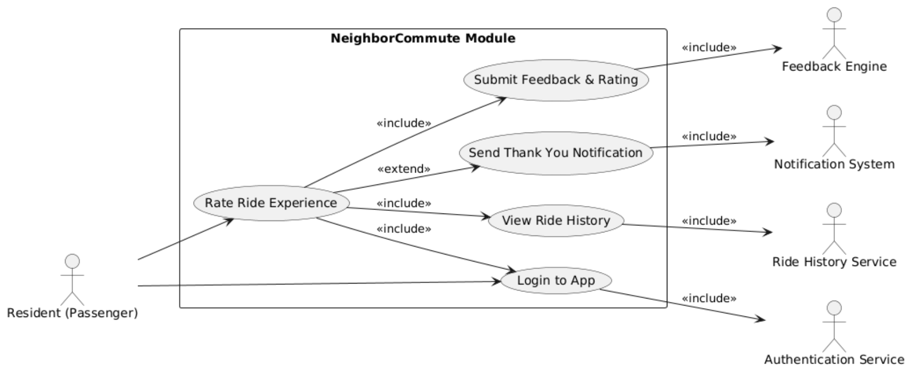


Figure 2.3: NeighborCommute High-Level Use Case Diagram

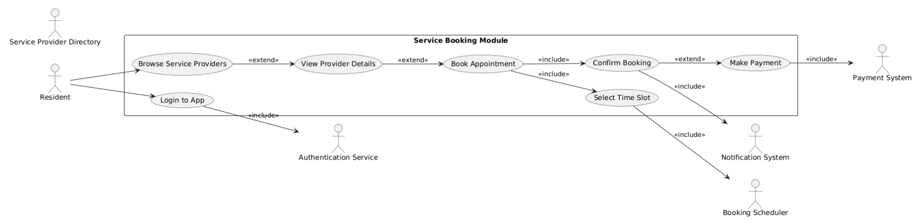


Figure 2.4: Neighborworks High-Level Use Case Diagram

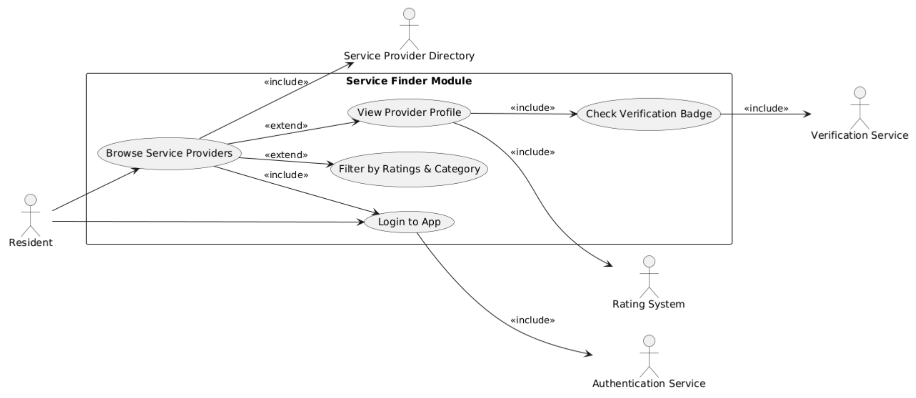


Figure 2.5: Neighborworks High-Level Use Case Diagram

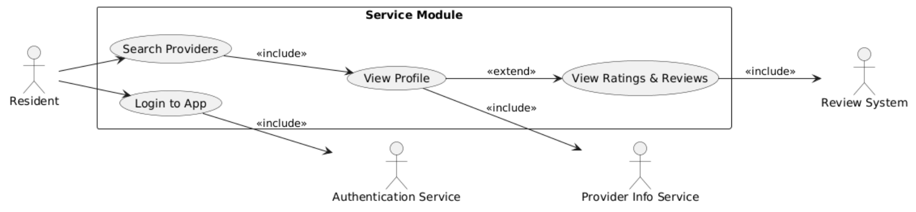


Figure 2.6: Neighborworks High-Level Use Case Diagram

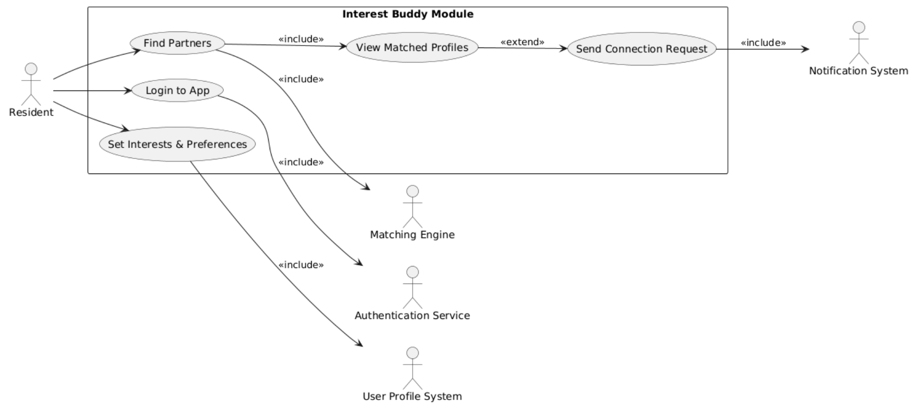


Figure 2.7: VibeTribe High-Level Use Case Diagram

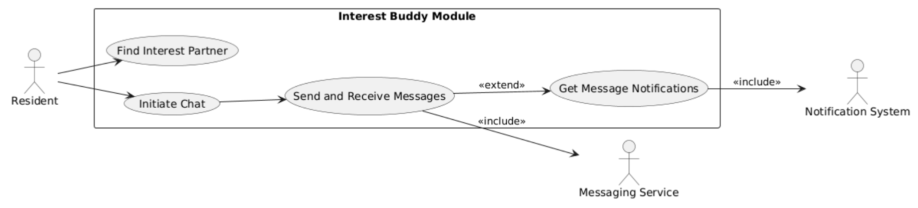


Figure 2.8: VibeTribe High-Level Use Case Diagram

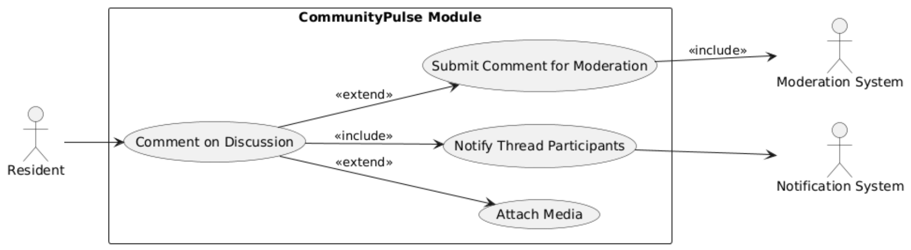


Figure 2.9: CommunityPulse High-Level Use Case Diagram

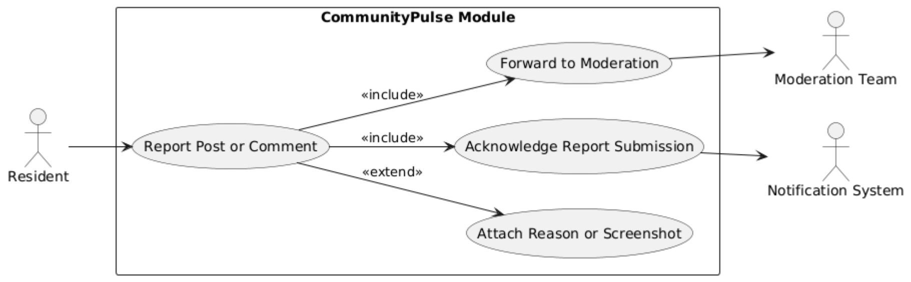


Figure 2.10: CommunityPulse High-Level Use Case Diagram

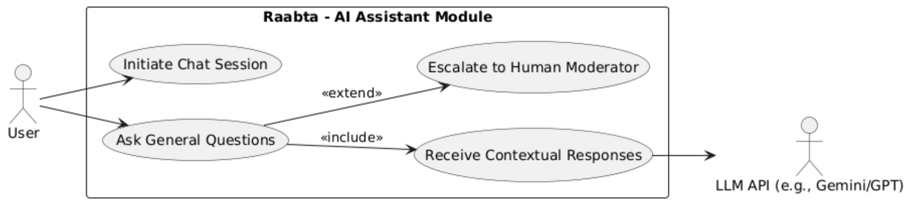

Raabta - AI Assistant Module
User
Initiate Chat Session
Ask General Questions
«extend»
Escalate to Human Moderator
«include»
Receive Contextual Responses
LLM API (e.g., Gemini/GPT)

Figure 2.11: RaabtaAI High-Level Use Case Diagram

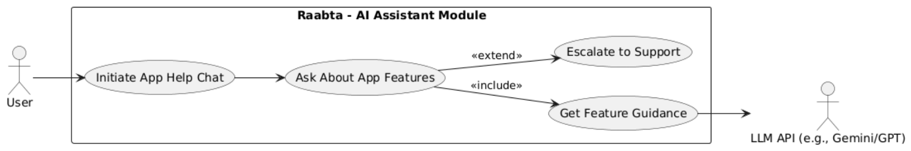


Figure 2.12: RaabtaAI High-Level Use Case Diagram

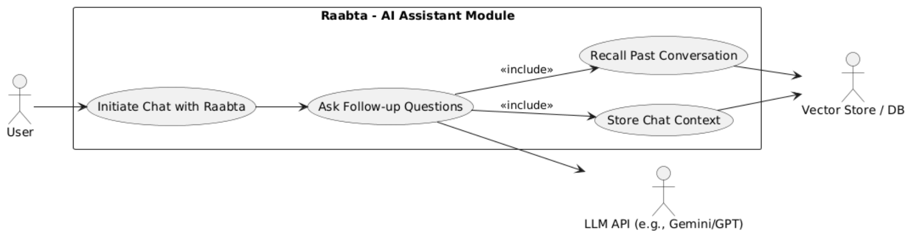


Figure 2.13: RaabtaAI High-Level Use Case Diagram

2.2 Functional Requirements

This section details the specific functional requirements for each module, including their core functionalities and interfaces. For each module, a dedicated space is provided to later include the corresponding SSD diagram.

2.2.0.1 NeighborCommute

NeighborCommute is designed to facilitate ride-sharing among neighbors. Its functional requirements include:

- **FR2.2.1.1 Ride Request Creation:** Users can create ride requests by entering origin, destination, and travel time.
- **FR2.2.1.2 Credibility-Based Matching:** The system matches ride requests based on verified user ratings, ride history, and endorsements.
- **FR2.2.1.3 Real-Time Tracking:** Once a match is made, the module provides live tracking of the ride via integration with location APIs.
- **FR2.2.1.4 Notifications and Alerts:** Automated notifications for ride confirmations, cancellations, and updates.

SSD Diagram for NeighborCommute:

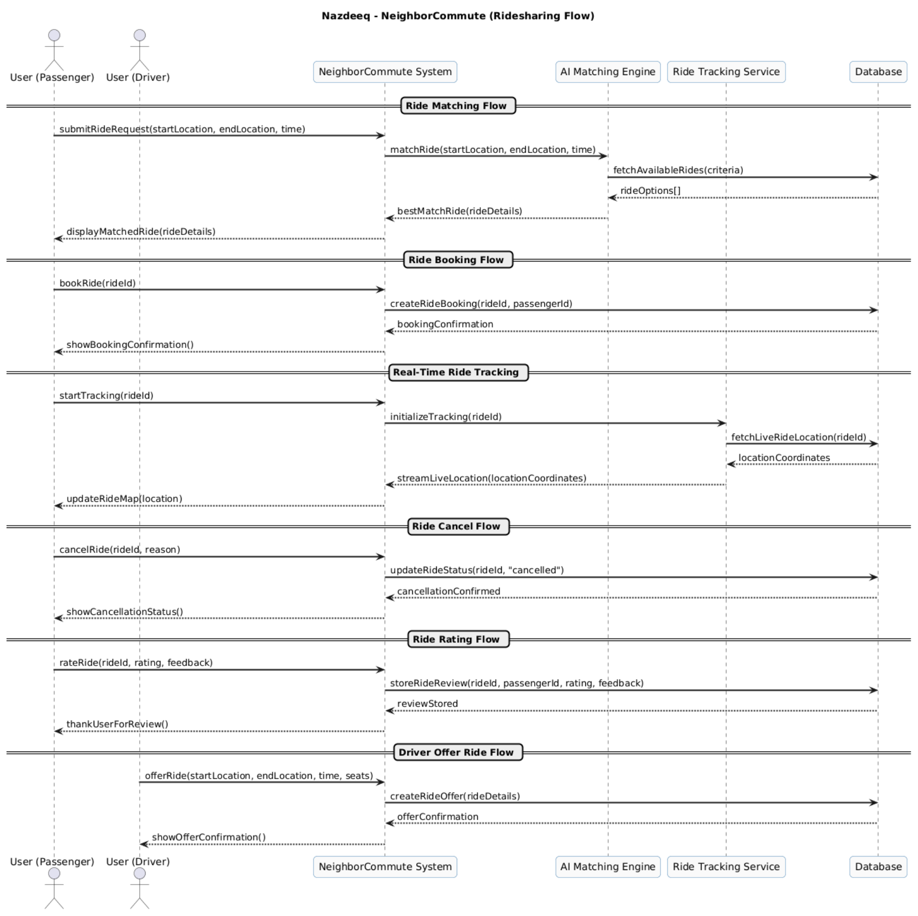

Nazdeeq - NeighborCommute (Ridesharing Flow)
User (Passenger)
User (Driver)
NeighborCommute System
AI Matching Engine
Ride Tracking Service
Database
Ride Matching Flow
submitRideRequest(startLocation, endLocation, time)
matchRide(startLocation, endLocation, time)
fetchAvailableRides(criteria)
rideOptions[]
bestMatchRide(rideDetails)
displayMatchedRide(rideDetails)
Ride Booking Flow
bookRide(rideId)
createRideBooking(rideId, passengerId)
bookingConfirmation
showBookingConfirmation()
Real-Time Ride Tracking
startTracking(rideId)
initializeTracking(rideId)
fetchLiveRideLocation(rideId)
locationCoordinates
streamLiveLocation(locationCoordinates)
updateRideMap(location)
Ride Cancel Flow
cancelRide(rideId, reason)
updateRideStatus(rideId, "cancelled")
cancellationConfirmed
showCancellationStatus()
Ride Rating Flow
rateRide(rideId, rating, feedback)
storeRideReview(rideId, passengerId, rating, feedback)
reviewStored
thankUserForReview()
Driver Offer Ride Flow
offerRide(startLocation, endLocation, time, seats)
createRideOffer(rideDetails)
offerConfirmation
showOfferConfirmation()

Figure 2.14: SSD Diagram for NeighborCommute

2.2.0.2 Neighborworks

Neighborworks provides a platform for connecting local service providers with residents in need of assistance. Its functional requirements include:

- **FR2.2.2.1 Service Request Management:** Users can create, view, and manage service requests (e.g., plumbing, electrical, tutoring).
- **FR2.2.2.2 Provider Verification:** Service providers undergo background checks and are verified through user reviews and multimedia evidence.
- **FR2.2.2.3 Instant Booking:** The system enables quick scheduling and confirmation of service appointments.
- **FR2.2.2.4 Service History Logging:** All service transactions are recorded for future reference and to build trust.

SSD Diagram for Neighborworks:

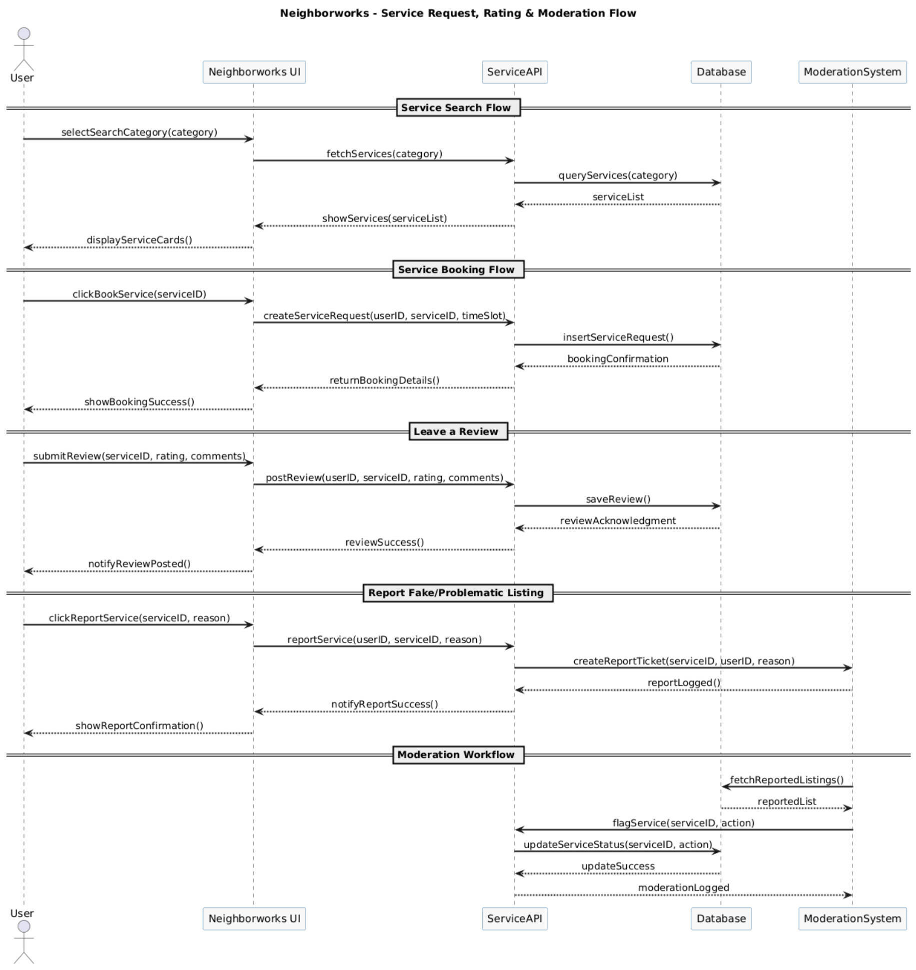


Figure 2.15: SSD Diagram for Neighborworks

2.2.0.3 VibeTribe

VibeTribe aims to connect residents with similar interests to foster community bonding and collaborative activities. Its functional requirements include:

- **FR2.2.3.1 Interest Profile Management:** Users can create, update, and manage their profiles including interests and hobbies.
- **FR2.2.3.2 Matching Algorithm:** The system employs personalized matching algorithms to suggest like-minded individuals.
- **FR2.2.3.3 Event Coordination:** Users can plan and schedule group activities and events, with calendar integration and automated reminders.

- **FR2.2.3.4 User Interaction:** Support for sending connection requests, joining groups, and participating in discussions related to shared interests.

SSD Diagram for VibeTribe:

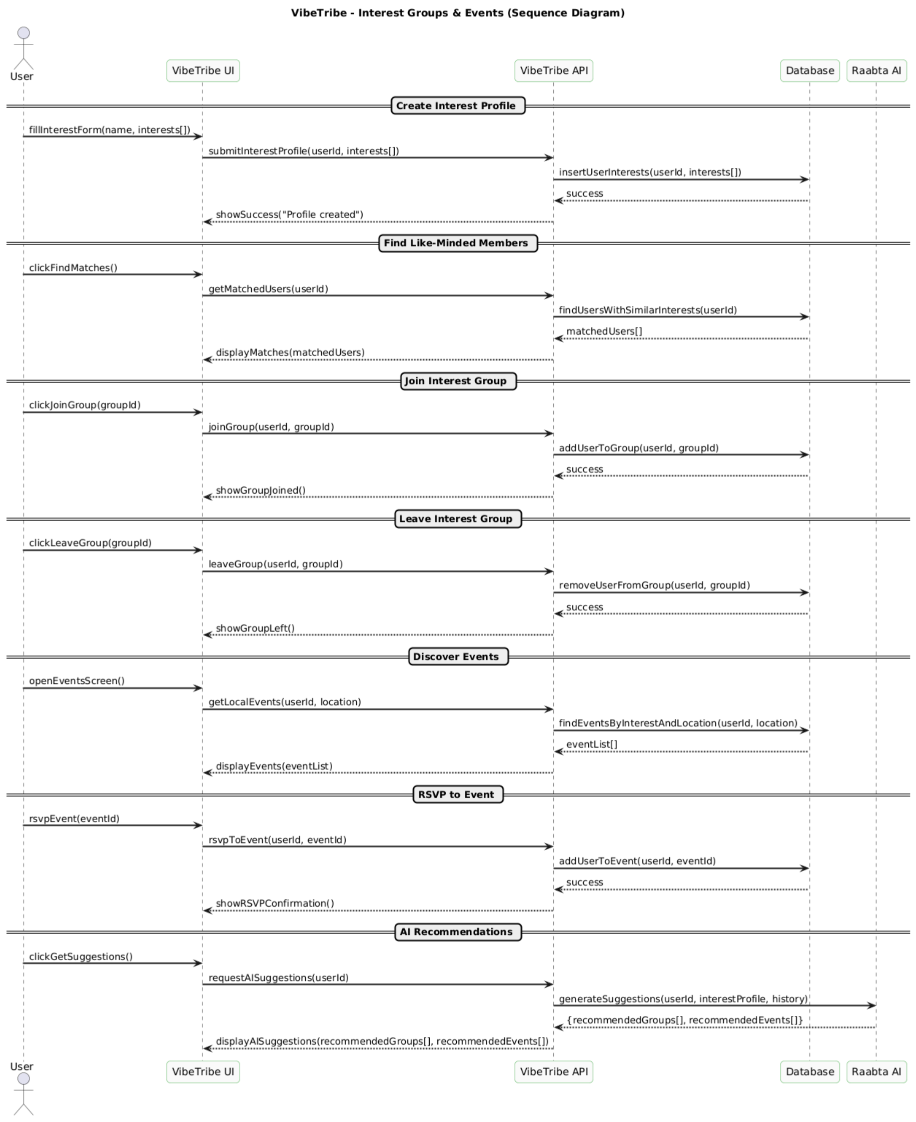

VibeTribe - Interest Groups & Events (Sequence Diagram)
User
VibeTribe UI
VibeTribe API
Database
Raabta AI
Create Interest Profile
fillInterestForm(name, interests[])
submitInterestProfile(userId, interests[])
insertUserInterests(userId, interests[])
success
showSuccess("Profile created")
Find Like-Minded Members
clickFindMatches()
getMatchedUsers(userId)
findUsersWithSimilarInterests(userId)
matchedUsers[]
displayMatches(matchedUsers)
Join Interest Group
clickJoinGroup(groupId)
joinGroup(userId, groupId)
addUserToGroup(userId, groupId)
success
showGroupJoined()
Leave Interest Group
clickLeaveGroup(groupId)
leaveGroup(userId, groupId)
removeUserFromGroup(userId, groupId)
success
showGroupLeft()
Discover Events
openEventsScreen()
getLocalEvents(userId, location)
findEventsByInterestAndLocation(userId, location)
eventList[]
displayEvents(eventList)
RSVP to Event
rsvpEvent(eventId)
rsvpToEvent(userId, eventId)
addUserToEvent(userId, eventId)
success
showRSVPConfirmation()
AI Recommendations
clickGetSuggestions()
requestAISuggestions(userId)
generateSuggestions(userId, interestProfile, history)
{recommendedGroups[], recommendedEvents[]}
displayAISuggestions(recommendedGroups[], recommendedEvents[])
User
VibeTribe UI
VibeTribe API
Database
Raabta AI

Figure 2.16: SSD Diagram for VibeTribe

2.2.0.4 CommunityPulse

CommunityPulse is the interactive forum for residents to discuss local issues, share opinions, and engage in community decision-making. Its functional requirements include:

- **FR2.2.4.1 Discussion Management:** Users can create, edit, and delete discussion topics.

- **FR2.2.4.2 Commenting and Voting:** Support for posting comments, upvoting/downvoting posts, and threaded discussions.

- **FR2.2.4.3 Sentiment Analysis:** Integration of AI modules to analyze post sentiment and rank discussions based on urgency and engagement.

- **FR2.2.4.4 Polls and Surveys:** Ability to create and participate in polls to gather community opinions.

- **FR2.2.4.5 Issue Tracking:** Residents can track ongoing discussions and receive notifications for updates.

2.2.0.5 Raabta

Raabta is the centralized AI-powered chatbot that provides instant assistance for various neighborhood queries. Its functional requirements include:

- **FR2.2.5.1 Chat Session Management:** Initiation, maintenance, and termination of chatbot sessions.

- **FR2.2.5.2 NLP and Query Processing:** Utilizes natural language processing to interpret user queries and fetch relevant responses.

- **FR2.2.5.3 Response Generation:** Generates context-aware and personalized responses, integrating data from various modules.

- **FR2.2.5.4 Continuous Learning:** Updates and refines response algorithms based on user feedback and new data.

- **FR2.2.5.5 24/7 Availability:** Ensures round-the-clock support with minimal response latency.

SSD Diagram for Raabta:

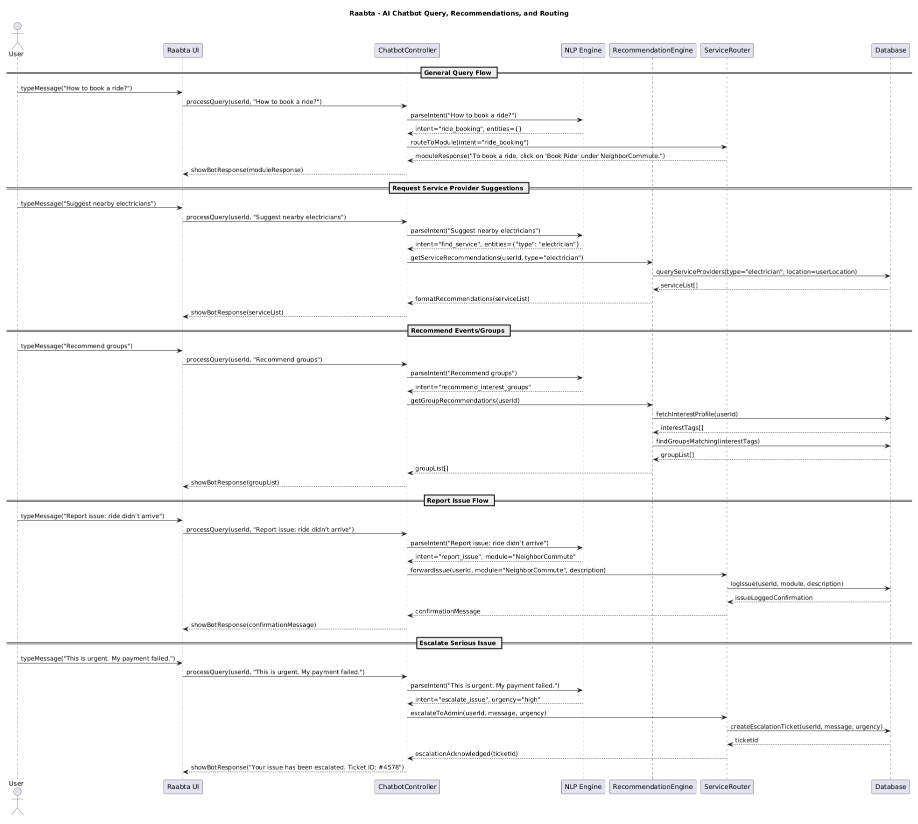

Raabta - AI Chatbot Query, Recommendations, and Routing
User
Raabta UI
ChatbotController
NLP Engine
RecommendationEngine
ServiceRouter
Database
General Query Flow
typeMessage("How to book a ride?")
processQuery(userId, "How to book a ride?")
parseIntent("How to book a ride?")
intent="ride_booking", entities={}
routeToModule(intent="ride_booking")
moduleResponse("To book a ride, click on 'Book Ride' under NeighborCommute.")
showBotResponse(moduleResponse)
Request Service Provider Suggestions
typeMessage("Suggest nearby electricians")
processQuery(userId, "Suggest nearby electricians")
parseIntent("Suggest nearby electricians")
intent="find_service", entities={"type": "electrician"}
getServiceRecommendations(userId, type="electrician")
queryServiceProviders(type="electrician", location=userLocation)
serviceList[]
formatRecommendations(serviceList)
showBotResponse(serviceList)
Recommend Events/Groups
typeMessage("Recommend groups")
processQuery(userId, "Recommend groups")
parseIntent("Recommend groups")
intent="recommend_interest_groups"
getGroupRecommendations(userId)
fetchInterestProfile(userId)
interestTags[]
findGroupsMatching(interestTags)
groupList[]
groupList[]
showBotResponse(groupList)
Report Issue Flow
typeMessage("Report issue: ride didn't arrive")
processQuery(userId, "Report issue: ride didn't arrive")
parseIntent("Report issue: ride didn't arrive")
intent="report_issue", module="NeighborCommute"
forwardIssue(userId, module="NeighborCommute", description)
logIssue(userId, module, description)
issueLoggedConfirmation
confirmationMessage
showBotResponse(confirmationMessage)
Escalate Serious Issue
typeMessage("This is urgent. My payment failed.")
processQuery(userId, "This is urgent. My payment failed.")
parseIntent("This is urgent. My payment failed.")
intent="escalate_issue", urgency="high"
escalateToAdmin(userId, message, urgency)
createEscalationTicket(userId, message, urgency)
ticketId
escalationAcknowledged(ticketId)
showBotResponse("Your issue has been escalated. Ticket ID: #4578")
User
Raabta UI
ChatbotController
NLP Engine
RecommendationEngine
ServiceRouter
Database

Figure 2.17: SSD Diagram for Raabta

2.3 Non-Functional Requirements

This section specifies non-functional requirements for Nazdeeq. These quality requirements are specific, quantitative, and verifiable.

2.3.1 Reliability

Reliability requirements ensure the system remains available and correct under expected conditions.

- **REL-1:** Nazdeeq shall maintain an uptime of at least 99.5% each calendar month.
- **REL-2:** In case of any single service failure, 95% of dependent API calls shall failover to backup instances within 5 seconds.
- **REL-3:** Critical data (rides, bookings, messages) shall be synchronously replicated across two availability zones with zero data loss.

- **REL-4:** After a total system outage, full recovery (RTO) shall complete within 30 minutes, verified by quarterly drills.

2.3.2 Usability

Usability requirements guide the design of a clear, intuitive, and accessible user experience.

- **USE-1:** 90% of new users shall complete their first ride booking or service request within 3 minutes without external guidance.
- **USE-2:** The UI shall conform to WCAG 2.1 Level AA; all interactive elements must be operable via keyboard and labeled for screen readers.
- **USE-3:** Onboarding screens shall load and transition in under 1 second on a standard 4G connection.
- **USE-4:** An in-app feedback widget shall be present on 100% of key screens; user submissions must be acknowledged within 2 business hours.

2.3.3 Performance

Performance requirements define response times, throughput, and scalability targets for critical operations.

- **PERF-1:** 95% of API calls for ride search, service booking, and group discovery shall respond within 2 seconds under up to 1,000 concurrent users.
- **PERF-2:** The system shall sustain at least 200 transactions per second (TPS) during peak usage (e.g., 89 AM commute).
- **PERF-3:** CPU and memory usage shall remain below 70% on average; any resource spike above 85% must trigger auto-scaling within 60 seconds.
- **PERF-4:** Page and screen transitions in the mobile app shall render fully within 300 ms after data load.

2.3.4 Security

Security requirements protect user data and ensure only authorized access.

- **SEC-1:** All communications between client and server shall use TLS 1.2 or higher; no unencrypted traffic permitted.
- **SEC-2:** JWT access tokens shall expire after 24 hours; refresh tokens shall expire after 7 days.
- **SEC-3:** Role-based access control shall enforce that unauthorized API calls return HTTP 403 within 50 ms.
- **SEC-4:** All security-sensitive events (logins, data changes) shall be logged and retained for a minimum of 12 months; audit queries shall complete within 5 seconds.
- **SEC-5:** Security incidents shall be detected and alerted to the operations team within 15 minutes, with containment measures initiated within 1 hour.

Chapter 3

System Overview

3.1 Architectural Design

Nazdeeq is built on a modular, scalable architecture comprising the following layers:

- **Mobile Frontend:** Developed in React Native to ensure cross-platform accessibility.
- **API Gateway:** A Node.js/Express server acting as a central entry point for all client requests.
- **Microservices:** Each module (NeighborCommute, Neighborworks, VibeTribe, CommunityPulse, Raabta) operates as an independent microservice for modularity and ease of scaling.
- **AI/ML Components:** Integrated for functionalities such as sentiment analysis (CommunityPulse) and natural language processing (Raabta).
- **Database:** MongoDB serves as the NoSQL datastore managing dynamic data, ride/service histories, user profiles, and more.
- **Authentication:** A hybrid approach using Firebase Authentication for social logins and OAuth/JWT for traditional sign-ups.

Nazdeeq Architecture Diagram

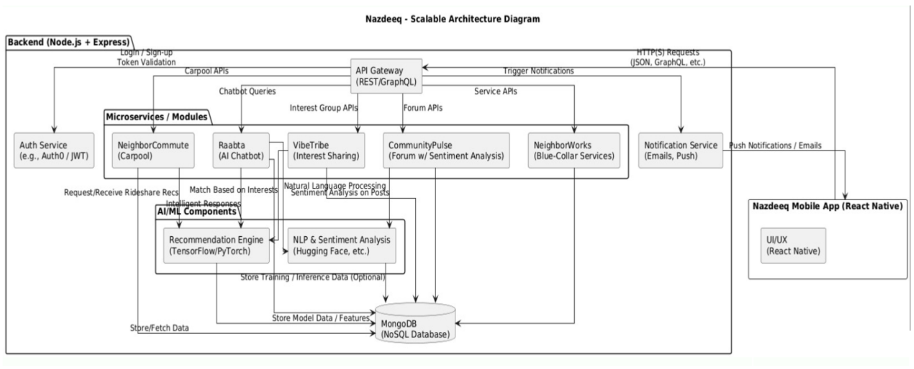


Figure 3.1: Nazdeeq Architecture Diagram

3.2 Data Design

In Nazdeeq, each realworld concept (user, ride, service, group, etc.) is represented by a MongoDB collection. Collections are schemaless but follow the logical structures defined below. Data is stored as BSON documents, allowing nested JSON objects (e.g. location coordinates and poll options). Key points:

- **Users Collection:** Stores profile, credentials, trustScore, array fields for roles and interests. Geospatial index on `location` supports nearby searches.
- **RideRequests & Rides Collections:** Ride requests capture pending queries; matched rides hold driver offers. References via ObjectId link requests to rides and passengers.
- **Services & ServiceRequests Collections:** Providers list offerings in `services`; `serviceRequests` track bookings, statuses, and link to `reviews`.
- **Groups, Events, Discussions, Polls, Comments Collections:** Support VibeTribe and CommunityPulseeach collection stores nested arrays (e.g. `attendeeIds`, poll `options`) for many-to-many relationships.
- **Chats & Messages Collections:** Persist AI chatbot sessions and message histories, enabling analytics and audit logging.
- **Indexes:** All foreign-key fields (`userId`, `groupId`, etc.) and high-traffic search fields (`dateTime`, `status`) are indexed to ensure sub-second query performance.

3.3 Domain Model

The domain model for Nazdeeq comprises the following entities and relationships:

User Attributes: `_id`, name, email, passwordHash, phone, trustScore, roles, interests Relationships: one-to-many with rideRequests, serviceRequests; many-to-many with rides.passengerIds and groups.memberIds; one-to-many with messages, discussions, comments.

RideRequest & Ride Attributes: origin, destination, dateTime, status, seatsAvailable, fare Relationships: each RideRequest references one Ride; a Ride references one driver and many passengers.

Service & ServiceRequest Attributes: title, description, pricePerHour, verification, status Relationships: one-to-many between services and serviceRequests, linking seeker and provider.

Group & Event Attributes: name, description, attendeeIds Relationships: a Group can have many Events; users attend events via attendeeIds.

Discussion, Comment, Poll Attributes: title, content, sentiment, options Relationships: one-to-many from Discussion to Comments and Polls.

Chat & Message Attributes: startedAt, endedAt, sender, content Relationships: each Chat session contains multiple Messages.

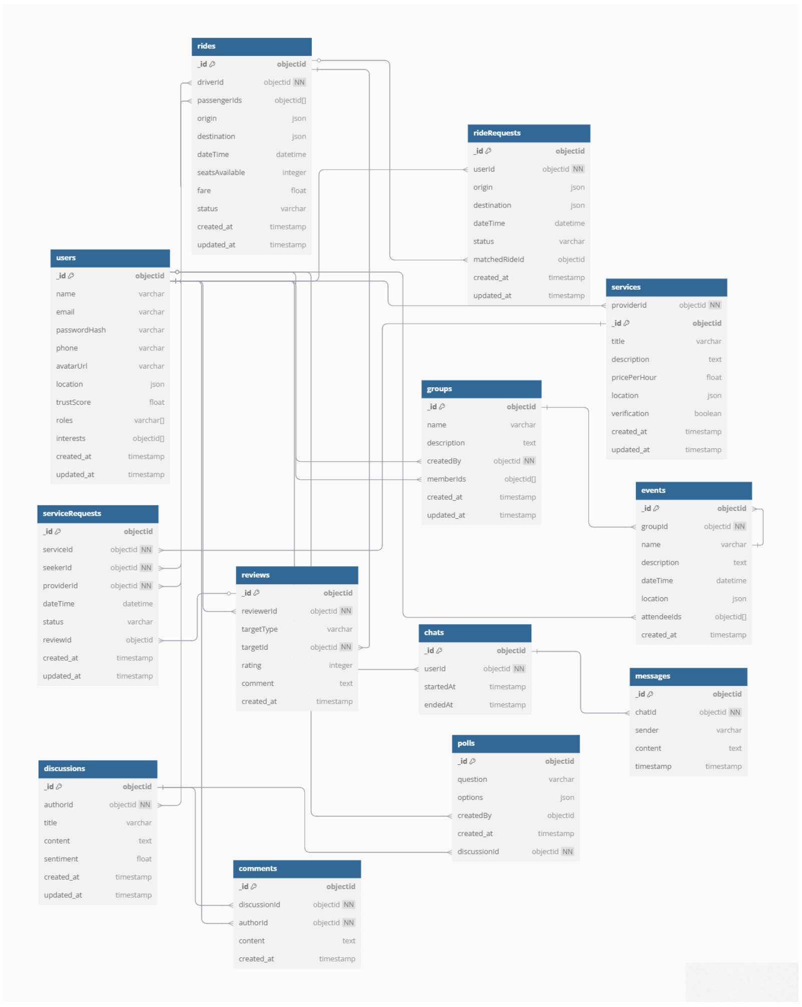


Figure 3.2: ERD for Nazdeeq entities and relationships

3.4 Design Models

To guide implementation and validate design, we will produce the following diagrams:

3.4.1 Activity Diagrams

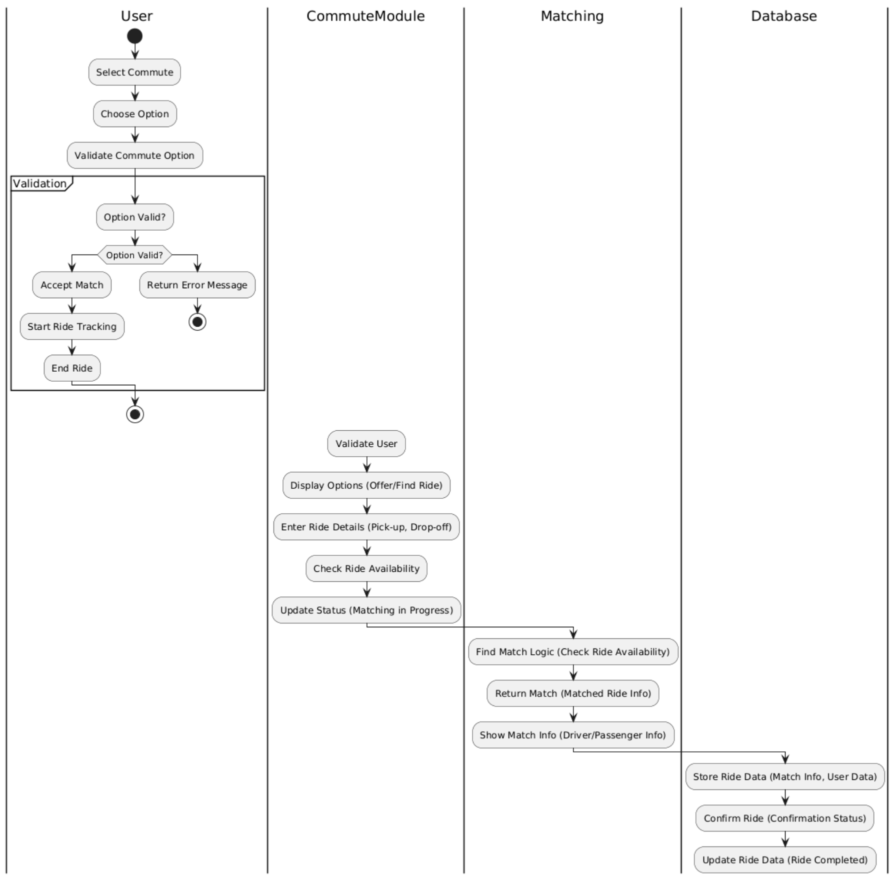

User
CommuteModule
Matching
Database
Select Commute
Choose Option
Validate Commute Option
Validation
Option Valid?
Option Valid?
Accept Match
Return Error Message
Start Ride Tracking
End Ride
Validate User
Display Options (Offer/Find Ride)
Enter Ride Details (Pick-up, Drop-off)
Check Ride Availability
Update Status (Matching in Progress)
Find Match Logic (Check Ride Availability)
Return Match (Matched Ride Info)
Show Match Info (Driver/Passenger Info)
Store Ride Data (Match Info, User Data)
Confirm Ride (Confirmation Status)
Update Ride Data (Ride Completed)

Figure 3.3: Activity Diagram: NeighborCommute Booking Flow

3.4.2 Data Flow Diagrams

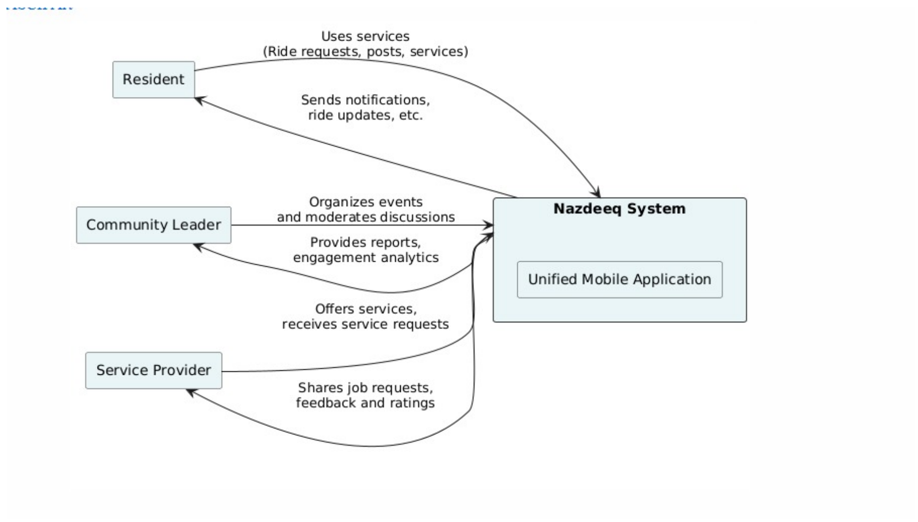


Figure 3.4: Level 0 Data Flow Diagram for Nazdeeq

3.4.3 Data Flow Diagrams

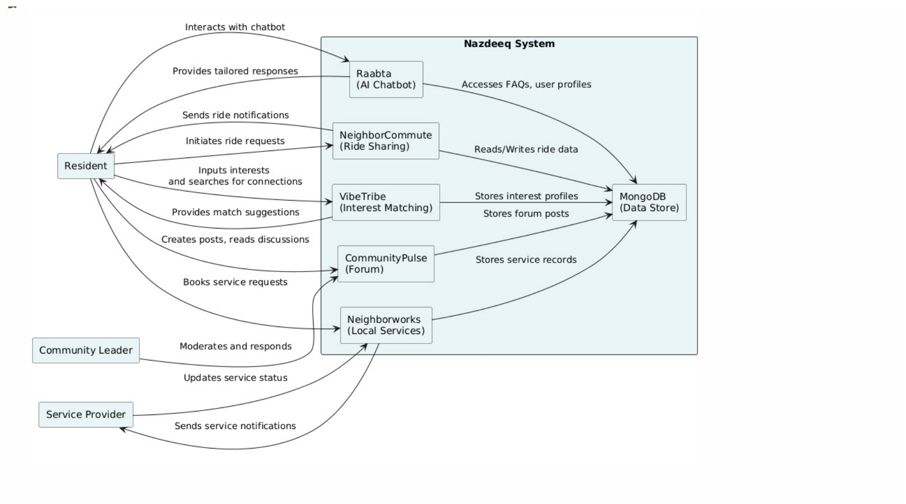


Figure 3.5: Level 1 Data Flow Diagram for Nazdeeq

3.4.4 Data Flow Diagrams

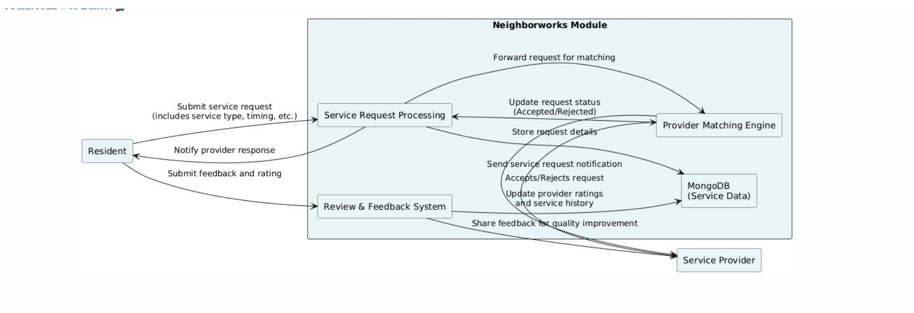

Neighborworks Module
Forward request for matching
Submit service request (includes service type, timing, etc.)
Service Request Processing
Update request status (Accepted/Rejected)
Provider Matching Engine
Store request details
Resident
Notify provider response
Submit feedback and rating
Send service request notification
Accepts/Rejects request
MongoDB (Service Data)
Update provider ratings and service history
Review & Feedback System
Share feedback for quality improvement
Service Provider

Figure 3.6: Level 2 Data Flow Diagram for Nazdeeq

Chapter 4

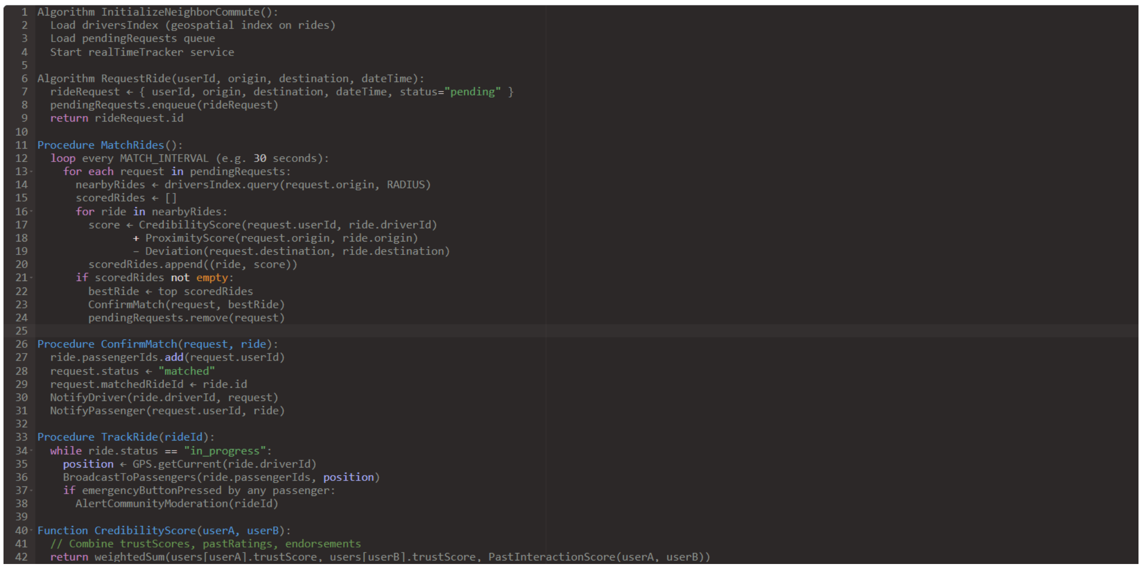
Implementation and Testing

4.1 Context and Design:

Nazdeeqs architecture is based on a microservice approach, with each module (NeighborCommute, NeighborWorks, VibeTribe, CommunityPulse, and Raabta) implemented as a separate service communicating via RESTful APIs. All data is stored in a MongoDB cluster, with collections for users, rides, services, groups, discussions, and chat sessions. The mobile UI is built in React Native, ensuring cross-platform consistency, and the AI components use TensorFlow and Hugging Face models hosted on serverless functions for scalability.

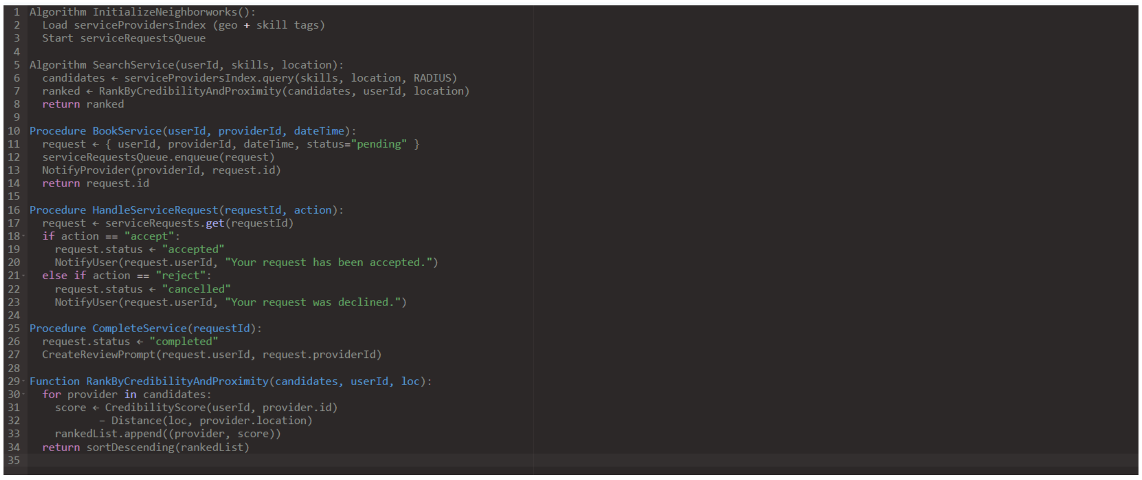
4.2 Algorithm Design

Below are the architecture patterns and algorithmic workflows designed for two core modules as part of FYPI.



```
Algorithm InitializeNeighborCommute():
  Load driversIndex (geospatial index on rides)
  Load pendingRequests queue
  Start realTimeTracker service

Algorithm RequestRide(userId, origin, destination, dateTime):
  rideRequest ← { userId, origin, destination, dateTime, status="pending" }
  pendingRequests.enqueue(rideRequest)
  return rideRequest.id

Procedure MatchRides():
  loop every MATCH_INTERVAL (e.g. 30 seconds):
    for each request in pendingRequests:
      nearbyRides ← driversIndex.query(request.origin, RADIUS)
      scoredRides ← []
      for ride in nearbyRides:
        score ← CredibilityScore(request.userId, ride.driverId)
               + ProximityScore(request.origin, ride.origin)
               - Deviation(request.destination, ride.destination)
        scoredRides.append((ride, score))
      if scoredRides not empty:
        bestRide ← top scoredRides
        ConfirmMatch(request, bestRide)
        pendingRequests.remove(request)

Procedure ConfirmMatch(request, ride):
  ride.passengerIds.add(request.userId)
  request.status ← "matched"
  request.matchedRideId ← ride.id
  NotifyDriver(ride.driverId, request)
  NotifyPassenger(request.userId, ride)

Procedure TrackRide(rideId):
  while ride.status == "in_progress":
    position ← GPS.getCurrent(ride.driverId)
    BroadcastToPassengers(ride.passengerIds, position)
    if emergencyButtonPressed by any passenger:
      AlertCommunityModeration(rideId)

Function CredibilityScore(userA, userB):
  // Combine trustScores, pastRatings, endorsements
  return weightedSum(users[userA].trustScore, users[userB].trustScore, PastInteractionScore(userA, userB))
```

Figure 4.1: NeighborCommute Algorithm Design Workflow



```
Algorithm InitializeNeighborworks():
  Load serviceProvidersIndex (geo + skill tags)
  Start serviceRequestsQueue

Algorithm SearchService(userId, skills, location):
  candidates ← serviceProvidersIndex.query(skills, location, RADIUS)
  ranked ← RankByCredibilityAndProximity(candidates, userId, location)
  return ranked

Procedure BookService(userId, providerId, dateTime):
  request ← { userId, providerId, dateTime, status="pending" }
  serviceRequestsQueue.enqueue(request)
  NotifyProvider(providerId, request.id)
  return request.id

Procedure HandleServiceRequest(requestId, action):
  request ← serviceRequests.get(requestId)
  if action == "accept":
    request.status ← "accepted"
    NotifyUser(request.userId, "Your request has been accepted.")
  else if action == "reject":
    request.status ← "cancelled"
    NotifyUser(request.userId, "Your request was declined.")

Procedure CompleteService(requestId):
  request.status ← "completed"
  CreateReviewPrompt(request.userId, request.providerId)

Function RankByCredibilityAndProximity(candidates, userId, loc):
  for provider in candidates:
    score ← CredibilityScore(userId, provider.id)
          - Distance(loc, provider.location)
    rankedList.append((provider, score))
  return sortDescending(rankedList)
```

Figure 4.2: NeighborWorks Algorithm Design Workflow

API/SDKs - Technical Components

Authentication & User Management

- Firebase Authentication
- Social login (Google, Apple, Facebook)
- Email/password login
- Token/session management

Conversational AI / Chatbot
- Google Gemini API or OpenAI API
- Conversational assistant for answering queries
- Module navigation via natural language
- Summarizing or interpreting discussions
- Generating suggestions or content replies

Cloud Backend / Hosting
- Firebase Firestore / Firebase Functions
- Scalable backend for storing forum posts, rides, profiles
- Trigger functions (e.g., notify users when a ride is posted)
- MongoDB Atlas (as vector database)
- Will be used as a vectorDB to store data embeddings for chatbot RAG purpose

Security
- JWT
- Used in server for stateless authentication

4.3 Unit Testing

Each unit test is designed to test a specific function or method independently from other components, helping to identify issues directly related to the functionality being tested.

Following are the unit test cases for the **NeighborCommute** and **Neighborworks** modules:

Test Case ID	Test Objective	Preconditions	Steps	Test Data	Expected Result	Post-condition	Actual Result	Pass/Fail
TC-NC-01	Verify user can search for available rides	User is logged in	Open app Go to NeighborCommute Enter location and time Tap 'Search'	Location: Islamabad Time: 8:00 AM	List of available rides is shown	Ride search results displayed on screen	As Expected	Pass
TC-NC-02	Verify user can post a ride offer	User is logged in	Go to Offer Ride Enter details and submit	From: Islamabad To: Rawalpindi Time: 9:00 AM	Ride offer saved successfully	Ride appears in active offers	As Expected	Pass
TC-NC-03	Verify recurring rides can be scheduled	User is logged in	Tap 'Schedule Recurring Ride' Select days and submit	Days: Mon-Fri Time: 8:30 AM	Confirmation of recurring rides saved	Rides visible in upcoming schedule	As Expected	Pass
TC-NW-01	Verify browsing of service providers	User is logged in	Go to Neighborworks Tap on a category	Category: Electricians	List of verified providers shown	Profiles loaded	As Expected	Pass
TC-NW-02	Verify booking a ser-	User is logged in	Select provider Tap 'Book'	Provider: Ahmed Time: 5:00	Booking confirmed mes-	Booking saved in user	As Expected	Pass

Bibliography

[1] Nextdoor. Nextdoor: The neighborhood network. https://nextdoor.com/, 2024.

[2] R. D. Putnam. *Bowling alone: The collapse and revival of American community*. Simon and Schuster, 2000.

[3] TaskRabbit. Taskrabbit: How it works. https://www.taskrabbit.com/how-it-works, 2024.

[4] Thumbtack. Thumbtack: Find local professionals. https://www.thumbtack.com/, 2024.

[5] A. Torre et al. Hyperlocal geosocial networking: A study of its impact on community engagement. *Journal of Urban Informatics*, 4(2):45–58, 2012.

[6] X. Zhu et al. Trust mechanisms in online marketplaces: Challenges and opportunities. *ACM Computing Surveys*, 50(2):1–37, 2018.