**NEARBUY PICK**

The app’s concept is for a person **traveling by bus or other transport**, and while they’re en route, they want to **buy items from a convenience store or hotel** along the way using the app.

kind of like ordering ahead from places you pass during your trip!

**Here’s how I imagine this app could work:**

**Key Concept:**

**Users order items from convenience stores or hotels located at stops or stations along their travel route. The app coordinates ordering and delivery so the user can pick up the items when the bus or transport stops there — or possibly have the items delivered to the bus if feasible.**

**Core Features:**

* **Route-Based Store Discovery**
  + User inputs their travel route or bus line.
  + The app shows convenience stores or hotels on or near that route.
* **Pre-Ordering Items**
  + User selects items from those stores/hotels in advance.
  + The app estimates the stop time and order readiness.
* **Pickup Coordination**
  + The app notifies the store/hotel about the user’s arrival time.
  + User picks up the order at the stop.
* **Live Tracking of Transport**
  + Optionally track the bus or vehicle to time order preparation.
* **Payment Integration**
  + Pay through the app in advance.
* **Notifications & Alerts**
  + Alerts to remind users when their stop is approaching and order is ready.

**User Flow Example:**

1. User enters travel info (e.g., bus route, bus number, start and end stops).
2. App lists stores/hotels on the route with available items.
3. User selects items, pays via the app.
4. Store receives order with estimated arrival time.
5. User boards bus, app tracks progress.
6. App alerts user when bus nears the store stop.
7. User picks up the items quickly and continues traveling.

**Additional questions:**

* Will delivery to the bus itself be needed, or only pickup at stops?
* Will stores/hotels manage their own menus and inventory in the app?
* Will the app integrate with transport schedules or GPS tracking for buses?

## 1. Main Screens / UI Flow

Here’s a rough flow for your app’s core user journey:

### Screen 1: Welcome / Login

* Sign up or log in (email, phone number)

### Screen 2: Enter Travel Info

* Input route details (bus number, origin, destination)
* Optionally, allow GPS tracking for live route detection

### Screen 3: Stores on Route

* List of convenience stores & hotels available on that route
* Show location, distance, ratings, estimated arrival time at each stop

### Screen 4: Store Menu / Items

* List of available products from selected store/hotel
* Ability to search, filter, add to cart

### Screen 5: Cart & Checkout

* Review items
* Select pickup stop (auto-suggest based on route)
* Payment options (card, wallet, etc.)
* Confirm order

### Screen 6: Order Tracking

* Show current bus location and estimated time to the pickup stop
* Show order status (preparing, ready for pickup, picked up)
* Notifications and alerts for pickup time

### Screen 7: Profile & Order History

* User info
* Past orders and ratings

## 2. Database Design

### Core Entities & Tables:

| **Table/Entity** | **Description** | **Key Fields** |
| --- | --- | --- |
| **Users** | Customer info and login credentials | user\_id, name, email, phone, password\_hash |
| **Routes** | Travel routes (bus number, stops, schedules) | route\_id, bus\_number, stops (list), timings |
| **Stores** | Convenience stores and hotels | store\_id, name, location, contact\_info |
| **Items** | Products offered by stores | item\_id, store\_id, name, price, category |
| **Orders** | User orders | order\_id, user\_id, store\_id, pickup\_stop, status, total\_price, payment\_status |
| **OrderItems** | Items in each order | order\_item\_id, order\_id, item\_id, quantity |
| **Stops** | Bus stops along routes | stop\_id, route\_id, stop\_name, arrival\_time |
| **Notifications** | Alerts for users and stores | notification\_id, user\_id/store\_id, message, timestamp |

## 3. System Architecture (High-level)

### Components:

* **Mobile App** (iOS/Android) for users
* **Store Portal** (web or mobile) for stores/hotels to receive & manage orders
* **Backend API** (RESTful) to handle requests, business logic, and database operations
* **Database** to store all data
* **Payment Gateway** integration
* **GPS & Maps API** for route & bus tracking
* **Push Notification Service** for alerts

### Flow:

User app <--> Backend API <--> Database  
User app <--> Maps/GPS API (bus tracking)  
Backend API <--> Payment Gateway  
Backend API <--> Store Portal (order notifications)  
Backend API <--> Push Notification Service

**Visual Mockups for Your Travel Delivery App**

**Screen 1: Welcome / Login**

**Screen 2: Enter Travel Info**

**Screen 3: Stores on Route**

**Screen 4: Store Menu / Items**

### ****Screen 5: Cart & Checkout****

### ****Screen 6: Order Tracking****

### ****Screen 7: Profile & Order History****

## Entities and Their Attributes

### 1. ****User****

| **Attribute** | **Description** |
| --- | --- |
| user\_id (PK) | Unique identifier for the user |
| name | User’s full name |
| email | User’s email (unique) |
| phone | User’s phone number (unique) |
| password\_hash | Hashed password |
| created\_at | Timestamp of account creation |

### 2. ****Route****

| **Attribute** | **Description** |
| --- | --- |
| route\_id (PK) | Unique identifier for the route |
| bus\_number | Identifier/number of the bus route |
| origin\_stop | Starting stop name |
| destination\_stop | Ending stop name |

### 3. ****Stop****

| **Attribute** | **Description** |
| --- | --- |
| stop\_id (PK) | Unique identifier for the stop |
| route\_id (FK) | Reference to associated route |
| stop\_name | Name of the stop |
| stop\_sequence | Order of stop in the route (e.g., 1, 2, 3) |
| estimated\_arrival | Estimated arrival time at this stop |

### 4. ****Store****

| **Attribute** | **Description** |
| --- | --- |
| store\_id (PK) | Unique identifier for the store |
| name | Store or hotel name |
| location | Physical address or coordinates |
| contact\_info | Phone/email or other contact details |
| rating | Average customer rating (decimal) |
| route\_id (FK) | Reference to the route where store is located |

### 5. ****Item****

| **Attribute** | **Description** |
| --- | --- |
| item\_id (PK) | Unique identifier for the item |
| store\_id (FK) | Reference to the store offering the item |
| name | Name of the item |
| category | Category/type (e.g., food, drink) |
| price | Price of the item |
| available | Boolean to indicate availability |

### 6. ****Order****

| **Attribute** | **Description** |
| --- | --- |
| order\_id (PK) | Unique identifier for the order |
| user\_id (FK) | Reference to the user placing order |
| store\_id (FK) | Store fulfilling the order |
| pickup\_stop\_id (FK) | Stop where user will pick up order |
| total\_price | Total price of the order |
| status | Current status (Pending, Ready, etc.) |
| payment\_status | Payment state (Paid, Unpaid, etc.) |
| created\_at | Timestamp when order was created |

### 7. ****OrderItem****

| **Attribute** | **Description** |
| --- | --- |
| order\_item\_id (PK) | Unique identifier for this record |
| order\_id (FK) | Reference to the order |
| item\_id (FK) | Reference to the item |
| quantity | Number of items ordered |

### 8. ****Notification**** (Optional)

| **Attribute** | **Description** |
| --- | --- |
| notification\_id (PK) | Unique identifier for the notification |
| user\_id (FK) | User receiving the notification |
| message | Notification text |
| created\_at | Timestamp of notification creation |
| read | Boolean if notification was read |

## 1. Live Bus Tracking

### How it works:

* **GPS Data Source:**  
  Each bus involved in your app’s routes would have a GPS device or mobile app sending its real-time location (latitude, longitude, timestamp) to a backend server.
* **Backend API:**  
  The backend exposes an API endpoint (e.g., /api/routes/:routeId/tracking) that returns the current position of the bus(es) on that route, plus estimated arrival times at stops.
* **Estimating Arrival Times:**  
  Using the bus’s speed and location data plus the fixed stop locations on the route, the backend calculates ETA for each stop.
* **Client App:**  
  The app periodically fetches the tracking API (e.g., every 15-30 seconds) or opens a WebSocket connection to receive live location updates of the bus.
* **Map Display:**  
  The user sees the bus moving on a map in real-time, helping them know how close the bus is to their pickup stop.

### Technologies involved:

* GPS hardware or mobile device sending location data (via MQTT, HTTP, or WebSocket).
* Backend with a real-time data ingestion service.
* WebSocket or Server-Sent Events (SSE) for pushing real-time updates to clients.
* Map APIs like Google Maps or Mapbox for displaying location visually.

## 2. Notifications

### Use Cases:

* Order status updates (e.g., “Your order is being prepared”, “Ready for pickup”).
* Bus arrival alerts (“Bus arriving at your stop in 3 minutes”).
* Promotions or app announcements.

### How it works:

* **Trigger Events:**  
  Backend triggers notifications based on events such as order status changes or bus proximity.
* **Notification Storage:**  
  Notifications are stored in the database linked to user accounts.
* **Delivery:**
  + **Push Notifications:** Use services like Firebase Cloud Messaging (FCM) or Apple Push Notification Service (APNs) to send real-time alerts to users’ devices.
  + **In-App Notifications:** Display notifications inside the app, fetched from the backend via API.
* **Read Status:**  
  Users can mark notifications as read, and the backend updates notification status.

### Technologies involved:

* Push notification services (FCM, APNs).
* Backend event system (e.g., Node.js event emitters, message queues like RabbitMQ or Kafka).
* Database to store and track notifications.
* Client-side SDK integration to receive and display notifications.

## Summary Flow Example:

1. Bus GPS updates location every few seconds → backend updates position and ETA.
2. User’s app requests bus location or subscribes via WebSocket → app displays bus on map.
3. When bus nears user’s pickup stop → backend triggers “bus arriving soon” notification → sent via push or in-app notification.
4. Order status changes → backend sends notification to user.

### 1. Detecting Bus Proximity on Mobile App

### Your app regularly fetches bus location and ETA via your backend API or listens to real-time updates (WebSocket).

### The app compares the bus’s current location or ETA to the user’s pickup stop.

### When the bus is within a certain distance threshold (e.g., 500 meters) or ETA is below a set time (e.g., 3 minutes), the app triggers the notification.

### 2. Triggering Vibration and Voice Alerts

### On Mobile (Web or Native):

### Vibration:

### On native apps (iOS/Android), you can use vibration APIs.

### On web apps, the Vibration API can trigger vibration if supported.

### Voice Notification:

### Use the Web Speech API’s speechSynthesis for text-to-speech on web.

### On native, use platform TTS libraries.