LiftTogether - Design Document

📋 Project Overview

Project Name: LiftTogether
Version: 1.0.0
Platform: Android (Kotlin + Jetpack Compose)
Architecture: MVVM with Repository Pattern
Backend: Firebase Authentication + Mock Data System

🎯 Mission Statement

LiftTogether is a community-driven ride-sharing application that connects riders in need with volunteer drivers, prioritizing requests based on urgency levels. The app fosters community support by enabling volunteers to provide transportation assistance to those who need it most.

🎨 User Interface Design

Design Principles

- **Accessibility First:** Clear, large touch targets, high contrast
- **Urgency-Based UI:** Visual hierarchy emphasizes urgent requests
- **Community Feel:** Warm, supportive color palette
- **Simplicity:** Minimal steps to complete core actions

Color Scheme

Primary	Secondary	Success	Warning	Error
#6200EE	#03DAC6	#4CAF50	#FF9800	#F44336
Trust, reliability	Community, support	Completed rides	High priority	Emergency, cancellations

Background #FFFFFF

Clean, accessible

📱 Screen Specifications

1. Authentication Screens

Login Screen

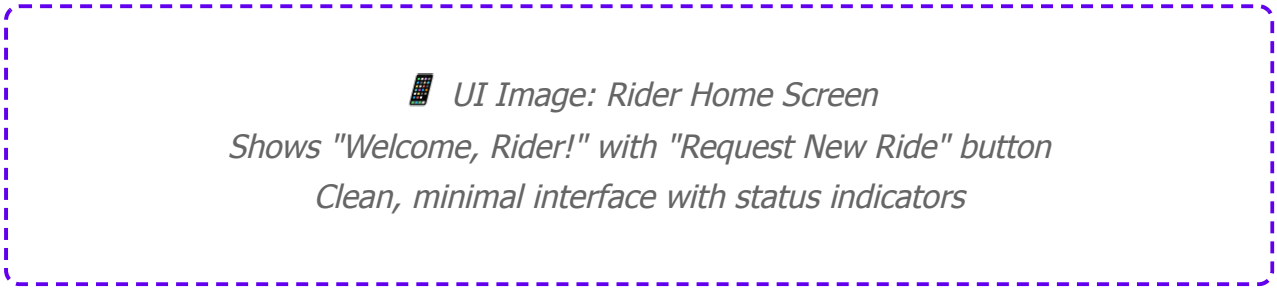
Purpose: User authentication entry point

Components:

- App logo and branding
- Email/phone input field
- Password input field
- User type selection (Rider/Volunteer)
- Login/Sign Up toggle
- Authentication mode indicator
- Mock user help (when Firebase unavailable)

2. Rider Screens

Rider Home Screen

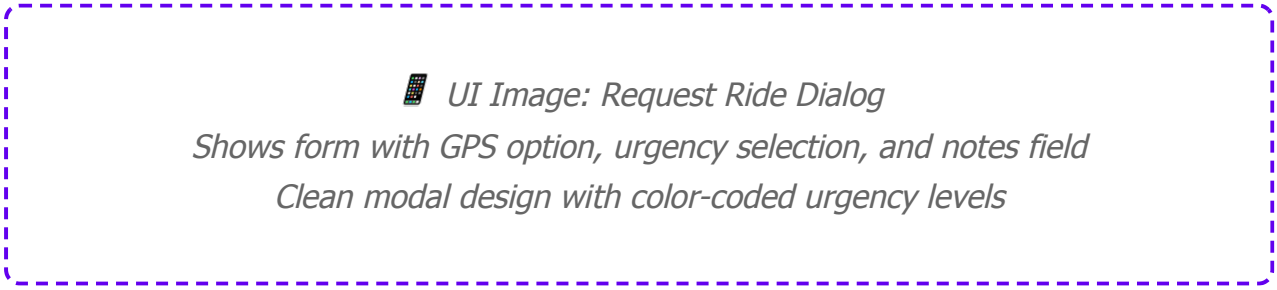


Purpose: Main dashboard for riders

Key Features:

- Welcome message with logout button
- Ride requests list
- Request new ride button (disabled when active)
- Status indicators
- Help text for active rides

Request Ride Dialog

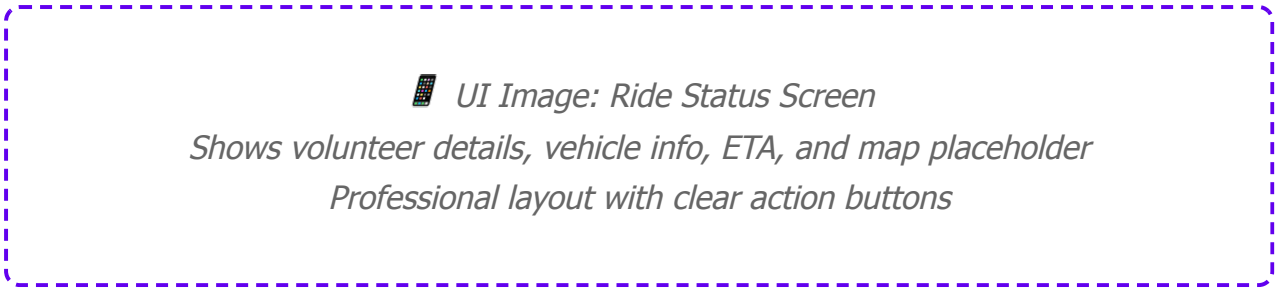


Purpose: Create new ride requests

Key Features:

- GPS integration option
- Color-coded urgency levels (Emergency=Red, High=Orange, Medium=Blue, Low=Green)
- Form validation
- Clean modal design

Ride Status Screen



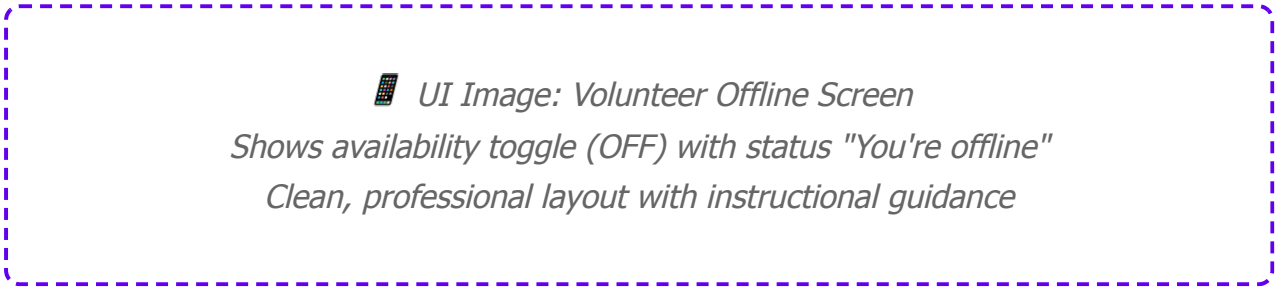
Purpose: Track active ride

Key Features:

- Professional volunteer info display
- Real-time ETA updates
- Map integration placeholder
- Clear action buttons

3. Volunteer Screens

Volunteer Home Screen (Offline)

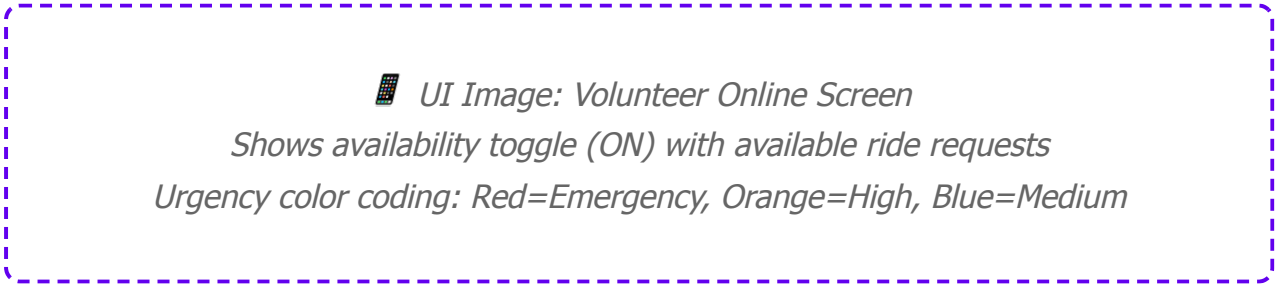


Purpose: Main dashboard for volunteers when offline

Key Features:

- Clear availability toggle
- Status messaging
- Instructional guidance
- Clean, professional layout

Volunteer Home Screen (Online)

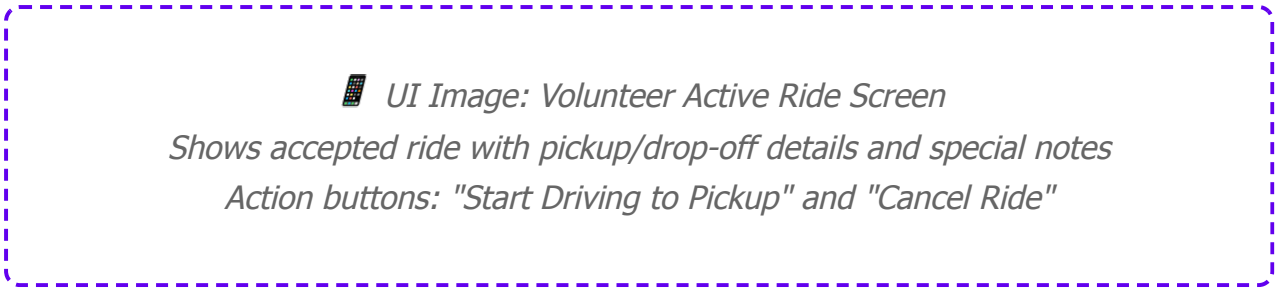


Purpose: Display available ride requests

Key Features:

- Urgency color coding (Red=Emergency, Orange=High, Blue=Medium)
- Clear request details
- Prominent action buttons
- Professional card layout

Active Ride Management



Purpose: Manage accepted rides

Key Features:

- Clear ride information
- Action-oriented buttons
- Status tracking
- Professional layout

🔧 Core Features

Authentication System

Firebase Authentication (Primary)

- Email/password authentication
- User registration
- Session management
- Automatic login persistence
- Secure logout

Mock Authentication (Fallback)

- Offline authentication
- Pre-configured test users
- Same UI/UX as Firebase
- Automatic fallback detection

Ride Request System

- Pickup Location: GPS detection + manual entry
- Drop-off Location: Manual entry with validation
- Urgency Levels: Emergency, High, Medium, Low
- Notes: Optional special requirements
- Single Active Ride: Only one ride request at a time
- Status Tracking: Real-time status updates

Volunteer System

- Availability Toggle: On/off availability
- Status Display: Clear availability indicator
- Request List: Sorted by urgency and proximity
- Request Details: Full information display
- Accept/Decline: Simple action buttons
- Status Updates: Progress tracking

🗄️ Data Models

Core Data Classes

```
 // User Model
data class User(
    val id: String,
    val email: String,
    val userType: String, // "Rider" or "Volunteer"
    val name: String?,
    val phone: String?
)

// Ride Request Model
data class RideRequest(
    val id: String,
    val riderId: String,
    val pickupLocation: String,
    val dropoffLocation: String,
    val urgency: UrgencyLevel,
    val notes: String?,
    val timestamp: Long,
    val status: RideStatusType
)

// Volunteer Model
data class Volunteer(
    val id: String,
    val name: String,
    val vehicleInfo: VehicleInfo,
    val rating: Float,
    val isAvailable: Boolean
)
```

Enums

```
 enum class UrgencyLevel(val displayName: String) {
    EMERGENCY("Emergency"),
    HIGH("High"),
```

```
    MEDIUM("Medium"),
    LOW("Low")
}

enum class RideStatusType {
    PENDING, ASSIGNED, ON_THE_WAY,
    ARRIVED, IN_PROGRESS, COMPLETED, CANCELLED
}
```

🧪 Testing Strategy

Mock Data System

Complete App Testing: All features work without backend

Pre-configured Test Users:

- `manoj@gmail.com` (Rider) - Password: `password123`
- `user1@gmail.com` (Volunteer) - Password: `password123`
- `rider@gmail.com` (Rider) - Password: `password123`
- `volunteer@gmail.com` (Volunteer) - Password: `password123`
- `test@gmail.com` (Rider) - Password: `password123`

Test Scenarios

1. **Authentication Flow**
    - Firebase login success/failure
    - Mock authentication fallback
    - User type detection
    - Session persistence
2. **Ride Request Flow**
    - Create ride request
    - Form validation
    - Single ride limitation
    - Status updates

3. **Volunteer Flow**
    - Availability toggle
    - Request acceptance
    - Status management
    - Ride completion

🚀 Performance Considerations

UI Performance

- **Lazy Loading:** LazyColumn for ride lists
- **State Optimization:** Minimal recomposition
- **Image Loading:** Placeholder for future map integration
- **Smooth Animations:** Material Design transitions

Memory Management

- **State Cleanup:** Proper state disposal
- **List Optimization:** Efficient list rendering
- **Background Tasks:** Proper coroutine management

🔮 Future Enhancements

Phase 2 Features

- **Real-time Updates:** WebSocket integration
- **Push Notifications:** Ride status updates
- **Maps Integration:** Google Maps/OpenStreetMap
- **Payment System:** Optional donations
- **Rating System:** User feedback

Phase 3 Features

- **Advanced Matching:** AI-powered volunteer matching
- **Route Optimization:** Efficient pickup/drop-off routes
- **Analytics:** Usage statistics and insights
- **Multi-language:** Internationalization
- **Accessibility:** Enhanced accessibility features

📊 Success Metrics

User Engagement

- **Daily Active Users:** Track app usage
- **Ride Completion Rate:** Successful ride percentage
- **User Retention:** Monthly active users
- **Volunteer Participation:** Active volunteer count

Technical Metrics

- **App Performance:** Load times, responsiveness
- **Crash Rate:** Stability monitoring
- **Authentication Success:** Login success rate
- **Feature Usage:** Most used features

📝 Conclusion

LiftTogether represents a community-focused approach to ride-sharing, prioritizing accessibility and volunteer engagement. The dual authentication system ensures reliability, while the mock data system enables comprehensive testing and development.

The app's design emphasizes simplicity and accessibility, making it easy for users of all technical levels to request and provide transportation assistance. The modular architecture allows for future enhancements while maintaining the core community-driven mission.

Key Success Factors:

- Reliable authentication system
- Intuitive user interface

- Community-focused features
- Robust testing framework
- Scalable architecture

Document Version: 1.0 | Last Updated: December 2024 | Author: Development Team