Location-Based Social App: Technical Architecture & Implementation

Architecture Overview

```
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────
┐
│  Flutter App    │   │  Flutter App    │   │  Flutter App    │
│   (User A)      │   │   (User B)      │   │   (User C)      │
└────────┬────────┘   └────────┬────────┘   └────────┬────────
┘
         │                     │                     │
         └─────────────────────┼─────────────────────┘
                               │
                    ┌──────────▼──────────┐
                    │   Load Balancer     │
                    │   (nginx/AWS)       │
                    └──────────┬──────────┘
                               │
         ┌─────────────────────┼─────────────────────┐
         │                     │                     │
┌────────▼────────┐   ┌────────▼────────┐   ┌────────▼────────
┐
│ API Gateway     │   │  WebSocket      │   │  Location       │
│ (Room CRUD)     │   │  Server         │   │  Service        │
│                 │   │  (Real-time)    │   │  (Positioning)  │
└────────┬────────┘   └────────┬────────┘   └────────┬────────
┘
         │                     │                     │
         └─────────────────────┼─────────────────────┘
                               │
                    ┌──────────▼──────────┐
                    │   Message Queue     │
                    │   (Redis)           │
                    └──────────┬──────────┘
```

```
                  │
         ┌──────────────────────┼──────────────────────┐
         │          │          │
    ┌────────▼────────┐ ┌──────▼──────┐ ┌─────────▼─────────┐
    │   PostgreSQL   │ │   Redis   │ │   Notification  │
    │   (PostGIS)   │ │  (Cache)  │ │   Service     │
    │           │ │       │ │          │
    └─────────────────┘ └─────────────┘ └───────────────────┘
```

Core Components Architecture

1. Client Layer (Flutter)

```
// Core Location Service
class LocationService {
  static const locationSettings = LocationSettings(
    accuracy: LocationAccuracy.best,
    distanceFilter: 10, // meters
  );

  Stream<Position> getLocationStream() {
    return Geolocator.getPositionStream(
      locationSettings: locationSettings,
    );
  }

  Future<Position> getCurrentLocation() async {
    return await Geolocator.getCurrentPosition();
  }
}

// Room Management
class RoomService {
  final WebSocketChannel _channel;

  void joinRoom(String roomId, Position userLocation) {
    final message = {
      'action': 'join_room',
      'roomId': roomId,
      'location': {
        'lat': userLocation.latitude,
        'lng': userLocation.longitude,
      },
      'timestamp': DateTime.now().toIso8601String(),
    };
    _channel.sink.add(json.encode(message));
  }

  void updateLocation(Position location) {
    final message = {
      'action': 'location_update',
      'location': {
        'lat': location.latitude,
        'lng': location.longitude,
        'accuracy': location.accuracy,
      },
      'timestamp': DateTime.now().toIso8601String(),
    };
    _channel.sink.add(json.encode(message));
  }
}

// Real-time State Management
class RoomBloc extends Bloc<RoomEvent, RoomState> {
  final LocationService _locationService;
  final RoomService _roomService;

  StreamSubscription? _locationSubscription;

  void _onJoinRoom(JoinRoom event, Emitter<RoomState> emit) {
    _locationSubscription = _locationService
        .getLocationStream()
        .listen((position) {
      _roomService.updateLocation(position);
    });

    _roomService.joinRoom(event.roomId, event.initialLocation);
    emit(RoomJoined(roomId: event.roomId));
  }
}
```

2. Backend Services (Node.js + TypeScript)

WebSocket Server

```
// Real-time Communication Server
class WebSocketServer {
  private io: Server;
  private rooms: Map<string, RoomManager> = new Map();

  constructor(server: any) {
    this.io = new Server(server, {
      cors: { origin: "*" },
      pingTimeout: 60000,
    });

    this.setupHandlers();
  }

  private setupHandlers() {
    this.io.on('connection', (socket: Socket) ⇒ {
      console.log(`User connected: ${socket.id}`);

      socket.on('join_room', async (data) ⇒ {
        await this.handleJoinRoom(socket, data);
      });

      socket.on('location_update', async (data) ⇒ {
        await this.handleLocationUpdate(socket, data);
      });

      socket.on('disconnect', () ⇒ {
        this.handleUserDisconnect(socket);
      });
    });
  }
```

```
private async handleJoinRoom(socket: Socket, data: any) {
  const { roomId, location, userId } = data;

  // Get or create room manager
  if (!this.rooms.has(roomId)) {
    this.rooms.set(roomId, new RoomManager(roomId));
  }

  const room = this.rooms.get(roomId)!;
  await room.addUser(socket, userId, location);

  socket.join(roomId);

  // Notify other users
  socket.to(roomId).emit('user_joined', {
    userId,
    location,
    timestamp: new Date(),
  });
}

private async handleLocationUpdate(socket: Socket, data: any) {
  const user = await this.getUserFromSocket(socket);
  if (!user) return;

  const { location } = data;

  // Update location in database
  await LocationService.updateUserLocation(user.id, location);

  // Broadcast to room members
  socket.to(user.currentRoom).emit('location_updated', {
    userId: user.id,
    location,
    timestamp: new Date(),
  });

  // Check geofence boundaries
  await this.checkGeofenceBoundaries(user, location);
  }
}
```

Room Manager

```
class RoomManager {
  private roomId: string;
  private users: Map<string, UserSession> = new Map();
  private boundary: GeoJSON.Polygon;

  constructor(roomId: string) {
    this.roomId = roomId;
    this.loadRoomData();
  }

  async addUser(socket: Socket, userId: string, location: Location) {
    // Check if user is within room boundaries
    const isWithinBounds = await this.isLocationWithinBounds(location);
    if (!isWithinBounds) {
      socket.emit('error', { message: 'Outside room boundaries' });
      return;
    }

    const userSession: UserSession = {
      userId,
      socketId: socket.id,
      location,
      joinedAt: new Date(),
      lastSeen: new Date(),
    };

    this.users.set(userId, userSession);

    // Send current room state to new user
    socket.emit('room_state', {
      roomId: this.roomId,
      users: Array.from(this.users.values()),
      boundary: this.boundary,
    });
  }

  private async isLocationWithinBounds(location: Location): Promise<boolean> {
    // Use PostGIS for precise boundary checking
    const query = `
      SELECT ST_Contains(
        ST_GeomFromGeoJSON($1),
        ST_Point($2, $3)
      ) as within_bounds
    `;

    const result = await db.query(query, [
      JSON.stringify(this.boundary),
      location.lng,
      location.lat,
    ]);

    return result.rows[0].within_bounds;
  }
}
```

Location Service

```
class LocationService {
  static async updateUserLocation(userId: string, location: Location) {
    // Update real-time location in Redis
    await redisClient.hset(
      `user:${userId}:location`,
      {
        lat: location.lat,
        lng: location.lng,
        accuracy: location.accuracy,
        timestamp: Date.now(),
      }
    );

    // Add to location history in PostgreSQL
    await db.query(
      `INSERT INTO location_history (user_id, location, timestamp)
```

```
     VALUES ($1, ST_Point($2, $3), $4)`,
    [userId, location.lng, location.lat, new Date()]
  );
}

static async getUsersInProximity(
  location: Location,
  radiusMeters: number
): Promise<User[]> {
  const query = `
    SELECT u.id, u.username,
           ST_X(lh.location) as lng,
           ST_Y(lh.location) as lat,
           ST_Distance(
             ST_Point($1, $2)::geography,
             lh.location::geography
           ) as distance
    FROM users u
    JOIN LATERAL (
      SELECT location
      FROM location_history
      WHERE user_id = u.id
      ORDER BY timestamp DESC
      LIMIT 1
    ) lh ON true
    WHERE ST_DWithin(
      ST_Point($1, $2)::geography,
      lh.location::geography,
      $3
    )
    ORDER BY distance;
  `;

  const result = await db.query(query, [
    location.lng,
    location.lat,
    radiusMeters,
  ]);

  return result.rows;
  }
}
```

3. Database Schema

```
-- PostgreSQL with PostGIS Extension

-- Users table
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  created_at TIMESTAMP DEFAULT NOW(),
  last_active TIMESTAMP DEFAULT NOW()
);

-- Rooms table
CREATE TABLE rooms (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name VARCHAR(100) NOT NULL,
  description TEXT,
  creator_id UUID REFERENCES users(id),
  boundary GEOMETRY(POLYGON, 4326), -- GeoJSON polygon
  max_participants INTEGER DEFAULT 50,
  is_public BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT NOW(),
  expires_at TIMESTAMP
);

-- Room participants
CREATE TABLE room_participants (
  room_id UUID REFERENCES rooms(id),
  user_id UUID REFERENCES users(id),
  joined_at TIMESTAMP DEFAULT NOW(),
  left_at TIMESTAMP,
  is_active BOOLEAN DEFAULT true,
  PRIMARY KEY (room_id, user_id)
);
```

```
-- Location history
CREATE TABLE location_history (
  id BIGSERIAL PRIMARY KEY,
  user_id UUID REFERENCES users(id),
  location GEOMETRY(POINT, 4326),
  accuracy FLOAT,
  timestamp TIMESTAMP DEFAULT NOW()
);

-- Spatial indexes for performance
CREATE INDEX idx_rooms_boundary ON rooms USING GIST (boundary);
CREATE INDEX idx_location_history_location ON location_history USING GIST (location);
CREATE INDEX idx_location_history_user_time ON location_history (user_id, timestamp DESC);
```

Data Flow Logic

1. Room Creation Flow

```
User Creates Room → Validate Location → Store in PostgreSQL →
Create Geofence → Notify Nearby Users → Return Room ID
```

2. Join Room Flow

```
User Requests Join → Check Location vs Geofence →
Add to WebSocket Room → Update Participants Table →
Broadcast User Joined → Send Room State
```

3. Real-time Location Updates

```
Client Location Change → Send to WebSocket →
Update Redis Cache → Broadcast to Room →
Store in PostgreSQL (batched) → Check Boundaries
```

4. Geofence Monitoring

```
Location Update → PostGIS Boundary Check →
If Outside: Trigger Leave Event →
Notify User & Room → Update Status
```

Key Implementation Considerations

Battery Optimization

```
class AdaptiveLocationService {
  LocationAccuracy getLocationAccuracy() {
    // Adapt based on user activity
    if (isStationary) return LocationAccuracy.low;
    if (isWalking) return LocationAccuracy.medium;
    return LocationAccuracy.best;
  }

  Duration getUpdateInterval() {
    if (isStationary) return Duration(minutes: 2);
    if (isWalking) return Duration(seconds: 30);
    return Duration(seconds: 10);
  }
}
```

Error Handling & Offline Support

```
class OfflineLocationQueue {
  final List<PendingLocationUpdate> _queue = [];

  void queueUpdate(Location location) {
    _queue.add(PendingLocationUpdate(location, DateTime.now()));
  }

  Future<void> syncPendingUpdates() async {
    for (final update in _queue) {
      try {
        await LocationAPI.updateLocation(update.location);
        _queue.remove(update);
      } catch (e) {
        // Retry later
      }
    }
  }
}
```

Security & Privacy

```
// Input validation middleware
function validateLocation(req: Request, res: Response, next: NextFunction) {
  const { lat, lng } = req.body.location;

  if (!isValidCoordinate(lat, lng)) {
    return res.status(400).json({ error: 'Invalid coordinates' });
  }

  // Rate limiting check
  if (isRateLimited(req.user.id)) {
    return res.status(429).json({ error: 'Too many updates' });
  }

  next();
}

// Data anonymization for analytics
function anonymizeLocation(location: Location): AnonymizedLocation {
  return {
    // Reduce precision to ~100m
    lat: Math.round(location.lat * 1000) / 1000,
    lng: Math.round(location.lng * 1000) / 1000,
    timestamp: Math.floor(Date.now() / 300000) * 300000, // 5-min buckets
  };
}
```

This architecture provides scalable real-time location sharing with efficient data storage, privacy controls, and battery optimization. The key is balancing real-time performance with resource consumption while maintaining precise location accuracy for room boundaries.