GroMotion Mobile App - Technical Architecture Documentation

Executive Summary

GroMotion is a React-based mobile fitness application that gamifies physical activity through AR (Augmented Reality) experiences, energy generation tracking, and community challenges. The app uses Capacitor for mobile deployment, Firebase Firestore for data persistence, and web-based AR technologies for immersive user experiences.

Key Finding: The application does not contain actual AI/ML implementations. The "AI" functionality consists of rule-based logic with static fake data generation, not machine learning algorithms.

---

1. AI System Architecture

1.1 AI Role and Logic

Reality Check: The application contains NO actual artificial intelligence or machine learning implementations. The "AI" referenced in the codebase is:

- **Rule-based logic** with random number generation
- **Static fake data** that never changes or learns
- **Simple mathematical calculations** (steps × calories conversion)
- **Predefined thresholds** for user progress evaluation

1.2 Fake Data Generation System

Location: `src/services/fakeData.ts`

The app generates all user data through static functions:

```
export const fakeUsers = Array.from({ length: 10 }, (_, i) => {
  const stepsToday = rnd(2000, 10000);
  const energyGenerated = rfloat(0.5, 5.0, 2);
  return {
    id: i === 0 ? "demo-user" : `user-${i + 1}`,
    energyGenerated,
    co2Saved: rfloat(0.1, 3.0, 2),
    calories: Math.round(stepsToday * 0.05),
    points: coins * 10,
  };
});
```

Data Types Generated:

- User profiles with random stats
- Step history with fabricated dates
- Leaderboard rankings
- Challenge data
- Energy generation metrics

1.3 "AI" Evaluation Logic

Location: `src/services/fake.ts`

The "AI evaluation" is a simple threshold check:

```
export const evaluateUserWithAI = (userId: string) => {
  const stats = getUserStats(userId);

  // Simple threshold-based "AI" - NOT machine learning
  if (stats.energyKwh > 2.0) return "excellent";
  if (stats.energyKwh > 1.0) return "good";
  return "needs_improvement";
};
```

Evaluation Criteria:

- Energy > 2.0 kWh = "Excellent"
- Energy > 1.0 kWh = "Good"
- Energy ≤ 1.0 kWh = "Needs Improvement"

---

2. Database Structure and Technologies

2.1 Firebase Firestore Architecture

Primary Database: Firebase Firestore (NoSQL Document Database)
Location: `src/lib/firebase.ts`

```
const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID,
};
```

2.2 Data Collections Structure

Users Collection (`users/{userId}`)

```
interface UserStats {
  calories: number;
  points: number;
  energyKwh: number;
  co2SavedKg: number;
  todaySteps: number;
  goalSteps: number;
}
```

Step History Collection (`steps_history/{historyId}`)

```
interface HistoryEntry {
  userId: string;
  date: string;
  distanceKm: number;
  kcal: number;
  steps: number;
  bpm?: number;
  timestamp: Timestamp;
}
```

Challenges Collection (`challenges/{challengeId}`)

- Stores community challenges and competitions
- Ordered by start date

Leaderboard Collection (`leaderboard/{userId}`)

- User rankings by points
- Ordered by descending points

AR Sessions Collection (`ar_sessions/{sessionId}`)

```
interface ARSession {
  userId: string;
  duration: number;
  distance: number;
  coinsCollected: number;
  energyGenerated: number;
  createdAt: Timestamp;
}
```

2.3 Data Fallback Strategy

Critical Architecture Decision: The app implements a dual-data strategy:

1. **Primary**: Firebase Firestore real-time listeners
2. **Fallback**: Static fake data when backend unavailable

Implementation Pattern:

```
export const listenUserStats = (userId: string, cb: (s: UserStats) => void) => {
  const ref = doc(db, "users", userId);
  return onSnapshot(ref, (snap) => {
    const d = snap.data();
    if (!d) {
      cb(getUserStats(userId)); // FALLBACK TO FAKE DATA
      return;
    }
    // Use real data from Firestore
    cb({
      calories: d.calories || 0,
      points: d.points || 0,
      energyKwh: d.energyKwh || 0,
      // ... map real data
    });
  }, (_err) => {
    cb(getUserStats(userId)); // ERROR FALLBACK TO FAKE DATA
  });
};
```

---

3. API Layer and Endpoints

3.1 API Architecture

API Base URL: Configurable via environment variable

- Development: `http://localhost:4000`
- Production: `VITE_API_URL` environment variable

Primary API Endpoint:

```
POST ${API_URL}/api/ar/sessions
```

3.2 AR Session Logging API

Purpose: Records AR session analytics for user tracking

Request Format:

```
{
  userId: string;
  duration: number;      // Session duration in seconds
  distance: number;      // Distance traveled in meters
  coinsCollected: number;
  energyGenerated: number;
  timestamp: string;
}
```

Error Handling Strategy:

1. **Primary**: Send to backend API
2. **Fallback**: Store in local Firestore collection
3. **Offline**: Queue for later sync (not implemented)

3.3 Real-time Data Listeners

Firebase Real-time Listeners:

```
// User stats listener
listenUserStats(userId, callback)

// History preview listener (last 3 entries)
listenUserHistoryPreview(userId, callback)

// Full history listener
listenUserHistoryFull(userId, callback)

// Challenges listener
listenChallenges(callback)

// Leaderboard listener
listenLeaderboard(callback)
```

---

4. Complete Project Workflow

4.1 User Onboarding Flow

```
Splash Screen → Onboarding Steps → Welcome → Authentication → Profile Setup → Main App
```

Onboarding Steps:

1. **Splash Screen**: Brand introduction
2. **Onboarding Steps**: Feature demonstration (3 steps)
3. **Welcome**: Value proposition
4. **Authentication**: Login/Register/Forgot Password
5. **Profile Setup**: Gender → Age → Weight → Height → Community
6. **Main App**: Home dashboard

4.2 Main Application Flow

Navigation Structure:

```
Home (Dashboard)
├── History (Activity Log)
├── Challenges (Community Competitions)
├── Map/AR (AR Experience)
├── Profile (User Settings)
└── Leaderboard (Rankings)
```

4.3 AR Session Workflow

Pre-AR:

1. User navigates to Map/AR tab
2. Views 2D map with route preview
3. Taps "Start AR Session"
4. Confirms readiness in dialog

During AR:

1. Camera permission request
2. AR engine initialization
3. Location services activation
4. Real-time coin collection
5. Quiz triggers every 5 coins
6. Energy generation tracking
7. Distance/time monitoring

Post-AR:

1. Session summary display
2. Data sync to backend/Firestore
3. Achievement notifications
4. Return to map view

4.4 Data Synchronization Strategy

Real-time Sync: Firebase listeners update UI immediately
Session Data: AR sessions logged via API endpoint
Offline Capability: Limited - falls back to fake data
Conflict Resolution: Last-write-wins (Firebase default)

---

5. AR System Integration

5.1 AR Technology Stack

Core Technologies:

- **WebRTC**: Camera access via `navigator.mediaDevices.getUserMedia()`
- **WebGL**: 3D rendering through HTML5 Canvas
- **Geolocation API**: GPS positioning via `navigator.geolocation`
- **Device Orientation**: Accelerometer/gyroscope (not implemented)

5.2 AR Engine Architecture

Location: `src/native/AREngine.ts`

Initialization Sequence:

```
const arConfig: ARConfig = {
  enableCamera: true,
  enableLocation: true,
  enablePlaneDetection: true,
  enableHitTesting: true,
};
await arEngine.initialize(arConfig);
```

Camera Pipeline:

1. Request rear-facing camera (`facingMode: 'environment'`)
2. Set ideal resolution (1920×1080)
3. Create video element for camera stream

4. Position as background layer

Rendering Pipeline:

1. **Background Layer**: Live camera feed
2. **AR Overlay Layer**: WebGL canvas with transparency
3. **UI Layer**: React components and HUD

5.3 3D Rendering System

WebGL Implementation:

- Canvas positioned absolutely over video
- 100% viewport coverage
- Transparent background (`gl.clearColor(0, 0, 0, 0)`)
- Real-time rendering at 60 FPS

AR Path Rendering:

```
private renderARPath(): void {
  // Create perspective-correct path geometry
  const pathGeometry = this.createPathGeometry(this.arPath);

  // Render green lanes with transparency
  this.renderPathLanes(pathGeometry);

  // Add directional arrows
  this.renderPathArrows(pathGeometry);
}
```

Coin Collection System:

- Virtual coins positioned along AR path
- 3D world coordinates converted to 2D screen space
- Distance-based collision detection (0.5m radius)
- Pulsing animation with glow effects
- Lightning bolt icon overlay

5.4 Location and Tracking

GPS Integration:

```
navigator.geolocation.watchPosition(
  (position) => {
    this.userPosition = {
      latitude: position.coords.latitude,
      longitude: position.coords.longitude,
      accuracy: position.coords.accuracy
    };
  },
  {
    enableHighAccuracy: true,
    timeout: 5000,
    maximumAge: 0
  }
);
```

Position Tracking:

- High accuracy GPS mode
- 5-second update intervals
- Real-time position updates
- Accuracy validation (not implemented)

5.5 Quiz Integration System

Trigger Mechanism: Every 5 coins collected
Question Source: Static predefined questions
Display Method: Modal overlay during AR session
Reward System: Bonus points for correct answers

5.6 Session Management

Start Process:

1. Initialize camera and location services
2. Generate AR path and coin placement
3. Begin rendering loop
4. Start timers and metrics

Stop Process:

1. Cancel animation frame loop
2. Stop camera tracks
3. Remove DOM elements
4. Calculate final statistics
5. Sync session data

---

6. Mobile Deployment Architecture

6.1 Capacitor Framework Integration

Platform Support: iOS and Android via Capacitor
Web View: Native webview wrapper around React app
Native Features: Camera, GPS, file system access

6.2 APK Build Process

Build Command: `npm run build:android`
Output: Signed APK ready for distribution
Requirements: Android SDK, Java JDK, Keystore for signing

6.3 Performance Considerations

Web-based AR Limitations:

- No native ARKit/ARCore integration
- Limited 3D tracking capabilities
- Battery intensive camera usage
- Performance depends on device hardware

Optimization Strategies:

- 60 FPS rendering target
- Efficient WebGL state management
- Canvas size optimization
- Memory cleanup on session end

---

7. Security and Data Privacy

7.1 Data Protection

Local Storage: User preferences stored in browser localStorage
Firebase Security: Rules-based access control (not implemented)
API Security: Basic HTTP requests (no authentication)
Camera Privacy: Explicit user permission required

7.2 Identified Security Gaps

1. **No API Authentication**: AR session endpoint unsecured
2. **No Data Validation**: Client-side data accepted without verification
3. **Local Storage Exposure**: Sensitive data stored in plain text
4. **No Rate Limiting**: API endpoints vulnerable to abuse
5. **Firebase Rules**: No security rules implemented

---

8. Technical Limitations and Recommendations

8.1 Current Limitations

AI System: No actual machine learning implementation
AR Technology: Web-based only, no native AR frameworks
Data Quality: Static fake data, no real user analytics
Backend: Minimal API layer, mostly Firebase-dependent
Offline Support: Limited offline functionality

8.2 Recommended Improvements

AI Implementation:

- Integrate actual ML models for user behavior analysis
- Implement predictive analytics for energy generation
- Add personalized workout recommendations
- Create adaptive difficulty scaling

AR Enhancement:

- Migrate to native ARKit (iOS) and ARCore (Android)
- Implement proper 3D object tracking
- Add environmental understanding
- Improve surface detection and plane estimation

Data Architecture:

- Replace fake data with real user data collection
- Implement proper data validation and sanitization
- Add data analytics and reporting capabilities
- Create data backup and recovery systems

Security Hardening:

- Implement API authentication and authorization
- Add rate limiting and DDoS protection
- Secure Firebase with proper security rules
- Encrypt sensitive user data

Performance Optimization:

- Implement proper caching strategies
- Add lazy loading for heavy components
- Optimize WebGL rendering pipeline
- Reduce battery consumption during AR sessions

---

9. Conclusion

GroMotion represents a well-structured React-based mobile application with ambitious AR features but significant technical gaps between marketing claims and actual implementation. The application successfully demonstrates core concepts of gamified fitness tracking but requires substantial development to achieve production-ready AI/ML capabilities and native AR performance.

Key Takeaways:

- Strong foundation with React + Capacitor architecture
- Effective use of Firebase for real-time data synchronization
- Creative AR implementation within web technology constraints
- Critical need for actual AI/ML implementation
- Opportunity for significant enhancement through native AR migration

Next Steps:

1. Implement genuine AI/ML systems for user analytics
2. Migrate to native AR frameworks (ARKit/ARCore)
3. Replace fake data with real data collection systems
4. Enhance security and API architecture
5. Add comprehensive offline support

Document Generated: November 19, 2025
Analysis Based on Codebase Review of GroMotion Mobile Application