문서 1: 프로젝트 전체 개요 및 비즈니스 요구사항

프로젝트 명칭

우리무리(UriMuri / OurGroup) - 한국형 단체 위치 추적 모바일 애플리케이션

프로젝트 배경 및 목적

- **주요 목표**: 학생 단체 이동, 산악회 등에서 인원 누락 및 실종을 방지
- **타겟 사용자**: 학교 교사/인솔자, 산악회 리더, 단체 활동 주최자
- **핵심 가치**: 실시간 위치 공유를 통한 안전한 단체 이동 보장

핵심 비즈니스 요구사항

1. **실시간 위치 추적**: 그룹 구성원들의 위치를 실시간으로 공유
2. **백그라운드 동작**: 앱이 비활성 상태여도 위치 추적 지속
3. **사용자 동의**: 위치 추적 전 명시적 동의 획득
4. **그룹 관리**: 코드 기반 그룹 생성 및 참여
5. **안전 기능**: 응급상황 알림, 이탈자 감지

MVP(Minimum Viable Product) 범위

필수 기능

- 방 생성/입장 (6자리 코드)
- 사용자 프로필 등록 (이름, 전화번호, 색상)
- 실시간 위치 공유 (지도 기반)
- 백그라운드 위치 추적
- 방장 관리 기능 (구성원 승인/제거)
- 응급상황 알림
- 위치 공유 일시 중단/재개

제외 기능 (향후 개발)

- 복잡한 권한 체계
- 다국어 지원
- 고급 분석 기능
- 소셜 기능

기술적 제약사항

- **플랫폼**: iOS, Android 동시 지원
- **배터리 최적화**: 일반 사용 대비 20% 이내 배터리 소모 증가
- **네트워크**: 3G/4G/5G/WiFi 모든 환경 지원
- **개인정보보호**: 한국 개인정보보호법 준수

문서 2: 기술 아키텍처 및 스택 정의서

선택된 기술 스택

Frontend Framework

React Native (최신 버전)

- **선택 이유**:
    - iOS/Android 크로스플랫폼 개발
    - JavaScript 기반으로 개발 속도 빠름
    - Firebase와 우수한 연동성
    - 백그라운드 위치 추적 라이브러리 풍부

Backend Infrastructure

Firebase 생태계 (Google Cloud Platform)

- **Firebase Realtime Database**: 실시간 위치 데이터 동기화
- **Firebase Authentication**: 익명 인증 기반 사용자 관리
- **Firebase Cloud Functions**: 서버리스 백엔드 로직
- **Firebase Cloud Messaging**: 푸시 알림
- **Firebase Analytics**: 사용자 행동 분석

핵심 의존성 라이브러리

json

```
{
  "dependencies": {
    "react": "18.2.0",
    "react-native": "0.72.0",
    "@react-native-firebase/app": "^18.6.1",
    "@react-native-firebase/auth": "^18.6.1",
```

```
    "@react-native-firebase/database": "^18.6.1",
    "@react-native-firebase/messaging": "^18.6.1",
    "@react-native-community/geolocation": "^3.1.0",
    "react-native-maps": "^1.8.0",
    "react-native-background-job": "^1.2.1",
    "react-native-background-timer": "^2.4.1",
    "@react-native-async-storage/async-storage": "^1.19.0",
    "@react-native-community/netinfo": "^9.4.0",
    "react-navigation": "^6.0.0"
  }
}
```

시스템 아키텍처 다이어그램

text

```
[Mobile App (React Native)]
            ↕
[Firebase Realtime Database] ← → [Other Mobile Apps]
            ↕
[Firebase Cloud Functions]
            ↕
[Firebase Authentication]
            ↕
[Firebase Cloud Messaging]

[Local Services in App:]
- Geolocation API
- Background Job Scheduler
- AsyncStorage
- Network Monitor
```

데이터베이스 스키마 설계

Firebase Realtime Database 구조

json

```
{
  "users": {
    "userId": {
      "name": "string (필수, max 20자)",
      "phoneNumber": "string (선택, max 15자)",
      "profileColor": "string (hex color)",
      "createdAt": "number (timestamp)"
    }
  },
  "rooms": {
    "roomCode": {
      "leaderId": "string (사용자 ID)",
      "createdAt": "number (timestamp)",
      "members": {
        "userId": {
          "approvedAt": "number (timestamp)",
          "location": {
            "latitude": "number",
            "longitude": "number",
            "timestamp": "number"
          },
          "isLocationSharingPaused": "boolean"
        }
      },
      "pendingRequests": {
```

```
        "userId": {
          "requestedAt": "number (timestamp)"
        }
      },
      "alerts": {
        "alertId": {
          "type": "string (battery_low, network_lost, etc.)",
          "userId": "string",
          "data": "object (상황별 데이터)",
          "timestamp": "number"
        }
      },
      "emergencyAlerts": {
        "alertId": {
          "userId": "string",
          "message": "string",
          "timestamp": "number"
        }
      }
    }
  }
}
```

보안 및 데이터베이스 규칙

```
{
  "rules": {
    "users": {
      "$userId": {
```

```
        ".read": "$userId == auth.uid",
        ".write": "$userId == auth.uid"
      }
    },
    "rooms": {
      "$roomId": {
        ".read": "auth != null && (root.child('rooms').child($roomId).child('members').hasChild(auth.uid)
|| root.child('rooms').child($roomId).child('leaderId').val() == auth.uid)",
        ".write": "auth != null && (root.child('rooms').child($roomId).child('members').hasChild(auth.uid)
|| root.child('rooms').child($roomId).child('leaderId').val() == auth.uid)",
        "members": {
          "$userId": {
            "location": {
              ".write": "$userId == auth.uid"
            }
          }
        },
        "leaderId": {
          ".write": "false"
        }
      }
    }
  }
}
```

문서 3: 백그라운드 위치 추적 핵심 구현 가이드

크로스플랫폼 위치 추적 서비스

메인 위치 서비스 클래스

javascript

```
// LocationTrackingService.js
```

```
import { AppState, Platform, Alert } from 'react-native';
import Geolocation from '@react-native-community/geolocation';
import BackgroundJob from 'react-native-background-job';
import BackgroundTimer from 'react-native-background-timer';
import database from '@react-native-firebase/database';
import auth from '@react-native-firebase/auth';

class LocationTrackingService {
  constructor() {
    this.watchId = null;
    this.backgroundTimer = null;
    this.isTracking = false;
    this.currentRoomCode = null;
    this.updateInterval = 30000; // 30초 기본 간격
    this.lastKnownLocation = null;
  }

  // 위치 추적 시작
  async startTracking(roomCode) {
    if (this.isTracking) {
      console.log('이미 위치 추적이 활성화되어 있습니다.');
      return;
    }

    this.currentRoomCode = roomCode;
    this.isTracking = true;

    // 플랫폼별 구현 분기
```

```
    if (Platform.OS === 'ios') {
      this.startIOSLocationTracking();
    } else {
      this.startAndroidLocationTracking();
    }

    // 앱 상태 변화 리스너 등록
    AppState.addEventListener('change', this.handleAppStateChange);
  }

  // iOS 위치 추적 구현
  startIOSLocationTracking() {
    // iOS watchPosition 사용 (백그라운드에서도 동작)
    this.watchId = Geolocation.watchPosition(
      this.onLocationUpdate,
      this.onLocationError,
      {
        enableHighAccuracy: false, // 배터리 절약
        distanceFilter: 10, // 10미터 이동 시에만 업데이트
        interval: this.updateInterval,
        fastestInterval: 15000, // 최소 15초 간격
        timeout: 20000,
        maximumAge: 60000, // 1분간 캐시된 위치 사용
      }
    );
  }

  // Android 위치 추적 구현
```

```
startAndroidLocationTracking() {
    // Android Background Job 사용
    BackgroundJob.register({
        jobKey: 'locationTracking',
        job: () => {
            this.getCurrentLocationAndUpdate();
        }
    });

    BackgroundJob.start({
        jobKey: 'locationTracking',
        period: this.updateInterval,
    });

    // Foreground에서는 실시간 watchPosition 사용
    if (AppState.currentState === 'active') {
        this.watchId = Geolocation.watchPosition(
            this.onLocationUpdate,
            this.onLocationError,
            {
                enableHighAccuracy: false,
                distanceFilter: 10,
                interval: this.updateInterval,
            }
        );
    }
}
```

```
// 현재 위치 가져오기 및 업데이트
getCurrentLocationAndUpdate = () => {
  Geolocation.getCurrentPosition(
    this.onLocationUpdate,
    this.onLocationError,
    {
      enableHighAccuracy: false,
      timeout: 20000,
      maximumAge: 60000,
    }
  );
};

// 위치 업데이트 콜백
onLocationUpdate = (position) => {
  const { latitude, longitude, accuracy } = position.coords;

  // 정확도가 너무 낮으면 무시 (100미터 이상 오차)
  if (accuracy > 100) {
    console.log('위치 정확도가 낮아 업데이트를 건너뜁니다.');
    return;
  }

  const locationData = {
    latitude,
    longitude,
    accuracy,
    timestamp: Date.now(),
```

```
    };

    this.lastKnownLocation = locationData;
    this.updateLocationToFirebase(locationData);
};

// Firebase에 위치 정보 업데이트
updateLocationToFirebase = async (locationData) => {
    try {
        const currentUser = auth().currentUser;
        if (!currentUser || !this.currentRoomCode) {
            console.log('사용자 또는 방 정보가 없습니다.');
            return;
        }

        await database()
            .ref(`rooms/${this.currentRoomCode}/members/${currentUser.uid}/location`)
            .set(locationData);

        console.log('위치 업데이트 성공:', locationData);
    } catch (error) {
        console.error('위치 업데이트 실패:', error);

        // 네트워크 오류 시 로컬에 저장 후 나중에 재시도
        this.saveLocationLocally(locationData);
    }
};
```

```
// 위치 오류 처리
onLocationError = (error) => {
  console.warn('위치 오류:', error);

  // 마지막 알려진 위치가 있으면 그것을 사용
  if (this.lastKnownLocation) {
    const updatedLocation = {
      ...this.lastKnownLocation,
      timestamp: Date.now(),
      isEstimated: true, // 추정 위치임을 표시
    };
    this.updateLocationToFirebase(updatedLocation);
  }
};

// 앱 상태 변화 처리
handleAppStateChange = (nextAppState) => {
  if (nextAppState === 'background') {
    this.onAppGoesToBackground();
  } else if (nextAppState === 'active') {
    this.onAppComesToForeground();
  }
};

// 앱이 백그라운드로 갈 때
onAppGoesToBackground = () => {
  console.log('앱이 백그라운드로 이동');
```

```
    if (Platform.OS === 'android') {
      // Android에서는 Foreground Service 시작
      this.startForegroundService();
    }
    // iOS는 별도 처리 불필요 (watchPosition이 백그라운드에서도 동작)
};

// 앱이 포그라운드로 올 때
onAppComesToForeground = () => {
    console.log('앱이 포그라운드로 복귀');

    // 로컬에 저장된 위치 데이터가 있으면 동기화
    this.syncLocalLocationData();
};

// Android Foreground Service 시작
startForegroundService = () => {
    // React Native에서는 네이티브 모듈 필요
    // 또는 BackgroundTimer 사용
    this.backgroundTimer = BackgroundTimer.setInterval(() => {
      this.getCurrentLocationAndUpdate();
    }, this.updateInterval);
};

// 위치 추적 중단
stopTracking = () => {
    this.isTracking = false;
```

```
  // WatchPosition 중단
  if (this.watchId) {
    Geolocation.clearWatch(this.watchId);
    this.watchId = null;
  }

  // Android BackgroundJob 중단
  if (Platform.OS === 'android') {
    BackgroundJob.stop({
      jobKey: 'locationTracking',
    });
  }

  // BackgroundTimer 중단
  if (this.backgroundTimer) {
    BackgroundTimer.clearInterval(this.backgroundTimer);
    this.backgroundTimer = null;
  }

  // 이벤트 리스너 제거
  AppState.removeEventListener('change', this.handleAppStateChange);

  console.log('위치 추적이 중단되었습니다.');
};

// 위치 공유 일시 중단
pauseLocationSharing = async () => {
  const currentUser = auth().currentUser;
```

```
    if (currentUser && this.currentRoomCode) {
      await database()
        .ref(`rooms/${this.currentRoomCode}/members/${currentUser.uid}/isLocationSharingPaused`)
        .set(true);
    }
  };

  // 위치 공유 재개
  resumeLocationSharing = async () => {
    const currentUser = auth().currentUser;
    if (currentUser && this.currentRoomCode) {
      await database()
        .ref(`rooms/${this.currentRoomCode}/members/${currentUser.uid}/isLocationSharingPaused`)
        .set(false);

      // 즉시 현재 위치 업데이트
      this.getCurrentLocationAndUpdate();
    }
  };

  // 로컬 위치 데이터 저장 (오프라인 대응)
  saveLocationLocally = async (locationData) => {
    try {
      const AsyncStorage = require('@react-native-async-storage/async-storage').default;
      const key = `location_${Date.now()}`;
      await AsyncStorage.setItem(key, JSON.stringify(locationData));
    } catch (error) {
      console.error('로컬 위치 저장 실패:', error);
```

```
    }
  };

  // 로컬 위치 데이터 동기화
  syncLocalLocationData = async () => {
    try {
      const AsyncStorage = require('@react-native-async-storage/async-storage').default;
      const keys = await AsyncStorage.getAllKeys();
      const locationKeys = keys.filter(key => key.startsWith('location_'));

      for (const key of locationKeys) {
        const locationDataStr = await AsyncStorage.getItem(key);
        if (locationDataStr) {
          const locationData = JSON.parse(locationDataStr);
          await this.updateLocationToFirebase(locationData);
          await AsyncStorage.removeItem(key); // 동기화 후 삭제
        }
      }
    } catch (error) {
      console.error('위치 데이터 동기화 실패:', error);
    }
  };
}

// 싱글톤 인스턴스 생성
const locationTrackingService = new LocationTrackingService();
export default locationTrackingService;
```

권한 관리 서비스

javascript

```
// PermissionManager.js
import { PermissionsAndroid, Platform, Alert, Linking } from 'react-native';

class PermissionManager {
  // 위치 권한 요청
  static async requestLocationPermissions() {
    if (Platform.OS === 'android') {
      return await this.requestAndroidLocationPermissions();
    } else {
      return await this.requestIOSLocationPermissions();
    }
  }

  // Android 위치 권한 요청
  static async requestAndroidLocationPermissions() {
    try {
      // 정밀한 위치 권한 요청
      const fineLocationGranted = await PermissionsAndroid.request(
        PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
        {
          title: '우리무리 위치 권한',
          message: '안전한 단체 이동을 위해 정확한 위치 정보가 필요합니다.',
          buttonNeutral: '나중에',
          buttonNegative: '거절',
          buttonPositive: '허용',
        }
      );
```

```
    if (fineLocationGranted !== PermissionsAndroid.RESULTS.GRANTED) {
      Alert.alert('권한 필요', '위치 권한이 필요합니다.');
      return false;
    }

    // Android 10 이상에서 백그라운드 위치 권한 요청
    if (Platform.Version >= 29) {
      const backgroundLocationGranted = await PermissionsAndroid.request(
        PermissionsAndroid.PERMISSIONS.ACCESS_BACKGROUND_LOCATION,
        {
          title: '백그라운드 위치 권한',
          message: '앱이 백그라운드에서도 위치를 추적하여 안전을 보장합니다.₩n₩n"항상 허용"을 선택해주세요.',
          buttonNeutral: '나중에',
          buttonNegative: '거절',
          buttonPositive: '설정으로 이동',
        }
      );

      if (backgroundLocationGranted !== PermissionsAndroid.RESULTS.GRANTED) {
        Alert.alert(
          '백그라운드 위치 권한 필요',
          '설정에서 "위치" → "항상 허용"을 선택해주세요.',
          [
            { text: '취소', style: 'cancel' },
            { text: '설정으로 이동', onPress: () => Linking.openSettings() }
          ]
        );
```

```
            return false;
          }
        }

        return true;
      } catch (error) {
        console.error('Android 권한 요청 오류:', error);
        return false;
      }
    }

    // iOS 위치 권한 요청
    static async requestIOSLocationPermissions() {
      // iOS는 Info.plist 설정과 함께 자동으로 처리됨
      // Geolocation.requestAuthorization() 사용
      return new Promise((resolve) => {
        const Geolocation = require('@react-native-community/geolocation').default;

        Geolocation.requestAuthorization(
          () => resolve(true),  // 성공
          (error) => {
            console.error('iOS 위치 권한 오류:', error);
            Alert.alert(
              '위치 권한 필요',
              '설정 → 개인정보보호 → 위치 서비스에서 '우리무리'를 "항상"으로 설정해주세요.',
              [
                { text: '취소', style: 'cancel' },
                { text: '설정으로 이동', onPress: () => Linking.openSettings() }
```

```
            ]
          );
          resolve(false);
        }
      );
    });
  }

  // 권한 상태 확인
  static async checkLocationPermissionStatus() {
    if (Platform.OS === 'android') {
      const fineLocation = await PermissionsAndroid.check(
        PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION
      );

      let backgroundLocation = true;
      if (Platform.Version >= 29) {
        backgroundLocation = await PermissionsAndroid.check(
          PermissionsAndroid.PERMISSIONS.ACCESS_BACKGROUND_LOCATION
        );
      }

      return fineLocation && backgroundLocation;
    }

    // iOS는 실제 위치 요청 시에 권한 확인
    return true;
  }
```

```
}

export default PermissionManager;
```

문서 4: 사용자 인터페이스 및 사용자 경험 구현

메인 네비게이션 구조

javascript

```
// App.js - 메인 앱 구조
import React, { useEffect, useState } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import auth from '@react-native-firebase/auth';

import HomeScreen from './screens/HomeScreen';
import UserProfileScreen from './screens/UserProfileScreen';
import CreateRoomScreen from './screens/CreateRoomScreen';
import JoinRoomScreen from './screens/JoinRoomScreen';
import MapScreen from './screens/MapScreen';
import RoomManagementScreen from './screens/RoomManagementScreen';

const Stack = createStackNavigator();

const App = () => {
  const [user, setUser] = useState(null);
  const [hasProfile, setHasProfile] = useState(false);

  useEffect(() => {
    // Firebase 익명 인증
    const initializeAuth = async () => {
```

```
      try {
        if (!auth().currentUser) {
          await auth().signInAnonymously();
        }
        setUser(auth().currentUser);

        // 사용자 프로필 확인
        const database = require('@react-native-firebase/database').default;
        const userSnapshot = await database()
          .ref(`users/${auth().currentUser.uid}`)
          .once('value');

        setHasProfile(userSnapshot.exists());
      } catch (error) {
        console.error('인증 초기화 오류:', error);
      }
    };

    initializeAuth();
  }, []);

  if (!user) {
    return null; // 로딩 화면
  }

  return (
    <NavigationContainer>
      <Stack.Navigator
```

```
    initialRouteName={hasProfile ? 'Home' : 'UserProfile'}
    screenOptions={{
      headerStyle: { backgroundColor: '#2E86AB' },
      headerTintColor: '#FFFFFF',
      headerTitleStyle: { fontWeight: 'bold' },
    }}
  >
    <Stack.Screen
      name="UserProfile"
      component={UserProfileScreen}
      options={{ title: '프로필 설정', headerLeft: null }}
    />
    <Stack.Screen
      name="Home"
      component={HomeScreen}
      options={{ title: 'UriMuri' }}
    />
    <Stack.Screen
      name="CreateRoom"
      component={CreateRoomScreen}
      options={{ title: '방 만들기' }}
    />
    <Stack.Screen
      name="JoinRoom"
      component={JoinRoomScreen}
      options={{ title: '방 참여하기' }}
    />
    <Stack.Screen
```

```
            name="Map"
            component={MapScreen}
            options={{ title: '위치 공유' }}
          />
          <Stack.Screen
            name="RoomManagement"
            component={RoomManagementScreen}
            options={{ title: '방 관리' }}
          />
        </Stack.Navigator>
      </NavigationContainer>
  );
};

export default App;
```

사용자 프로필 화면

javascript

```
// screens/UserProfileScreen.js
import React, { useState } from 'react';
import {
  View,
  Text,
  TextInput,
  TouchableOpacity,
  ScrollView,
  Alert
} from 'react-native';
import auth from '@react-native-firebase/auth';
```

```
import database from '@react-native-firebase/database';
import { globalStyles } from '../styles/globalStyles';

const UserProfileScreen = ({ navigation }) => {
  const [name, setName] = useState('');
  const [phoneNumber, setPhoneNumber] = useState('');
  const [selectedColor, setSelectedColor] = useState('#FF6B6B');

  const profileColors = [
    '#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4',
    '#FFEAA7', '#DDA0DD', '#98D8C8', '#F7DC6F',
    '#FF9999', '#87CEEB', '#FFB347', '#90EE90'
  ];

  const validateInputs = () => {
    if (!name.trim()) {
      Alert.alert('입력 오류', '이름을 입력해주세요.');
      return false;
    }

    if (name.trim().length > 20) {
      Alert.alert('입력 오류', '이름은 20자 이하로 입력해주세요.');
      return false;
    }

    if (phoneNumber && phoneNumber.length > 15) {
      Alert.alert('입력 오류', '전화번호는 15자 이하로 입력해주세요.');
      return false;
```

```
    }

    return true;
  };

  const saveProfile = async () => {
    if (!validateInputs()) return;

    try {
      const user = auth().currentUser;
      const profileData = {
        name: name.trim(),
        phoneNumber: phoneNumber.trim(),
        profileColor: selectedColor,
        createdAt: Date.now(),
      };

      await database()
        .ref(`users/${user.uid}`)
        .set(profileData);

      Alert.alert('성공', '프로필이 저장되었습니다.', [
        { text: '확인', onPress: () => navigation.replace('Home') }
      ]);
    } catch (error) {
      console.error('프로필 저장 오류:', error);
      Alert.alert('오류', '프로필 저장에 실패했습니다. 다시 시도해주세요.');
    }
```

```
};

return (
  <ScrollView style={globalStyles.container}>
    <Text style={globalStyles.title}>프로필 설정</Text>
    <Text style={globalStyles.subtitle}>
      안전한 단체 이동을 위해 기본 정보를 입력해주세요.
    </Text>

    <TextInput
      style={globalStyles.input}
      placeholder="이름 (필수)"
      value={name}
      onChangeText={setName}
      maxLength={20}
      returnKeyType="next"
    />

    <TextInput
      style={globalStyles.input}
      placeholder="전화번호 (선택사항)"
      value={phoneNumber}
      onChangeText={setPhoneNumber}
      keyboardType="phone-pad"
      maxLength={15}
      returnKeyType="done"
    />
```

```
        <Text style={globalStyles.label}>지도에서 표시될 색상 선택</Text>
        <View style={globalStyles.colorContainer}>
          {profileColors.map((color) => (
            <TouchableOpacity
              key={color}
              style={[
                globalStyles.colorOption,
                { backgroundColor: color },
                selectedColor === color && globalStyles.selectedColor
              ]}
              onPress={() => setSelectedColor(color)}
            />
          ))}
        </View>

        <TouchableOpacity style={globalStyles.primaryButton} onPress={saveProfile}>
          <Text style={globalStyles.buttonText}>프로필 저장</Text>
        </TouchableOpacity>
      </ScrollView>
    );
};

export default UserProfileScreen;
```

홈 화면

javascript

```
// screens/HomeScreen.js
import React from 'react';
import { View, Text, TouchableOpacity, Image } from 'react-native';
```

```
import { globalStyles } from '../styles/globalStyles';


const HomeScreen = ({ navigation }) => {
  return (
    <View style={globalStyles.container}>
      <View style={globalStyles.logoContainer}>
        <Text style={globalStyles.logoText}>📍</Text>
        <Text style={globalStyles.appTitle}>우리무리</Text>
        <Text style={globalStyles.appSubtitle}>안전한 단체 이동의 시작</Text>
      </View>

      <View style={globalStyles.buttonContainer}>
        <TouchableOpacity
          style={[globalStyles.primaryButton, globalStyles.largeButton]}
          onPress={() => navigation.navigate('CreateRoom')}
        >
          <Text style={globalStyles.buttonText}>새 방 만들기</Text>
          <Text style={globalStyles.buttonSubtext}>단체를 이끌고 있다면</Text>
        </TouchableOpacity>

        <TouchableOpacity
          style={[globalStyles.secondaryButton, globalStyles.largeButton]}
          onPress={() => navigation.navigate('JoinRoom')}
        >
          <Text style={globalStyles.buttonText}>방 참여하기</Text>
          <Text style={globalStyles.buttonSubtext}>방 코드가 있다면</Text>
        </TouchableOpacity>
      </View>
```

```
        <View style={globalStyles.infoContainer}>
          <Text style={globalStyles.infoText}>
            • 실시간 위치 공유로 안전한 단체 이동{'₩n'}
            • 배터리 최적화된 백그라운드 추적{'₩n'}
            • 응급상황 즉시 알림
          </Text>
        </View>
      </View>
    );
};

export default HomeScreen;
```

방 생성 화면

javascript

```
// screens/CreateRoomScreen.js
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, Alert } from 'react-native';
import auth from '@react-native-firebase/auth';
import database from '@react-native-firebase/database';
import PermissionManager from '../services/PermissionManager';
import locationTrackingService from '../services/LocationTrackingService';
import { globalStyles } from '../styles/globalStyles';

const CreateRoomScreen = ({ navigation }) => {
  const [isCreating, setIsCreating] = useState(false);

  const generateRoomCode = () => {
```

```
  const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
  let result = '';
  for (let i = 0; i < 6; i++) {
    result += chars.charAt(Math.floor(Math.random() * chars.length));
  }
  return result;
};

const createRoom = async () => {
  if (isCreating) return;

  setIsCreating(true);

  try {
    // 위치 권한 확인
    const hasPermission = await PermissionManager.requestLocationPermissions();
    if (!hasPermission) {
      setIsCreating(false);
      return;
    }

    const currentUser = auth().currentUser;
    const roomCode = generateRoomCode();

    // 방 생성
    const roomData = {
      leaderId: currentUser.uid,
      createdAt: Date.now(),
```

```
      members: {
        [currentUser.uid]: {
          approvedAt: Date.now(),
          location: null,
          isLocationSharingPaused: false,
        }
      }
    };

    await database().ref(`rooms/${roomCode}`).set(roomData);

    // 위치 추적 시작
    await locationTrackingService.startTracking(roomCode);

    Alert.alert(
      '방 생성 완료',
      `방 코드: ${roomCode}₩n₩n이 코드를 다른 참가자들에게 공유해주세요.`,
      [
        {
          text: '확인',
          onPress: () => navigation.replace('Map', { roomCode, isLeader: true })
        }
      ]
    );
  } catch (error) {
    console.error('방 생성 오류:', error);
    Alert.alert('오류', '방 생성에 실패했습니다. 다시 시도해주세요.');
    setIsCreating(false);
```

```
    }
};

return (
    <View style={globalStyles.container}>
      <View style={globalStyles.contentContainer}>
        <Text style={globalStyles.title}>새 방 만들기</Text>
        <Text style={globalStyles.description}>
          단체 위치 공유를 위한 새로운 방을 만듭니다.{'₩n'}
          방장으로서 모든 구성원의 위치를 실시간으로 확인할 수 있습니다.
        </Text>

        <View style={globalStyles.featureList}>
          <Text style={globalStyles.featureItem}>✓ 실시간 위치 공유</Text>
          <Text style={globalStyles.featureItem}>✓ 구성원 승인 관리</Text>
          <Text style={globalStyles.featureItem}>✓ 응급상황 알림 수신</Text>
          <Text style={globalStyles.featureItem}>✓ 이탈자 감지 알림</Text>
        </View>
      </View>

      <TouchableOpacity
        style={[
          globalStyles.primaryButton,
          globalStyles.largeButton,
          isCreating && globalStyles.disabledButton
        ]}
        onPress={createRoom}
        disabled={isCreating}
```

```
        >
          <Text style={globalStyles.buttonText}>
            {isCreating ? '방 생성 중...' : '방 만들기'}
          </Text>
        </TouchableOpacity>
      </View>
    );
};

export default CreateRoomScreen;
```

방 참여 화면

javascript

```
// screens/JoinRoomScreen.js
import React, { useState } from 'react';
import { View, Text, TextInput, TouchableOpacity, Alert } from 'react-native';
import auth from '@react-native-firebase/auth';
import database from '@react-native-firebase/database';
import PermissionManager from '../services/PermissionManager';
import locationTrackingService from '../services/LocationTrackingService';
import { globalStyles } from '../styles/globalStyles';

const JoinRoomScreen = ({ navigation }) => {
  const [roomCode, setRoomCode] = useState('');
  const [isJoining, setIsJoining] = useState(false);

  const validateRoomCode = (code) => {
    return /^[A-Z0-9]{6}$/.test(code.toUpperCase());
  };
```

```
const joinRoom = async () => {
  if (isJoining) return;

  const code = roomCode.toUpperCase().trim();

  if (!validateRoomCode(code)) {
    Alert.alert('입력 오류', '방 코드는 6자리 영문 대문자와 숫자로 구성됩니다.');
    return;
  }

  setIsJoining(true);

  try {
    // 위치 권한 확인
    const hasPermission = await PermissionManager.requestLocationPermissions();
    if (!hasPermission) {
      setIsJoining(false);
      return;
    }

    // 방 존재 확인
    const roomSnapshot = await database().ref(`rooms/${code}`).once('value');
    if (!roomSnapshot.exists()) {
      Alert.alert('오류', '존재하지 않는 방 코드입니다.');
      setIsJoining(false);
      return;
    }
```

```
const currentUser = auth().currentUser;

// 이미 참여 중인지 확인
const memberSnapshot = await database()
  .ref(`rooms/${code}/members/${currentUser.uid}`)
  .once('value');

if (memberSnapshot.exists()) {
  Alert.alert('알림', '이미 참여 중인 방입니다.', [
    {
      text: '확인',
      onPress: () => navigation.replace('Map', { roomCode: code, isLeader: false })
    }
  ]);
  setIsJoining(false);
  return;
}

// 참여 요청 추가
await database()
  .ref(`rooms/${code}/pendingRequests/${currentUser.uid}`)
  .set({
    requestedAt: Date.now(),
  });

Alert.alert(
  '참여 요청 완료',
```

```
        '방장의 승인을 기다리고 있습니다.₩n승인되면 자동으로 위치 공유가 시작됩니다.',
        [
          {
            text: '확인',
            onPress: () => {
              // 승인 대기 화면으로 이동하거나 홈으로 이동
              navigation.navigate('Home');
              // 승인 대기 중 알림 리스너 설정
              waitForApproval(code);
            }
          }
        ]
      );
    } catch (error) {
      console.error('방 참여 오류:', error);
      Alert.alert('오류', '방 참여에 실패했습니다. 다시 시도해주세요.');
    } finally {
      setIsJoining(false);
    }
  };

  const waitForApproval = (code) => {
    const currentUser = auth().currentUser;

    // 승인 상태 리스너
    const memberRef = database().ref(`rooms/${code}/members/${currentUser.uid}`);
    const approvalListener = memberRef.on('value', async (snapshot) => {
      if (snapshot.exists()) {
```

```
      // 승인됨
      memberRef.off('value', approvalListener);

      Alert.alert(
        '승인 완료',
        '방에 참여되었습니다. 위치 공유를 시작합니다.',
        [
          {
            text: '확인',
            onPress: async () => {
              await locationTrackingService.startTracking(code);
              navigation.navigate('Map', { roomCode: code, isLeader: false });
            }
          }
        ]
      );
    }
  });

  // 요청 거절 확인 리스너 (pendingRequests에서 제거되었는지 확인)
  const requestRef = database().ref(`rooms/${code}/pendingRequests/${currentUser.uid}`);
  const rejectionListener = requestRef.on('value', (snapshot) => {
    if (!snapshot.exists()) {
      requestRef.off('value', rejectionListener);
      memberRef.off('value', approvalListener);

      Alert.alert('참여 거절', '방장이 참여 요청을 거절했습니다.');
    }
```

```
    });
};

return (
    <View style={globalStyles.container}>
      <View style={globalStyles.contentContainer}>
        <Text style={globalStyles.title}>방 참여하기</Text>
        <Text style={globalStyles.description}>
          방장으로부터 받은 6자리 코드를 입력하세요.
        </Text>

        <TextInput
          style={[globalStyles.input, globalStyles.codeInput]}
          placeholder="방 코드 입력 (예: ABC123)"
          value={roomCode}
          onChangeText={(text) => setRoomCode(text.toUpperCase())}
          maxLength={6}
          autoCapitalize="characters"
          autoComplete="off"
          autoCorrect={false}
          returnKeyType="join"
          onSubmitEditing={joinRoom}
        />

        <View style={globalStyles.infoBox}>
          <Text style={globalStyles.infoTitle}>참여 후 제공되는 기능:</Text>
          <Text style={globalStyles.infoItem}>• 실시간 위치 자동 공유</Text>
          <Text style={globalStyles.infoItem}>• 응급상황 즉시 알림</Text>
```

```
            <Text style={globalStyles.infoItem}>• 위치 공유 일시 중단/재개</Text>
          </View>
        </View>

        <TouchableOpacity
          style={[
            globalStyles.primaryButton,
            globalStyles.largeButton,
            (!roomCode.trim() || isJoining) && globalStyles.disabledButton
          ]}
          onPress={joinRoom}
          disabled={!roomCode.trim() || isJoining}
        >
          <Text style={globalStyles.buttonText}>
            {isJoining ? '참여 요청 중...' : '방 참여하기'}
          </Text>
        </TouchableOpacity>
      </View>
    );
};

export default JoinRoomScreen;
```

문서 5: 지도 및 위치 시각화 구현

메인 지도 화면

javascript

```
// screens/MapScreen.js
import React, { useEffect, useState, useRef } from 'react';
import {
```

```
  View,
  Text,
  TouchableOpacity,
  Alert,
  Modal,
  FlatList,
  Dimensions
} from 'react-native';
import MapView, { Marker, Callout, Circle } from 'react-native-maps';
import auth from '@react-native-firebase/auth';
import database from '@react-native-firebase/database';
import locationTrackingService from '../services/LocationTrackingService';
import { globalStyles, mapStyles } from '../styles/globalStyles';

const { width, height } = Dimensions.get('window');

const MapScreen = ({ route, navigation }) => {
  const { roomCode, isLeader } = route.params;
  const mapRef = useRef(null);

  const [members, setMembers] = useState([]);
  const [currentUser, setCurrentUser] = useState(null);
  const [isLocationSharingPaused, setIsLocationSharingPaused] = useState(false);
  const [showMemberList, setShowMemberList] = useState(false);
  const [mapRegion, setMapRegion] = useState({
    latitude: 37.5665,
    longitude: 126.9780,
    latitudeDelta: 0.01,
```

```
    longitudeDelta: 0.01,
  });

  useEffect(() => {
    initializeMap();
    setupDataListeners();

    return () => {
      // 컴포넌트 언마운트 시 위치 추적 중단
      locationTrackingService.stopTracking();
    };
  }, []);

  const initializeMap = async () => {
    const user = auth().currentUser;
    setCurrentUser(user);

    // 현재 위치로 지도 중심 이동
    navigator.geolocation.getCurrentPosition(
      (position) => {
        const { latitude, longitude } = position.coords;
        setMapRegion({
          latitude,
          longitude,
          latitudeDelta: 0.01,
          longitudeDelta: 0.01,
        });
      },
```

```
      (error) => console.log('현재 위치 가져오기 실패:', error),
      { enableHighAccuracy: true, timeout: 20000, maximumAge: 1000 }
    );
};

const setupDataListeners = () => {
  // 구성원 위치 실시간 업데이트
  const membersRef = database().ref(`rooms/${roomCode}/members`);

  membersRef.on('value', async (snapshot) => {
    const membersData = snapshot.val() || {};
    const membersList = [];

    for (const [userId, memberInfo] of Object.entries(membersData)) {
      try {
        // 사용자 프로필 정보 가져오기
        const userSnapshot = await database().ref(`users/${userId}`).once('value');
        const userProfile = userSnapshot.val() || {};

        membersList.push({
          id: userId,
          name: userProfile.name || '이름없음',
          phoneNumber: userProfile.phoneNumber || '',
          profileColor: userProfile.profileColor || '#FF6B6B',
          location: memberInfo.location,
          lastUpdate: memberInfo.location?.timestamp,
          isLocationSharingPaused: memberInfo.isLocationSharingPaused || false,
          isCurrentUser: userId === currentUser?.uid,
```

```
        });
      } catch (error) {
        console.error('사용자 프로필 로드 오류:', error);
      }
    }

    setMembers(membersList);

    // 첫 번째 위치 업데이트 시 지도 중심 조정
    if (membersList.length > 0) {
      fitMapToMembers(membersList);
    }
  });

  // 응급상황 알림 리스너 (방장만)
  if (isLeader) {
    const emergencyRef = database().ref(`rooms/${roomCode}/emergencyAlerts`);
    emergencyRef.on('child_added', async (snapshot) => {
      const alert = snapshot.val();
      const userSnapshot = await database().ref(`users/${alert.userId}`).once('value');
      const userName = userSnapshot.val()?.name || '알 수 없는 사용자';

      Alert.alert(
        '🚨 응급상황 발생',
        `${userName}님이 응급상황을 신고했습니다.₩n즉시 확인해주세요.`,
        [
          { text: '확인', onPress: () => focusOnMember(alert.userId) }
        ]
```

```
        );
      });
    }
  };

  const fitMapToMembers = (membersList) => {
    const locations = membersList
      .filter(member => member.location && !member.isLocationSharingPaused)
      .map(member => member.location);

    if (locations.length === 0) return;

    if (locations.length === 1) {
      const location = locations[0];
      setMapRegion({
        latitude: location.latitude,
        longitude: location.longitude,
        latitudeDelta: 0.01,
        longitudeDelta: 0.01,
      });
    } else {
      // 모든 구성원이 보이도록 지도 범위 조정
      const latitudes = locations.map(loc => loc.latitude);
      const longitudes = locations.map(loc => loc.longitude);

      const minLat = Math.min(...latitudes);
      const maxLat = Math.max(...latitudes);
      const minLng = Math.min(...longitudes);
```

```
      const maxLng = Math.max(...longitudes);

      const centerLat = (minLat + maxLat) / 2;
      const centerLng = (minLng + maxLng) / 2;
      const deltaLat = (maxLat - minLat) * 1.5; // 여유 공간 추가
      const deltaLng = (maxLng - minLng) * 1.5;

      mapRef.current?.animateToRegion({
        latitude: centerLat,
        longitude: centerLng,
        latitudeDelta: Math.max(deltaLat, 0.01),
        longitudeDelta: Math.max(deltaLng, 0.01),
      });
    }
  };

  const focusOnMember = (memberId) => {
    const member = members.find(m => m.id === memberId);
    if (member && member.location) {
      mapRef.current?.animateToRegion({
        latitude: member.location.latitude,
        longitude: member.location.longitude,
        latitudeDelta: 0.005,
        longitudeDelta: 0.005,
      });
    }
  };
```

```
const toggleLocationSharing = async () => {
  try {
    if (isLocationSharingPaused) {
      await locationTrackingService.resumeLocationSharing();
      setIsLocationSharingPaused(false);
      Alert.alert('알림', '위치 공유가 재개되었습니다.');
    } else {
      await locationTrackingService.pauseLocationSharing();
      setIsLocationSharingPaused(true);
      Alert.alert('알림', '위치 공유가 일시 중단되었습니다.');
    }
  } catch (error) {
    Alert.alert('오류', '위치 공유 설정 변경에 실패했습니다.');
  }
};

const sendEmergencyAlert = () => {
  Alert.alert(
    '🚨 응급상황 알림',
    '방장에게 응급상황을 알리시겠습니까?₩n이 알림은 즉시 전송됩니다.',
    [
      { text: '취소', style: 'cancel' },
      {
        text: '긴급 전송',
        style: 'destructive',
        onPress: async () => {
          try {
            await database()
```

```
                .ref(`rooms/${roomCode}/emergencyAlerts`)
                .push({
                  userId: currentUser.uid,
                  timestamp: Date.now(),
                  message: '응급상황 발생',
                });

              Alert.alert('전송 완료', '응급상황이 방장에게 전송되었습니다.');
            } catch (error) {
              Alert.alert('전송 실패', '응급상황 알림 전송에 실패했습니다.');
            }
          },
        },
      ]
    );
};

const renderMemberMarker = (member) => {
  if (!member.location || member.isLocationSharingPaused) return null;

  const isStale = Date.now() - member.lastUpdate > 300000; // 5분 이상 업데이트 없음

  return (
    <Marker
      key={member.id}
      coordinate={{
        latitude: member.location.latitude,
        longitude: member.location.longitude,
```

```
        }}
        pinColor={isStale ? '#CCCCCC' : member.profileColor}
        opacity={isStale ? 0.5 : 1.0}
      >
        <Callout onPress={() => focusOnMember(member.id)}>
          <View style={mapStyles.calloutContainer}>
            <Text style={mapStyles.memberName}>{member.name}</Text>
            {member.phoneNumber && (
              <Text style={mapStyles.phoneNumber}>{member.phoneNumber}</Text>
            )}
            <Text style={mapStyles.lastUpdate}>
              마지막 업데이트: {new Date(member.lastUpdate).toLocaleTimeString('ko-KR')}
            </Text>
            {isStale && (
              <Text style={mapStyles.staleWarning}>⚠ 위치 정보가 오래됨</Text>
            )}
            {member.isCurrentUser && (
              <Text style={mapStyles.currentUserIndicator}>내 위치</Text>
            )}
          </View>
        </Callout>
      </Marker>
    );
  };

  const calculateNearbyCount = () => {
    const currentUserMember = members.find(m => m.isCurrentUser);
    if (!currentUserMember || !currentUserMember.location) return 0;
```

```
  const currentLocation = currentUserMember.location;
  let nearbyCount = 0;

  members.forEach(member => {
    if (member.isCurrentUser || !member.location || member.isLocationSharingPaused) return;

    const distance = getDistanceFromLatLonInM(
      currentLocation.latitude,
      currentLocation.longitude,
      member.location.latitude,
      member.location.longitude
    );

    if (distance <= 50) {
      nearbyCount++;
    }
  });

  return nearbyCount;
};

// 두 지점 간 거리 계산 (미터)
const getDistanceFromLatLonInM = (lat1, lon1, lat2, lon2) => {
  const R = 6371; // 지구 반지름 (km)
  const dLat = deg2rad(lat2 - lat1);
  const dLon = deg2rad(lon2 - lon1);
  const a =
```

```
      Math.sin(dLat/2) * Math.sin(dLat/2) +
      Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
      Math.sin(dLon/2) * Math.sin(dLon/2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    const d = R * c; // 거리 (km)
    return d * 1000; // 미터로 변환
};

const deg2rad = (deg) => {
    return deg * (Math.PI/180);
};

return (
    <View style={{ flex: 1 }}>
      <MapView
        ref={mapRef}
        style={{ flex: 1 }}
        region={mapRegion}
        onRegionChangeComplete={setMapRegion}
        showsUserLocation={true}
        showsMyLocationButton={false}
      >
        {members.map(renderMemberMarker)}

        {/* 현재 사용자 주변 50미터 원 표시 */}
        {members.find(m => m.isCurrentUser)?.location && (
          <Circle
            center={{
```

```
            latitude: members.find(m => m.isCurrentUser).location.latitude,
            longitude: members.find(m => m.isCurrentUser).location.longitude,
          }}
          radius={50}
          fillColor="rgba(46, 134, 171, 0.1)"
          strokeColor="rgba(46, 134, 171, 0.3)"
          strokeWidth={2}
        />
      )}
    </MapView>

    {/* 상단 정보 패널 */}
    <View style={mapStyles.topPanel}>
      <View style={mapStyles.infoCard}>
        <Text style={mapStyles.roomCodeText}>방 코드: {roomCode}</Text>
        <Text style={mapStyles.memberCountText}>
          총 {members.length}명 참여 중
        </Text>
        <Text style={mapStyles.nearbyCountText}>
          내 주변 50m: {calculateNearbyCount()}명
        </Text>
      </View>

      {isLeader && (
        <TouchableOpacity
          style={mapStyles.managementButton}
          onPress={() => navigation.navigate('RoomManagement', { roomCode })}
        >
```

```
          <Text style={mapStyles.managementButtonText}>방 관리</Text>
        </TouchableOpacity>
      )}
    </View>

    {/* 하단 컨트롤 패널 */}
    <View style={mapStyles.bottomPanel}>
      <TouchableOpacity
        style={mapStyles.memberListButton}
        onPress={() => setShowMemberList(true)}
      >
        <Text style={mapStyles.controlButtonText}>
          구성원 목록 ({members.length})
        </Text>
      </TouchableOpacity>

      <TouchableOpacity
        style={[
          mapStyles.controlButton,
          isLocationSharingPaused && mapStyles.pausedButton
        ]}
        onPress={toggleLocationSharing}
      >
        <Text style={mapStyles.controlButtonText}>
          {isLocationSharingPaused ? '위치 공유 재개' : '위치 공유 일시중단'}
        </Text>
      </TouchableOpacity>
```

```
    <TouchableOpacity
      style={[mapStyles.controlButton, mapStyles.emergencyButton]}
      onPress={sendEmergencyAlert}
    >
      <Text style={mapStyles.controlButtonText}>🚨 응급상황</Text>
    </TouchableOpacity>
  </View>

  {/* 구성원 목록 모달 */}
  <Modal
    visible={showMemberList}
    animationType="slide"
    transparent={true}
    onRequestClose={() => setShowMemberList(false)}
  >
    <View style={mapStyles.modalOverlay}>
      <View style={mapStyles.modalContent}>
        <View style={mapStyles.modalHeader}>
          <Text style={mapStyles.modalTitle}>구성원 목록</Text>
          <TouchableOpacity onPress={() => setShowMemberList(false)}>
            <Text style={mapStyles.closeButton}>✕</Text>
          </TouchableOpacity>
        </View>

        <FlatList
          data={members}
          keyExtractor={(item) => item.id}
          renderItem={({ item }) => (
```

```
<TouchableOpacity
  style={mapStyles.memberListItem}
  onPress={() => {
    setShowMemberList(false);
    focusOnMember(item.id);
  }}
>
  <View
    style={[
      mapStyles.memberColorIndicator,
      { backgroundColor: item.profileColor }
    ]}
  />
  <View style={mapStyles.memberInfo}>
    <Text style={mapStyles.memberListName}>
      {item.name} {item.isCurrentUser && '(나)'}
    </Text>
    {item.phoneNumber && (
      <Text style={mapStyles.memberListPhone}>{item.phoneNumber}</Text>
    )}
    <Text style={mapStyles.memberListStatus}>
      {item.isLocationSharingPaused
        ? '위치 공유 중단됨'
        : item.location
          ? `최근 업데이트: ${new Date(item.lastUpdate).toLocaleTimeString('ko-KR')}`
          : '위치 정보 없음'
      }
    </Text>
```

```
                        </View>
                        <Text style={mapStyles.focusButton}>👁</Text>
                      </TouchableOpacity>
                    )}
                  />
                </View>
              </View>
            </Modal>
          </View>
    );
};

export default MapScreen;
```

문서 6: 글로벌 스타일시트 및 디자인 시스템

```
javascript
// styles/globalStyles.js
import { StyleSheet, Dimensions } from 'react-native';

const { width, height } = Dimensions.get('window');

// 색상 팔레트
const colors = {
  primary: '#2E86AB',        // 메인 파란색
  secondary: '#A23B72',      // 보조 분홍색
  success: '#27AE60',        // 성공 녹색
  warning: '#F39C12',        // 경고 주황색
  danger: '#E74C3C',         // 위험 빨간색
  light: '#ECF0F1',          // 밝은 회색
```

```
  dark: '#2C3E50',              // 어두운 회색
  white: '#FFFFFF',
  black: '#000000',

  // 투명도 적용된 색상
  primaryLight: 'rgba(46, 134, 171, 0.1)',
  dangerLight: 'rgba(231, 76, 60, 0.1)',
};

// 간격 및 크기
const spacing = {
  xs: 4,
  sm: 8,
  md: 16,
  lg: 24,
  xl: 32,
  xxl: 48,
};

const fontSize = {
  xs: 12,
  sm: 14,
  md: 16,
  lg: 18,
  xl: 24,
  xxl: 32,
};
```

```
// 글로벌 스타일
export const globalStyles = StyleSheet.create({
  // 기본 컨테이너
  container: {
    flex: 1,
    backgroundColor: colors.white,
    padding: spacing.md,
  },

  contentContainer: {
    flex: 1,
    justifyContent: 'center',
  },

  // 텍스트 스타일
  title: {
    fontSize: fontSize.xxl,
    fontWeight: 'bold',
    textAlign: 'center',
    marginBottom: spacing.lg,
    color: colors.primary,
  },

  subtitle: {
    fontSize: fontSize.lg,
    textAlign: 'center',
    marginBottom: spacing.md,
    color: colors.dark,
```

```
    lineHeight: 24,
  },

  description: {
    fontSize: fontSize.md,
    textAlign: 'center',
    marginBottom: spacing.lg,
    color: colors.dark,
    lineHeight: 22,
  },

  label: {
    fontSize: fontSize.md,
    fontWeight: '600',
    marginBottom: spacing.sm,
    color: colors.dark,
  },

  // 홈 화면 특별 스타일
  logoContainer: {
    alignItems: 'center',
    marginBottom: spacing.xxl,
  },

  logoText: {
    fontSize: 80,
    marginBottom: spacing.sm,
  },
```

```
  appTitle: {
    fontSize: fontSize.xxl,
    fontWeight: 'bold',
    color: colors.primary,
    marginBottom: spacing.xs,
  },

  appSubtitle: {
    fontSize: fontSize.md,
    color: colors.dark,
  },

  // 버튼 스타일
  primaryButton: {
    backgroundColor: colors.primary,
    padding: spacing.md,
    borderRadius: 12,
    marginVertical: spacing.sm,
    elevation: 3,
    shadowColor: colors.black,
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.25,
    shadowRadius: 4,
  },

  secondaryButton: {
    backgroundColor: colors.secondary,
```

```
    padding: spacing.md,
    borderRadius: 12,
    marginVertical: spacing.sm,
    elevation: 3,
    shadowColor: colors.black,
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.25,
    shadowRadius: 4,
  },

  largeButton: {
    padding: spacing.lg,
    marginVertical: spacing.md,
  },

  disabledButton: {
    backgroundColor: colors.light,
    elevation: 0,
    shadowOpacity: 0,
  },

  buttonText: {
    color: colors.white,
    fontSize: fontSize.lg,
    fontWeight: 'bold',
    textAlign: 'center',
  },
```

```
  buttonSubtext: {
    color: colors.white,
    fontSize: fontSize.sm,
    textAlign: 'center',
    marginTop: spacing.xs,
    opacity: 0.9,
  },

  // 입력 필드
  input: {
    borderWidth: 2,
    borderColor: colors.light,
    backgroundColor: colors.white,
    padding: spacing.md,
    borderRadius: 12,
    fontSize: fontSize.md,
    marginVertical: spacing.sm,
  },

  codeInput: {
    fontSize: fontSize.xl,
    textAlign: 'center',
    letterSpacing: 4,
    fontWeight: 'bold',
  },

  // 색상 선택기
  colorContainer: {
```

```
    flexDirection: 'row',
    flexWrap: 'wrap',
    justifyContent: 'space-between',
    marginVertical: spacing.md,
  },

  colorOption: {
    width: 50,
    height: 50,
    borderRadius: 25,
    margin: spacing.xs,
    borderWidth: 3,
    borderColor: 'transparent',
  },

  selectedColor: {
    borderColor: colors.dark,
    borderWidth: 4,
  },

  // 정보 박스
  infoContainer: {
    backgroundColor: colors.primaryLight,
    padding: spacing.lg,
    borderRadius: 12,
    marginTop: spacing.xl,
  },
```

```
infoBox: {
    backgroundColor: colors.light,
    padding: spacing.md,
    borderRadius: 12,
    marginVertical: spacing.md,
},

infoTitle: {
    fontSize: fontSize.md,
    fontWeight: 'bold',
    marginBottom: spacing.sm,
    color: colors.dark,
},

infoText: {
    fontSize: fontSize.sm,
    color: colors.dark,
    lineHeight: 20,
},

infoItem: {
    fontSize: fontSize.sm,
    color: colors.dark,
    marginVertical: spacing.xs,
},

// 기능 목록
featureList: {
```

```
    marginVertical: spacing.lg,
  },

  featureItem: {
    fontSize: fontSize.md,
    color: colors.success,
    marginVertical: spacing.xs,
    fontWeight: '500',
  },

  // 버튼 컨테이너
  buttonContainer: {
    marginVertical: spacing.xl,
  },
});

// 지도 화면 전용 스타일
export const mapStyles = StyleSheet.create({
  // 상단 패널
  topPanel: {
    position: 'absolute',
    top: 50,
    left: spacing.md,
    right: spacing.md,
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'flex-start',
  },
```

```
infoCard: {
    backgroundColor: colors.white,
    padding: spacing.md,
    borderRadius: 12,
    elevation: 5,
    shadowColor: colors.black,
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.25,
    shadowRadius: 4,
    flex: 1,
    marginRight: spacing.sm,
},

roomCodeText: {
    fontSize: fontSize.md,
    fontWeight: 'bold',
    color: colors.primary,
},

memberCountText: {
    fontSize: fontSize.sm,
    color: colors.dark,
    marginTop: spacing.xs,
},

nearbyCountText: {
    fontSize: fontSize.sm,
```

```
    color: colors.success,
    marginTop: spacing.xs,
    fontWeight: '600',
  },

  managementButton: {
    backgroundColor: colors.secondary,
    padding: spacing.sm,
    borderRadius: 8,
    elevation: 5,
  },

  managementButtonText: {
    color: colors.white,
    fontSize: fontSize.sm,
    fontWeight: 'bold',
  },

  // 하단 패널
  bottomPanel: {
    position: 'absolute',
    bottom: 30,
    left: spacing.md,
    right: spacing.md,
    flexDirection: 'row',
    justifyContent: 'space-between',
  },
```

```
controlButton: {
    backgroundColor: colors.primary,
    padding: spacing.sm,
    borderRadius: 8,
    flex: 0.32,
    elevation: 5,
    shadowColor: colors.black,
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.25,
    shadowRadius: 4,
},

memberListButton: {
    backgroundColor: colors.dark,
    padding: spacing.sm,
    borderRadius: 8,
    flex: 0.32,
    elevation: 5,
    shadowColor: colors.black,
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.25,
    shadowRadius: 4,
},

pausedButton: {
    backgroundColor: colors.warning,
},
```

```
emergencyButton: {
    backgroundColor: colors.danger,
},

controlButtonText: {
    color: colors.white,
    fontSize: fontSize.xs,
    fontWeight: 'bold',
    textAlign: 'center',
},

// 마커 콜아웃
calloutContainer: {
    padding: spacing.sm,
    minWidth: 180,
    maxWidth: 250,
},

memberName: {
    fontSize: fontSize.md,
    fontWeight: 'bold',
    marginBottom: spacing.xs,
    color: colors.dark,
},

phoneNumber: {
    fontSize: fontSize.sm,
    color: colors.dark,
```

```
    marginBottom: spacing.xs,
  },

  lastUpdate: {
    fontSize: fontSize.xs,
    color: '#666',
  },

  staleWarning: {
    fontSize: fontSize.xs,
    color: colors.warning,
    fontWeight: 'bold',
    marginTop: spacing.xs,
  },

  currentUserIndicator: {
    fontSize: fontSize.xs,
    color: colors.primary,
    fontWeight: 'bold',
    marginTop: spacing.xs,
  },

  // 모달 스타일
  modalOverlay: {
    flex: 1,
    backgroundColor: 'rgba(0, 0, 0, 0.5)',
    justifyContent: 'flex-end',
  },
```

```
  modalContent: {
    backgroundColor: colors.white,
    borderTopLeftRadius: 20,
    borderTopRightRadius: 20,
    paddingTop: spacing.lg,
    maxHeight: height * 0.7,
  },

  modalHeader: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    paddingHorizontal: spacing.lg,
    paddingBottom: spacing.md,
    borderBottomWidth: 1,
    borderBottomColor: colors.light,
  },

  modalTitle: {
    fontSize: fontSize.lg,
    fontWeight: 'bold',
    color: colors.dark,
  },

  closeButton: {
    fontSize: fontSize.xl,
    color: colors.dark,
```

```
    fontWeight: 'bold',
  },

  // 구성원 목록 아이템
  memberListItem: {
    flexDirection: 'row',
    alignItems: 'center',
    padding: spacing.md,
    borderBottomWidth: 1,
    borderBottomColor: colors.light,
  },

  memberColorIndicator: {
    width: 30,
    height: 30,
    borderRadius: 15,
    marginRight: spacing.md,
  },

  memberInfo: {
    flex: 1,
  },

  memberListName: {
    fontSize: fontSize.md,
    fontWeight: 'bold',
    color: colors.dark,
  },
```

```
  memberListPhone: {
    fontSize: fontSize.sm,
    color: colors.dark,
    marginTop: spacing.xs,
  },

  memberListStatus: {
    fontSize: fontSize.xs,
    color: '#666',
    marginTop: spacing.xs,
  },

  focusButton: {
    fontSize: fontSize.lg,
    padding: spacing.sm,
  },
});

// 방 관리 화면 스타일
export const roomManagementStyles = StyleSheet.create({
  section: {
    marginVertical: spacing.lg,
  },

  sectionTitle: {
    fontSize: fontSize.lg,
    fontWeight: 'bold',
```

```
    color: colors.dark,
    marginBottom: spacing.md,
  },

  memberItem: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    padding: spacing.md,
    backgroundColor: colors.light,
    borderRadius: 8,
    marginVertical: spacing.xs,
  },

  memberItemText: {
    fontSize: fontSize.md,
    color: colors.dark,
    flex: 1,
  },

  actionButtons: {
    flexDirection: 'row',
    gap: spacing.sm,
  },

  approveButton: {
    backgroundColor: colors.success,
    padding: spacing.sm,
```

```
    borderRadius: 6,
  },

  rejectButton: {
    backgroundColor: colors.danger,
    padding: spacing.sm,
    borderRadius: 6,
  },

  removeButton: {
    backgroundColor: colors.warning,
    padding: spacing.sm,
    borderRadius: 6,
  },

  dangerButton: {
    backgroundColor: colors.danger,
    padding: spacing.md,
    borderRadius: 8,
    marginTop: spacing.xl,
  },

  leaveButton: {
    backgroundColor: '#95A5A6',
    padding: spacing.md,
    borderRadius: 8,
    marginTop: spacing.md,
  },
```

```
  roomCodeDisplay: {
    backgroundColor: colors.primaryLight,
    padding: spacing.lg,
    borderRadius: 12,
    alignItems: 'center',
    marginBottom: spacing.lg,
  },

  roomCodeText: {
    fontSize: fontSize.xl,
    fontWeight: 'bold',
    color: colors.primary,
    letterSpacing: 2,
  },

  shareButton: {
    backgroundColor: colors.primary,
    padding: spacing.sm,
    borderRadius: 6,
    marginTop: spacing.sm,
  },
});

export { colors, spacing, fontSize };
```

문서 7: 최종 개발 로드맵 및 배포 가이드

전체 개발 일정 (4주 MVP)

1주차: 기본 인프라 및 프로젝트 설정

Day 1-2: 프로젝트 초기화

- React Native 프로젝트 생성
- Firebase 프로젝트 설정 및 연동
- 기본 네비게이션 구조 구현
- 글로벌 스타일시트 적용

Day 3-4: 인증 및 프로필 시스템

- Firebase Anonymous Authentication 구현
- 사용자 프로필 화면 개발
- 프로필 데이터 Firebase 저장/로드

Day 5-7: 기본 UI 완성

- 홈 화면 구현
- 방 생성/참여 화면 기본 틀
- 권한 관리 시스템 구현

2주차: 핵심 위치 추적 기능

Day 8-10: 위치 서비스 개발

- LocationTrackingService 클래스 구현
- 크로스플랫폼 백그라운드 위치 추적
- Firebase Realtime Database 연동

Day 11-12: 방 관리 시스템

- 방 생성/코드 생성 로직
- 구성원 승인/거절 시스템
- 실시간 데이터 동기화

Day 13-14: 기본 지도 화면

- React Native Maps 통합
- 기본 마커 표시
- 실시간 위치 업데이트

3주차: 고급 기능 및 UX 개선

Day 15-17: 지도 기능 고도화

- 구성원 정보 콜아웃
- 50미터 반경 표시
- 지도 자동 중심 조정

Day 18-19: 안전 기능

- 응급상황 알림 시스템
- 위치 공유 일시 중단/재개
- 배터리/네트워크 상태 모니터링

Day 20-21: 방 관리 고도화

- 구성원 목록 모달
- 방장 전용 관리 기능
- 방 해체 및 나가기

4주차: 테스트, 최적화 및 배포 준비

Day 22-24: 통합 테스트

- 전체 플로우 테스트
- 다양한 네트워크 환경 테스트
- 배터리 최적화 검증

Day 25-26: 버그 수정 및 성능 최적화

- 메모리 누수 체크
- 백그라운드 동작 안정성 검증
- UI/UX 개선

Day 27-28: 배포 준비

- 앱 스토어 메타데이터 준비
- 개인정보처리방침 작성
- 빌드 최적화 및 서명

Firebase 설정 가이드

1. Firebase 프로젝트 생성

bash

 # Firebase CLI 설치

npm install -g firebase-tools

 # Firebase 로그인

firebase login

 # 프로젝트 초기화

firebase init

2. Realtime Database 설정

json

```
{
  "rules": {
    "users": {
      "$userId": {
        ".read": "$userId == auth.uid",
        ".write": "$userId == auth.uid",
        ".validate": "newData.hasChildren(['name', 'profileColor', 'createdAt'])"
      }
    },
    "rooms": {
      "$roomId": {
        ".read": "auth != null && (root.child('rooms').child($roomId).child('members').hasChild(auth.uid)
|| root.child('rooms').child($roomId).child('leaderId').val() == auth.uid)",
        ".write": "auth != null && (root.child('rooms').child($roomId).child('members').hasChild(auth.uid)
|| root.child('rooms').child($roomId).child('leaderId').val() == auth.uid)",
        "leaderId": {
          ".write": "!data.exists()"
        },
        "members": {
          "$userId": {
```

```
        "location": {
          ".write": "$userId == auth.uid",
          ".validate": "newData.hasChildren(['latitude', 'longitude', 'timestamp'])"
        },
        "isLocationSharingPaused": {
          ".write": "$userId == auth.uid"
        }
      }
    }
  }
}
```

3. 환경 설정 파일

javascript

```
// config/firebase.js
export const firebaseConfig = {
  apiKey: process.env.FIREBASE_API_KEY,
  authDomain: process.env.FIREBASE_AUTH_DOMAIN,
  databaseURL: process.env.FIREBASE_DATABASE_URL,
  projectId: process.env.FIREBASE_PROJECT_ID,
  storageBucket: process.env.FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.FIREBASE_APP_ID,
};
```

플랫폼별 설정

Android 설정

xml

```
<!-- android/app/src/main/AndroidManifest.xml -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />

<application>
  <!-- Google Maps API Key -->
  <meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_GOOGLE_MAPS_API_KEY"/>

  <!-- Background Location Tracking -->
  <service android:name="com.yourapp.LocationTrackingService"
           android:enabled="true"
           android:exported="false" />
</application>
```

iOS 설정

```
xml
<!-- ios/UriMuri/Info.plist -->
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>우리무리는 단체 안전을 위해 백그라운드에서 위치를 추적합니다.</string>

<key>NSLocationWhenInUseUsageDescription</key>
<string>우리무리는 실시간 위치 공유를 위해 위치 정보가 필요합니다.</string>

<key>UIBackgroundModes</key>
```

```
<array>
  <string>location</string>
  <string>background-processing</string>
</array>
```

배포 체크리스트

공통 사항

- 모든 기능 정상 동작 확인
- 다양한 기기에서 테스트 완료
- 배터리 소모량 최적화 검증
- 네트워크 연결 불안정 상황 대응 확인
- 개인정보처리방침 및 이용약관 작성

Android 배포

- android/app/build.gradle 버전 업데이트
- Release APK 빌드 및 테스트
- Google Play Console 개발자 계정 준비
- 앱 서명 키 생성 및 보안 관리

iOS 배포

- Xcode 프로젝트 설정 확인
- Apple Developer 계정 준비
- App Store Connect 메타데이터 준비
- TestFlight 베타 테스트 진행

보안 고려사항

- Firebase 보안 규칙 최종 검토
- API 키 환경변수 처리
- 사용자 데이터 암호화
- GDPR 및 한국 개인정보보호법 준수

예상 비용 및 운영

- **Firebase 사용료**: 무료 tier로 시작, 사용자 증가 시 월 $25-100

- **Google Maps API**: 월 무료 할당량 $200 상당
- **개발자 계정**: Apple $99/년, Google $25 일회성
- **총 초기 비용**: 약 $150-200