작업과정

완벽하게 정리되었습니다. 이 **[거리 단계(Zone) 전략]**은 MVP로서 매우 훌륭한 접근입니다.

이제 **"시스템이 없는 상태"**에서 시작하여 **"실제 테스트"**까지 가는 구체적인 **작업 순서(Roadmap)**를 정리해 드립니다. 이 순서대로만 진행하면 3명이서 꼬이지 않고 완성할 수 있습니다.

📅 Phase 0. 프로젝트 세팅 (오늘 당장 - 1시간 컷)

개발 시작 전에 **"약속"**을 정하는 단계입니다. 이것만 정하면 바로 각자 집으로 흩어져도 됩니다.

1. [다같이] JSON 데이터 구조 확정 (단톡방 공지)

모든 팀원은 이 데이터 구조를 머릿속에 박제합니다.

JSON

```
// Firebase DB 구조 (root)
{
  "seats": {
    "seat_1": { "occupied": false, "userId": null }, // 비콘 1m 이내 (가장 가까움)
    "seat_2": { "occupied": false, "userId": null }, // 비콘 1~3m (중간)
    "seat_3": { "occupied": false, "userId": null }  // 비콘 3~5m (멈)
  }
}
```

2. [다같이] RSSI 기준값 약속

- **1번 좌석:** -59보다 클 때 (예: -40, -50)
- **2번 좌석:** -60 ~ -74 사이
- **3번 좌석:** -75 ~ -90 사이
- **대기:** -90보다 작거나 신호 없음

🏠 Phase 1. 각자 개발 (집에서 - 시뮬레이션 모드)

서로를 기다리지 않고 **"가짜"**로 기능을 완성하는 단계입니다.

📡 팀원 A (로직 담당)

목표: "랜덤 숫자가 들어왔을 때, 좌석 번호를 뱉어내는 로직 완성"

1. **가짜 생성기 만들기:** `setInterval`을 이용해 1초마다 -40 ~ -99 사이의 랜덤 숫자를 찍는 함수 작성.
2. **판별기 구현:** `if (rssi > -60) return "seat_1"` 형태의 함수 작성.
3. **[중요] 떨림 방지(Smoothing) 구현:**
   - 배열 `rssiHistory = []`를 만듭니다.
   - 랜덤 숫자가 들어오면 배열에 넣고, 최근 5개의 **평균값**을 구합니다.
   - 그 **평균값**으로 판별기를 돌려서 로그에 찍어봅니다.
   - *성공 기준: 로그에 "현재 좌석: seat_1"이 안정적으로 찍히면 끝.*

🔥 팀원 B (서버 담당)

목표: "파이어베이스 방을 만들고, 원격 조종 기능 만들기"

1. **프로젝트 생성:** 파이어베이스 콘솔에서 프로젝트 생성 후 `google-services.json` (또는 plist) 파일 팀원에게 배포.
2. **DB 초기화:** Realtime Database 메뉴로 가서 **Phase 0**에서 정한 JSON을 손으로 직접 입력해서 저장.
3. **함수 작성:**
   - `updateSeat(seatName, myUUID)` 함수 구현.
   - 로직: 해당 `seatName`만 `occupied: true`로 바꾸고, **나머지 좌석은 `false`로 초기화**하는 로직이 핵심. (한 사람이 두 자리를 차지하면 안 되니까!)
   - *성공 기준: 코드 실행 시 파이어베이스 콘솔 웹사이트에서 데이터가 슉슉 바뀌면 끝.*

🎨 팀원 C (화면 담당)

목표: "데이터에 따라 빨간불/초록불이 바뀌는 화면 완성"

1. **더미 데이터 박제:** 코드 상단에 `const MOCK_DATA = { ... }` 선언.
2. **UI 그리기:** 1번(앞), 2번(중간), 3번(뒤) 좌석 버튼 배치.
3. **조건부 스타일링:** `MOCK_DATA.seats.seat_1.occupied === true`면 배경색 `red`, 아니면 `green`.
4. **혼잡도 로직:** `Object.values`로 `true` 개수를 세서 상단 텍스트("여유/혼잡") 변경.

5. *성공 기준: 코드의 `MOCK_DATA` 값을 `true`로 고치고 저장했을 때, 화면 색이 바로 바뀌면 끝.*

🔗 Phase 2. 온라인 통합 (비콘 없이 서버 연결)

이제 **가짜 데이터(Mock)**를 **서버 데이터(Firebase)**로 교체합니다.

1. 팀원 C + 팀원 B 합체

- 팀원 C는 `MOCK_DATA` 변수를 지웁니다.
- 대신 파이어베이스 `onValue` 함수를 연결해 실시간 데이터를 받아옵니다.
- **테스트:** 팀원 B가 파이어베이스 콘솔 웹사이트에서 손으로 `false` → `true`로 바꿔봅니다. 팀원 C의 앱 화면이 바뀌면 성공!

2. 팀원 A + 팀원 B 합체

- 팀원 A는 판별기 결과(로그 찍던 것) 아래에 팀원 B의 `updateSeat()` 함수를 넣습니다.
- **테스트:** 팀원 A가 가짜 생성기(랜덤)를 켜둡니다. 파이어베이스 콘솔 웹사이트의 데이터가 혼자서 막 바뀌면 성공!

🏟️ Phase 3. 현장 테스트 (진짜 비콘 투입)

드디어 만나서 **"건전지 끼우는"** 날입니다.

1. 리얼 월드 전환 (팀원 A)

- `setInterval`(랜덤) 코드를 주석 처리합니다.
- 실제 `BleManager` 라이브러리의 스캔 함수를 연결합니다.
- 이제 `processSignal(rssi)` 함수에 **진짜 RSSI**가 들어갑니다.

2. 현장 튜닝 (Calibration)

- 비콘을 책상에 둡니다.
- 앱을 켜고 **1m 지점**에 서봅니다.
- 로그에 찍히는 RSSI를 봅니다. (생각보다 신호가 약할 수 있습니다. 예를 들어 -65가 나올 수도 있음)
- **기준값 수정:** 만약 1m에서 -65가 나온다면, 코드의 1번 좌석 기준을 `60`에서 `68` 정도로 수정합니다.

3. 최종 시나리오 점검

- 폰을 들고 1번 좌석(가까이) -> 2번 좌석(조금 뒤) -> 3번 좌석(멀리) 걸어 다닙니다.

- 화면 속 내 자리가 따라오는지 확인합니다.

📝 요약: 작업 체크리스트

1. **[오늘]** JSON 구조 단톡방 공지.
2. **[내일]**
   - **A:** 랜덤값으로 좌석 뱉는 로직 완성 (로그 확인).
   - **B:** 파이어베이스 세팅 및 `updateSeat` 함수 완성.
   - **C:** 더미 데이터로 색깔 바뀌는 UI 완성.
3. **[모레]** A, C가 각자 B의 파이어베이스 코드 붙이기.
4. **[만나는 날]** 랜덤 함수 떼고 실제 비콘 라이브러리 연결 & RSSI 숫자 튜닝.

이 순서라면 데이터가 없어도, 시스템이 없어도 개발하는 데 전혀 문제없습니다. **가장 먼저 B님이 파이어베이스 프로젝트를 파서 공유해주세요!**