MICE 행사 관리 시스템 개발 기획서 (v3.0 - 최종본)

이 문서는 `소그라_주제v2.pdf` 기획안과 최종 구현된 코드를 바탕으로 역산하여 작성된 최종 버전의 기획서입니다.

1. 프로젝트 개요

- **프로젝트명**: MICE 행사 관리 시스템
- **목표**: MICE(Meeting, Incentive, Convention, Exhibition) 행사를 위한 올인원 참가자 및 세션 관리 플랫폼을 개발합니다. 역할 기반 접근 제어와 안전한 QR 인증 시스템을 통해 행사 운영의 효율성과 보안을 극대화합니다.
- **주요 대상**: 행사 관리자, 연사, 참가자

2. 핵심 기능

1. 역할 기반 접근 제어 (RBAC)

- **Admin (관리자)**: 시스템의 모든 권한을 가지며, 사용자 및 세션의 CRUD(생성, 읽기, 갱신, 삭제)를 담당합니다.
- **Speaker (연사)**: 자신에게 할당된 세션의 정보를 확인하고, Q&A를 관리하며, 발표 자료를 업로드합니다.
- **Attendee (참가자)**: 전체 세션 정보를 조회하고, 질문을 등록하며, QR 코드를 통해 행사장 입장 및 세션 출석 체크를 수행합니다.

2. 하이브리드 QR 인증 시스템

- **정적 QR (나의 비표)**: 참가자별로 고유하게 발급되는 QR 코드로, 행사장 메인 게이트에서 신원 확인 및 입장용으로 사용됩니다.
- **동적 QR (세션 출석)**: 관리자가 세션룸의 스크린에 표시하는 60초마다 갱신되는 QR 코드입니다. 참가자는 이를 스캔하여 각 세션의 출석을 인증하며, 캡처된 이미지 공유를 통한 어뷰징을 방지합니다.

3. 인터랙티브 세션 관리

- **실시간 Q&A**: 참가자는 세션 상세 페이지에서 연사에게 질문을 등록할 수 있으며, 연사는 자신의 세션 관리 페이지에서 질문 목록을 확인할 수 있습니다.
- **발표 자료 관리**: 연사는 세션과 관련된 발표 자료(PDF, PPT 등)를 업로드할 수 있고, 참가자는 이를 다운로드할 수 있습니다.
- **즐겨찾기**: 참가자는 관심 있는 세션을 즐겨찾기 목록에 추가하여 별도로 관리할 수 있습니다.

3. 최종 기술 스택

Backend

- **Framework**: Node.js, Express.js
- **Database**: PostgreSQL
- **ORM**: Prisma
- **Authentication**: `jsonwebtoken` (JWT), `bcrypt` (비밀번호 해싱)
- **File Handling**: `multer` (파일 업로드)
- **In-Memory Cache**: `node-cache` (동적 QR 토큰 관리)
- **Environment**: `dotenv`

Frontend

- **Framework**: React 18
- **Build Tool**: Vite
- **Routing**: `react-router-dom`
- **Styling**: Tailwind CSS, PostCSS, Autoprefixer
- **API Client**: `axios`
- **QR Code**: `qrcode.react` (생성), `html5-qrcode` (스캔)
- **State Management**: React Context API (`AuthContext`)

4. 데이터베이스 스키마

`prisma/schema.prisma`에 정의된 최종 스키마입니다.

```
// 사용자 테이블
model User {
  id            Int       @id @default(autoincrement())
  email         String    @unique
  passwordHash  String    @map("password_hash")
  name          String
  role          Role      @default(ATTENDEE)
  organization  String?
  createdAt     DateTime  @default(now()) @map("created_at")
  updatedAt     DateTime  @updatedAt @map("updated_at")

  sessionsAsSpeaker Session[]
  questions         Question[]
  favorites         Favorite[]
  attendanceLogs    AttendanceLog[]
  uploadedMaterials SessionMaterial[]

  @@map("users")
}

// 역할 ENUM
enum Role {
  ADMIN
  SPEAKER
  ATTENDEE
}

// 세션 테이블
model Session {
  id          Int       @id @default(autoincrement())
  title       String
  description String    @db.Text
  startTime   DateTime  @map("start_time")
  endTime     DateTime  @map("end_time")
  speakerId   Int       @map("speaker_id")
  track       String?
  createdAt   DateTime  @default(now()) @map("created_at")
  updatedAt   DateTime  @updatedAt @map("updated_at")
```

```
  speaker         User
  materials       SessionMaterial[]
  questions       Question[]
  favorites       Favorite[]
  attendanceLogs  AttendanceLog[]

  @@map("sessions")
}

// 세션 자료 테이블
model SessionMaterial {
  id               Int      @id @default(autoincrement())
  sessionId        Int      @map("session_id")
  originalFileName String   @map("original_file_name")
  storedFileName   String   @map("stored_file_name")
  uploaderId       Int      @map("uploader_id")
  createdAt        DateTime @default(now()) @map("created_at")

  session  Session
  uploader User

  @@map("session_materials")
}

// 질문 테이블
model Question {
  id           Int      @id @default(autoincrement())
  sessionId    Int      @map("session_id")
  attendeeId   Int      @map("attendee_id")
  questionText String   @map("question_text") @db.Text
  createdAt    DateTime @default(now()) @map("created_at")

  session  Session
  attendee User

  @@map("questions")
}

// 즐겨찾기 테이블
model Favorite {
  userId    Int @map("user_id")
  sessionId Int @map("session_id")

  user    User
  session Session

  @@id([userId, sessionId])
  @@map("favorites")
}

// 출석 로그 테이블
model AttendanceLog {
  id          Int      @id @default(autoincrement())
  userId      Int      @map("user_id")
```

```
  sessionId   Int      @map("session_id")
  checkedInAt DateTime @default(now()) @map("checked_in_at")

  user    User
  session Session

  @@unique([userId, sessionId])
  @@map("attendance_logs")
}
```

5. API 엔드포인트 명세

경로 (Endpoint)	Method	허용 역할	설명	보안 (RBAC)
/api/auth/register	POST	Public	회원가입	-
/api/auth/login	POST	Public	로그인	-
/api/users/me	GET	All (Auth)	내 정보 조회	authMiddleware
/api/sessions	GET	All (Auth)	전체 세션 목록	authMiddleware
/api/sessions/:id	GET	All (Auth)	세션 상세 정보	authMiddleware
/api/sessions/:id/questions	POST	All (Auth)	세션에 질문 등록	authMiddleware
/api/sessions/:id/favorite	POST	All (Auth)	즐겨찾기 추가	authMiddleware
/api/sessions/:id/favorite	DELETE	All (Auth)	즐겨찾기 제거	authMiddleware
/api/sessions/:id/material	GET	All (Auth)	발표 자료 다운로드	authMiddleware
/api/sessions/check-in	POST	Attendee	세션 출석 체크	authMiddleware, isAttendee
/api/speaker/my-sessions	GET	Speaker	나의 세션 목록	authMiddleware, isSpeaker
/api/speaker/sessions/:id/questions	GET	Speaker	내 세션의 질문 목록	authMiddleware, isSpeaker

경로 (Endpoint)	Method	허용 역할	설명	보안 (RBAC)
`/api/speaker/sessions/:id/material`	`POST`	Speaker	발표 자료 업로드	`authMiddleware`, `isSpeaker`
`/api/admin/users`	`GET`, `POST`	Admin	사용자 관리	`authMiddleware`, `isAdmin`
`/api/admin/users/:id`	`PUT`, `DELETE`	Admin	사용자 수정/삭제	`authMiddleware`, `isAdmin`
`/api/admin/sessions`	`GET`, `POST`	Admin	세션 관리	`authMiddleware`, `isAdmin`
`/api/admin/sessions/:id`	`PUT`, `DELETE`	Admin	세션 수정/삭제	`authMiddleware`, `isAdmin`
`/api/admin/sessions/:id/dynamic-qr`	`GET`	Admin	동적 QR 생성	`authMiddleware`, `isAdmin`
`/api/admin/sessions/:id/attendance`	`GET`	Admin	세션 출석 현황	`authMiddleware`, `isAdmin`

6. 화면별 기능 명세

6.1. 공통

- `/login`: 이메일, 비밀번호로 로그인. 성공 시 역할에 맞는 대시보드로 이동.

6.2. Attendee (참가자)

- `/` **(대시보드)**: 전체 세션 목록을 카드로 표시. 세션 클릭 시 상세 페이지로 이동.
- `/sessions/:id` **(세션 상세)**: 세션의 상세 정보, 연사 정보, 발표 자료 다운로드, Q&A 등록 및 목록 확인 기능 제공.
- `/my-pass` **(나의 비표)**: 본인의 ID가 담긴 정적 QR 코드를 표시. 행사장 입장 시 사용.
- `/scan` **(출석 스캔)**: 카메라를 열어 세션룸의 동적 QR 코드를 스캔하고 출석을 인증.

6.3. Speaker (연사)

- `/speaker/dashboard` **(대시보드)**: 본인에게 할당된 세션 목록만 표시.
- `/speaker/sessions/:id` **(세션 관리)**: 세션 상세 정보, 등록된 Q&A 목록 확인, 발표 자료 업로드 기능 제공.

6.4. Admin (관리자)

- `/admin/dashboard` **(대시보드)**: 전체 사용자, 세션, 참가자 수 등 주요 통계 정보를 표시. 사용자/세션 관리 페이지로 이동하는 링크 제공.
- `/admin/users` **(사용자 관리)**: 전체 사용자 목록 표시. 사용자 생성, 역할 변경, 정보 수정, 삭제 기능 제공.
- `/admin/sessions` **(세션 관리)**: 전체 세션 목록 표시. 세션 생성, 정보 수정, 삭제 기능 제공.

- `/admin/sessions/:id/qr-display` **(동적 QR 표시)**: 특정 세션의 출석 체크용 동적 QR 코드를 전체 화면으로 표시. 60초마다 자동 갱신.

7. 배포 아키텍처

- **Frontend (Vercel)**: React로 빌드된 정적 에셋(`dist` 폴더)을 Vercel에 배포. Vercel의 글로벌 CDN을 통해 사용자에게 빠르고 안정적으로 콘텐츠를 제공.
- **Backend (Railway)**: Node.js/Express 애플리케이션을 Railway에 배포. GitHub 저장소와 연동하여 `main` 브랜치 푸시 시 자동 빌드 및 배포.
- **Database (Railway)**: Railway에서 PostgreSQL 서비스를 생성하고, 백엔드 서비스에 `DATABASE_URL` 환경 변수로 연결.
- **CI/CD**:
    1. 로컬에서 코드 변경 후 `main` 브랜치에 `git push`.
    2. Vercel과 Railway가 각각 Webhook을 통해 푸시를 감지.
    3. Vercel은 `frontend` 디렉토리에서 `npm run build`를 실행하여 프론트엔드를 재배포.
    4. Railway는 `backend` 디렉토리에서 `npm run build` (`prisma generate`) 후 `npm start` (`prisma migrate deploy && node server.js`)를 실행하여 백엔드를 재배포.

8. 보안 구현 상세

- **인증 (Authentication)**: 로그인 시 `jsonwebtoken`을 사용하여 7일 유효기간의 JWT를 발급. 이후 모든 API 요청의 `Authorization: Bearer <token>` 헤더를 통해 사용자를 식별.
- **인가 (Authorization - RBAC)**:
    - **Backend**: `authMiddleware`로 JWT를 검증하여 `req.user`에 사용자 정보를 주입. 이후 `isAdmin`, `isSpeaker` 등의 RBAC 미들웨어가 `req.user.role`을 확인하여 각 API에 대한 접근을 통제.
    - **Frontend**: `ProtectedRoute` 컴포넌트가 `AuthContext`의 `user.role`을 확인하여 역할에 맞지 않는 페이지 접근 시 리다이렉트.
- **어뷰징 방지 (Anti-Abuse)**:
    - **동적 QR**: 세션 출석용 QR은 `node-cache`를 이용해 60초의 유효기간을 갖는 일회성 토큰으로 생성. 동일 QR 재사용 및 공유를 통한 대리 출석을 방지.
- **안전한 파일 처리**:
    - **업로드**: `multer` 라이브러리를 사용. 서버 사이드에서 확장자(pdf, pptx 등)를 엄격히 검사하여 웹쉘 등 악성 파일 업로드를 차단. 파일명은 UUID로 변경하여 저장.
    - **다운로드**: `GET /api/sessions/:id/material` API는 인증을 거치도록 하여, 인증된 사용자만 파일에 접근할 수 있도록 통제.