**Gen Z 세대를 위한 공유 캘린더**

[간신히 Gen z]  
구현 기술 보고서

2024-10-25

2271246 김소진, 2071111 이준용, 2091019 채희석, 2071012 이연준

목차

[1.크로스 플랫폼 2](#_Toc181878930)

[**1.1 [React Native]** 2](#_Toc181878931)

[**1.2 [Flutter]** 3](#_Toc181878932)

[2.백엔드 서비스 5](#_Toc181878933)

[**2.1 Firebase** 5](#_Toc181878934)

[**2.2 AWS Amplify(아마존 웹 서비스)** 6](#_Toc181878935)

[3.요구사항 구현 가능 여부 조사 결과 7](#_Toc181878936)

[**3.1 Firebase를 통한 백엔드 서비스 구현** 7](#_Toc181878937)

[**3.2 Flutter를 통한 크로스 플랫폼 구현** 10](#_Toc181878938)

[4.최종 선정 구현 기술 11](#_Toc181878939)

[**4.1 Firebase를 사용하여 백엔드 서비스 선정** 11](#_Toc181878940)

[**4.1 Flutter를 통한 크로스 플랫폼 구현** 11](#_Toc181878941)

## 크로스 플랫폼

### **1.1 [React Native]**

Facebook이 개발한 프레임워크로, JavaScript를 사용하여 iOS와 Android 애플리케이션을 개발할 수 있다.

--특징--

▶JavaScript 사용: 웹 개발자들이 이미 많이 사용하는 JavaScript를 기반으로 하고 있어 접근이 용이함.

▶React 기반: React의 컴포넌트 기반 아키텍처를 채택하여 재사용성 높은 UI를 개발할 수 있음.

▶네이티브 모듈 사용 가능: 필요할 때는 Java나 Swift 등의 네이티브 코드를 통해 성능을 최적화할 수 있음.

▶활발한 커뮤니티: 개발 커뮤니티가 크고 많은 오픈소스 라이브러리 및 도구들이 존재하여 빠르게 개발 가능.

--장점--

▶기존 웹 개발자에게 친숙: 웹 개발자들이 JavaScript와 React에 익숙하다면, React Native로 쉽게 전환할 수 있음.

▶빠른 개발 속도: 많은 오픈소스 라이브러리와 도구들이 있어 빠른 애플리케이션 개발이 가능.

▶핫 리로드(Hot Reload): 코드 수정 후 애플리케이션을 다시 빌드하지 않고도 즉시 결과를 확인할 수 있음.

▶크로스 플랫폼: 단일 코드베이스로 iOS와 Android 앱을 동시에 개발할 수 있음.

--단점--

▶성능 한계: Flutter에 비해 성능이 다소 떨어질 수 있으며, 복잡한 애니메이션 처리나 성능이 중요한 앱에서는 네이티브 코드가 필요할 수 있음.

▶네이티브 모듈 의존: 특정 기능에서는 네이티브 모듈이 필요해, 각 플랫폼별 코드를 따로 작성해야 하는 경우가 생길 수 있음.

▶UI 일관성 문제: 네이티브 컴포넌트를 사용하다 보니, 플랫폼마다 UI가 다르게 보일 수 있음.

### **1.2 [Flutter]**

Flutter는 Google에서 개발한 오픈소스 UI 툴킷으로, Dart 언어를 사용하여 iOS와 Android 애플리케이션을 개발할 수 있다.

--특징--

▶Dart 언어 사용: Flutter는 Dart라는 언어를 사용하며, 이 언어는 학습 곡선이 있지만 최적화된 성능을 제공함.

▶위젯 기반: Flutter는 자체적으로 모든 위젯을 렌더링하여, 플랫폼 독립적인 일관된 UI를 제공.

▶우수한 성능: 네이티브 성능에 가까운 애플리케이션을 개발할 수 있음.

▶다양한 애니메이션: 복잡한 애니메이션과 그래픽을 쉽게 구현할 수 있음.

--장점--

▶일관된 UI/UX: Flutter는 모든 위젯을 자체적으로 렌더링하기 때문에, iOS와 Android에서 일관된 UI/UX를 제공.

▶우수한 성능: Flutter는 네이티브 성능에 근접한 성능을 제공하며, 특히 복잡한 UI나 애니메이션에서 뛰어난 성능을 발휘.

▶풍부한 위젯: 많은 기본 위젯과 UI 컴포넌트를 제공하여, 커스터마이징과 애니메이션 구현이 쉬움.

▶빠른 개발: Hot Reload 기능과 잘 구성된 문서 덕분에 개발 속도가 빠름.

--단점--

▶Dart 언어 학습 필요: Dart 언어는 JavaScript만큼 널리 사용되지 않아서 새로운 언어를 학습해야 함.

▶앱 크기: Flutter로 개발된 앱은 상대적으로 파일 크기가 큼.

▶네이티브 기능 제한: React Native처럼 특정 네이티브 기능을 사용할 때는 별도의 네이티브 코드 작업이 필요할 수 있음.

React Native와 Flutter 비교 분석

|  |  |  |
| --- | --- | --- |
| **항목** | **React Native** | **Flutter** |
| **언어** | JavaScript | Dart |
| **UI** | 네이티브 컴포넌트를 사용해 플랫폼 별로 UI가 다름 | 자체 위젯을 사용해 일관된 UI 제공 |
| **성능** | 성능이 약간 떨어질 수 있음 | 네이티브에 가까운 성능 제공 |
| **생태계** | 커뮤니티와 라이브러리가 풍부 | 구글의 적극적 지원과 확장성 있는 위젯 제공 |
| **개발 속도** | 빠름 | 빠름 |
| **애니메이션** | 복잡한 애니메이션 구현이 다소 복잡 | 복잡한 애니메이션도 쉽게 구현 가능 |
| **코드 재사용성** | 높은 수준의 재사용성 | 높은 수준의 재사용성 |

## 백엔드 서비스

### **2.1 Firebase**

웹서비스를 만들면서 반복해서 사용하는 기능들은 서비스 주체만 다를 뿐 그 절차와 기능이 비슷하다. 그런 기능들을 정형화해 서비스로 제공하는 것이 파이어베이스이다. 파이어베이스를 이용하면 서버 없이 개발이 가능하다.

--장점--

실시간 데이터베이스

- 실시간 데이터베이스를 제공하여 실시간으로 데이터를 동기화하고 업데이트할 수 있다. 일정이 즉시 업데이트되어 여러 사용자가 동시에 접근해도 원활하게 공유된다.

인증 및 보안

- Firebase Authentication을 통해 사용자 인증을 쉽게 구현할 수 있다. 구글, 페이스북, 이메일 등을 통한 인증을 손쉽게 통합할 수 있어 사용자 관리가 용이하다.

Firebase는 데이터 전송 중에 SSL을 사용하여 보안을 유지한다.

실시간 알림

- Firebase Cloud Messaging(FCM)을 통해 푸시 알림을 전송할 수 있다. 사용자에게 중요한 업데이트를 실시간으로 전달할 수 있다.

호스팅

- Firebase Hosting을 통해 프론트엔드 웹 애플리케이션 배포가 가능하며, 모바일 백엔드와 통합이 원활하다.

빠르고 쉬운 개발

- Firebase는 사용하기 쉬운 API 및 SDK를 제공하여 애플리케이션 개발을 간편화한다. 서버 관리 및 구성에 대한 걱정을 덜어주어 빠르게 개발할 수 있다.

실시간 기능

- Firebase의 실시간 데이터베이스 및 실시간 알림은 실시간 상호 작용이 필요한 애플리케이션에 적합하다.

다양한 통합

- 다양한 기능이 통합되어 있어 개발자들이 다양한 요구에 대응할 수 있다. 인증, 데이터베이스, 스토리지, 호스팅 등이 통합되어 제공된다.

--단점--

- 벤더 종속성

Google 생태계에 의존적이다. Firebase에서 다른 서비스로 이전하기가 쉽지 않다.

- 데이터베이스 가격

데이터가 많아질 경우 실시간 데이터베이스의 가격이 급격히 증가할 수 있다.

### **2.2 AWS Amplify(아마존 웹 서비스)**

AWS 기반의 풀스택 애플리케이션을 빠르게 구성할 수 있도록 도와주는 프레임워크이다. AWS를 사용하면 백엔드 구성이나 배포를 손쉽게 할 수 있게 도와주며 AWS에 있는 서버리스 관련 서비스를 빠르게 연결할 수 있어 아주 편리하다.

--장점--

- AWS 서비스 연결이 쉽다.

기존 서비스 연결 방법은, 각 서비스의 고유값을 가져와 코드에 지정하고 각 서비스의 권한 생성을 해야했다. AWS Amplify(아마존 웹 서비스)는 단 한줄의 명령어로 서비스 연결이 가능하다.

- 코드 수정 및 배포 프로세스 간소화

풀스택 CI/CD 배포 파이프라인을 제공해 코드 한 줄로 배포까지 가능하다.

- 프레임워크가 무료

AWS Amplify(아마존 웹 서비스) 프레임워크 자체는 무료이고, 연결해서 사용한 AWS 서비스에 대해서만 비용을 지불하기 때문에 비용에 부담이 없다.

- 오프라인 기능

Amplify는 오프라인 데이터 동기화 기능을 제공해 인터넷 연결이 불안정할 때도 데이터 업데이트가 가능하다.

--단점--

- 설정 복잡성

Firebase에 비해 설정과 구성이 복잡하며, 학습 곡선이 있을 수 있다.

- 비용 구조의 복잡성

다양한 서비스와 사용량에 따라 가격이 달라져 관리가 다소 까다로울 수 있다.

## 요구사항 구현 가능 여부 조사 결과

### **3.1 Firebase를 통한 백엔드 서비스 구현**

**RF 1.0 캘린더 기능**

**RF 1.1 캘린더 보기** : Firebase Firestore 또는 Realtime Database에 저장된 일정 데이터를 조회하여 캘린더 형태로 표시한다. 사용자는 Firestore에서 특정 날짜에 해당하는 일정만 조회할 수 있다. 일정의 startDate, endDate 필드를 기준으로 쿼리하여 해당 기간 동안의 일정을 가져온다. Firestore의 효율적인 쿼리 기능을 활용해 필요한 일정 데이터만 가져오므로 빠르게 캘린더에 표시가 가능하다.

**RF 1.2 일정 등록**: 일정 등록 UI에서 사용자가 입력한 데이터를 Firestore에 새 문서로 생성한다. Firestore.instance.collection('schedules').add(scheduleData)로 일정 데이터를 id, title, description, startDate, endDate, category, isPublic 등과 같은 필드로 구성하여 저장한다.

**RF 1.3 일정 삭제**: 특정 일정이 선택되면 해당 일정의 문서 ID를 기반으로 delete() 메서드를 호출하여 삭제한다. Firestore.instance.collection("schedules").doc(scheduleId).delete();와 같은 코드로 Firestore에서 해당 일정을 삭제할 수 있으며, 삭제 후 실시간 업데이트가 동기화되어 클라이언트에 즉시 반영된다.

**RF 1.4 일정 변경**: 일정 수정UI에서 사용자가 일정 내용을 수정한 후, Firestore의 update() 메서드를 사용하여 문서 내 데이터를 변경한다, Firestore.instance.collection('schedules').doc(scheduleId).update(updatedData);로 일정을 업데이트하며, 변경 사항이 실시간으로 동기화된다.

**RF 1.5 한눈에 캘린더 보기**: 전체 일정을 한 번에 볼 수 있도록 monthView와 같은 설정으로 달력에 일정을 표시한다. 이때 Firestore에서 orderBy를 사용해 날짜별로 일정을 정렬하고, 한 달 단위로 일정을 불러와 사용자에게 제공한다.

**RF 1.6 할 일, 루틴, 습관 등록**: Firestore 문서에 type 필드를 추가하여 ‘할 일’, ‘루틴’, ‘습관’ 등의 유형을 지정할 수 있다. 이러한 태그나 유형별로 데이터를 분류할 수 있도록 하여, 나중에 일정 목록에서 해당 유형별로 일정을 쉽게 조회 가능하게 한다.

**RF 1.7 일정 메모**: Firestore에 저장된 일정 데이터에 memo 필드를 추가하여, 사용자가 일정에 관련된 추가 메모를 기재할 수 있도록 한다. 일정 상세 페이지에서 memo 데이터를 불러와 메모 내용을 확인 가능하며, 메모가 있을 때만 표시되도록 설정할 수 있다.

**RF 2.0 사용자 등록 기능**

**RF 2.1 회원가입**: Firebase Authentication을 통해 이메일과 비밀번호 회원가입 기능을 구현한다. createUserWithEmailAndPassword(email, password) 메서드를 사용하여 사용자를 생성하며, 가입 시 Firestore에 사용자 정보를 저장할 수 있도록 한다. 예를 들어 users라는 Firestore 컬렉션에 사용자 UID를 문서 ID로 저장하고, 사용자 이름, 이메일 등을 문서 필드로 추가한다.

**RF 2.2 로그인**: Firebase Authentication을 사용해 이메일과 비밀번호로 로그인 기능을 제공한다. signInWithEmailAndPassword(email, password) 메서드로 로그인 처리를 하고, 로그인 성공 시 사용자 UID를 사용하여 Firestore에서 해당 사용자 데이터를 불러온다.

**RF 2.3 사용자 정보 저장**: 사용자가 가입 후 추가적인 정보를 입력할 수 있도록 Firestore에 사용자 정보를 저장한다. users 컬렉션을 생성하여, 사용자 UID를 문서 ID로 사용하고, 사용자 이름, 생일, 프로필 이미지 URL과 같은 세부 정보를 추가로 저장할 수 있다.

**RF 3.0 공유 그룹 기능**

**RF 3.1 그룹 생성**: 사용자가 그룹 이름과 설명을 입력해 Firestore의 groups 컬렉션에 그룹을 생성한다. 생성된 그룹에는 groupId와 ownerId(현재 사용자의 UID)가 포함되며, members 필드에 참여자 리스트를 저장할 수 있다.

**RF 3.2 그룹 삭제**: Firestore에서 그룹 문서와 해당 그룹 내 일정 데이터를 삭제한다. 그룹 소유자인 사용자만 삭제 가능하도록 Firestore Security Rules를 설정하여 권한을 관리할 수 있다.

**RF 3.3 그룹 초대 및 참여**: 그룹 초대를 위한 초대 코드를 생성하여 Firestore에 저장하거나, 초대 링크를 생성해 Firestore에 초대 정보를 저장한다. 초대 코드로 그룹에 참여할 수 있도록 하고, 참여자가 초대 코드를 입력하면 Firestore에서 해당 그룹의 members 필드에 참여자의 UID를 추가한다.

**RF 3.4 그룹 탈퇴 및 제외**: 사용자가 그룹에서 탈퇴하거나 관리자가 특정 사용자를 제외할 때, Firestore에서 그룹 문서의 members 필드에서 해당 UID를 삭제한다. 삭제 후 실시간으로 갱신하여 남은 그룹 멤버들에게도 반영된다.

**RF 3.5 그룹 내 일정 조율**: 그룹의 일정 데이터 수정 권한을 Firestore에서 schedulePermissions와 같은 필드로 관리하여 그룹 내에서 일정 수정을 조율할 수 있도록 한다. 예를 들어, editPermission을 가진 사용자만 일정을 변경할 수 있도록 설정한다.

**RF 3.6 그룹 참여자 확인**: Firestore에서 members 필드 데이터를 조회하여 그룹 참여자 리스트를 확인하고, 이를 UI에 표시한다.

**RF 4.0 알림 기능**

**RF 4.1 일정 알림 기능**: Firebase Cloud Messaging (FCM)을 사용하여 특정 일정에 대한 알림을 보낸다. 사용자가 일정 등록 시 알림 시간을 설정하고, 설정된 알림 시간에 맞춰 FCM으로 푸시 알림을 발송한다.

**RF 5.0 일정 관리 기능**

**RF 5.1 일정 카테고리 분류**: Firestore에 일정 등록 시 category 필드를 추가하여 일정 데이터를 카테고리별로 구분한다. 예를 들어, meeting, task, event 등의 카테고리로 구분하여 각 카테고리에 따라 필터링 기능을 제공한다.

**RF 5.2 일정 분석**: Firestore에 저장된 일정 데이터를 일정 기간 동안 수집해 분석하는 Firebase Functions를 작성할 수 있다. 예를 들어 월별 일정 수, 카테고리별 일정 수 등을 분석하여 분석 결과를 별도의 컬렉션에 저장하고, 클라이언트 앱에서 조회하여 차트로 표시한다.

**RF 5.3 일정 공개여부 설정**: 일정 등록 시 isPublic 필드를 추가해 공개 여부를 저장한다. Firestore 보안 규칙을 통해 isPublic이 true인 일정만 다른 사용자가 조회할 수 있도록 설정하여 비공개 일정은 본인만 볼 수 있도록 처리한다.

**RF 6.0 동기화 및 연동 기능**

**RF 6.1 캘린더 동기화**: Firestore의 실시간 동기화 기능을 사용하여 일정이 추가, 삭제, 변경될 때마다 즉시 클라이언트 앱에 반영되도록 구현한다. Firestore의 실시간 리스너를 추가하여 일정 데이터의 변경을 감지하고 UI에 반영한다.

**RF 6.2 캘린더 연동**: 외부 캘린더 API (예: 구글 캘린더)와의 연동을 위해 Google Calendar API를 호출해 Firestore에 일정 데이터를 저장하거나 업데이트한다. 외부 캘린더에서 일정을 가져올 때 Firestore에 저장하여 앱 내에서 외부 캘린더 일정과 함께 표시할 수 있다.

### **3.2 Flutter를 통한 크로스 플랫폼 구현**

**캘린더 UI 구성**: table\_calendar 또는 syncfusion\_flutter\_calendar와 같은 Flutter용 캘린더 패키지를 사용하여 일정 조회 및 등록, 삭제, 변경 기능을 구성한다. 캘린더 위젯을 화면에 추가하고, 각 날짜에 맞는 일정을 표시한다.

**사용자 등록/로그인 UI**: Firebase Authentication과 FlutterFire 패키지를 사용하여 로그인 및 회원가입 UI를 구현한다. FirebaseAuth 인스턴스를 사용하여 사용자 등록 및 로그인 기능을 제공하고, 성공 시 Firestore에서 사용자 프로필을 불러온다.

**그룹 관리 및 초대**: Flutter로 그룹 UI를 구성하여 초대 링크를 생성하고, Firestore의 그룹 데이터를 표시한다. 초대 코드를 생성하여 사용자가 그룹에 참여할 수 있도록 한다.

**알림 기능**: FCM을 활용하여 Flutter 앱에서 일정 알림을 제공한다. 일정 등록 시 알림 시간을 지정하고, 그 시간에 맞춰 알림을 발송하여 Flutter의 로컬 알림 패키지로 표시한다.

**캘린더 동기화**: Firestore의 실시간 동기화 기능을 이용해 캘린더가 즉시 업데이트되도록 구성합니다.

**기타 캘린더 및 일정 연동**: Flutter의 url\_launcher 패키지를 사용하여 외부 캘린더와 연동하여 추가적인 일정을 가져와 Firestore에 저장한다.

## 최종 선정 구현 기술

### **4.1 Firebase를 사용하여 백엔드 서비스 선정**

Firebase는 서버리스 환경에서 실시간 데이터베이스와 인증, 푸시 알림 등의 기능을 제공하며, 백엔드 인프라를 손쉽게 관리할 수 있다는 점에서 본 프로젝트에 적합하다. 특히, 실시간 동기화와 다양한 인증 방법을 손쉽게 구현할 수 있어 Gen Z 세대를 위한 공유 캘린더 애플리케이션의 기본 요구 사항인 실시간 업데이트와 안정적인 사용자 관리를 충족할 수 있다. 또한, Firebase Cloud Messaging (FCM)을 통해 중요한 일정에 대해 알림을 제공함으로써 사용자 경험을 향상시킬 수 있다고 판단된다.

### **4.1 Flutter를 통한 크로스 플랫폼 구현**

Flutter는 크로스 플랫폼 개발을 지원하며, 일관된 UI/UX를 제공할 수 있어 iOS와 Android에서 모두 동일한 사용자 경험을 제공한다. 특히 위젯 기반 구조를 통해 복잡한 UI와 애니메이션을 손쉽게 구현할 수 있어, Gen Z 세대의 다양한 요구를 반영한 UI와 상호작용을 구현하기에 적합하다. Hot Reload 기능을 통한 빠른 개발과 Firebase와의 매끄러운 연동 덕분에 효율적인 개발 환경을 제공해, 프로젝트의 일정 내 완성도를 높이는 데 도움이 될 것 같다.