Pusan National University
Computer Science and Engineering
Technical Report 2025-09

어린이가 작성한 이야기 기반 그림 동화 생성기

201824527 여현준

201824551 이승민

지도교수 탁성우

목 차

1. 서론

1.1. 연구 배경

최근 AI 기술의 발전은 콘텐츠 제작 방식에 커다란 변화를 가져오고 있다. 특히, LLM 의 자연어 처리와 Stable Diffusion 과 같은 생성형 AI 기술을 이용하여 간단한 프롬프트를 통해 누구든지 콘텐츠를 생산하고 소비할 수 있는 수준에 이르렀다. 이러한 시대적 흐름 속에 인간만이 가지는 고유한 능력인 창의력의 중요성이 더욱 부각되고 있다. 특히 어린 시절은 창의성을 기르는 데 가장 중요한 시기이므로, AI 과 공존하며 고유한 가치를 발휘하기 위해서는 어릴 때부터 창의력을 마음껏 펼치고 키울 수 있는 환경을 조성해 주는 것이 좋다.

1.2. 기존 문제점

1.2.1. 어린이의 창의성 표현의 한계

어린이들이 상상한 내용을 글로는 쉽게 표현할 수 있지만, 그림으로 직접 옮기는 데에는 기술적인 한계나 어려움을 겪을 수 있다. 이로 인해 아이들의 풍부한 상상력이 온전히 시각적으로 구현되지 못하는 경우가 많다.

1.2.2. 수동적인 콘텐츠 소비 문화

현재 어린이 콘텐츠 시장은 어른들이 창작한 이야기를 아이들이 일방적으로 소비하는 구조가 주를 이룬다. 이로 인해 어린이 스스로 이야기를 만들어내고 시각화 하는 능동적인 창작 활동 경험이 부족하다는 문제점이 있다.

1.2.3. 단조로운 창의적 활동 경험

아이들의 창의적인 글쓰기와 그림 그리기 활동이 별개의 과정으로 분리되어 있어, 두 가지 활동을 유기적으로 연결하고 확장하는 경험이 부족하다.

1.2.4. 생성형 AI의 한계

AI를 활용하여 그림을 그릴 수 있게 된 현재에도 기존 범용 AI를 통한 자연어 처리로는 간단하고 단편적인 그림만을 그릴 수 있다. 더 복잡한 그림을 그리기 위해서는 적절한 프롬프트와 자세한 상황 설명이 필요하다. 이를 어린 아이가 직접 하기에는 어려움이 존재한다.

1.3. 연구 목표

본 과제에서는 어른이 창작한 "동화" 라는 컨텐츠를 어린이들이 수동적으로 소비하는 것이 아닌 어린이들이 능동적으로 자신의 이야기를 펼치고, 이를 그림 동화로 확인할 수 있게 하여 창의적인 콘텐츠 창작자로의 전환을 가능하게 하는 것이 목표이다.
따라서 본 과제에서는 어린이가 작성한 이야기를 분석하고, 자동으로 프롬프트를 작성하여 생성형 인공지능에 전달해 그림동화를 자동으로 생성하는 시스템을 구현하고자 한다.

2. 연구 배경

2.1. 배경 지식

2.1.1. 모바일 앱 개발 트렌드: 크로스 플랫폼

오늘날 모바일 애플리케이션 개발은 iOS와 Android 두 개의 주요 플랫폼을 중심으로 이루어진다. 각 플랫폼별로 네이티브 언어(Swift, Kotlin)를 사용하여 개발할 수 있지만, 이는 두 개의 코드베이스를 관리해야 하는 부담을 야기한다.

이러한 문제점을 해결하기 위해 등장한 것이 크로스 플랫폼(Cross-Platform) 개발 프레임워크이다. 하나의 코드베이스로 iOS와 Android 애플리케이션을 동시에 구축할 수 있어 개발 효율성과 생산성을 크게 향상시킨다. 본 과제 역시 이러한 트렌드를 반영하여 React Native를 핵심 프레임워크로 채택했습니다. 이를 통해 사용자에게 두 플랫폼 모두에서 일관된 경험을 제공하는 동시에, 개발 리소스를 효율적으로 활용할 수 있다.

2.1.2. 클라우드 기반 서비스 아키텍처 (Serverless)

전통적인 애플리케이션 개발은 서버를 직접 구축하고 관리하는 복잡한 과정을 거쳐야 한다. 그러나 클라우드 서비스의 발전은 이러한 부담을 줄여주며 개발자들이 오직 핵심 기능 개발에만 집중할 수 있게 한다.

본 과제는 Firebase를 활용하여 서버 관리 없이도 앱의 백엔드 기능을 구현했다.

Firebase Authentication으로 사용자 인증 시스템을 간편하게 구축하며 보안성도 확보했다.

Cloud Firestore라는 NoSQL 데이터베이스를 사용해 사용자의 일기나 동화 데이터를 실시간으로 저장하고 동기화할 수 있다.

Firebase Storage는 AI가 생성한 이미지 파일을 효율적으로 관리하는 데 사용된다.

2.1.3. 인공지능(AI)과 생성형 모델의 활용

최근 AI 기술은 단순한 데이터 분석을 넘어, 텍스트나 이미지를 스스로 생성하는 생성형 AI로 빠르게 발전하고 있다. 이는 창작과 예술의 영역까지 확장되며 새로운 가능성을 열고 있다.

본 과제는 이와 같은 생성형 AI의 최신 기술을 활용한다. Google Gemini AI 모델을 사용하여 사용자가 작성한 일기 텍스트를 분석하고, 문단별로 AI가 이해할 수 있는 프롬프트로 변환한다. 이 프롬프트를 기반으로 Gemini Image Preview(Nano Banana) 모델이 각 문단에 어울리는 이미지를 생성한다. 이를 통해 어린이는 자신의 이야기에 맞는 독창적인 그림을 손쉽게 얻을 수 있다.

2.1.4. JavaScript 개발 생태계

모바일 앱 개발은 웹 개발 기술과 유사한 생태계를 공유한다. React Native는 JavaScript를 기반으로 하며, 이 언어의 유연성과 확장성은 다양한 개발 도구와 결합하여 높은 생산성을 가능하게 한다. 다음과 같은 도구들로 지속적인 유지보수와 기능 확장을 용이하게 한다.

TypeScript를 도입하여 코드에 정적 타입(Static Typing)을 적용함으로써, 복잡한 프로젝트에서도 오류를 줄이고 안정적인 코드를 작성할 수 있다.

ESLint와 Prettier는 코드의 스타일을 일관되게 유지하고 잠재적인 버그를 미리 찾아내어 코드 품질을 향상시키는 데 기여한다.

2.2. 서비스 기획 및 수정 사항

2.2.1. 화면 설계

<그림 1>과 같이 모바일 환경에 맞춰 기본적으로 세로 화면으로 구성하며, 반응형UI를 사용하여 다른 기기 및 해상도에서도 사용을 용이하게 한다. 주 사용층이 아이임을 고려하여 아기자기 하면서 사용하기 쉽게 직관적인 UI/UX를 구성한다.

<그림 1>. UI 설계

2.2.2. API설정 및 데이터 베이스 구축

2.2.2.1. API 엔드포인트 설정

본 프로젝트의 모든 API 엔드포인트는 Firebase Functions를 기반으로 설계되었으며, 이를 통해 클라이언트와 백엔드 간의 안전하고 효율적인 통신이 가능하다. 각 엔드포인트는 역할에 따라 인증, AI 서비스, 데이터 관리, 이미지 처리 등 카테고리로 구분되어 있으며, 엔드포인트는 <표 1>과 같다.

카테고리	API 엔드포인트	주요 기능
인증 시스템	signUpWithEmail()	이메일로 회원가입
	signInWithEmail()	이메일로 로그인
	signOutUser()	로그아웃
	sendPasswordResetEmail()	비밀번호 재설정 이메일 발송
AI 서비스	generateText(prompt)	텍스트 생성
	divideStoryIntoParagraphs(storyText)	동화 문단 나누기
	generateScenePrompt()	장면 프롬프트 생성
	generateImage(prompt, options)	이미지 생성
	convertToWebP(imageData)	WebP 형식 변환
일기 관리	createDiaryEntry()	일기 생성
	updateDiaryEntry()	일기 수정
	deleteDiaryEntry()	일기 삭제
	getDiaryEntry()	특정 일기 조회

	getUserDiaries()	사용자 일기 목록 조회
동화 관리	createFairytaleEntry()	동화 생성
	updateFairytaleEntry()	동화 수정
	deleteFairytaleEntry()	동화 삭제
	getFairytaleEntry()	특정 동화 조회
	getUserFairytales()	사용자 동화 목록 조회
이미지 시스템	generateImage()	AI 이미지 생성
	generateImagePrompt()	이미지 프롬프트 생성
	uploadImageToStorage()	이미지 저장
	deleteImageFromStorage()	이미지 삭제
스토리 처리	processStory(storyText)	동화 텍스트 처리
	divideStoryIntoParagraphs()	문단 분할
	generateImageDescriptionsForParagraphs()	문단별 이미지 설명 생성
API 키 관리	setApiKey()	API 키 설정
	getApiKey()	API 키 조회
	isApiKeyValid()	API 키 유효성 검사

<표 1>. 엔드포인트 목록

2.2.2.2. 데이터베이스 구축

- 데이터베이스 선택: 본 프로젝트의 데이터베이스는 Firebase를 기반으로 한 NoSQL 아키텍처를 채택하였다. 메인 데이터 저장소인 Firestore와 미디어 파일을 위한 Firebase Storage로 구성되어 있다. 일기와 동화 컬렉션의 paragraphs 필드는 중첩된 데이터 구조를 가지며, 주요 필드와 역할은 <표 2>와 같다.

필드 이름	역할	데이터 타입
id	문서 고유 ID	string
userId	사용자 고유 ID (Firebase Auth UID)	string
title	일기/동화의 제목	string
content	일기/동화의 전체 내용	string
date	일기/동화의 날짜	string (YYYY-MM-DD 형식)
createdAt	문서 생성 시간	Timestamp

updatedAt	문서 최종 수정 시간	Timestamp
mainImageUrl	스토리북 대표 이미지 URL	string?
paragraphs	문단별 상세 데이터 배열	DiaryParagraph[]
mood	일기/동화의 분위기	string?
weather	날씨 또는 배경 설정	string?
tags	태그 배열	string[]

<표 2>. diaries 및 fairytales 컬렉션의 주요 필드와 역할

- 데이터모델링 설계: 데이터베이스 구축 시 데이터를 하나의 문서에 중첩 구조로 저장하여, 여러 번의 쿼리 없이 한 번의 읽기 작업으로 필요한 모든 정보를 가져올 수 있게 한다.

- 쿼리 최적화 및 인덱스 설정: 데이터베이스의 성능을 최적화하기 위해 효율적인 쿼리 전략을 수립했다. 사용자별로 일기나 동화 목록을 조회하는 쿼리가 가장 빈번하므로, userId와 createdAt, date 필드에 복합 인덱스를 설정했다. 이를 통해 대규모 데이터 속에서도 사용자의 특정 데이터를 빠르게 검색할 수 있다. 모든 데이터는 asia-northeast3(서울) 리전(region)에 저장되어 낮은 지연시간을 갖는다.

- 보안 및 데이터 일관성: Firebase Security Rules를 통해 각 사용자가 자신의 데이터(userId 필드 기반)에만 접근하고 수정할 수 있도록 권한 제어를 설정했다. 또한, 데이터의 일관성을 보장하기 위해 문서 생성 및 업데이트 시 트랜잭션을 사용하고, Firebase Functions에서 데이터 유효성 검사를 수행한다.

- 파일 스토리지 구조: AI가 생성한 이미지 파일은 Firebase Storage에 저장되며, 효율적인 관리를 위해 diaries와 fairytales 폴더 아래 사용자 ID를 기준으로 데이터를 분리했다. 이 구조는 데이터 격리를 용이하게 한다.

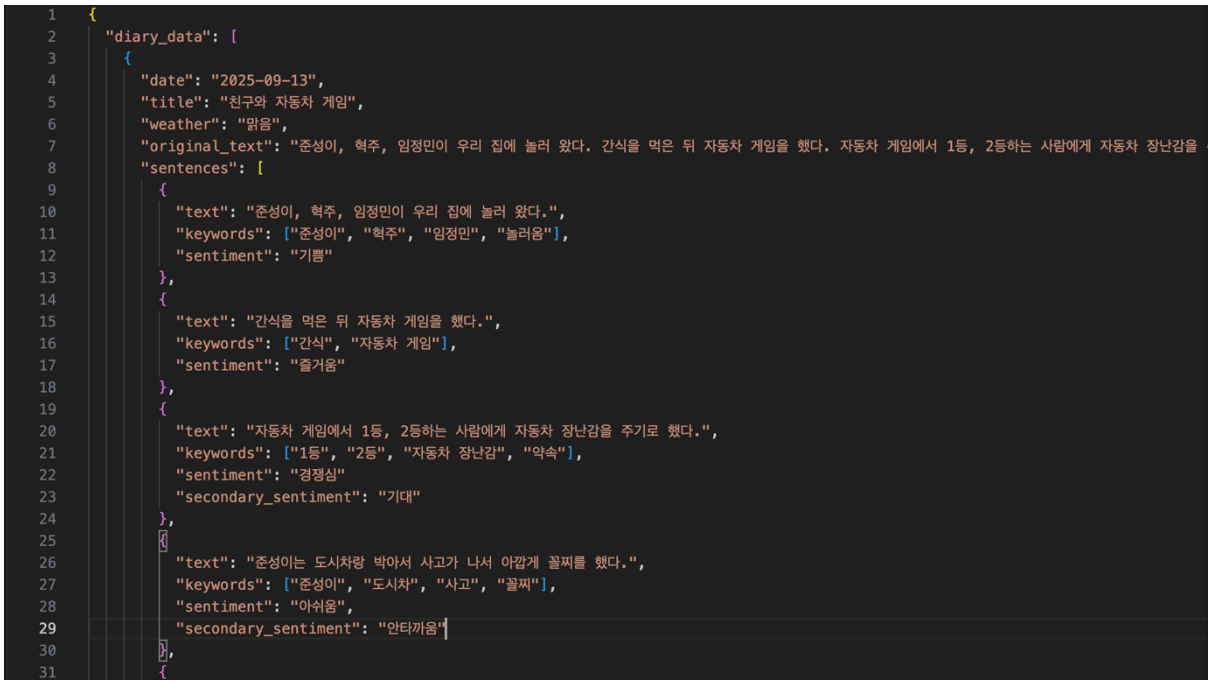
2.2.3. LLM 및 생성형 AI 선택 및 파인튜닝

2.2.3.1. AI 선택 및 파인튜닝 준비

본 과제를 진행하는 데에 가장 적절한 AI를 선택 한다. Google Gemini, OpenAI ChatGPT, Anthropic Claude와 같은 범용 LLM에 기존에 존재하는 동화를 사용하여 이를 문단별로 나누고, 간단한 프롬프트 엔지니어링으로 그 속에 담긴 감정과 주요 키워드를 추출하고 생성형 AI용 프롬프트를 생성하게 하여 가장 결과가 좋았던(기존 동화와 가장

유사한) LLM을 선별한다. 이렇게 작성된 프롬프트를 Stable Diffusion, OpenAI DALL-E, Google Nano Banana등의 생성형 AI에 입력하고, 가장 적절한 색깔, 그림체 사용 및 자연스러운 사물의 배치를 하는 생성형 AI를 선별한다.

실제 초등학교 저학년의 일기를 구해 학습에 적절한 일기를 선별하고, <그림 2>와 같이 문장별로 분리 하여 해당 문장의 주요 키워드와 감정을 정해 JSON 파일로 만든다.



```
{
  "diary_data": [
    {
      "date": "2025-09-13",
      "title": "친구와 자동차 게임",
      "weather": "맑음",
      "original_text": "준성이, 혁주, 임정민이 우리 집에 놀러 왔다. 간식을 먹은 뒤 자동차 게임을 했다. 자동차 게임에서 1등, 2등하는 사람에게 자동차 장난감을
      "sentences": [
        {
          "text": "준성이, 혁주, 임정민이 우리 집에 놀러 왔다.",
          "keywords": ["준성이", "혁주", "임정민", "놀러옴"],
          "sentiment": "기쁨"
        },
        {
          "text": "간식을 먹은 뒤 자동차 게임을 했다.",
          "keywords": ["간식", "자동차 게임"],
          "sentiment": "즐거움"
        },
        {
          "text": "자동차 게임에서 1등, 2등하는 사람에게 자동차 장난감을 주기로 했다.",
          "keywords": ["1등", "2등", "자동차 장난감", "약속"],
          "sentiment": "경쟁심"
          "secondary_sentiment": "기대"
        },
        {
          "text": "준성이는 도시차랑 박아서 사고가 나서 아깝게 꼴찌를 했다.",
          "keywords": ["준성이", "도시차", "사고", "꼴찌"],
          "sentiment": "아쉬움",
          "secondary_sentiment": "안타까움"
        },
        {
```

<그림 2>. JSON파일 일부

하이퍼파라미터(Epoch, 학습률, batch size등)를 설정하여 파인튜닝 스크립트를 작성하고, 앞서 작성한 JSON파일과 함께 입력 하여 훈련시킨다. 훈련 과정동안 지속적인 모니터링으로 과적합(Overfitting)현상을 방지한다.

학습이 완료 되면 학습에 사용하지 않은 다른 일기를 사용하여 적절한 결과가 도출 되는지 성능을 평가한다. 결과가 만족스럽지 못하다면 하이퍼파라미터를 변경하여 재학습시키고, 만족스럽다면 모델은 서비스에 올린 후 엔드포인트를 생성하여 배포한다.

2.2.3.2. 수정 사항

프로젝트 초반에는 LLM은 Google Gemini, 생성형 AI는 OpenAI DALL-E가 선별 되었으나, 프로젝트 진행 중 계속해서 API의 성능을 모니터링 한 결과 그림 생성에 있어 Google Nano Banana가 DALL-E보다 더 나은 결과를 보여주는 것이 확인되어 변경하였다.

Google Gemini 2.5 Flash 는 현재 별도의 파인튜닝을 지원하지 않아 Gemini 2.5 Flash 기반 파인튜닝을 지원하는 Vertex AI 를 사용하려고 했으나 React Native 환경에서 Vertex AI SDK 의 안정적인 연동을 위한 공식 레퍼런스와 자료가 매우 부족했으며, 보안에 민감한 환경 변수 처리 방식에 대한 명확한 가이드라인을 찾기 어려웠다.

이러한 문제점들은 프로젝트 진행 속도를 저해하고 개발 안정성을 떨어뜨릴 수 있다고 판단하여 프로젝트의 목표를 효과적으로 달성하기 위해 React Native 환경에 최적화된 @google/generative-ai SDK 를 통해 접근 가능한 Google Gemini 모델로 기술 스택을 변경했다. 파인튜닝을 통한 최적화는 불가능 하지만 현재 성능으로도 프롬프트 엔지니어링을 통해 간단한 문단 분석과 키워드 추출 및 프롬프트 작성에는 문제가 없다고 판단했다.

2.3. 개발 환경

2.3.1. 프론트엔드

2.3.1.1. 핵심 프레임워크

- React Native 0.81.1: 크로스 플랫폼 모바일 앱 개발 프레임워크
- React 19.1.0: 사용자 인터페이스 구축을 위한 JavaScript 라이브러리
- TypeScript 5.8.3: 정적 타입 검사를 통한 코드 안정성 향상

2.3.1.2. 네비게이션

- @react-navigation/native 7.1.17: React Native 네비게이션 라이브러리
- @react-navigation/stack 7.4.8: 스택 기반 네비게이션 구현
- react-native-screens 4.16.0: 네이티브 스크린 최적화
- react-native-safe-area-context 5.6.1: 안전 영역 처리
- react-native-gesture-handler 2.28.0: 제스처 처리

2.3.1.3. UI/UX 라이브러리

- react-native-vector-icons 10.3.0: 아이콘 라이브러리
- react-native-svg 15.13.0: SVG 그래픽 지원

- react-native-image-picker 8.2.1: 이미지 선택 및 촬영

- @react-native-async-storage/async-storage 1.24.0: 로컬 데이터 저장

- react-native-get-random-values 1.11.0: Node.js crypto polyfill

- react-native-url-polyfill 2.0.0: URL polyfill

2.3.1.4. 상태 관리 및 컨텍스트

- React Context API: 전역 상태 관리 (인증 상태, 사용자 데이터)

- Custom Hooks: 재사용 가능한 로직 분리

2.3.2. 백엔드 및 클라우드 서비스

2.3.2.1. Firebase 서비스

- Firebase Authentication: 사용자 인증 및 계정 관리

- Cloud Firestore: NoSQL 데이터베이스 (일기, 동화 데이터 저장)

- Firebase Storage: 이미지 파일 저장 및 관리

2.3.2.2. AI 서비스

- Google Gemini AI 2.5 Flash: 텍스트 생성 및 분석

- Google Gemini 2.5 Flash Image Preview: AI 이미지 생성

- @google/generative-ai 0.24.1: Google AI SDK

2.3.3. 개발 도구 및 빌드 시스템

2.3.3.1. 번들러 및 빌드 도구

- Metro: React Native 번들러

- Babel: JavaScript 트랜스파일러

- @react-native/babel-preset 0.81.1: React Native Babel 프리셋

- react-native-dotenv 3.4.11: 환경 변수 관리

2.3.3.2. 개발 환경 설정

- @react-native-community/cli 20.0.0: React Native CLI

- @react-native-community/cli-platform-android 20.0.0: Android 플랫폼 지원

- @react-native-community/cli-platform-ios 20.0.0: iOS 플랫폼 지원

2.3.3.3. 코드 품질 도구

- ESLint 8.19.0: 코드 린팅

- Prettier 2.8.8: 코드 포맷팅

- @react-native/eslint-config 0.81.1: React Native ESLint 설정

2.3.3.4. 테스팅

- Jest 29.6.3: JavaScript 테스팅 프레임워크

- @types/jest 29.5.13: Jest TypeScript 타입 정의

- react-test-renderer 19.1.0: React 컴포넌트 테스팅

- @types/react-test-renderer 19.1.0: React Test Renderer 타입 정의

2.3.4. 플랫폼별 설정

2.3.4.1. Android 설정

- Gradle: Android 빌드 시스템

- Kotlin: Android 네이티브 코드

- Android SDK: 최소 SDK 21, 타겟 SDK 34

- ProGuard: 코드 난독화 및 최적화

2.3.4.2. iOS 설정

- CocoaPods: iOS 의존성 관리

- Swift: iOS 네이티브 코드

- Xcode: iOS 개발 환경

- iOS Deployment Target: 11.0 이상

2.3.5. 아키텍처 패턴

2.3.5.1. 폴더 구조

```
src/
├── components/          # 재사용 가능한 UI 컴포넌트
├── screens/         # 화면 컴포넌트
├── navigation/          # 네비게이션 설정
├── services/            # 비즈니스 로직 및 API 서비스
├── contexts/            # React Context
├── types/               # TypeScript 타입 정의
├── constants/           # 상수 정의
├── config/              # 설정 파일
└── utils/               # 유틸리티 함수
```

2.3.5.2. 서비스 레이어

- AI 서비스: Gemini AI를 통한 텍스트 및 이미지 생성
- 인증 서비스: Firebase Authentication 관리
- 데이터 서비스: Firestore 데이터 CRUD 작업
- 이미지 서비스: 이미지 생성 및 저장 관리
- 스토리 처리 서비스: 일기/동화 텍스트 처리

2.3.5.3. 상태 관리 패턴

- Context API: 전역 상태 (인증, 사용자 데이터)
- Local State: 컴포넌트별 로컬 상태

- Custom Hooks: 비즈니스 로직 캡슐화

2.3.6. 주요 기능별 기술 구현

2.3.6.1. AI 텍스트 처리

- Gemini 2.5 Flash: 텍스트 생성 및 분석

- 프롬프트 엔지니어링: 문단 분리, 키워드 추출, 그림 생성용 프롬프트 생성

2.3.6.2. AI 이미지 생성

- Gemini 2.5 Flash Image Preview: AI 이미지 생성

2.3.6.3. 데이터 관리

- Firestore: 실시간 데이터베이스

- AsyncStorage: 로컬 캐싱

- 이미지 최적화: Base64 인코딩 및 압축

- 오프라인 지원: 네트워크 연결 상태 관리

2.3.7. 보안 및 성능

- API 키 관리: 환경 변수를 통한 안전한 키 관리

- 사용자 인증: Firebase Authentication

- 데이터 암호화: Firestore 보안 규칙

2.3.8. 배포 및 배포 환경

2.3.8.1. Android 배포

- APK 빌드: Debug/Release 모드

- Google Play Store: 앱 스토어 배포 준비

- 서명 설정: ProGuard 및 코드 난독화

2.3.8.2. iOS 배포

- Xcode Archive: 앱 빌드 및 패키징

- App Store Connect: 앱 스토어 배포 준비

- 코드 사이닝: 개발자 인증서 설정

3. 연구 내용

3.1. 기술구현

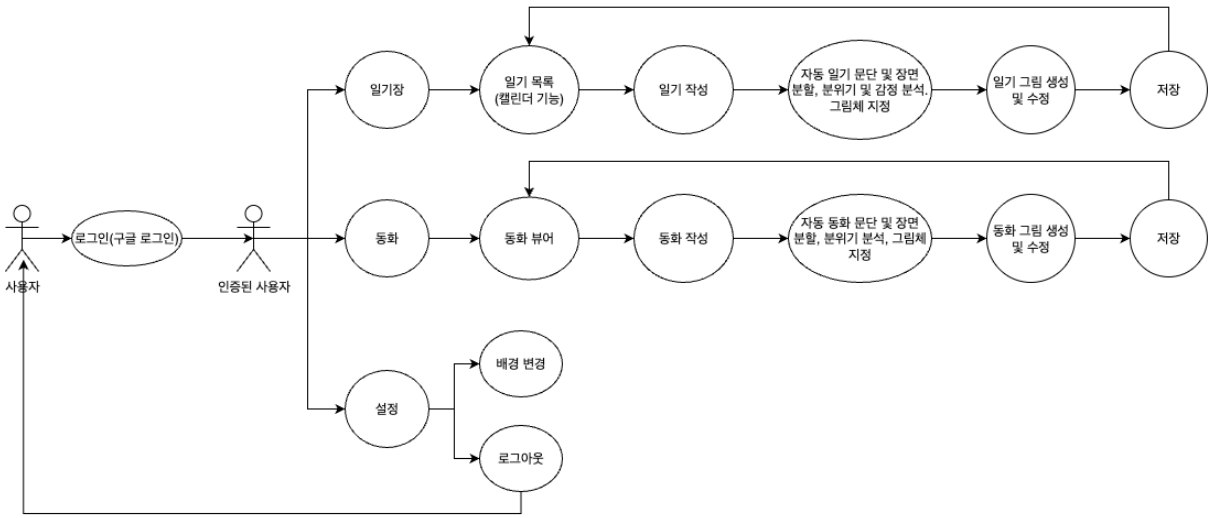
3.1.1. 서비스 구성도 및 유즈케이스

React Native App			
Presentation Layer (UI Components)			
Screens	Components	Navigation	Context
Landing	Calendar	Stack	Auth
SignIn	DateSlider	Navigator	Context
Main	Icons		
Diary	ImageStyle		
Fairytale	Selector		
Settings			
Business Logic Layer			
Auth Service	Data Service	AI Service	Storage Service
SignUp	Diary	Gemini	Image
SignIn	Fairytale	Firebase	Upload
SignOut	CRUD	Text Gen	Download
Reset	Query	Image Gen	Delete
Data Layer (Types & Utils)			
Types	Utils	Constants	Config
Diary	Date	API	Firebase
AI	Image	Style	
Auth	RN Image	UI	

External Services			
Firebase Auth	Google AI	Firebase Database	Firebase Storage

Email/Pass	Gemini 2.5 Flash	Documents	Images
Session	Gemini 2.5 Flash Image Preview	Collections	Files
		Queries	Metadata

<표 3>, <표 4>. 서비스 구성표



<그림 3>. 유즈케이스 다이어그램

3.1.2. React Native를 통한 크로스플랫폼 개발

구현 이유: iOS와 Android 서로 다른 두 모바일 운영체제를 동시에 지원하기 위해 각 플랫폼의 네이티브 언어(Swift, Kotlin)로 개발하면 막대한 시간과 리소스가 필요하다. 이 비효율성을 극복하고 핵심 기능 구현에 집중하기 위해 하나의 코드베이스로 두 플랫폼을 모두 지원하는 하이브리드 앱 개발 방식을 선택했다.

구현 방법: React Native를 핵심 프레임워크로 채택하여 JavaScript/TypeScript 기반의 단일 코드로 iOS와 Android 애플리케이션을 빌드한다.

Metro 번들러를 통해 소스 코드를 번들링하여 네이티브 앱 형태로 변환했으며, React Native CLI를 사용하여 양 플랫폼의 빌드 환경을 관리한다.

사용자 인터페이스와 전역 상태 관리를 위해 React Context API를 사용하여 사용자 인증 상태나 데이터와 같은 핵심 정보를 중앙에서 관리하도록 설계한다. 이는 불필요한 props drilling을 방지하고 코드의 유지보수성을 높이는 결과를 가져온다.

3.1.3. 프롬프트 엔지니어링 파이프라인

구현 이유: 순수한 자연어는 AI 모델이 정확히 이해하기 어려운 경우가 많다. 특히, 아동의 문장은 문법적으로 단순하거나 모호한 표현이 포함될 수 있다. 특히, Google Gemini 2.5 Flash 는 별도의 파인튜닝을 지원하지 않으므로 프롬프트 엔지니어링을 최대한 활용하여 최적화를 시켜야 한다. 사용자가 복잡한 프롬프트를 직접 작성하지 않아도 고품질의 그림을 얻을 수 있도록, 프롬프트 엔지니어링 파이프라인을 자동화하는 데 연구의 초점을 맞췄다.

구현 방법: <그림 4>와 같이 프로그램에 사전에 미리 문단 분리 및 키워드 추출용 프롬프트 및 그림 생성프롬프트의 핵심 내용을 정의해 둔다. 사용자가 서비스를 이용 할 때 사용자의 텍스트 데이터와 함께 자동으로 사전 정의 프롬프트가 Google Gemini 2.5 Flash에 전달되고, Gemini는 프롬프트에 따라 사용자 텍스트를 분석 하고, 문단 분리 한다.

```
12
13  /**
14   * 이미지 생성용 문단 나누기 프롬프트
15   * 사용 위치: Firebase Functions (divideIntoImageGenerationParagraphs)
16   */
17  export const DIVIDE_INTO_IMAGE_GENERATION_PARAGRAPHS_PROMPT = `
18  다음 일기/동화를 이미지 생성에 적합한 문단으로 나누어주세요. 각 문단은 그림으로 표현할 수 있는 구체적인 장면이어야 합니다.
19
20  원본 텍스트:
21  {storyText}
22
23  요구사항:
24  1. 각 문단은 하나의 시각적 장면을 담고 있어야 합니다
25  2. 문단은 2-4문장 정도의 적당한 길이여야 합니다
26  3. 각 문단에 대해 그림을 그리기 위한 장면 프롬프트도 생성해주세요
27  4. 장면 프롬프트는 이미지 생성을 위해 영어로 작성하되 다른 언어 요소가 있다면 적절히 영어로 번역해주세요
28  5. 각 문단에 대해 이미지 검색에 적합한 키워드 3-5개를 추출해주세요
29  6. 장면 프롬프트에는 절대 이미지 스타일(애니메이션, 사실적, 동화 등), 분위기, 색상 팔레트를 포함하지 마세요
30  7. 장면 프롬프트는 장면의 내용만 설명하세요
31  8. 전체 텍스트의 내용과 분위기를 분석하여 적절한 이미지 스타일, 분위기, 색상 팔레트를 자동으로 선택해주세요
32
33  이미지 스타일 옵션: anime, realistic, fairytale, cartoon, watercolor, digital_art, children_book
34  분위기 옵션: happy, sad, mysterious, adventurous, peaceful, exciting
35  색상 팔레트 옵션: vibrant, pastel, monochrome, warm, cool
36
37  응답 형식:
38  문단1: [첫 번째 장면 문단]
39  장면프롬프트1: [첫 번째 장면을 그리기 위한 상세한 프롬프트 - 스타일 제외]
40  키워드1: [키워드1, 키워드2, 키워드3, 키워드4, 키워드5]
41  문단2: [두 번째 장면 문단]
42  장면프롬프트2: [두 번째 장면을 그리기 위한 상세한 프롬프트 - 스타일 제외]
43  키워드2: [키워드1, 키워드2, 키워드3, 키워드4, 키워드5]
44  문단3: [세 번째 장면 문단]
45  장면프롬프트3: [세 번째 장면을 그리기 위한 상세한 프롬프트 - 스타일 제외]
46  키워드3: [키워드1, 키워드2, 키워드3, 키워드4, 키워드5]
47
48  추천 이미지 설정:
49  스타일: [anime, realistic, fairytale, cartoon, watercolor, digital_art, children_book 중 하나]
50  분위기: [happy, sad, mysterious, adventurous, peaceful, exciting 중 하나]
51  색상팔레트: [vibrant, pastel, monochrome, warm, cool 중 하나]
52  `;
```

<그림 4>. 사전 정의 프롬프트 코드의 일부

자동으로 분리된 각 장면에 대해, 모델은 핵심 키워드 추출 및 장면 그림 프롬프트 생

성 작업을 수행한다. 이 과정은 @google/generative-ai SDK를 통해 프로그램적으로 구현되어, AI 모델을 호출하고 응답을 받는 모든 과정을 백엔드에서 처리한다.

최종적으로 생성된 프롬프트는 Gemini 2.5 Flash Image Preview 모델로 전달되어 이미지를 생성한다. 이 모든 과정은 백엔드에서 이루어지므로, 사용자는 단지 텍스트를 입력하는 것만으로 완성된 결과물을 받을 수 있다.

3.1.4. 클라우드 기반 아키텍처 및 데이터 관리

구현 이유: AI 모델 호출, 데이터 저장, 사용자 인증과 같은 백엔드 기능은 모바일 앱 내에서 직접 처리하기보다 전문적인 클라우드 서비스를 활용하는 것이 안정성과 보안 측면에서 유리하다. 서버를 직접 구축하고 관리하는 부담을 줄이기 위해 서버리스(Serverless) 아키텍처를 채택했다.

구현 방법: Firebase Authentication을 통해 사용자 인증을 처리하여 앱의 보안을 강화했다.

Cloud Firestore에 사용자의 이야기 텍스트와 메타데이터를 저장하여 실시간으로 동기화한다. 이는 사용자가 여러 기기에서 접속해도 동일한 데이터를 볼 수 있게 한다.

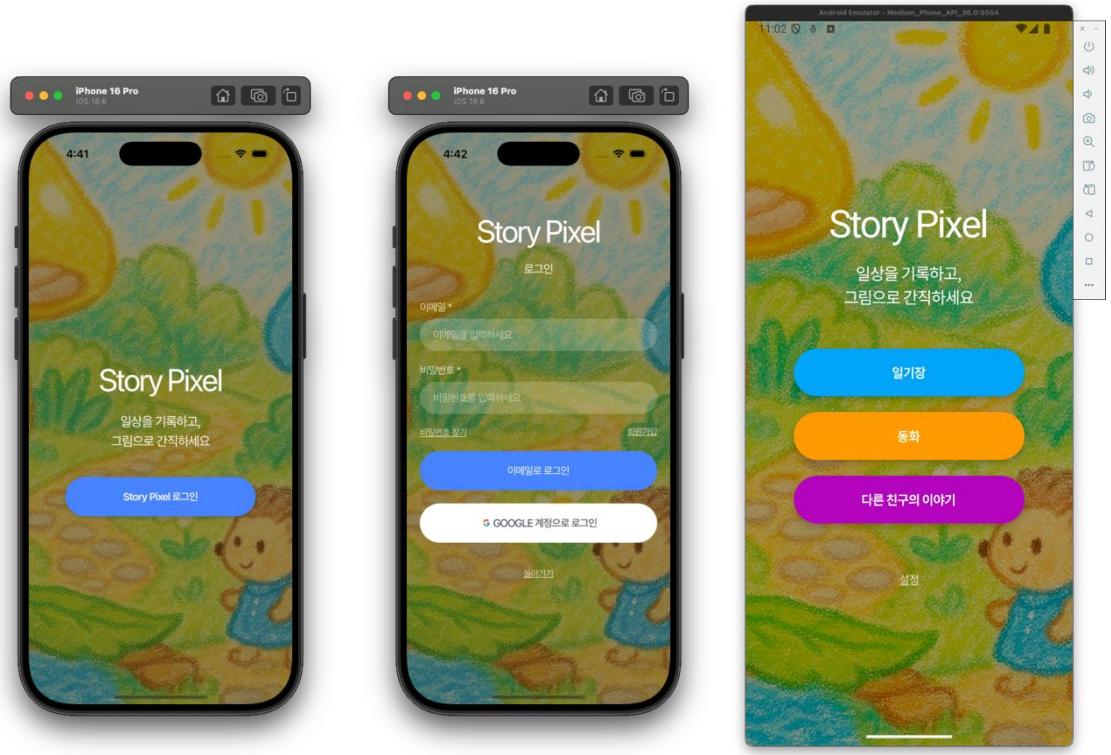
AI가 생성한 이미지 파일은 Firebase Storage에 저장하고, Firestore에는 이미지 파일의 URL만 저장하여 데이터베이스의 효율성을 높였다.

Firebase Cloud Functions를 활용해 AI 모델을 호출하는 로직을 구현했다. 모바일 앱은 이 함수 엔드포인트에 요청을 보내고, 함수가 Google API 키를 안전하게 사용하여 AI 모델을 호출한 뒤 결과를 반환한다. 이 방식은 API 키의 앱 노출을 원천적으로 막는 보안 조치이다.

3.2. 연구 결과

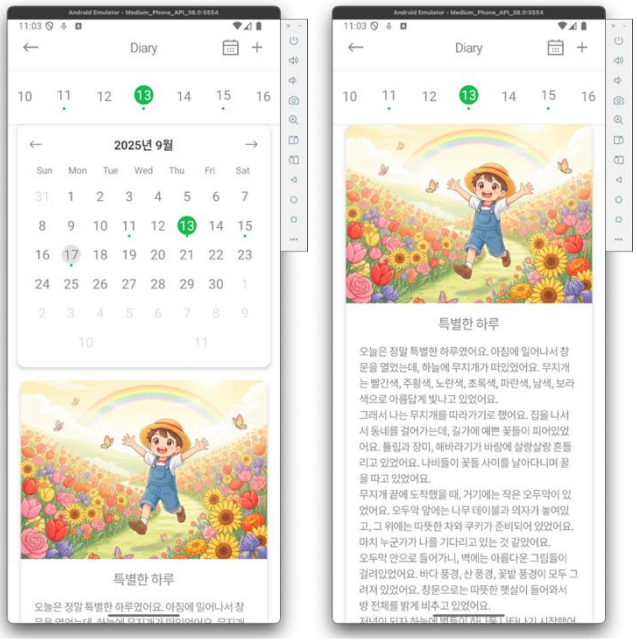
본 항목에서는 실제 구현한 앱의 스크린샷과 함께 기능에 대해 설명한다.

<그림 5>는 초기 화면 및 로그인, 로그인 후 첫 화면이다. 회원가입 및 로그인 기능으로 사용자는 개인 일기와 동화를 저장할 수 있다. 초기 화면에서는 일기장 및 동화 메뉴를 선택할 수 있고, 공유 기능은 아직 구현되지 않았다.



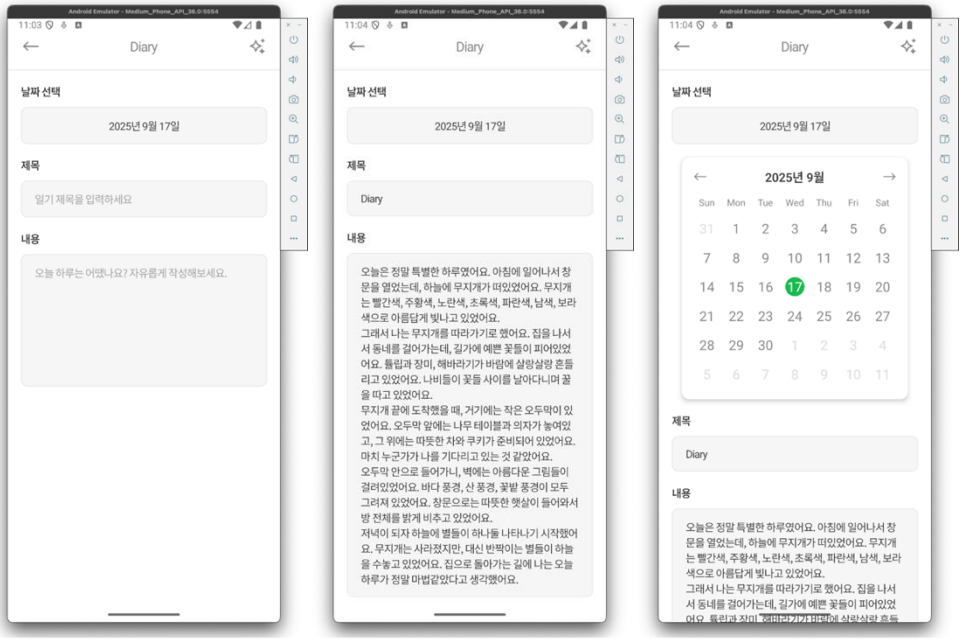
<그림 5>. 초기 화면 및 로그인 화면, 사용자 인증 후 초기 화면

<그림 6>은 일기 메뉴이다. 이미 저장된 일기가 있다면 캘린더를 통해 해당 날짜로 이동하여 저장된 일기를 확인 가능하고, 일기를 새로 작성하고 싶다면 '+' 버튼을 통해 일기 작성 페이지로 넘어갈 수 있다.



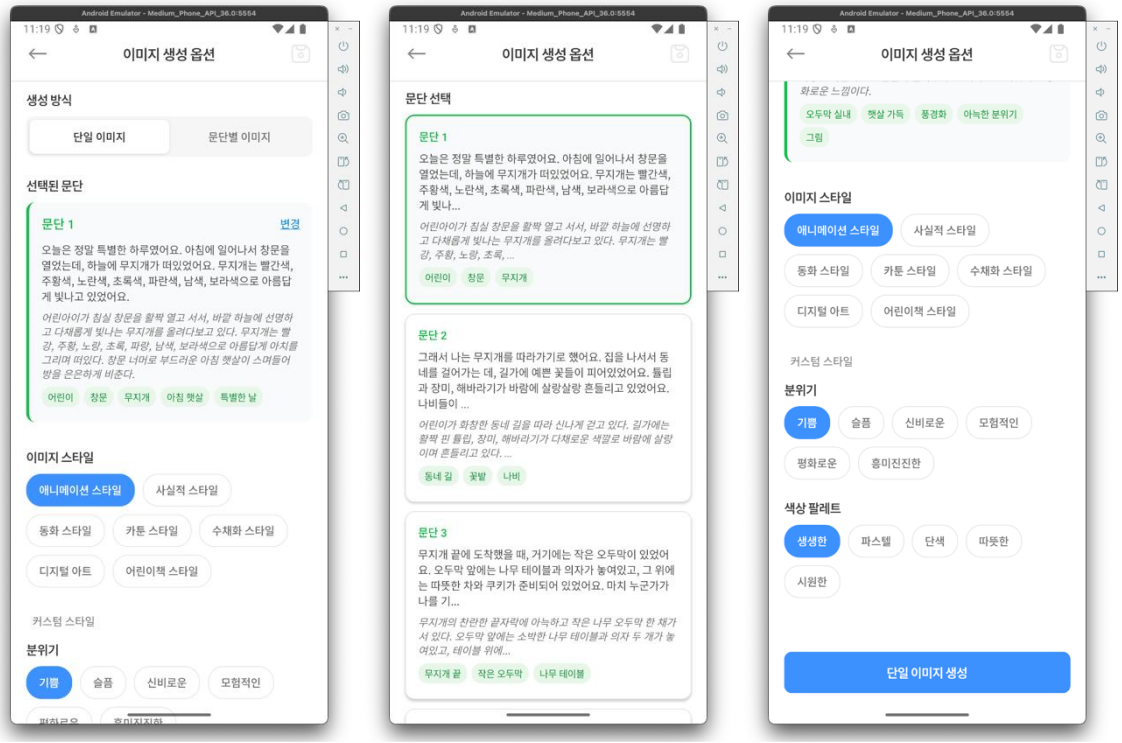
<그림 6>. 일기 메뉴

<그림 7>은 일기 작성 메뉴 이다. 사용자는 날짜를 선택하고, 일기의 제목과 내용을 작성한다. 그 후 우 상단 AI버튼을 누르면 일기 문단 분석이 시작된다.



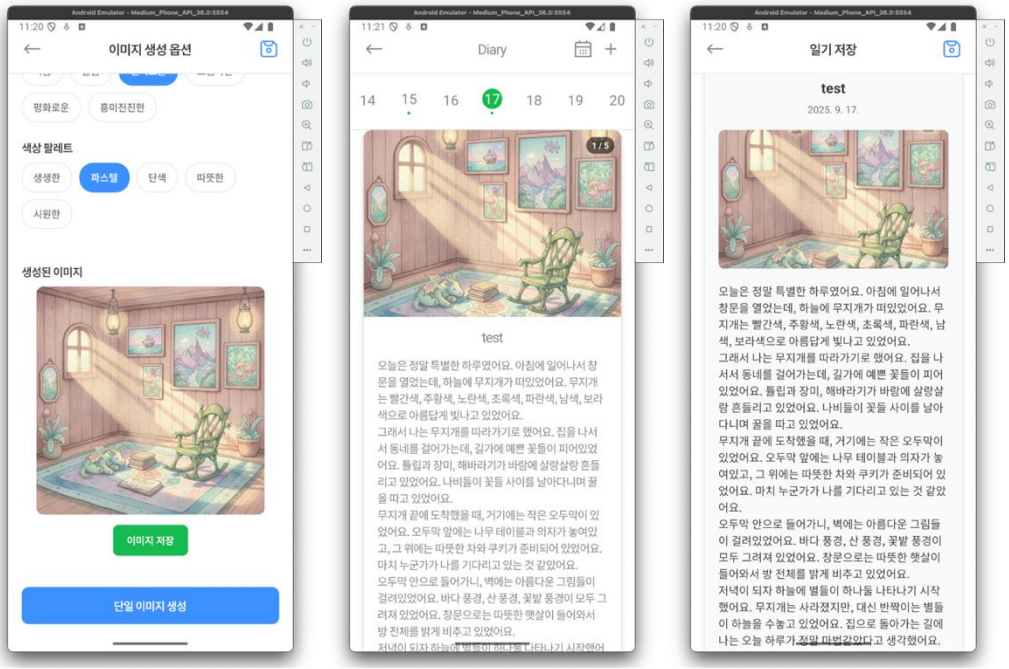
<그림 7>. 일기 쓰기 메뉴

<그림 8>은 AI를 통한 문단 분석 및 키워드 추출 메뉴 이다. 미리 정해진 프롬프트와 사용자가 작성한 내용으로 Google Gemini 2.5 Flash가 문단을 나누고, 등장인물이나 상황, 분위기를 판단한다. 문단이 나뉘어진 후 사용자는 해당 메뉴에서 그림을 생성하기 전 그림 생성에 사용할 문단을 선택하거나 전체 문단을 선택할 수 있고 원하는 그림체, 분위기, 색상 팔레트를 수동으로 결정할 수 있다. 이미지 생성 버튼을 누르면 사용자의 설정에 따라 그림 생성 프롬프트가 생성 되고 Nano Banana에 그림 생성을 요청 한다.



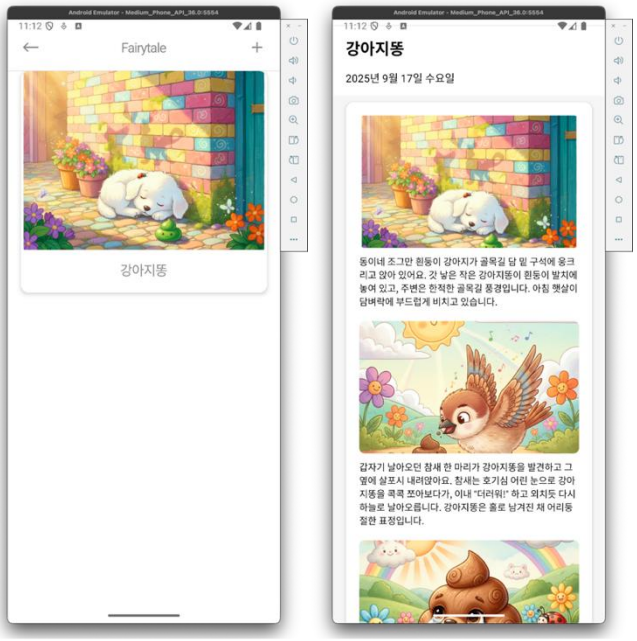
<그림 8>. 일기 AI 문단 분석 및 키워드 추출 기능

<그림 9>는 생성된 이미지를 확인하고 저장하는 단계이다. 사용자는 완성된 이미지를 따로 저장이 가능하고, 일기와 이미지를 함께 저장 할 수 있다 저장된 일기는 일기 메뉴에서 확인 가능하다..



<그림 9>. 이미지 생성 및 저장

<그림 10>은 동화 메뉴이다. 날짜를 선택하는 메뉴가 없을 뿐 전체적인 기능과 작동 방식은 일기와 동일하다.



<그림 10>. 동화 메뉴

4. 연구 결과 분석 및 평가

4.1. 연구 결과

4.1.1. 성능 평가

- 문단 분석 및 키워드 추출 성능: 약 20개의 서로 다른 공백을 포함한 200~300자 정도의 일기를 사용하여 문단을 나누고 키워드를 추출 하는 데에 걸린 시간을 측정 하고, 문단 분리 및 키워드 추출의 품질을 확인한다. 결과는 <표 5>와 같으며, 평균 약 43초, 최대 56.33초, 최소 4.54초가 소요 되었다. 가장 빠른 시간과 가장 느린 시간 사이 간극이 커서 최적화가 좀 더 필요하다. 문단 분리 기능은 글의 분위기 전환이나 실제 문단의 분리에 맞춰 잘 작동 하였으며, 키워드 추출 기능은 주요 등장 인물과 사건, 분위기나 기분을 잘 추출 하여 준수한 성능을 보여 주었다.

시도(회차)	걸린 시간(초)	비고
1	35.67	
2	04.54	최소 소요 시간
3	40.15	
4	50.41	
5	49.17	
6	27.81	
7	51.02	
8	45.04	
9	46.78	
10	42.06	
11	32.19	
12	44.11	
13	56.33	최대 소요 시간
14	52.88	
15	38.92	
16	43.44	
17	47.95	
18	41.38	
19	53.16	
20	48.33	

<표 5>. 문단 분석 및 키워드 추출 소요 시간

- 그림 생성 성능: 앞서 분리한 문단을 바탕으로 그림을 생성하기 위한 프롬프트 생성 및 그림 하나를 생성 하는 데에 소요되는 시간을 측정한다. 결과가 나오면 글의 분위기와 상황에 적절한 그림이 생성 되었는지 평가 한다. 결과는 <표 6>과 같으며 평균 약 39초, 최대 62.71초, 최소 6.68초가 소요 되었다. 마찬가지로 최대 시간과 최소 시간의 간극이 커 최적화가 필요하다. 전반적으로 글의 분위기와 등장인물, 상황에 맞는 그림이 생성 되었고, 사물이 자연스럽게 배치 되었다.

시도(회차)	걸린 시간(초)	비고
1	22.96	
2	50.00	
3	46.75	
4	15.55	
5	38.68	
6	62.71	최대 소요 시간
7	62.65	
8	38.04	
9	60.30	
10	24.02	
11	34.11	
12	08.88	
13	35.63	
14	53.91	
15	23.58	
16	55.47	
17	41.68	
18	36.67	
19	61.95	
20	06.68	최소 소요 시간

<표 6>. 그림 생성 소요 시간

4.1.2. 사용자 경험(UX) 평가

- 직관적인 UI: <그림 11>과 같이 주변 지인 18명의 평가에 따르면 성인의 관점에서 보았을 때 충분히 직관적이고 사용성이 좋다는 평가를 받았다. (개인 평가 5점 만점 중 평균 4.6점) 그러나 그림 생성 아이콘이 너무 작아 튜토리얼 없이 처음 사용 하면 어려울 수도 있다는 피드백을 받았다.

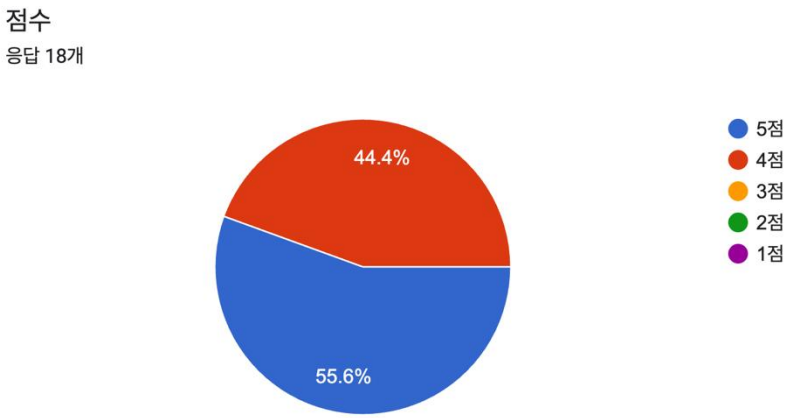


그림 11. 프로젝트 평가 점수 그래프

그림 12. 프로젝트 평가 의견

- 어린아이가 사용하기에 용이한지 여부: 본 과제의 주 타겟 연령층은 초등 저학년의 어린아이이다. 해당 연령층이 사용할 수 있을 만큼 직관적이고 쉬운 UI/UX인지 여부를 확인 하기 위해 실제 어린아이에게 테스트할 수 있는 환경이 아니었기 때문에 해당 항목은 확인 할 수 없었다.

4.1.3. 기술적 한계점

- AI 이미지의 미세 조정: 현재 AI 모델은 사용자가 직접 세부적인 이미지를 수정하는 데 한계가 있다.

- Vertex AI 적용 실패: 처음 계획했던 Vertex AI 와 React Native 환경의 안정적인 연결을 위한 레퍼런스 및 가이드라인이 부족하여 대신 Gemini AI를 사용하는 결정을 내리게 되었다. 이는 프로젝트의 안정성과 효율성을 확보 했으나 기능이 제한 적이다.(파인튜닝 불가능 등)

5. 결론 및 향후 연구 방향

5.1. 결론

본 과제에서 React Native 를 활용한 효율적인 크로스 플랫폼 개발로 프로젝트의 신속한 구현을 달성했으며, Google Gemini AI 를 중심으로 구축된 자동화된 프롬프트 엔지니어링 파이프라인을 통해 사용자가 느낄 AI 기술의 복잡성을 완화했다. 이러한 기술적 연구는 사용자인 어린이가 별도의 학습 없이도 자신의 이야기를 시각화하고 창의력을 발휘할 수 있는 직관적인 환경을 제공하는 데 결정적인 역할을 할 것으로 기대된다. 성인 사용자 테스트를 통해 앱의 직관적인 사용성과 창작 활동에 대한 긍정적인 반응(성인 18 명을 대상. 5 점 만점 중 평균 4.6 점)을 확인하며, 본 프로젝트가 설정했던 목표의 일부를 달성했음을 확인 하였다.

5.2. 향후 계획

5.2.1. 기능 추가

- 개인화(페르소나 구축) 기능 추가: 현재 AI는 일반적인 묘사를 기반으로 그림을 생성하고 있다. 향후 사용자가 가족, 친구, 반려동물의 사진과 특징을 등록하면, Gemini의 멀티모달 기능을 활용하여 텍스트 프롬프트와 함께 사용자의 이미지를 직접 참조하는 방식으로 기술을 고도화 시켜 AI가 이를 기억하고 이야기 속에 등장하는 인물들을 일관된 모습으로 그려주는 개인화 기능을 추가할 계획이다.

- 음성 인식 기능: 텍스트 입력 외에 음성으로 이야기를 말하면 자동으로 텍스트로 변환해주는 기능을 추가하여, 글쓰기에 어려움을 느끼는 어린이나 더 어린 연령대의 사용자

도 쉽게 앱을 이용할 수 있도록 접근성을 높일 것이다.

- 다국어 지원: i18n(국제화) 기술을 적용하여 한국어 외에 다양한 언어를 지원함으로써, 글로벌 사용자들이 자신의 언어로 창의력을 펼칠 수 있는 환경을 제공할 예정이다.

- 소셜 기능: 완성된 스토리북을 다른 사용자들과 공유하는 소셜 기능을 추가하여 창작의 즐거움을 나누고 동기를 부여하는 커뮤니티를 구축할 예정이다.

- UI/UX 개선 및 튜토리얼 개설: 피드백을 받은 대로 앱 아이콘의 위치나 크기 등 UI/UX를 개선하고, 앱을 처음 사용하는 사용자를 위한 튜토리얼을 추가할 예정이다.

5.2.2. 기능 외 사항

- 실제 타겟층 대상 테스트: 아직 실제 타겟 연령층인 어린아이들을 상대로 어플리케이션을 테스트 한 적이 없으므로 지인 혹은 교육기관과 연계하여 이를 진행할 예정이다.