



픽소 iOS 과제 - 김시종

2024.08.07 ~ 2024.08.13 진행한 픽소 iOS 과제 전형입니다.

기능 구현 gif 파일로 넣었으나 pdf에서 제대로 작동되지 않는 것 같아 노션 링크 남겨드립니다.

감사합니다.

[노션링크](#)

1. 프로젝트 설명

Custom Album

사전 과제 주제인 iOS 사진앱과 같은 앱을 구현하기 위해 PHAsset을 통해 사진 라이브러리에 저장되어있는 사진을 자동으로 동기화하여 관리하고 다양한 필터 및 조정, 자르기, 블러 등 사진을 편집할 수 있는 기능을 구현했습니다.

CoreImage를 활용해 가져온 사진을 편집(필터, 자르기, 조정, 블러 등)해 저장하여 사진 라이브러리에 추가할 수 있고 CoreData를 활용해 즐겨찾기 된 사진을 별도로 저장해 관리할 수 있도록 구현했습니다.

아키텍처 구조 : MVVM

사용 언어 : SwiftUI

비동기 처리 : Combine, Swift Concurrency, GCD

프레임 워크 : Core Image, Core Data, Photos(PHAsset), CoreLocation, Combine

2. 주요 트러블 슈팅

1. 디바이스 연결 시 대량의 사진 라이브러리 데이터로 앱 과부하 현상

- 처음 구현 시 사진과 연결하여 앨범에 있는 사진을 동기화 할 수 있도록 구현하였고 다수의 데이터 문제는 LazyGrid를 활용하면 해결 될 것으로 생각했으나, 모든 데이터를 로드하게 되므로써 앱이 과부하 되어 정상적으로 작동하지 않는 문제가 있었습니다.

→ pagination을 활용해 스크롤이 내려갔을 때 50장 씩 사진을 로드하는 방향으로 구현하여 데이터를 과도하게 가져와 앱이 과부하 되는 현상을 해결할 수 있었고 보다 빠르게 데이터를 전달 받아 효율성이 더욱 높아질 수 있었습니다.

2. AlbumView, FavoriteView가 하나의 FullScreenPhotoView를 공유하면서 타입 불일치 오류

- 각각의 사진을 가지고 있는 뷰에서 사진 클릭 후 FullScreenPhotoView로 넘어갈 때 Photo, Favorite의 배열로 각각 저장되어 있어 타입 불일치 오류

→ Favorite의 값을 Photo에 다시 매치 해주면서 타입을 Photo로 일치시켜 해당 오류 수정

```
extension Favorite {
    func toPhoto() -> Photo {
        var asset: PHAsset? = nil
        if let assetIdentifier = self.assetIdentifier {
            let fetchResult = PHAsset.fetchAssets(withLocalIdentifiers: [assetIdentifier], options: nil)
            asset = fetchResult.firstObject
        }

        return Photo(
            id: self.id ?? UUID().uuidString,
            image: UIImage(data: Data(base64Encoded: self.image ?? "") ?? Data()) ?? UIImage(),
            date: self.date,
            location: self.location,
            isFavorite: self.isFavorite,
            asset: asset,
            assetIdentifier: self.assetIdentifier
        )
    }
}
```

3. 이미지 편집 중 필터, 조정, 자르기, 블러 등 각각의 이미지 편집 과정 충돌

- 편집 과정 중 각 편집 값이 충돌하여 정상적으로 변경 값을 가지지 못해 사진이 변경된 사진이 아닌 원본사진이 추가적으로 저장되는 문제 및 다른 편집 기능이 불가능한 오류가 있었습니다.

로직 확인 결과, 각각의 이미지 편집 과정은 정상적으로 작동하였으나, 이미지 적용

중 편집된 이미지의 충돌이 있었고 이를 통해 다른 편집 기능에 영향을 주고 있다는 사실을 인지했습니다.

→ 각 편집 과정에서 적용되는 이미지를 설정했습니다. 처음에는 `editedImage` 하나로 통합해 변경된 이미지를 관리하려 했으나, 각 이미지 변경 값에 대한 이미지를 넣어주고 기능 별 VM으로 나눠 이미지를 관리할 수 있도록 수정해 최종 이미지를 전달 하는 방식으로 로직을 수정했습니다.

```
private var displayedImage: UIImage {
    var finalImage = viewModel.currentPhoto.image

    if editViewModel.selectedAction == .adjustment, let adjustedImage = adjustmentViewModel.adjustedImage {
        finalImage = adjustedImage
    }

    if editViewModel.selectedAction == .filter, let filteredImage = filterViewModel.filteredImage {
        finalImage = filteredImage
    }

    if editViewModel.selectedAction == .blur, let blurredImage = blurViewModel.blurredImage {
        finalImage = blurredImage
    }

    if editViewModel.selectedAction == .crop, let croppedImage = cropViewModel.croppedImage {
        return croppedImage
    }

    return finalImage
}
```

3. 주요 기능



1. 사진 동기화

- 사진 라이브러리에서 사진을 동기화 하여 앨범을 관리하고 즐겨찾기에 추가하거나 삭제하는 기능
- 모든 사진을 스크롤 해 탐색이 가능하며, 사진을 터치해 상세 정보를 확인할 수 있습니다.

2. 사진 편집

- 다양한 필터를 적용해 사진을 변경할 수 있습니다.
- 밝기, 대비, 채도, 선명도, 노출 등 다양한 속성을 조정해 세밀한 이미지 편집이 가능합니다.
- CropBox를 활용해 원하는 이미지 사이즈로 자르기가 가능합니다.
- 원하는 곳을 클릭해 블러 효과를 적용할 수 있습니다.

3. 데이터 저장

- CoreData를 사용해 별도 저장이 가능하도록 구현했습니다.

4. 이미지 처리

1. 필터 적용

- CoreImage의 다양한 필터를 활용해 사진에 효과를 적용할 수 있습니다.
- 사용 가능한 필터는 Sepia Tone, Noir, Chrome, Instant, Fade, Monochrome, Posterize, Vignette 중 선택할 수 있도록 했습니다.
- applyFilter 메서드를 통해 필터 이름을 지정하고 필터가 적용된 이미지를 반환하는 로직으로 구현했습니다.

2. 자르기 적용

- 사진의 특정 영역을 자르거나, 정사각형, 4:3 등 이미지를 자를 수 있는 기능을 제공합니다.
- CropBox를 활용해 원하는 만큼 사이즈를 조정해 사용할 수 있습니다.

3. 이미지 조정

- CoreImage를 활용해 밝기, 대비, 채도, 선명도, 노출, 생동감, 하이라이트 등 이미지 속성을 세밀하게 조정할 수 있습니다.
- 편집한 설정값을 실시간으로 이미지가 조정됩니다.

4. 블러 효과

- 사용자가 화면에 터치한 위치에 블러 효과를 적용할 수 있습니다.
- applyBlur 메서드를 활용해 사용자가 터치한 위치를 중심으로 블러가 적용되며, 마스크를 활용해 자연스러운 블러 효과를 생성합니다.

5. 사진 관리

1. CoreData

- saveFavoritePhoto , saveFavoritePhoto 메서드를 활용해 즐겨찾기 상태의 사진을 저장하거나 삭제 할 수 있습니다.
CoreData를 통해 저장된 데이터를 fetchFavoritePhotos 메서드를 통해 실시간으로 로드합니다.

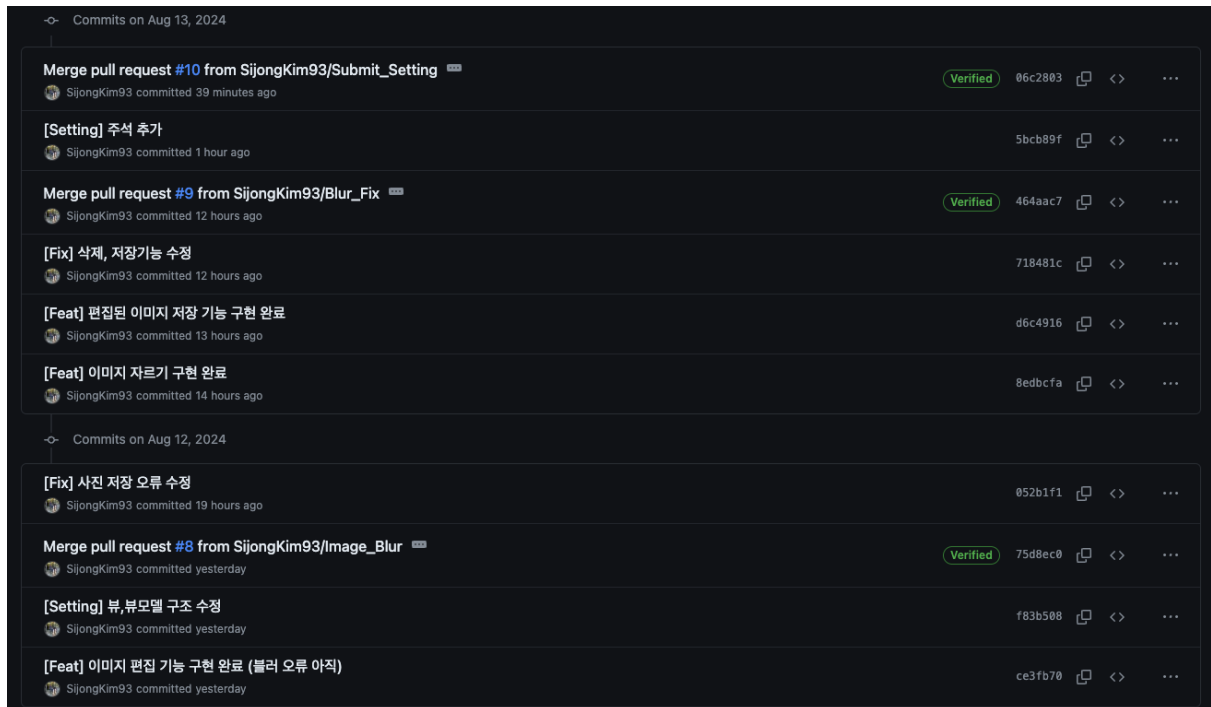
2. 사진 라이브러리 접근

- PermissionManager 를 활용해 앱 실행 시 사용자의 사진 라이브러리 접근 권한을 확인하고 권한이 승인되면 PHAsset을 활용해 사진 라이브러리에 저장되어있



















는 사진을 로드합니다.

- 권한이 승인되지 않으면 `openSettingsURLString`를 활용해 권한을 다시 승인할 수 있도록 유도하는 방향으로 구현했습니다.

6. GitHub



private repository를 통해 각 상황별 개발 진행 과정을 기록했습니다.

Branch	Updated
Submit_Setting 	 1 hour ago
Blur_Fix 	 12 hours ago
Edit_Save 	 14 hours ago
Image_Blur 	 yesterday
Image_Edit 	 2 days ago
Photo_info 	 3 days ago
Photo_Edit 	 4 days ago
Photo_UI 	 5 days ago
main 	 last week

기능 구현 별로 branch를 나눠 해당 하는 기능을 구현하고 해당 내용에 대한 히스토리 확인이 필요할때 적절하게 사용하였습니다.