모바일 프로그래밍 팀프로젝트 개별 보고서

제출자: 김미소, 202312602

김미소

1. UI/UX 디자인

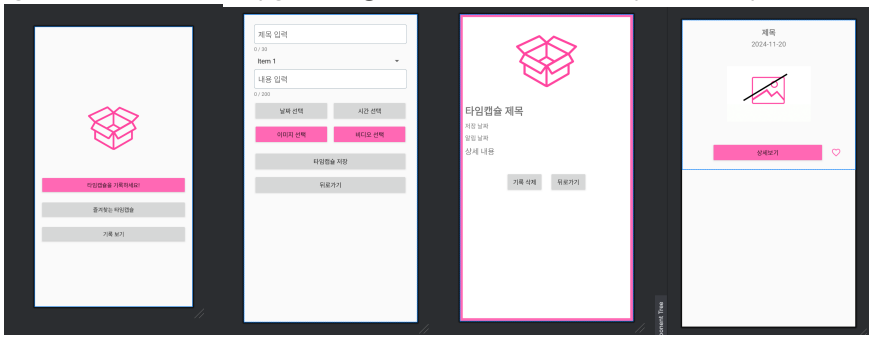
저는 프로젝트의 UI/UX 디자인과 앱의 전체적인 프론트엔드 설계를 담당했습니다. 사용자가 직관적으로 앱을 사용할 수 있도록 설계하며, 화면 간의 전환을 명확하게 정의하고, 각 화면을 시각적으로 깔끔하게 구성했습니다.

1-1 스플래시 화면: 사용자가 앱을 처음 실행할 때 3초 동안 보여지는 스플래시 화면을 구현했습니다. (앱 아이콘 및 모든 xml, 이미지요소는 직접 전부 제작함)


```
class SplashActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash)

        Handler(Looper.getMainLooper()).postDelayed({
            startActivity(Intent(this, MainActivity::class.java))
            finish()
        }, 3000)
    }
}
```

1-2 시각적 요소: RecyclerView를 사용하여 타임캡슐 목록을 표시하고
핑크,그레이,화이트 색상만 사용하여 심플하지만 기억에 남도록 디자인했습니다.



2. 데이터베이스 설계

앱에서 사용되는 데이터 모델을 설계하고, 이를 바탕으로 타임캡슐 데이터를 SQLite데이터베이스에 저장하고 관리할 수 있도록 했습니다. 각 타임캡슐 항목에는 제목, 날짜, 감정 태그, 이미지, 비디오 등의 정보가 포함됩니다. MemoryData모델 클래스를 정의하여 각 타임캡슐의 데이터를 객체로 저장했습니다.

3. 다중 이미지선택, 이미지, 비디오 저장 기능

앱에서 다중 이미지와 비디오를 선택할 수 있는 기능, 타임캡슐에 저장한 이미지와 비디오를 갤러리에 저장할 수 있는 기능을 구현하였습니다. 사용자가 여러 이미지를 선택하거나 비디오 파일을 선택하여 타임캡슐에 첨부할 수 있도록 했습니다. Intent를 사용하여 갤러리에서 여러 이미지를 선택하고, 이를 앱에 반영하도록 했습니다.

3-1 상세화면에서 이미지 저장 및 보기 기능 (비디오도 마찬가지로 로직 동일하므로 생략)

3-1.1 이미지 목록 표시 (RecyclerView 사용)

LinearLayoutManager를 이용해 수평 스크롤로 구성되었습니다.
각 이미지 항목은 Glide라이브러리를 사용해 로드됩니다.
이때 placeholder_image와 error_image를 사용하여 이미지가 로드되기 전과 오류가 발생했을 때 표시할 이미지를 설정합니다.

```
recyclerView.layoutManager = LinearLayoutManager(this, LinearLayoutManager.HORIZONTAL, false)
recyclerView.adapter = object : RecyclerView.Adapter<ImageViewHolder>() {
    // onCreateViewHolder와 onBindViewHolder에서 Glide를 사용하여 이미지를 로드하고, 클릭 시 이미지를 선택하도록 설정
}
```

3-1.2 이미지 상세보기 (풀스크린 모드)

viewFullScreenButton을 클릭하면 선택된 이미지를 풀스크린 모드로 볼 수 있게 됩니다. 이미지는 PhotoView를 사용하여 확대/축소할 수 있도록 표시됩니다.
AlertDialog를 사용하여 이미지를 풀스크린 모드로 보여주고, 사용자가 이미지를 터치하여 확대할 수 있도록 합니다.

```
private fun showImageInFullScreen(imageUri: String) {
    try {
        val photoView = PhotoView(this)
        photoView.layoutParams = FrameLayout.LayoutParams(
            FrameLayout.LayoutParams.MATCH_PARENT,
            FrameLayout.LayoutParams.MATCH_PARENT
        )

        Glide.with(this)
            .load(Uri.parse(imageUri))
            .error(R.drawable.error_image)
            .placeholder(R.drawable.placeholder_image)
            .into(photoView)

        val layout = FrameLayout(this)
        layout.addView(photoView)

        val dialog = AlertDialog.Builder(this)
            .setView(layout)
            .setCancelable(true)
            .create()

        dialog.show()
    } catch (e: Exception) {
        Toast.makeText(this, "이미지를 로드하는 중 오류 발생: ${e.localizedMessage}", Toast.LENGTH_SHORT).show()
```

```
    }
}
```

3-1.3 이미지 저장 (개별 이미지 및 전체 이미지)

개별 이미지를 갤러리에 저장할 때는 saveImage함수가 호출됩니다. 이미지를 Drawable로 로드한 후, Bitmap으로 변환하고 이를 갤러리에 저장합니다.
전체 이미지를 저장할 때는 saveAllImages함수가 호출되며, imageUris목록에 있는 모든 이미지를 저장하려 시도합니다. 저장 성공/실패 횟수를 Toast로 표시합니다.

```
private fun saveImage(imageUri: Uri) {
    val drawable = loadDrawableFromUri(imageUri)
    if (drawable != null) {
        val bitmap = convertDrawableToBitmap(drawable)
        if (bitmap != null && saveImageToGallery(bitmap)) {
            Toast.makeText(this, "이미지가 갤러리에 저장되었습니다.",
Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText(this, "이미지 저장에 실패했습니다.",
Toast.LENGTH_SHORT).show()
        }
    }
}

private fun saveAllImages() {
    var successCount = 0
    var failureCount = 0

    for (imageUri in imageUris) {
        val drawable = loadDrawableFromUri(imageUri)
        if (drawable != null) {
            val bitmap = convertDrawableToBitmap(drawable)
            if (bitmap != null && saveImageToGallery(bitmap)) {
                successCount++
            } else {
                failureCount++
            }
        }
    }

    Toast.makeText(this, "성공적으로 저장된 이미지: $successCount, 저장 실패:
$failureCount", Toast.LENGTH_SHORT).show()
}
```

3-1.4 이미지 로드 및 변환

이미지는loadDrawableFromUri함수를 통해 Uri에서 로드됩니다. Drawable.createFromStream메서드를 사용하여 Uri로부터 Drawable을 가져옵니다.
Drawable을 Bitmap으로 변환하는 과정은 convertDrawableToBitmap메서드에서 이루어집니다. Bitmap은 Canvas를 이용해 Drawable을 그려서 변환합니다.

```
private fun loadDrawableFromUri(uri: Uri): Drawable? {
    return try {
        val inputStream = contentResolver.openInputStream(uri)
        Drawable.createFromStream(inputStream, uri.toString())
    } catch (e: Exception) {
```

```
            Toast.makeText(this, "이미지 로드 실패: ${e.localizedMessage}",
Toast.LENGTH_SHORT).show()
            null
        }
    }
    private fun convertDrawableToBitmap(drawable: Drawable): Bitmap? {
        return try {
            val bitmap = Bitmap.createBitmap(
                drawable.intrinsicWidth,
                drawable.intrinsicHeight,
                Bitmap.Config.ARGB_8888
            )
            val canvas = android.graphics.Canvas(bitmap)
            drawable.setBounds(0, 0, canvas.width, canvas.height)
            drawable.draw(canvas)
            bitmap
        } catch (e: Exception) {
            Toast.makeText(this, "Drawable -> Bitmap 변환 실패: ${e.localizedMessage}",
Toast.LENGTH_SHORT).show()
            null
        }
    }
```

3-1.5 이미지 저장 위치 및 갤러리 저장

이미지는 Android의 MediaStore.Images.Media를 사용하여 갤러리에 저장됩니다. "DigitalMemoryBox"라는 폴더에 이미지를 저장하며, ContentValues를 이용해 이미지의 메타데이터를 설정합니다.

```
val contentValues = ContentValues().apply {
    put(MediaStore.Images.Media.DISPLAY_NAME,
"saved_image_${System.currentTimeMillis()}.jpg")
    put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg")
    put(MediaStore.Images.Media.RELATIVE_PATH, "Pictures/DigitalMemoryBox")
}
```

3-1.6 버튼 동작

이미지 저장 버튼 (saveImageButton): 현재 선택된 이미지를 저장합니다.
전체 이미지 저장 버튼 (saveAllImagesButton): 목록에 있는 모든 이미지를 저장합니다.
뒤로 가기 버튼 (backButton): 이전 화면으로 돌아갑니다.

```
saveImageButton.setOnClickListener {
    val currentUri = selectedImageUri ?: imageUris.firstOrNull()
    currentUri?.let {
        saveImage(it)
    }
}
saveAllImagesButton.setOnClickListener {
    saveAllImages()
}
backButton.setOnClickListener {
    finish()
}
```

3-1.7 에러 처리 및 사용자 알림

이미지를 로드하는 중에 오류가 발생하면, Toast를 통해 사용자에게 오류 메시지를 전달합니다.
Glide를 사용하여 이미지 로딩을 처리하며, 로딩 실패 시 기본 이미지를 보여줍니다.

```
lide.with(this@ImageManagerActivity)
    .load(imageUri)
    .apply(
        RequestOptions()
            .placeholder(R.drawable.placeholder_image)
            .error(R.drawable.error_image)
    )
    .into(holder.imageView)
 .
```

4. 즐겨찾기 상태 동기화 및 화면 반영

타임캡슐을 즐겨찾기 목록에 추가하거나 제거할 수 있는 기능을 구현하였습니다. 즐겨찾기 상태를 변경하면 즉시 UI가 반영됩니다.

```
val favoriteIcon = if (memory.isFavorite) R.drawable.ic_favorite else R.drawable.ic_favorite_border
itemHolder.favoriteButton.setImageResource(favoriteIcon)

itemHolder.favoriteButton.setOnClickListener {
    val isNowFavorite = !memory.isFavorite
    memory.isFavorite = isNowFavorite

    val dbHelper = MemoryDatabaseHelper(context)
    val isUpdated = dbHelper.updateMemoryFavoriteStatus(memory.id, isNowFavorite)

    if (isUpdated) {
        if (isNowFavorite) {
            notifyItemChanged(position)
        } else {
            onFavoriteStatusChanged?.invoke(memory, position)
        }

        Toast.makeText(
            context,
            if (isNowFavorite) "즐겨찾기에 추가되었습니다." else "즐겨찾기에서 제거되었습니다.",
            Toast.LENGTH_SHORT
        ).show()
    } else {
        Toast.makeText(context, "즐겨찾기 상태 업데이트에 실패했습니다.", Toast.LENGTH_SHORT).show()
    }
}
```