**스마트앱프로그래밍 Term Project 결과 보고서**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 분반 | 02 분반 | | | |
| 작 품 명 | Monster Henter Index | | | |
| 개발기간 | 2023 년 12 월 15 일 ～ 2023 년 12 월 16 일 | | | |
| 지도교수 | 컴퓨터공학과 | | 오병우 | |
| 구 분 | 학년 | 학 번 | | 성 명 |
| 책임자(팀장) | 3 | 20190558 | | 배진열 |
| 팀원1 | 해당 사항 없음 |  | |  |
| 팀원2 | 해당 사항 없음 |  | |  |
| 팀원3 | 해당 사항 없음 |  | |  |
| 본인은 스마트앱프로그래밍 Term Project 결과 보고서를 첨부와 같이 제출합니다. 프로젝트를 진행할 때 부정한 방법을 사용했거나 보고서에 사실과 다른 내용을 적었을 때는 모든 처벌을 감수하겠습니다.  2023년 12월 19일  팀 장 배진열.    컴퓨터공학심화프로그램 | | | | |

**결과 보고서**

**Monster Hunter Index**

**1. 서론**

1. **개발 목표**

주식회사 캡콤(CAPCOM)에서 출시한 Monster Hunter 시리지 중 가장 인기가 있는 Monster Hunter World와 확장 팩인 ICEBORN에 등장하는 다양한 대형 몬스터들의 정보를 손쉽게 접근하고 활용할 수 있는 몬스터 도감 앱을 개발합니다. 이 앱은 초보자들이 몬스터의 약점 및 특성을 쉽게 파악하여 사냥 전략을 세우고, 몬스터의 아이콘 및 이미지를 시각적으로 확인할 수 있도록 합니다.

1. **배경 설명**

**텍스트, 스크린샷, 구름, 하늘이(가) 표시된 사진

자동 생성된 설명**

<뉴스 민 기사 중 일부>

Monster Hunter World 시리즈는 대형 몬스터를 사냥하고 소재 및 광물 등을 수집하여 강화된 장비를 만들어 나아가는 액션 RPG게임입니다. 게임 내에 등장하는 몬스터는 각각 고유한 특성과 약점을 가지고 있어, 효과적인 전투 전략을 수립하기 위해서는 해당 몬스터에 대한 상세한 정보가 필요합니다. 이러한 필요성을 충족시키기 위해 몬스터 도감 앱을 개발하고자 합니다.

1. **개발동기**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

<Game Meca 기사 중 일부>

Monster Hunter World는 커다란 게임 세계와 다양한 몬스터들을 제공하면서도, 이에 대한 정보 제공은 제한적입니다. 특히 초보자들은 각 몬스터의 특성을 파악하는데 어려움을 겪는 경우가 많습니다. 따라서 이 앱은 사용자들이 게임 플레이를 보다 효과적으로 즐길 수 있도록 도움을 주고자 합니다.

1. **관련 기술 현황**
2. 데이터 수집 및 저장

Monster Hunter World에 있는 70여 마리의 대형 몬스터의 정보를 <https://cloudtype.io/> 사이트를 활용하여 MariaDB에 저장합니다. 정보에는 몬스터의 이름, 별명, 약점, 서식지, 몬스터 아이콘 링크, 몬스터 이미지 링크 등이 존재합니다.

1. 데이터 배포

Database에 있는 데이터들을 <https://cloudtype.io/>사이트를 활용하여 서버를 만들어 데이터를 배포합니다. 배포한 데이터를 Android Studio에서 개발한 애플리케이션으로 가져옵니다.

1. 사용자 인터페이스

Database에서 가져온 데이터인 Json파일 형식을 Data class를 사용하여 형식에 맞게 데이터들을 리스트에 저장하고 사용자가 보기 쉽게 데이터를 가공하여 화면에 표시합니다.

**2. 작품 개요**

**2.1 전체 구성도**

|  |  |
| --- | --- |
| 앱 실행 시 홈 화면 | 디아블로스 Detail 화면 |
| **텍스트, 스크린샷, 폰트, 우표이(가) 표시된 사진  자동 생성된 설명** |  |

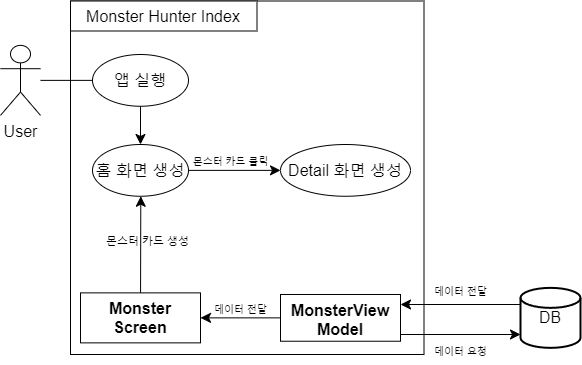
**2.2 작품 설명**

앱 실행 시 왼쪽 그림처럼 몬스터 헌터 월드에 나오는 대형 몬스터들이 표시가 되고 자세히 보고싶은 몬스터를 클릭을 하면 해당 몬스터의 실제 모습, 종류, 약점 속성, 서식지 등을 알려줍니다.

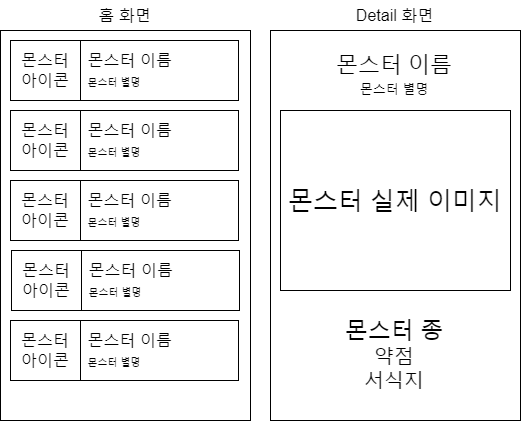
**3. 설계**

**3.1 Use Case**

**3.1.1 앱 실행 및 몬스터 카드 클릭**



**3.2 UI Design**

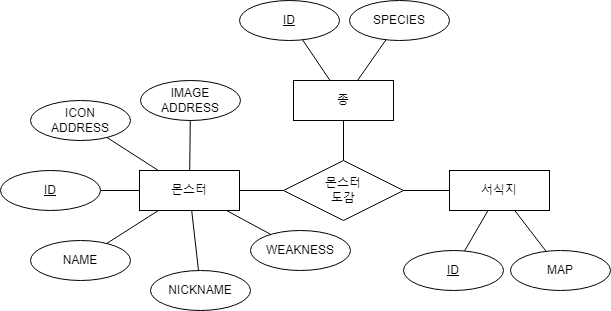
****

왼쪽 그림은 홈 화면으로 하나의 박스에 몬스터 아이콘, 몬스터 이름, 몬스터 별명을 표시합니다.

오른쪽 그림은 Detail 화면으로 상단에 몬스터 이름과 해당 몬스터의 별명이 표시가 되고 중간에 몬스터의 실제 이미지, 그 아래에 몬스터의 종, 약점, 서식지를 나타낸다.

**3.3 DB Design**

**3.3.1 Conceptual Design**

****

**3.3.2 Logical Design**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**- monster (몬스터)

|  |
| --- |
| create table monster (  id INT not null,  name VARCHAR(255) not null,  nickname VARCHAR(100),  weakness VARCHAR(255),  icon\_address TEXT,  image\_adress TEXT,  PRIMARY KEY(id)  ) |

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**- habiat (서식지)

|  |
| --- |
| create table habitat (  id INT not null,  map VARCHAR(255) not null,  PRIMARY KEY(id)  ) |

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**- species (종)

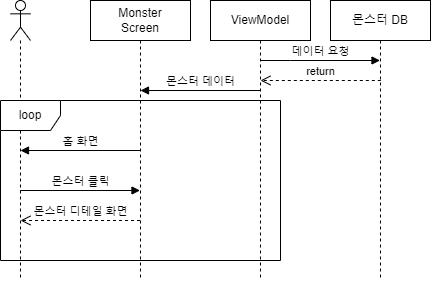
|  |
| --- |
| create table species (  id INT not null,  species VARCHAR(100),  PRIMARY KEY(id)  ) |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명- monsterdex (몬스터 도감)

|  |
| --- |
| create table monsterdex (  monsterid INT not null,  speciesid INT not null,  habitatid INT not null,  PRIMARY KEY(monstered,  speciesid, habitatid),    FOREIGN KEY(monsterid) REFERENCES monster(id)      ON DELETE CASCADE      ON UPDATE CASCADE,    FOREIGN KEY(speciesid) REFERENCES species(id)      ON DELETE CASCADE      ON UPDATE CASCADE,    FOREIGN KEY(habitatid) REFERENCES habitat(id)      ON DELETE CASCADE      ON UPDATE CASCADE,    INDEX(monsterid)  ) |

**3.4 Use Case별 Sequence Diagram**



**4. 구현**

**4.1 앱 실행**

**텍스트, 스크린샷, 폰트, 우표이(가) 표시된 사진

자동 생성된 설명**

<앱 실행 화면>

앱 실행 시 DB에서 데이터를 받아와 가공을 하여 화면에 출력한다.

**4.2 몬스터 카드 클릭**

**텍스트, 포유류, 공룡이(가) 표시된 사진

자동 생성된 설명**

<앱 실행 화면>

몬스터 카드를 클릭할 시 해당 몬스터의 상세 설명 화면으로 전환된다.

**4.3 소스코드 설명**

|  |
| --- |
| MainActivity.kt |
| package kr.ac.kumoh.ce.s20190558.monsterhunterindex  import android.os.Bundle import androidx.activity.ComponentActivity import androidx.activity.compose.setContent import androidx.activity.viewModels import androidx.compose.foundation.layout.fillMaxSize import androidx.compose.material3.MaterialTheme import androidx.compose.material3.Surface import androidx.compose.runtime.Composable import androidx.compose.runtime.getValue import androidx.compose.runtime.livedata.observeAsState import androidx.compose.ui.Modifier import kr.ac.kumoh.ce.s20190558.monsterhunterindex.ui.theme.MonsterHunterIndexTheme  class MainActivity : ComponentActivity() {  private val viewModel: MonsterViewModel by viewModels()   override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  setContent {  MainScreen(viewModel)  }  } }  @Composable fun MainScreen(viewModel: MonsterViewModel) {  val monsterList by viewModel.monsterList.observeAsState(emptyList())   MonsterHunterIndexTheme {  // A surface container using the 'background' color from the theme  Surface(  modifier = Modifier.fillMaxSize(),  color = MaterialTheme.colorScheme.background  ) {  MonsterApp(monsterList)  }  } } |

다음 코드는 앱 실행시 가장 먼저 실행되는 코드이다. MonsterViewModel을 만들어 MonsterApp에 넘겨준다.

|  |
| --- |
| MonsterViewModel.kt |
| package kr.ac.kumoh.ce.s20190558.monsterhunterindex  import android.util.Log import androidx.lifecycle.*LiveData* import androidx.lifecycle.MutableLiveData import androidx.lifecycle.*ViewModel* import androidx.lifecycle.viewModelScope import kotlinx.coroutines.launch import retrofit2.Retrofit import retrofit2.converter.gson.GsonConverterFactory  class MonsterViewModel : ViewModel() {  private val serverUrl = "https://port-0-s23-monsterhunterbe-5yc2g32mlomgpvzu.sel5.cloudtype.app/"  private val monsterApi: MonsterApi  private val \_monsterList = MutableLiveData<List<Monster>>()  val monsterList: *LiveData*<List<Monster>>  get() = \_monsterList   init {  val retrofit = Retrofit.Builder().baseUrl(serverUrl)  .addConverterFactory(GsonConverterFactory.create()).build()   monsterApi = retrofit.create(MonsterApi::class.java)  fetchData()  }   private fun fetchData() {  viewModelScope.launch {  try {  val response = monsterApi.getMonsters()  \_monsterList.value = response  } catch (e: *Exception*) {  Log.e("fetchData()", e.toString())  }  }  } } |

Class생성시 MonsterApi에 있는 함수인 getMonsters()에서 값을 가져와서 값을 초기화 합니다.

|  |
| --- |
| MonsterApi.kt |
| package kr.ac.kumoh.ce.s20190558.monsterhunterindex  import retrofit2.http.GET  interface MonsterApi {  @GET("monster")  suspend fun getMonsters(): List<Monster> } |

데이터베이스에서 get을 통해 /monster에 있는 Json형식의 데이터를 Monster형식으로 바꾸어 가져옵니다.

|  |
| --- |
| Monster.kt |
| package kr.ac.kumoh.ce.s20190558.monsterhunterindex package kr.ac.kumoh.ce.s20190558.monsterhunterindex  import com.google.gson.annotations.SerializedName  data class Monster(  val id: Int,  val name: String,  val nickname: String,  val weakness: String,  val species: String,  val maps: String,  @SerializedName("icon\_address") val iconAddress: String,  @SerializedName("image\_address") val imageAddress: String ) |

Json형식 데이터를 다음과 같이 가져온다.

|  |
| --- |
| Server.js |
| import express from "express"  import mysql from "mysql"  import dbconf from "./conf/auth.js"  const app = express()  const port = 3010  const db = mysql.createConnection(dbconf)  db.connect()  app.get('/', (req, res) => {    res.send('스마트앱 프로그래밍 과제 백엔드')  })  app.get('/monster', (req, res) => {    const sql = `      SELECT        monster.id,        monster.name,        monster.nickname,        monster.weakness,        species.species,        GROUP\_CONCAT(habitat.map) AS maps,        monster.icon\_address,        monster.image\_address      FROM monster      LEFT JOIN monsterdex ON monster.id = monsterdex.monsterid      LEFT JOIN habitat ON monsterdex.habitatid = habitat.id      LEFT JOIN species ON monsterdex.speciesid = species.id      GROUP BY        monster.id,        monster.name,        monster.nickname,        monster.weakness,        species.species,        monster.icon\_address,        monster.image\_address;    `;    db.query(sql, (err, rows) => {      if (err) {        res.json({result: "error"})        return console.log(err)      }      res.json(rows)    })  })  app.listen(port, () => {    console.log(`서버 실행됨 (port ${port})`)  }) |

다음과 같은 형식으로 서버에서 데이터를 보낸다.

|  |
| --- |
| MonsterScreen.js |
| package kr.ac.kumoh.ce.s20190558.monsterhunterindex  import androidx.compose.foundation.background import androidx.compose.foundation.clickable import androidx.compose.foundation.layout.Arrangement import androidx.compose.foundation.layout.Column import androidx.compose.foundation.layout.IntrinsicSize import androidx.compose.foundation.layout.PaddingValues import androidx.compose.foundation.layout.Row import androidx.compose.foundation.layout.Spacer import androidx.compose.foundation.layout.fillMaxSize import androidx.compose.foundation.layout.fillMaxWidth import androidx.compose.foundation.layout.height import androidx.compose.foundation.layout.padding import androidx.compose.foundation.layout.size import androidx.compose.foundation.layout.width import androidx.compose.foundation.lazy.LazyColumn import androidx.compose.foundation.rememberScrollState import androidx.compose.foundation.verticalScroll import androidx.compose.material3.Card import androidx.compose.material3.CardDefaults import androidx.compose.material3.Text import androidx.compose.runtime.Composable import androidx.compose.ui.Alignment import androidx.compose.ui.Modifier import androidx.compose.ui.draw.clip import androidx.compose.ui.graphics.Color import androidx.compose.ui.graphics.RectangleShape import androidx.compose.ui.layout.ContentScale import androidx.compose.ui.text.style.TextAlign import androidx.compose.ui.unit.dp import androidx.compose.ui.unit.sp import androidx.navigation.NavController import androidx.navigation.*NavType* import coil.compose.AsyncImage import androidx.navigation.compose.NavHost import androidx.navigation.compose.composable import androidx.navigation.compose.rememberNavController import androidx.navigation.navArgument  enum class MonsterScreen {  List,  Detail }  @Composable fun MonsterApp(monsterList: List<Monster>) {  val navController = rememberNavController()   NavHost(  navController = navController,  startDestination = MonsterScreen.List.name,  ) {  composable(route = MonsterScreen.List.name) {  MonsterList(navController, monsterList)  }  composable(  route = MonsterScreen.Detail.name + "/{index}",  arguments = listOf(navArgument("index") {  type = NavType.IntType  })  ) {  val index = it.arguments?.getInt("index") ?: -1  if (index >= 0)  MonsterDetail(monsterList[index])  }  } }  @Composable fun MonsterList(navController: NavController, list: List<Monster>) {  LazyColumn(  verticalArrangement = Arrangement.spacedBy(16.dp),  contentPadding = PaddingValues(horizontal = 8.dp)  ) {  items(list.size) {  MonsterItem(navController, list, it)  }  } } @Composable fun MonsterItem(navController: NavController, monsterList: List<Monster>, index: Int) {   Card(  modifier = Modifier.clickable{  navController.navigate(MonsterScreen.Detail.name + "/$index")  },  elevation = CardDefaults.cardElevation(8.dp)  ) {  Row(  modifier = Modifier  .fillMaxWidth()  .height(IntrinsicSize.Min)  .padding(8.dp)  ) {  AsyncImage(  model = monsterList[index].iconAddress,  contentDescription = "${monsterList[index].name} 아이콘",  contentScale = ContentScale.Crop,  modifier = Modifier  .size(100.dp)  .clip(RectangleShape),  )  Spacer(modifier = Modifier.width(10.dp))  Column(  modifier = Modifier.fillMaxSize(),  verticalArrangement = Arrangement.SpaceAround  ) {  TextName(monsterList[index].name)  TextNickname(monsterList[index].nickname)  }  }  } } @Composable fun TextName(name: String){  Text(name, fontSize = 30.sp, lineHeight = 35.sp) } @Composable fun TextNickname(nickname: String) {  Text(nickname, fontSize = 20.sp) } @Composable fun MonsterDetail(monster: Monster) {  Column(  modifier = Modifier  .background(Color(252, 235, 255, 255))  .padding(16.dp)  .verticalScroll(rememberScrollState()),  horizontalAlignment = Alignment.CenterHorizontally  ) {  Text(  monster.name,  fontSize = 40.sp,  textAlign = TextAlign.Center,  lineHeight = 45.sp  )  Text(  monster.nickname,  fontSize = 20.sp,  textAlign = TextAlign.Center,  lineHeight = 25.sp  )  Spacer(modifier = Modifier.height(16.dp))   AsyncImage(  model = monster.imageAddress,  contentDescription = "${monster.name} 이미지",  contentScale = ContentScale.Fit,  modifier = Modifier.size(400.dp),  )  Spacer(modifier = Modifier.height(16.dp))   Text(  monster.species,  fontSize = 40.sp,  textAlign = TextAlign.Center,  lineHeight = 45.sp  )  Spacer(modifier = Modifier.height(16.dp))   Text("약점: ${monster.weakness}", fontSize = 30.sp)  Spacer(modifier = Modifier.height(16.dp))   Text(  text = "서식지: ${monster.maps}",  fontSize = 30.sp,  textAlign = TextAlign.Center,  lineHeight = 35.sp,  )  } } |

앱 실행 시 몬스터리스트 데이터를 받아 MonsterApp라는 함수가 실행이 된다. 이 함수에서 startDestination값을 List로 하여 처음 실행시 List화면이 출력되게 합니다. Route가 List일시에 MonsterList함수가 호출이 되며, 가져온 데이터의 개수만큼 카드가 생성되며 MonsterItem함수를 통해 클릭 이벤트, 데이터 가공이 진행이 된다. 그리고 클릭이벤트 실행시 route를 Detail + “/현재 카드의 id”로 설정되어, MonsterApp함수 내에 65번코드를 통해 MonsterDetail 함수가 실행이 됩니다. MonsterDetail 함수는 현재 몬스터의 id에 따른 데이터를 가져와 가공하여 Detail 화면을 채웁니다.

|  |
| --- |
| monsterdb.sql |
| DROP DATABASE IF EXISTS monsterdb;  CREATE DATABASE IF NOT EXISTS monsterdb    DEFAULT CHARACTER SET utf8mb4    DEFAULT COLLATE utf8mb4\_general\_ci;  USE monsterdb;  CREATE TABLE monster (    id INT NOT NULL AUTO\_INCREMENT,    name VARCHAR(255) NOT NULL,    nickname VARCHAR(100),    weakness VARCHAR(255),    icon\_address TEXT,    image\_address TEXT,    PRIMARY KEY(id)  ) ENGINE = InnoDB    DEFAULT CHARACTER SET utf8mb4    DEFAULT COLLATE utf8mb4\_general\_ci;  CREATE TABLE habitat (    id INT NOT NULL AUTO\_INCREMENT,    map VARCHAR(255) NOT NULL,    PRIMARY KEY(id)  ) ENGINE = InnoDB    DEFAULT CHARACTER SET utf8mb4    DEFAULT COLLATE utf8mb4\_general\_ci;  CREATE TABLE species (    id INT NOT NULL AUTO\_INCREMENT,    species VARCHAR(100),    PRIMARY KEY(id)  ) ENGINE = InnoDB    DEFAULT CHARACTER SET utf8mb4    DEFAULT COLLATE utf8mb4\_general\_ci;  CREATE TABLE monsterdex (    monsterid INT NOT NULL,    speciesid INT NOT NULL,    habitatid INT NOT NULL,    PRIMARY KEY(monsterid, speciesid, habitatid),    FOREIGN KEY(monsterid) REFERENCES monster(id)      ON DELETE CASCADE      ON UPDATE CASCADE,    FOREIGN KEY(speciesid) REFERENCES species(id)      ON DELETE CASCADE      ON UPDATE CASCADE,    FOREIGN KEY(habitatid) REFERENCES habitat(id)      ON DELETE CASCADE      ON UPDATE CASCADE,    INDEX(monsterid)  ) ENGINE = InnoDB    DEFAULT CHARACTER SET utf8mb4  DEFAULT COLLATE utf8mb4\_general\_ci;  SELECT monster.id, monster.name, monster.nickname, monster.weakness, species.species, GROUP\_CONCAT(habitat.map) AS maps, monster.icon\_address, monster.image\_address      FROM monster      LEFT JOIN monsterdex ON monster.id = monsterdex.monsterid      LEFT JOIN habitat ON monsterdex.habitatid = habitat.id      LEFT JOIN species ON monsterdex.speciesid = species.id      GROUP BY        monster.id,        monster.name,        monster.nickname,        monster.weakness,        species.species,        monster.icon\_address,        monster.image\_address; |

**5. 결 론**

**5.1 개발 내용**

이 프로젝트에서는 Monster Hunter World와 ICEBORNE의 대형 몬스터로 Montser Hunter 도감 앱을 개발하였습니다. 사용자들은 몬스터에 대한 정보를 쉽게 찾아볼 수 있으며, 몬스터 리스트와 상세 정보를 효과적으로 표시하는 기능을 갖추고 있습니다. Jetpack Compose 및 Coil라이브러리를 사용하여 모던하고 유연한 UI를 구현하였습니다.

**5.2 향후 개선 과제**

향후에는 다음과 같은 개선 과제를 고려할 수 있습니다.

* **검색 기능 추가**: 사용자가 특정 조건에 따라 몬스터를 검색하거나 필터링할 수 있는 기능을 추가하여 사용자의 편의성을 더욱 증가시킵니다.
* **더 다양한 정보 제공**: 몬스터에 대한 더 다양한 정보를 제공하여 사용자들이 게임 플레이에 도움을 받을 수 있도록 확장할 수 있습니다. 예) 맵에 따른 베이스캠프 위치

**5.3 기대 효과**

이 프로젝트를 통해 기대되는 효과는 다음과 같습니다.

* **게임 난이도 하향**: 유저가 몬스터의 정보를 미리 알고 대비를 하기에 게임 난이도가 줄어들 것으로 기대된다.
* **신규 게이머 증가**: 높은 난이도로 입문을 어려워하던 게이머들이 정보 공유를 통해 줄어든 난이도로 인해 유입이 될 것이라 예상된다.

**6. 자체 분석과 평가**

**6.1 수행 수준의 평가**

수행 수준을 자체적으로 평가하자면, 주어진 목표를 대체로 잘 달성한 것으로 생각됩니다. 몬스터 리스트와 세부정보를 표시하는 기능이 잘 구현되었고, Compose를 사용한 UI 부분은 개선할 부분은 여럿 보이지만 전반적으로 과제의 목적을 이루는 데 성공했다고 생각합니다.

**6.2 과제를 수행하면서 어려웠던 점과 문제 해결**

DB에서 데이터를 Join하는 부분과 데이터를 Monster데이터 클래스로 받아오는 부분이 어려웠습니다. Join하는 부분은 ChatGPT를 활용하여 해결하였고,   
데이터를 받아오는 부분은 경고 메시지 때문에 스네이크 케이스에서 카멜케이스로 수정한 icon\_address, image\_address이 두개가 Json데이터의 이름과 달라 값을 가져오지 못하는 거였습니다. 그래서 이부분도 ChatGPT를 사용하여 해결하였습니다.

**참고문헌**

[1] <https://nano5.notion.site/Week09-Database-4f95cf4133cf4375bdb447d0bf248291?pvs=4>

[2] <https://nano5.notion.site/Week10-Backend-300c043e6349429592e3a1d5a790a88a?pvs=4>

[3] <https://nano5.notion.site/Week13-01-Song-List-53472b09f76f46cd9d3805ec52e9b6bf?pvs=4>

[4] <https://nano5.notion.site/Week13-02-AsyncImage-556f54b34dbd43df8105331c518eaa43?pvs=4>

[5] <https://nano5.notion.site/Week14-02-Song-Detail-Screen-c12d837638a14b3094e3b82be24d01a0?pvs=4>