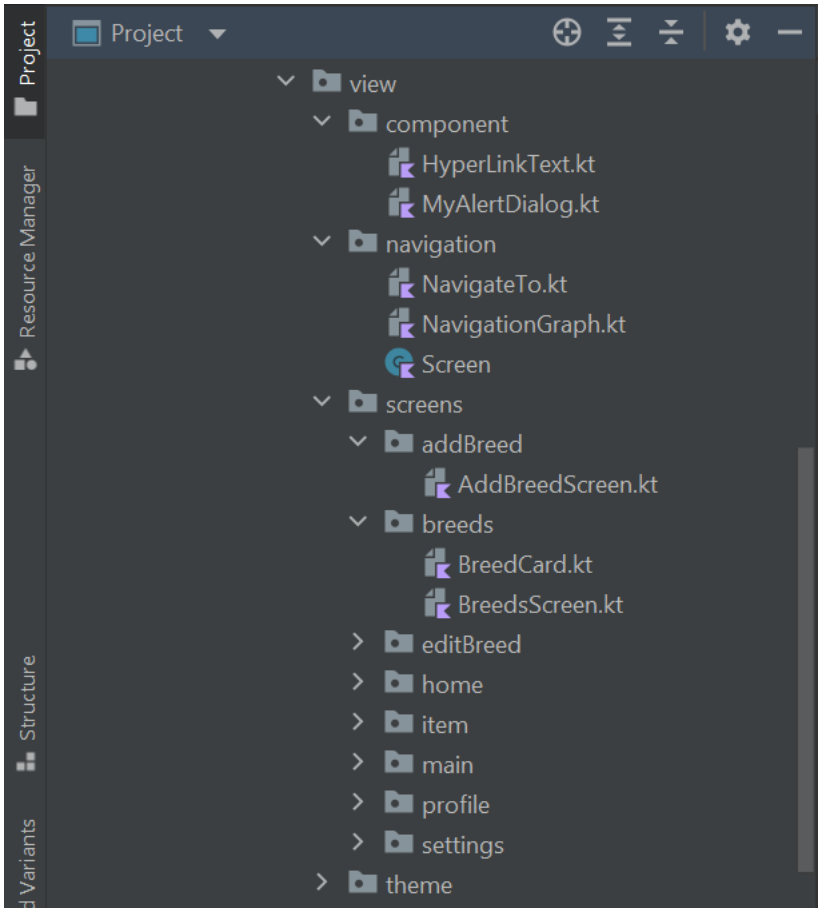
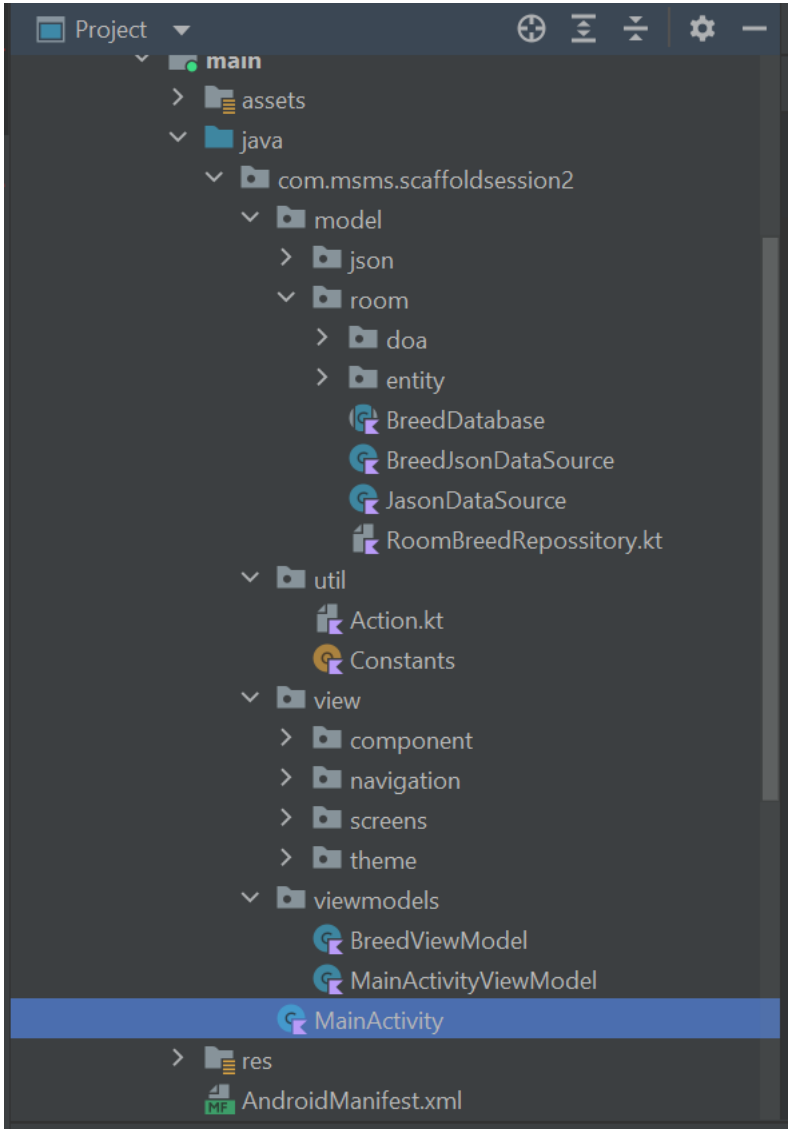
## Possible Project Structure following MVVM



## Breed class

```
@Entity(tableName = "breed_table")
data class Breed(
    @PrimaryKey
    var id:String="Missing",
    var metricWeight:String = "Missing",
//     var weight: Weight = Weight("",""),
    var description: String="Missing",
    var name: String="Missing",
    var origin:String="Missing",
    var wikipedia_url:String="Missing",
    var urlImage:String="Missing",
//     var image: Image = Image("",-1,-1,""),
){
    constructor(
        id: String
    ):this(
        id,
        metricWeight = "2 - 4",
        description = "No description",
        name = "No name",
        origin = "no origin",
        wikipedia_url = "https://en.wikipedia.org/wiki/American_Curl",
        urlImage = "https://cdn2.thecatapi.com/images/xnsqonbjW.jpg",
        )
}
```

## BreedDoa

```
@Dao
interface BreedDao {
    @Query("SELECT COUNT(id) FROM breed_table")
    fun getBreedCount():Int
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun addBreeds(breeds: List<Breed>)
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun addBreed(breed: Breed)
    @Query("SELECT * FROM breed_table ORDER BY id ASC")
    fun getAllBreeds(): Flow<List<Breed>>
    @Query("SELECT * FROM breed_table WHERE id = :breedId")
    fun getBreed(breedId: String) : Flow<Breed>
    @Delete
    fun deleteBreed(breed: Breed)
    @Update
    fun updateBreed(breed: Breed)
    @Query(value = "DELETE FROM breed_table")
    fun deleteAllBreeds()
    //…others
}
```

## Database

```
@Database(entities = [Breed::class], version = 4, exportSchema = false)
abstract class BreedDatabase : RoomDatabase() {
    abstract fun breedDao(): BreedDao

    companion object {
        private var INSTANCE: BreedDatabase? = null

        fun getInstance(context: Context): BreedDatabase{
            synchronized(this) {
                var instance = INSTANCE
                if (instance == null) {
                    instance = Room.databaseBuilder(
                        context.applicationContext,
                        BreedDatabase::class.java,
                        "breed_database"
                    ).fallbackToDestructiveMigration().build()
                    INSTANCE = instance
                }
                return instance
            }
        }
    }
}
```

## RoomRepository

```
class RoomBreedRepository( context: Context){
    private val roomDb by lazy {
        BreedDatabase.getInstance(context)

    }
    private val breedDao by lazy {
        roomDb.breedDao()
    }
    fun breedCount(): Int = breedDao.getBreedCount()
    suspend fun getAllBreeds(): Flow<List<Breed>> = breedDao.getAllBreeds()
    suspend fun getBreed(breedId: String): Flow<Breed> = breedDao.getBreed(breedId)
    suspend fun addBreed(breed: Breed) = breedDao.addBreed(breed)
    suspend fun deleteBreed(breed: Breed) = breedDao.deleteBreed(breed)
    suspend fun updateBreed(breed: Breed) = breedDao.updateBreed(breed)
    suspend fun deleteAllBreeds() = breedDao.deleteAllBreeds()
    //…others
    init {
        runBlocking {
            this.launch(Dispatchers.IO) {
                initDbFromJson(context)
            }
        }
    }
```

```
suspend fun initDbFromJson (context: Context): Unit{
        if (roomDb==null){ return }
        if (breedDao==null){ return }
        var breeds = listOf<com.msms.scaffoldsession2.model.json.Breed>()
        if (breedDao.getBreedCount() == 0) {
            val json = Json {
                ignoreUnknownKeys = true
                coerceInputValues = true
            }
            val breedJson = context.assets
                .open("breeds.json")
                .bufferedReader()
                .use {
                    it.readText()
                }
            breeds = json.decodeFromString(breedJson)
            val b = toModelBreeds(breeds)
            b.forEach {
                runBlocking {
                    this.launch {
                        breedDao.addBreed(it)
                    }
                }
            }
        }
    }
    private fun toModelBreeds(
        breeds: List<com.msms.scaffoldsession2.model.json.Breed>):List<Breed>{
        var modelBreeds = mutableListOf<Breed>()
        breeds.forEach {
            modelBreeds.add(
                Breed(
                it.id,it.weight.metric,it.description,it.name,it.origin,it.wikipedia_url,it.image.url
                )
            )
        }
        return modelBreeds.toList()
    }
}
```

## BreedViewModel

```
class BreedViewModel(val context: Application) : AndroidViewModel(context){
    private val roomBreedRepository = RoomBreedRepository(context)
    private val _breed = MutableStateFlow(Breed("","","","","","",""))
    val breed: StateFlow<Breed> = _breed.asStateFlow()
    fun setCurrentBreed(breed: Breed){
        _breed.value = breed
    }
    fun updateName(name: String){
        _breed.value = _breed.value.copy(name = name)
    }
    fun updateOrigin(origin: String){
        _breed.value = _breed.value.copy(origin = origin)
    }
    //…others
    private var _breeds = MutableStateFlow(emptyList<Breed>())
    var breeds: StateFlow<List<Breed>> = _breeds.asStateFlow()
    init {
        getAllBreeds()
    }
    fun getAllBreeds(){
        viewModelScope.launch{
            roomBreedRepository.getAllBreeds().collect{
                _breeds.value = it
            }
        }
    }
    fun addBreed(breed: Breed) = viewModelScope.launch(Dispatchers.IO) {
        roomBreedRepository.addBreed(breed)
    }
    fun updateBread(breed: Breed){
        viewModelScope.launch(Dispatchers.IO){
            roomBreedRepository.updateBreed(breed)
        }
    }
    //…others
//… other stuff too
}
```

## MainActivity

```
class MainActivity : ComponentActivity() {
    private lateinit var navController: NavHostController
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            navController = rememberNavController()
            val context = LocalContext.current
            val breedViewModel = viewModel<BreedViewModel>(
                                        viewModelStoreOwner = context as ComponentActivity
                                    )
//            val BreedViewModel1 = ViewModelProvider(this)[BreedViewModel::class.java]
//            val breedViewModel2 = ViewModelProvider.AndroidViewModelFactory(application).
//                                                        create(BreedViewModel::class.java)
//            val breedViewModel3 = BreedViewModel(application)
            ScaffoldSession1Theme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    MainScreen(navController)
                }
            }
        }
    }
}
```

## MainScreen

```
@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun MainScreen(
       navController: NavHostController,
       breedViewModel: BreedViewModel = viewModel(LocalContext.current as ComponentActivity)
){
    Scaffold (
        bottomBar = {
            BottomBar(navController = navController)
        }
    ){
        NavigationGraph(navController = navController)
    }
}

@Composable
fun BottomBar(navController: NavHostController){
//
}

@Composable
fun RowScope.AddItem(
    screen: Screen,
    currentDestination: NavDestination,
    navController: NavHostController
){
    //
}
```

## Navigation Graph

```
@Composable
fun NavigationGraph(navController: NavHostController){
    NavHost(
        navController = navController,
        startDestination = Screen.Home.route
    ) {
        composable(route = Screen.Home.route) {
            HomeScreen(navController)
        }
        composable(route = Screen.Breeds.route) {
            BreedsScreen(navController = navController)
        }
        composable(route = Screen.AddBreed.route) {
            AddBreedScreen(navController = navController)
        }
        composable(route = Screen.EditBreed.route) {
            EditBreedScreen(navController = navController)
        }
        //…
    }
}
```

## BreedsScreen

```
@Composable
fun BreedsScreen(

       navController: NavController,

       breedViewModel: BreedViewModel= viewModel(LocalContext.current as ComponentActivity)

) {

    val breeds: List<Breed> by breedViewModel.breeds.collectAsState()
    val showDialogState: Boolean by breedViewModel.showDialog.collectAsState()

    BreedsScreenContents(navController,breeds)
}
```

## BreedScreenContents

```
@Composable
fun BreedsScreenContents(
    navController: NavController,
    breeds: List<Breed>,
    breedViewModel: BreedViewModel = viewModel(LocalContext.current as ComponentActivity)
) {
    Column() {
        Row() {
            IconButton(onClick = {
                navController.navigate(route = Screen.AddBreed.route)
            }) {
                Icon(
                    imageVector = Icons.Default.Add,
                    contentDescription = "Add Icon"
                )
            }
            IconButton(onClick = {
                breedViewModel.deleteAllBreeds()
            }) {
                Icon(
                    imageVector = Icons.Default.Delete, contentDescription = "Delete All Icon"
                )
            }
            Text(
                text = "Delete All", modifier = Modifier.padding(top = 10.dp)
            )
        }
        if (breeds.isEmpty()) {
            Text("Breeds list is empty")
        } else {
            LazyColumn {
                items(breeds) {
                    BreedCard(navController, it)
                }
            }
            Spacer(modifier = Modifier.size(90.dp))
        }
    }
}
```

## BreedCard

```
@Composable
fun BreedCard(
       navController: NavController,
       breed: Breed,
       breedViewModel: BreedViewModel = viewModel(LocalContext.current as ComponentActivity)
) {
    var isExpanded by remember { mutableStateOf(false) }
    val surfaceColor by animateColorAsState(
                           targetValue = if (isExpanded) MaterialTheme.colors.primary
                                         else MaterialTheme.colors.surface)
    val id = breed.id
    Log.d("BreedCard", "in")
    Card(
        backgroundColor = Color.LightGray,
        modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth(),
        shape = RoundedCornerShape(CornerSize(10.dp)),
        elevation = 18.dp
    ) {
        Column() {
            Row(
                modifier = Modifier
                    .padding(1.dp)
                    .fillMaxWidth()
                    .background(Color.DarkGray, RectangleShape)
            ) {
                Image(
                    painter = rememberAsyncImagePainter(breed.urlImage),
                    contentDescription = "Cat Image",
                    modifier = Modifier
                        .size(80.dp)
                        .clip(CircleShape)
                        .border(1.5.dp, MaterialTheme.colors.secondaryVariant, CircleShape)
                )
                Spacer(modifier = Modifier.padding(20.dp))
                IconButton(
                    onClick = {
                        breedViewModel.deleteBread(breed)
```

```
                    }) {
                    Icon(
                        imageVector = Icons.Default.Delete,
                        contentDescription = "Delete Breed"
                    )
                }
                Spacer(modifier = Modifier.padding(20.dp))
                IconButton(
                    onClick = {
                        breedViewModel.setCurrentBreed(breed)
                        navController.navigate(route = Screen.EditBreed.route)
                    }) {
                    Icon(
                        imageVector = Icons.Default.Edit,
                        contentDescription = "Edit Breed"
                    )
                }
            }
            Surface(shape = MaterialTheme.shapes.medium,
                color = MaterialTheme.colors.surface,
                modifier = Modifier
                    .animateContentSize()
                    .fillMaxWidth()
                    .padding(1.dp)
                    .clickable { navController.navigate(route = "…") }
            ) {
                Column {
                    Text(
                        text = "Id: ${breed.id}",
                        style = MaterialTheme.typography.subtitle2
                    )
                    Text(
                        text = "Name: ${breed.name}",
                        style = MaterialTheme.typography.subtitle2
                    )
                    //…                 }
}
```

## EditBreed

```
@Composable
fun EditBreedScreen(navController: NavController, breedViewModel: BreedViewModel = viewModel(
    LocalContext.current as ComponentActivity)){

    val currentBreed: Breed by breedViewModel.breed.collectAsState()
    var id = currentBreed.id
    var name = currentBreed.name
    var origin = currentBreed.origin
    var description = currentBreed.description
    var image = currentBreed.urlImage
    var wikipedia = currentBreed.wikipedia_url
    var metricWeight = currentBreed.metricWeight

    Column ( modifier = Modifier.fillMaxSize().verticalScroll(rememberScrollState())
    ){
        Text(
            text = "Edit Breed Screen", color = MaterialTheme.colors.primary,
            fontSize = MaterialTheme.typography.h3.fontSize, fontWeight = FontWeight.Bold,
        )
        OutlinedTextField(
            value = id,
            onValueChange = {
                breedViewModel.updateId(it)
            },
            placeholder = { Text(text = "ID")}, modifier = Modifier.fillMaxWidth()
        )
        //…
        Button(
            onClick = {
                breedViewModel.updateBread(currentBreed)
                navController.navigate(Screen.Breeds.route)
            }
        ) {
            Text(text = "Update Breed")
        }
        Spacer(modifier = Modifier.size(70.dp))
    }
}
```

## AddBreed

```
@Composable
fun AddBreedScreen(
       navController: NavController,
       breedViewModel: BreedViewModel = viewModel(LocalContext.current as ComponentActivity)
){
    var id by remember { mutableStateOf("") }
    var name by remember{ mutableStateOf("") }
    var origin by remember { mutableStateOf("") }
    var description by remember { mutableStateOf("") }
    var image by remember { mutableStateOf("") }
    var wikipedia by remember { mutableStateOf("") }
    var metricWeight by remember { mutableStateOf("") }

    Column (
        modifier = Modifier.fillMaxSize(),
    ){
        Text(
            text = "Add Breed Screen", color = MaterialTheme.colors.primary,
            fontSize = MaterialTheme.typography.h3.fontSize,fontWeight = FontWeight.Bold,
        )
        OutlinedTextField(
            value = id,
            onValueChange = { id = it },
            placeholder = { Text(text = "ID") }
        )
        //…
        Button(
            onClick = {
                val breed = Breed(id)
                breedViewModel.addBreed(breed)
                navController.navigate(Screen.Breeds.route)
            }
        ) {
            Text(text = "Add Item")
            Icon(imageVector = Icons.Default.Add, contentDescription = "Add Breed")
        }
        Spacer(modifier = Modifier.size(70.dp))
    }
}
```

## Build.gradle

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'org.jetbrains.kotlin.plugin.serialization' version '1.7.10'
    id 'kotlin-kapt'
}

android {
    compileSdk 33

    defaultConfig {
        applicationId "com.msms.scaffoldsession2"
        minSdk 30
        targetSdk 33
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary true
        }
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        compose true
    }
    composeOptions {
        kotlinCompilerExtensionVersion compose_version
    }
    packagingOptions {
        resources {
            excludes += '/META-INF/{AL2.0,LGPL2.1}'
```

```
        }
    }
    namespace 'com.msms.scaffoldsession2'
}

dependencies {
    implementation 'androidx.core:core-ktx:1.9.0'
    implementation "androidx.compose.ui:ui:1.3.0-beta03"
    implementation "androidx.compose.material:material:1.3.0-beta03"
    implementation "androidx.compose.ui:ui-tooling-preview:1.3.0-beta03"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.5.1'
    implementation 'androidx.activity:activity-compose:1.6.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:1.3.0-beta03"
    debugImplementation "androidx.compose.ui:ui-tooling:1.3.0-beta03"
    debugImplementation "androidx.compose.ui:ui-test-manifest:1.3.0-beta03"
    implementation "org.jetbrains.kotlinx:kotlinx-serialization-json-jvm:1.4.0"
    implementation("io.coil-kt:coil-compose:2.2.0")
    //viewModel
    implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version")
    // ViewModel utilities for Compose
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:$lifecycle_version")
    // Lifecycles only (without ViewModel or LiveData)
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:$lifecycle_version")
    // Saved state module for ViewModel
    implementation("androidx.lifecycle:lifecycle-viewmodel-savedstate:$lifecycle_version")
    // if using Java8, use the following instead of lifecycle-compiler
    implementation("androidx.lifecycle:lifecycle-common-java8:$lifecycle_version")
    //navigation
    implementation('androidx.navigation:navigation-compose:2.5.2')
    ///Room
    implementation("androidx.room:room-runtime:$room_version")
    implementation ("androidx.room:room-ktx:$room_version")
    kapt "androidx.room:room-compiler:$room_version"
    // optional - Kotlin Extensions and Coroutines support for Room
    implementation("androidx.room:room-ktx:$room_version")
    // optional - Test helpers
    testImplementation("androidx.room:room-testing:$room_version")
}
```

## build.gadle at app

```
buildscript {
    ext {
        compose_version = '1.3.0'
        room_version = "2.4.3"
        lifecycle_version = '2.6.0-alpha02'
        arch_version = '2.1.0'
    }

        repositories {
            google()
            mavenCentral()
        }
    dependencies {

    }
}// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '7.3.0' apply false
    id 'com.android.library' version '7.3.0' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.10' apply false
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```