12

Persisting Data

Activity 12.01 – managing multiple persistence options

Solution

Perform the following steps to solve the activity:

1. Create a new Android Studio project with `Empty Activity`.
2. Add the `ksp` plugin dependency to `gradle/libs.versions.toml`:

    ```
    [versions]
    …
    ksp = "2.0.21-1.0.25"
    …
    [plugins]
    …
    ksp = { id = "com.google.devtools.ksp", version.ref = "ksp" }
    ```

3. In the root `build.gradle.kts` file, add the `ksp` plugin:

    ```
    plugins {
        …
        alias(libs.plugins.ksp) apply false
    }
    ```

4. In `app/build.gradle.kts`, add the `ksp` plugin:

    ```
    plugins {
        …
    ```

```
    alias(libs.plugins.ksp)

}
```

5. In gradle/libs.versions.toml, add the required dependencies, which include viewModel, room, retrofit, gson, and dataStore:

```
[versions]
…
viewModelCompose = "2.8.7"
room = "2.7.1"
retrofit="2.9.0"
retrofitGson="2.6.2"
gson="2.10.1"
dataStore="1.1.5"

[libraries]
…
androidx-viewmodel-compose = { group = "androidx.lifecycle",
    name = "lifecycle-viewmodel-compose", version.ref =
"viewModelCompose" }
androidx-room-runtime = { group = "androidx.room",
    name = "room-runtime", version.ref = "room" }
androidx-room-compiler = { group = "androidx.room",
    name = "room-compiler", version.ref = "room" }
androidx-room-ktx = { group = "androidx.room",
    name = "room-ktx", version.ref = "room" }
androidx-data-store-preferences = { group = "androidx.datastore",
    name = "datastore-preferences", version.ref = "dataStore" }
squareup-retrofit = { group = "com.squareup.retrofit2",
    name = "retrofit", version.ref = "retrofit" }
squareup-retrofit-gson = { group = "com.squareup.retrofit2",
    name = "converter-gson", version.ref = "retrofitGson" }
gson = { group = "com.google.code.gson", name = "gson",
    version.ref = "gson" }
…
[plugins]
```

6. Next, let's add the libraries in `app/build.gradle.kts`:

```
    implementation(libs.squareup.retrofit)
    implementation(libs.squareup.retrofit.gson)
    implementation(libs.gson)
    implementation(libs.androidx.viewmodel.compose)
    implementation(
        libs.androidx.data.store.preferences
    )
    implementation(libs.androidx.room.runtime)
    implementation(libs.androidx.room.ktx)
    ksp(libs.androidx.room.compiler)
```

7. Create a package called `api`.
8. Create a class called `Dog`, which will be a representation of the JSON data that the app will receive:

```
data class Dog(
    @SerializedName("status") val status: String,
    @SerializedName("message") val urls: List<String>
)
```

9. Now, create a `Retrofit` interface, which will define how we load the list of `Dog` objects and how the download will be executed through a dynamic URL:

```
interface DogService {

    @GET("breed/hound/images/random/{number}")
    suspend fun getDogs(
        @Path("number") number: Int
    ): Dog

    @GET
    suspend fun downloadFile(
        @Url fileUrl: String
    ): ResponseBody
}
```

The @Path annotation allows us to dynamically set certain parts of the path, and @Url allows us to place a dynamic URL in the download function. The method will return a ResponseBody object, which will contain methods to allow us to access the bytes (through InputStream) of the file.

10. Add the INTERNET permission to the AndroidManifest.xml file:

```
<uses-permission android:name="android.permission.INTERNET" />
```

11. Create the storage package and, inside this, create the room package.
12. Create a DogEntity class, which will contain an ID and the URL for the dog photo:

```
@Entity(tableName = "dogs")
data class DogEntity(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id") val id: Long,
    @ColumnInfo(name = "url") val url: String
)
```

13. Create a DogDao interface, which will contain the method to insert a list of Dog objects, query the existing dogs, and delete all the dogs in the table:

```
@Dao
interface DogDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertDogs(dogs: List<DogEntity>)

    @Query("SELECT * FROM dogs")
    fun loadDogs(): Flow<List<DogEntity>>

    @Query("DELETE FROM dogs")
    suspend fun deleteAll()
}
```

14. Create the DogDatabase class, which will connect the entity and the data access object:

```
@Database(
    entities = [DogEntity::class],
    version = 1
)
```

```
abstract class DogDatabase : RoomDatabase() {

    abstract fun dogDao(): DogDao
}
```

15. Create the `xml` resource directory inside the `res` folder.
16. Create a `provider_paths.xml` file inside the `xml` directory that will point to the external media folder. In this example, we will save the files directly in the root folder:

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <external-cache-path
        name="my-media"
        path="." />
</paths>
```

17. Inside the `storage` package, create the `filesystem` package.
18. Create the `FileToUriMapper` class, which will convert a file into a URI to allow us to test the other classes better:

```
class FileToUriMapper {

    fun getUriForFile(
        context: Context, file: File
    ): Uri {
        return FileProvider.getUriForFile(
            context,
            "com.packt.android.dogs",
            file
        )
    }
}
```

19. Create a `ProviderFileHandler` class, which will be responsible for writing inside a file that will belong to `FileProvider`:

```
class ProviderFileHandler(
    private val context: Context,
    private val fileToUriMapper: FileToUriMapper
) {
    suspend fun writeStream(
        name: String, inputStream: InputStream
    ) {
        withContext(Dispatchers.IO) {
            val fileToSave =
                File(context.externalCacheDir, name)
            val outputStream = context
                .contentResolver.openOutputStream(
                    fileToUriMapper.getUriForFile(
                        context, fileToSave
                    ),
                "rw"
            )
            outputStream?.let {
                inputStream.copyTo(outputStream)
            }
        }
    }
}
```

20. Make sure you have the provider in the `AndroidManifest.xml` file:

```
        <provider
            android:name=
                "androidx.core.content.FileProvider"
            android:authorities=
                "com.packt.android.dogs"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name=
                    "android.support
                    .FILE_PROVIDER_PATHS"
                android:resource=
                    "@xml/provider_paths" />
        </provider>
```

Make sure to set the same authority here as you did in `FileToUriMapper`.

21. Create a package named `preference` in the `storage` package.
22. In the `preference` package, create `DownloadPreferences`, which will be responsible for retrieving the number of results we want to display on the screen. In case there is no value saved, we will default to `10` results:

```
val Context.dataStore: DataStore<Preferences> by
preferencesDataStore(name = "download_preferences")

private val KEY_NO_RESULTS =
    intPreferencesKey("key_no_results")

class DownloadPreferences(
    private val context: Context
) {

    val resultNumberFlow: Flow<Int> =
    context.dataStore.data
        .map { preferences ->
            preferences[KEY_NO_RESULTS] ?: 10
        }

    suspend fun saveResultNumber(resultNo: Int) {
        context.dataStore.edit { preferences ->
            preferences[KEY_NO_RESULTS] = resultNo
        }
    }
}
```

23. Now, move on to the `Repository` aspect and create a new package called `repository`.
24. Create a class named `Result`, which will have three outputs: `Loading`, `Success`, and `Error`. We can achieve this through the Kotlin `sealed class` feature:

```
sealed class Result<T> {

    class Loading<T> : Result<T>()
    data class Success<T>(val data: T) : Result<T>()
    class Error<T> : Result<T>()
}
```

25. Define a model that will be used by our user interface layer:

```
data class DogUi(val url: String)
```

26. Define a `Mapper` class that will convert one type of model into another:

```
class DogMapper {

    fun mapServiceToEntity(dog: Dog):
        List<DogEntity> = dog.urls.map
    {
        DogEntity(0, it)
    }

    fun mapEntityToUi(dogEntity: DogEntity):
        DogUi = DogUi(dogEntity.url)
}
```

27. Next, define our `Repository` interface and name it `DogRepository`:

```
interface DogRepository {

    fun loadDogList(): Flow<Result<List<DogUi>>>

    suspend fun updateNumberOfResults(
        numberOfResults:Int
    ): Result<Unit>

    suspend fun downloadFile(url: String):
        Result<Unit>
}
```

28. Provide the implementation for the repository. The implementation for retrieving the list of URLs will set the `Loading` state first, and then it will monitor any changes in the database and start the request. When the request finishes, it will insert the data in the database, which should then provide a notification regarding the changes to the data model:

```
class DogRepositoryImpl(
    private val downloadPreferences:
        DownloadPreferences,
    private val providerFileHandler:
        ProviderFileHandler,
    private val dogService: DogService,
    private val dogDao: DogDao,
    private val dogMapper: DogMapper
) : DogRepository {

    @OptIn(ExperimentalCoroutinesApi::class)
    override fun loadDogList():
        Flow<Result<List<DogUi>>>
    {
        return downloadPreferences
            .resultNumberFlow.flatMapLatest
        {
            flowOf(dogService.getDogs(it))
        }.map {
            dogMapper.mapServiceToEntity(it)
        }.onEach {
            dogDao.insertDogs(it)
        }.map {
            Result.Success(
                it.map { dogMapper.mapEntityToUi(it) }
            ) as Result<List<DogUi>>
        }.catch {
            emit(Result.Error())
        }.onStart {
            emit(Result.Loading())
        }
    }

}
```

In the XE "multiple persistence options:managing" same file, add a function that will download a file and save it using ProviderFileHandler:

```
class DogRepositoryImpl(
    private val downloadPreferences:
        DownloadPreferences,
    private val providerFileHandler:
        ProviderFileHandler,
    private val dogService: DogService,
    private val dogDao: DogDao,
    private val dogMapper: DogMapper
) : DogRepository {
...
    override suspend fun downloadFile(url: String):
        Result<Unit>
    {
        try {
            val body = dogService.downloadFile(url)
            val name =
                url.substring(
                    url.lastIndexOf("/") + 1
                )
            providerFileHandler.writeStream(
                name, body.byteStream()
            )
            return Result.Success(Unit)
        }catch (e: Exception){
            return Result.Error()
        }
    }

}
```

29. In the same file, add a function that will update the number of results using `DownloadPreferences`:

```
class DogRepositoryImpl(
    private val downloadPreferences:
        DownloadPreferences,
    private val providerFileHandler:
        ProviderFileHandler,
    private val dogService: DogService,
    private val dogDao: DogDao,
    private val dogMapper: DogMapper
) : DogRepository {
```

```
...
    override suspend fun updateNumberOfResults(
        numberOfResults: Int
    ): Result<Unit> {
        downloadPreferences.saveResultNumber(
            numberOfResults
        )
        return Result.Success(Unit)
    }

}
```

The implementation for downloading a file will set the `Loading` state when the download is started. Then, if the connection to the server isn't established, the server replies with an error, or the download cannot be performed, it will set the state to `Error`. If the download is completed successfully, it will show a success message.

30. Move on to `DogViewModel`, which will have a reference to `DogRepository` and will invoke its functions to show the list and download each dog:

```
class DogViewModel(
    private val dogRepository: DogRepository
) : ViewModel() {

    private val _state = MutableStateFlow<
        Result<List<DogUi>>
    >(
        Result.Loading()
    )
    val state: StateFlow<Result<List<DogUi>>> = _state

    init {
        viewModelScope.launch {
            loadDogList()
        }
    }

    fun saveNumberOfResult(numberOfResults: Int) {
        viewModelScope.launch {
            dogRepository.updateNumberOfResults(
                numberOfResults
            )
            loadDogList()
        }
    }

    fun downloadDog(url: String) {
        viewModelScope.launch {
            dogRepository.downloadFile(url)
        }
    }

    private suspend fun loadDogList() {
        dogRepository.loadDogList()
            .collect {
                _state.emit(it)
            }
    }
}
```

31. Make sure that the following string is present in `res/values/strings.xml`:

```
<string name="click_me">Click Me</string>
```

32. In `MainActivity`, create `@Composable` functions that will represent the `Loading` state and `Error` state:

```
@Composable
fun Loading(modifier: Modifier) {
    CircularProgressIndicator(
        modifier = modifier.width(64.dp)
    )
}

@Composable
fun Error(modifier: Modifier) {
    Text(
        text = "Something went wrong",
        modifier = modifier
    )
}
```

33. In `MainActivity`, create a `@Composable` function that will show a `TextField` element and a `Button` element to indicate how many results we want to display, and below these elements, the list of URLs for each dog:

```
@Composable
fun DogScreen(
    dogs: List<DogUi>,
    onNumberOfResultsClicked: (Int) -> Unit,
    onRowClicked: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    LazyColumn(modifier = modifier) {
        item {
            var textFieldText by remember {
                mutableStateOf(dogs.size.toString())
            }
            TextField(value = textFieldText,
                keyboardOptions = KeyboardOptions(
                    keyboardType =
                        KeyboardType.Decimal
                ),
                onValueChange = {
                    textFieldText = it
                })
            Button(onClick = {
                onNumberOfResultsClicked(
                    textFieldText.toInt()
                )
            }) {
                Text(
                    text = stringResource(
                        id = R.string.click_me
                    )
                )
            }
        }
        items(dogs.size) {
            val index = it
            Column(modifier = Modifier
                .padding(16.dp)
                .clickable(onClick = {
                    onRowClicked(dogs[index].url)
                })) {
                Text(text = dogs[it].url)
            }
        }
    }
}
```

34. In `MainActivity`, create a `@Composable` function, `Dog`, that will connect `DogViewModel` with `DogScreen`, `Error`, or `Loading`, depending on the state:

```
@Composable
fun Dog(
    dogViewModel: DogViewModel,
    modifier: Modifier
) {

    when (
        val result = dogViewModel
            .state.collectAsState().value
    ) {
        is Result.Success -> {
            DogScreen(
                dogs = result.data,
                onNumberOfResultsClicked = {
                    dogViewModel
                    .saveNumberOfResult(it)
                },
                onRowClicked = {
                    dogViewModel.downloadDog(it)
                },
                modifier = modifier
            )
        }

        is Result.Loading -> {
            Loading(modifier = modifier)
        }

        is Result.Error -> {
            Error(modifier = modifier)
        }
    }
}
```

35. In `MainActivity`, modify the `onCreate` function to initialize `DogRepository`, `DogDatabase`, and the rest of the required storage components:

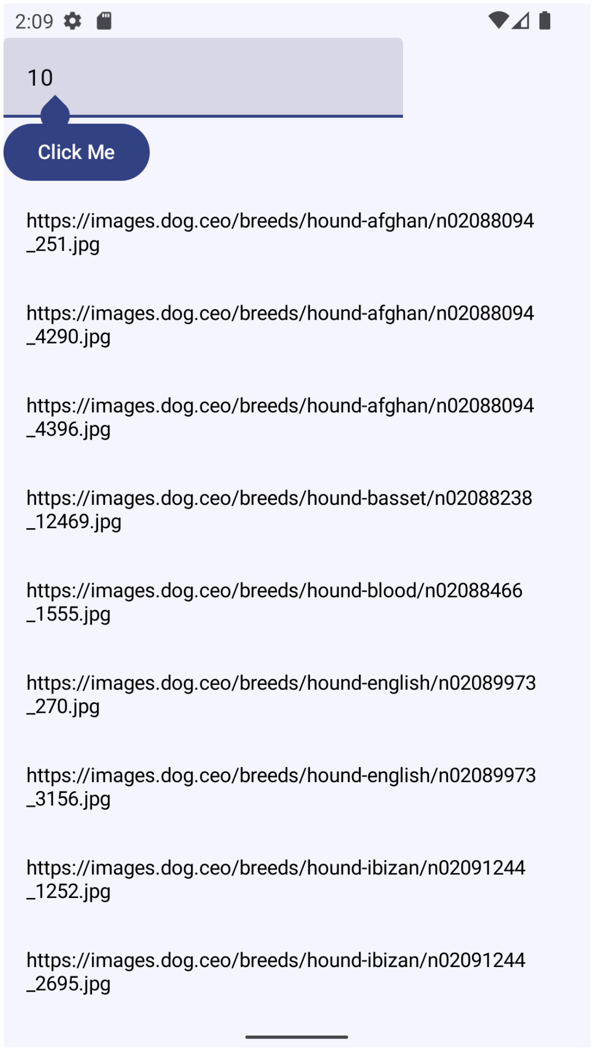
```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        val retrofit = Retrofit.Builder()
            .baseUrl("https://dog.ceo/api/")
            .addConverterFactory(
                GsonConverterFactory.create()
            )
            .build()
        val downloadService = retrofit
            .create<DogService>(
                DogService::class.java
            )
        val database = Room.databaseBuilder(
            applicationContext,
            DogDatabase::class.java,
            "dog-db"
        )
            .build()
        val dogRepository = DogRepositoryImpl(
            DownloadPreferences(applicationContext),
            ProviderFileHandler(
                applicationContext,
                FileToUriMapper()
            ),
            downloadService,
            database.dogDao(),
            DogMapper()
        )
...
    }
```

36. In the same function, add to the `Scaffold` function the `Dog` function defined previously, with all the required components:

```
override fun onCreate(savedInstanceState: Bundle?) {
...
    setContent {
        Activity1201Theme {
            Scaffold(
                modifier = Modifier.fillMaxSize()
            ) { innerPadding ->
                val viewModel =
                    viewModel<DogViewModel>(
                       factory = object :
                           ViewModelProvider.Factory {
                               override fun
                                   <T : ViewModel>
                                       create(
                                           modelClass:
                                           Class<T>
                                       ): T {
                                return DogViewModel(
                                    dogRepository
                                ) as T
                            }
                        })
                    Dog(
                        dogViewModel = viewModel,
                        modifier = Modifier
```

```
                                .padding(innerPadding)
                                .fillMaxSize()
                        )
                    }
                }
            }
        }
```

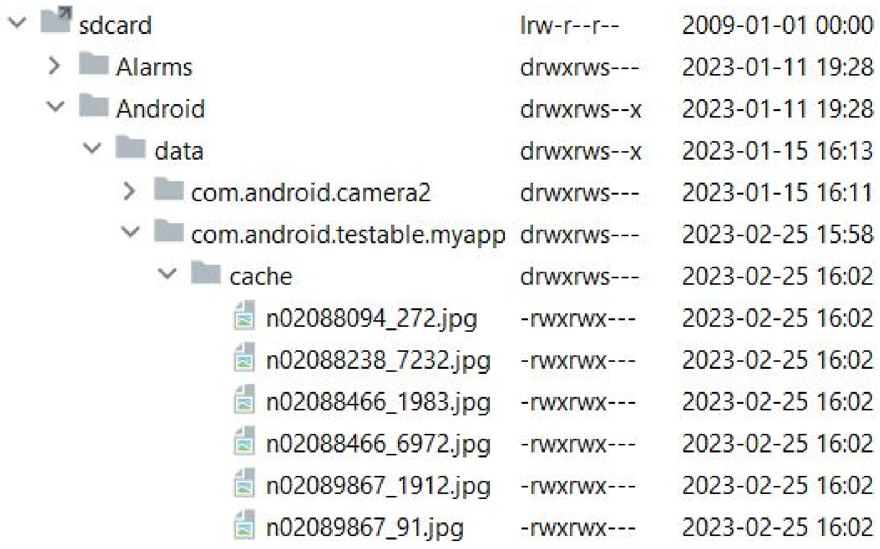
37. If you run the application now, you will see the following output:



```
2:09
10
Click Me
https://images.dog.ceo/breeds/hound-afghan/n02088094
_251.jpg
https://images.dog.ceo/breeds/hound-afghan/n02088094
_4290.jpg
https://images.dog.ceo/breeds/hound-afghan/n02088094
_4396.jpg
https://images.dog.ceo/breeds/hound-basset/n02088238
_12469.jpg
https://images.dog.ceo/breeds/hound-blood/n02088466
_1555.jpg
https://images.dog.ceo/breeds/hound-english/n02089973
_270.jpg
https://images.dog.ceo/breeds/hound-english/n02089973
_3156.jpg
https://images.dog.ceo/breeds/hound-ibizan/n02091244
_1252.jpg
https://images.dog.ceo/breeds/hound-ibizan/n02091244
_2695.jpg
```

Figure 12.1 – Output of Activity 12.01

38. Clicking the items will trigger the download for each individual item. You can view the files using **Device File Explorer**:



```
sdcard                          lrw-r--r--    2009-01-01 00:00
  Alarms                        drwxrwx---    2023-01-11 19:28
  Android                       drwxrws--x    2023-01-11 19:28
    data                        drwxrws--x    2023-01-15 16:13
      com.android.camera2       drwxrws---    2023-01-15 16:11
      com.android.testable.myapp drwxrws---   2023-02-25 15:58
        cache                   drwxrws---    2023-02-25 16:02
          n02088094_272.jpg     -rwxrwx---    2023-02-25 16:02
          n02088238_7232.jpg    -rwxrwx---    2023-02-25 16:02
          n02088466_1983.jpg    -rwxrwx---    2023-02-25 16:02
          n02088466_6972.jpg    -rwxrwx---    2023-02-25 16:02
          n02089867_1912.jpg    -rwxrwx---    2023-02-25 16:02
          n02089867_91.jpg      -rwxrwx---    2023-02-25 16:02
```

Figure 12.2 – View of downloaded files on the device