Assignment 2

100 points

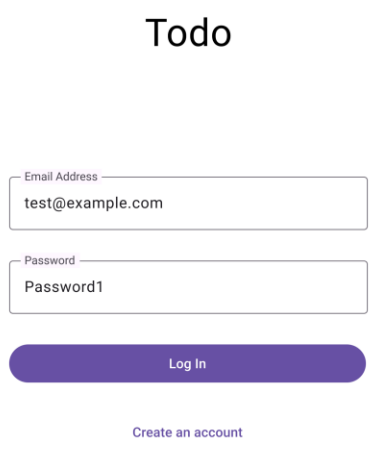
Due: August 7, 2024 @ 6:00PM

The goal in this assignment is to take the work you did in Assignment 1 and connect it to a web service to manage the application data so that it persists across restarts of the application. We will be using a free API at https://todos.simpleapi.dev This API requires the use of an API key to make API calls. Visit https://todos.simpleapi.dev/get-key.php to create an account and get an API key. You will use this key in every request you make as a query parameter called "apikey". There are two modes for interacting with the API: 1) generic todos that are tied to your API key, or 2) user todos.

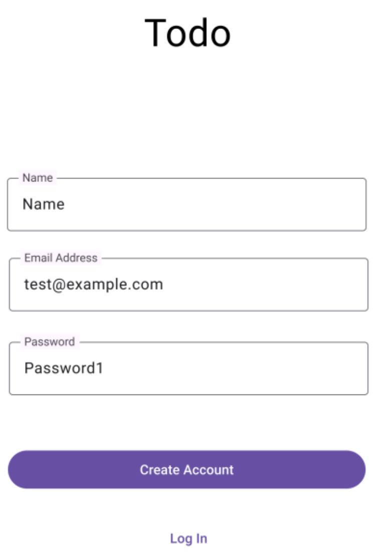
User Interface

You will be implementing both a create account and a log in screen.

LOGIN



CREATE ACCOUNT



Tapping on the “Create an account” button on the “Log In” screen will take the user to the “Create Account” screen. Tapping on the “Log In” button on the “Create Account” screen will take the user to the “Log In” screen.

On both screens if a user attempts to “Log In” or “Create Account” and the form fields are empty, show an alert to the user indicating that the email and password need to be entered.

After a user has successfully created an account or logged in, the app should navigate to the home screen you created in Assignment 1.

HTTP and REST API

Create an API Service using Retrofit.

The documentation for the TODO API is here: https://todos.simpleapi.dev/docs.php

Create methods for each of the following routes:

GET /api/users/{user_id}/todos - fetches the list of todos for user with id {user_id}

POST /api/users/{user_id}/todos - creates a new todo for user with id {user_id}

PUT /api/users/{user_id}/todos/{id} - updates a todo item for id {user_id} with todo id {id}

POST /api/users/register - creates a new user returns a user object with token and id.

POST /api/users/login - logs a user in and returns a user object with token and id.

Look at the documentation for each route for details on the request bodies and responses.

Create data classes for each of request and response types. Use those types in your Retrofit function definitions. Do NOT use Strings to represent complex data types. But do note that the same data types are used in multiple routes. You can and should reuse data classes when possible. For example, POST /api/users/register and POST /api/users/login both return the same data type, you should use the same return type for both of those functions.

Architecture

Create three View Models.

1. Log In View Model – manages the Log In screen and the related operations.
2. Create Account View Model – manages the Create Account screen and the related operations.
3. Todo List View Model – manages the main Todo List screen.

The view models will each have an instance of the API service you create with the Retrofit Builder. Remember to set up Retrofit to use Moshi to serialize and deserialize your objects to and from Json. Examples of this can be found in the slides or on many places just a search away.

The Log In View Model and the Create Account View Model will need to get the token and the user id from either the log in response or the create account response and store them in a place where other parts of the application can make use of it. I suggest using SharedPreferences which is a built-in part of the Android SDK.

Application Behavior

For the main section of your application these are the requirements:

1. The main screen of the application should GET the list of the logged in user's todos from the server.
2. When tapping the check box to "complete" a todo item, the app should PUT that to the server.
3. When creating a new todo from the bottom sheet, the app should POST that new todo to the server.
4. After creating a new todo, make sure to GET the list of the logged in user's todos again so that the new todo can be displayed.
5. If you quit the application and log in again, the todos for the user should be fetched and displayed.
6. If you quit the application and log in with a different user, the different user's todos should be fetched and displayed.

For this application, the API is the data layer. Therefore, you do not need to implement a database to store todo items that are fetched from the server. You should store the current list of todos in the Todo List View Model. This list should be *replaced* when GETting the list of the user's todos.

Error Handling

Every time you make an API call, there is a chance for an error to occur. If an error occurs your application ***MUST*** display an alert to a user indicating that something has gone wrong and giving them instructions on how to fix the issue if possible (for example, email address is incorrect so check that) and if it's not possible ask them to try again (for example, creating a new todo fails for some unknown reason).

Requirements

- Implement the UI in Jetpack Compose as shown above.
    - You may use whichever colors for the components you wish provided you adhere to the following:
        - The save button must be filled
        - The cancel button must be unfilled but have an outline
        - The text field must be an outlined text field
        - The todo items list must have the correct horizontal padding (12dp).
- Your submission must be pushed to GitHub in a branch called “assignment2”.
- You must submit the link to your merge request as your submission.
- You must use MVVM.
- You must use Jetpack compose.
- You must use Kotlin Coroutines (i.e. suspend functions) in your Retrofit functions. (Remember that ViewModel super class provides viewModelScope for calling creating coroutines. You can use this in any function not just the init block like we did in lab 6).
- You must use good coding practices that we’ve discussed so far (e.g. state hoisting and dependency injection).

Tips for success

- Get started early. This will take you longer than you think it will.
- I do not pre-grade assignments. Please read the requirements when you are unclear of how to proceed. If you ask me to look over your implementation to tell you if you did everything correctly, I will not do so.
- All strings that are not part of the data must be placed in the strings.xml file (this includes the labels for the text fields and buttons). ***Do not forget this***.
- Double and triple check that what you have pushed to GitHub is what you expect. If your submission is not there by the deadline it will incur a late penalty even if you finished it before the deadline but didn’t push all the code to GitHub.
- Make sure that the link you are submitting is the link to the Merge Request. Submitting any other link than the link to the Merge Request will result in a penalty.
- Make sure your branch name is correct.
- Remember when asking for help you must tell me these things:
    - What are you trying to accomplish?
    - What have you tried to do to accomplish that?
    - What specific question do you have?

Assessment

30 points – Correct implementation of user interface as shown in the images with allowances for exceptions as described above.

55 points – Correct behavior of user interface as described above.

10 points – Correct coding style. Strings externalized into strings.xml, code formatted properly, variables named appropriately, vertical whitespace is appropriate, proper null handling, proper state hoisting.

5 points – Proper submission of assignment by creating an MR in GitHub, proper branch name, and proper link submitted to D2L.

APPENDIX

Dependency versions

Open the file “libs.versions.toml” and replace all of the content with the following text that is between the lines of “#”.

##

```
[versions]

agp = "8.4.0"

kotlin = "1.9.0"

coreKtx = "1.13.1"

junit = "4.13.2"

junitVersion = "1.2.1"

espressoCore = "3.6.1"

lifecycleRuntimeKtx = "2.8.3"

activityCompose = "1.9.0"

composeBom = "2024.04.01"


[libraries]

androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }

junit = { group = "junit", name = "junit", version.ref = "junit" }
```

```
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }

androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core",
version.ref = "espressoCore" }

androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-
ktx", version.ref = "lifecycleRuntimeKtx" }

androidx-activity-compose = { group = "androidx.activity", name = "activity-compose",
version.ref = "activityCompose" }

androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref =
"composeBom" }

androidx-ui = { group = "androidx.compose.ui", name = "ui" }

androidx-ui-graphics = { group = "androidx.compose.ui", name = "ui-graphics" }

androidx-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }

androidx-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }

androidx-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }

androidx-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }

androidx-material3 = { group = "androidx.compose.material3", name = "material3" }

moshi = { group = "com.squareup.moshi", name = "moshi", version = "1.15.1"}

moshi-kotlin = { group = "com.squareup.moshi", name = "moshi-kotlin", version = "1.15.1"}

android-coroutines = { group = "org.jetbrains.kotlinx", name = "kotlinx-coroutines-android",
version = "1.7.3" }

androidx-lifecycle-viewmodel = { group = "androidx.lifecycle", name = "lifecycle-viewmodel",
version = "2.8.3" }

androidx-lifecycle-viewmodel-compose = { group = "androidx.lifecycle", name = "lifecycle-
viewmodel-compose", version = "2.8.3" }

androidx-lifecycle-viewmodel-ktx = { group = "androidx.lifecycle", name = "lifecycle-
viewmodel-ktx", version = "2.8.3" }


[plugins]

android-application = { id = "com.android.application", version.ref = "agp" }

jetbrains-kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }

########################################
```

Build.gradle updates

Open the build.gradle file that is in the “app” directory. There are two build.gradle files, make sure you’re opening the correct one. You will know if you have opened the correct one if there is a “dependencies” section. Replace the entire dependency block with the text between the lines of “/”.

```
//////////////////////////////
dependencies {

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    implementation(libs.moshi)
    implementation(libs.moshi.kotlin)
    implementation(libs.android.coroutines)
    implementation(libs.androidx.lifecycle.viewmodel)
    implementation(libs.androidx.lifecycle.viewmodel.compose)
    implementation(libs.androidx.lifecycle.viewmodel.ktx)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)
}
//////////////////////
```

Sync

Once you have done those two modifications, sync your project by clicking on the elephant with an arrow icon on the top right toolbar.