Lab 10 - Data Store

`COMP4107 - SDDT - HKBU - Spring2023`

Today we'll continue building our InfoDay app using Jetpack Compose. We'll discuss how to **save data in key-value storage**.

Clone your project from GitHub https://classroom.github.com/a/t86LHCCf and let's keep developing.

Saving Small Data with Jetpack DataStore

Sometimes we want to **save a small amount of data like settings, status, or records permanently in our app**. This can be easily handled with **Jetpack Datastore**, which is simply a **permanent key-value store**.

Now, let's add the following to your module-level gradle:

```
implementation "androidx.datastore:datastore-preferences:1.0.0"
```

Then, let's create a new Kotlin file `UserPerferences.kt` as follows:

```
import android.content.Context
import androidx.datastore.core.DataStore
import androidx.datastore.preferences.core.Preferences
import androidx.datastore.preferences.core.booleanPreferencesKey
import androidx.datastore.preferences.core.edit
import androidx.datastore.preferences.preferencesDataStore
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map

class UserPreferences(private val context: Context) {

    // to make sure there's only one instance
    companion object {
        private val Context.dataStore: DataStore<Preferences> by
preferencesDataStore("settings")
        val DARK_MODE = booleanPreferencesKey("dark_mode")
    }
}
```

In this companion object, the first line configures a **Jetpack DataStore** named `settings` to hold **key-value pairs**. The second line sets up a `dark_mode` key expected to have a Boolean value.

Then implement these two functions in this class to support **basic read/write operations**:

```
// get the mode
val getMode: Flow<Boolean?> = context.dataStore.data
    .map { preferences ->
        preferences[DARK_MODE] ?: false
    }

// save mode into datastore
suspend fun saveMode(mode: Boolean) {
    context.dataStore.edit { preferences ->
        preferences[DARK_MODE] = mode
    }
}
```

In Kotlin, **Flow** is a reactive stream library that provides a way to **emit multiple values asynchronously** in a sequential manner. It is a **cold stream**, meaning that it starts **emitting values only when a collector** is attached to it.

Next, proceed to `InfoScreen.kt` and develop a list item with a **switch**:

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SettingList() {

    var checked by remember { mutableStateOf(true) }

    ListItem(
        headlineText = { Text("Dark Mode") },
        leadingContent = {
            Icon(
                Icons.Filled.Settings,
                contentDescription = null
            )
        },
        trailingContent = {
            Switch(
                modifier = Modifier.semantics { contentDescription = "Demo" },
                checked = checked,
                onCheckedChange = {
                    checked = it
                })
        }
    )
}
```

When the user toggles the switch, the `onCheckedChange` lambda will be called with `it` holding the changed value. We store this value in the state `checked`. This value is protected by the `remember` keyword, making it immutable to recomposition.

We want to **persist** `checked` even after app restarts by storing it in the `DataStore`. Construct the following in the `SettingList` composable:

```
val dataStore = UserPreferences(LocalContext.current)
val coroutineScope = rememberCoroutineScope()
```

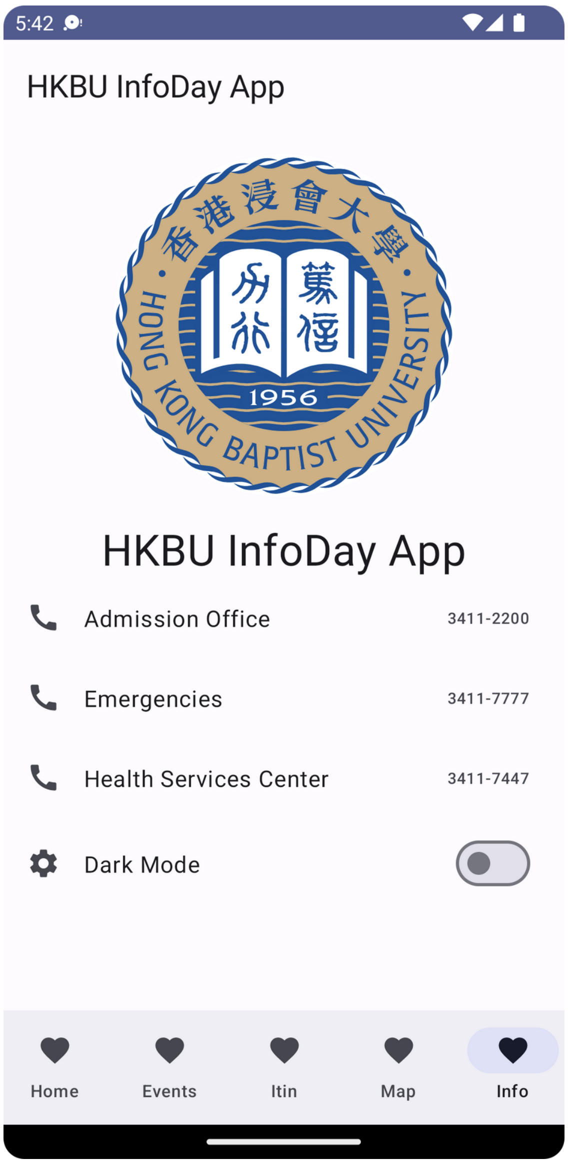
The first line creates a `UserPreferences` instance and establishes a DataStore. The second line establishes a **coroutine scope** needed to save DataStore changes as follows:

```
coroutineScope.launch {
    dataStore.saveMode(it)
}
```

Add the above to the `onCheckedChange` lambda expression. Any changes to the switch will now also save in the DataStore.

Finally, add `SettingList` to `InfoScreen` as follows:

```
fun InfoScreen() {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        InfoGreeting()
        PhoneList()
        SettingList()
    }
}
```


5:42

HKBU InfoDay App

HKBU InfoDay App

Admission Office 3411-2200

Emergencies 3411-7777

Health Services Center 3411-7447

Dark Mode

Retrieving Data from Data Store

To apply dark mode, we only need to provide a boolean `darkTheme` argument to the `InfoDayTheme()` composable function. You can view the `Theme.kt` file under `ui.theme` to learn more about this class.

In `MainActivity.kt`, let's retrieve the stored value and pass it as an argument to `InfoDayTheme()` to initialize it.

```
setContent {

    val dataStore = UserPreferences(LocalContext.current)
    val mode by dataStore.getMode.collectAsState(initial = false)

    InfoDayTheme(darkTheme = mode ?: false) {
        // A surface container using the 'background' color from the theme
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
//              Greeting("Android")
            ScaffoldScreen()
        }
    }
}
```

In Kotlin, `collectAsState()` is a method provided by the `StateFlow` class in the Kotlin coroutines library.

`collectAsState()` is used to collect values emitted by a `StateFlow` and represent them as a state object that can be observed by the UI. When the value of the `StateFlow` changes, the state object is updated, and the UI automatically recomposes to reflect the new state.

In short, updates to the dataStore will change the `mode` value, re-rendering `InfoDayTheme` to display the changes.

Toggling the switch now changes the app theme. Dark mode preference persists even after restarting the app!

However, the switch on `InfoScreen` will stay checked. This is because we reset `checked` to true whenever we return to this screen with:

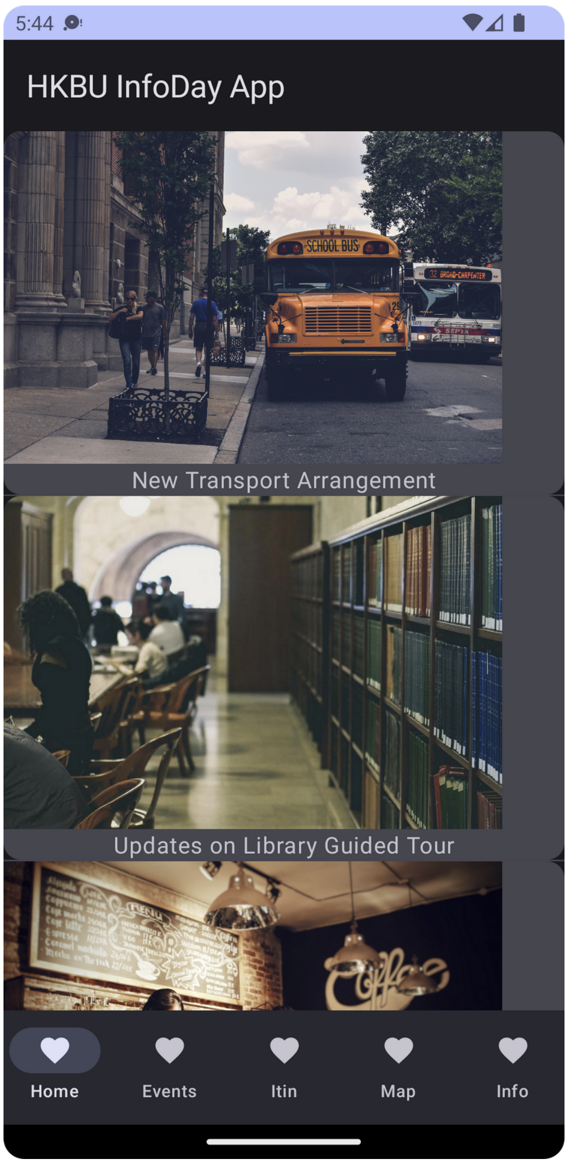
```
var checked by remember { mutableStateOf(true) }
```

To solve this, we can directly bind the switch with the DataStore value as follows:

```
val checked by dataStore.getMode.collectAsState(initial = false)
//    var checked by remember { mutableStateOf(true) }
```

We should remove the `checked = it` statement, as `checked` is no longer a mutable state and cannot be modified by calling the dataStore function.

```
Switch(
    modifier = Modifier.semantics { contentDescription = "Demo" },
//                  checked = checked,
    checked = checked ?: true,
    onCheckedChange = {
//                      checked = it
        coroutineScope.launch {
            dataStore.saveMode(it)
        }
    })
```



Commit and push your project to the private repo.

09/03/2023 00:32