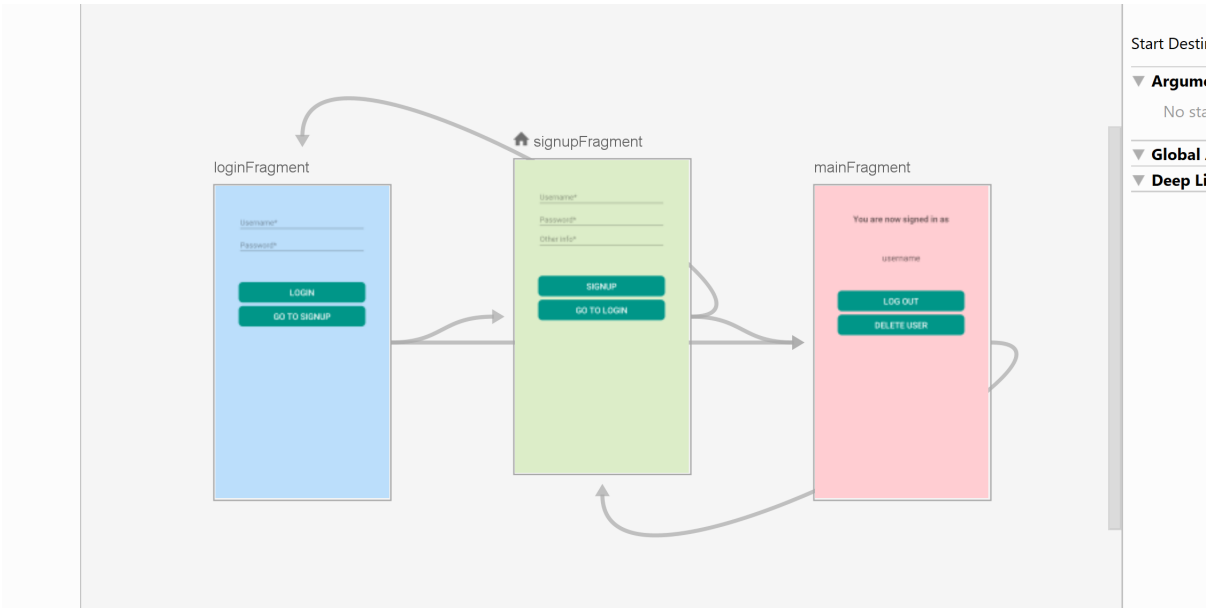
Room and Coroutines

로그인 화면 만들기

아래와 같이 사인업 - 로그인 - 로그인 완료 상태의 fragment 를 사용한다.



login fragment

- login
- signup 으로 가기

signup fragment

- signup
- login 으로 가기

main fragment

- logout
- delete user

0. 레이아웃 디자인, 그래이들 설정

```
def nav_version = "2.1.0"

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'


    implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"

    def room_version = "2.2.2"
    implementation "androidx.room:room-runtime:$room_version"
    kapt "androidx.room:room-compiler:$room_version"
    implementation "androidx.room:room-ktx:$room_version"

    def coroutines_version = "1.3.0"
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:$coroutines_version"
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:$coroutines_version"

    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.2.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.1.0'
}
```

1. Activity 셋팅하기

activity_main.xml 에 default nav host 를 true 로 설정한다. nav graph 또한 navigation.xml 파일로 설정해준다.

```
<fragment
        android:id="@+id/fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/navigation" />
```

2. Database 셋팅하기

a. User 엔티티 클래스 작성

b. DAO 인터페이스 작성

c. database 생성하는 object 클래스 작성

d. 이와 별도로 Login State 를 담는 State 클래스 작성

User.kt

```
package com.devtides.coroutinesroom.db

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class User(
    val username : String,

    //데이터베이스에는 name 과 같이 필드 이름이 저장된다.
    @ColumnInfo(name = "password_hash")
    val passwordHash : Int,

    val info :String
){

    @PrimaryKey
    var id : Long = 0;
}
```

UserDao.kt

```
package com.devtides.coroutinesroom.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query

@Dao
interface UserDao{

    //return 은 PK 이다.
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(user : User) : Long

    @Query("SELECT * FROM user WHERE username = :username")
    suspend fun getUser(username: String) : User

    @Query("DELETE FROM user WHERE id = :id")
    suspend fun deleteUser(id : Long)
}
```

UserDatabase.kt

```
package com.devtides.coroutinesroom.db

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = arrayOf(User::class), version = 1)
abstract class UserDatabase: RoomDatabase() {
    abstract fun userDao(): UserDao

    companion object {
        @Volatile private var instance: UserDatabase? = null
        private val LOCK = Any()

        operator fun invoke(context: Context) = instance
            ?: synchronized(LOCK) {
            instance
                ?: buildDatabase(
                    context
                ).also {
                instance = it
            }
        }

        private fun buildDatabase(context: Context) = Room.databaseBuilder(
            context.applicationContext,
            UserDatabase::class.java,
            "userdatabase"
        ).build()
    }
}
```

LoginState.kt

```
package com.devtides.coroutinesroom.db

object LoginState{

    var isLoggedIn = false
    var user: User? = null

    fun logout(){
        isLoggedIn = false
        user = null
    }

    fun login(user : User){
        isLoggedIn = true
```

```
        this.user = user
    }
}
```

3. Fragment 에 상응하는 ViewModel 클래스 안에서 코루틴을 사용해 Database Operation 을 한다.

SignupViewModel.kt (나머지 ViewModel 들도 유사하게 작업을 처리해준다)

```
package com.devtides.coroutinesroom.viewmodel

import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.MutableLiveData
import com.devtides.coroutinesroom.db.LoginState
import com.devtides.coroutinesroom.db.User
import com.devtides.coroutinesroom.db.UserDatabase
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

class SignupViewModel(application: Application) : AndroidViewModel(application) {

    private val coroutineScope = CoroutineScope(Dispatchers.IO)
    //android view model 을 사용하므로 application context 를 사용한다.
    private val db by lazy{ UserDatabase(getApplication()).userDao()}
    //lazy - 필요할 때 사용하겠다

    val signupComplete = MutableLiveData<Boolean>()
    val error = MutableLiveData<String>()

    fun signup(username: String, password: String, info: String) {
        coroutineScope.launch {
            //이미 유저가 있는지 확인해야 한다.
            val user = db.getUser(username)
            if(user != null){
                //update the interface - main dispatcher 사용한다.
                withContext(Dispatchers.Main){
                    error.value = "User already exists!"
                }
            }else{
                val user = User(username, password.hashCode(), info)
                val userId = db.insertUser(user) //들어간 아이디가 반환된다.
                user.id = userId
                LoginState.login(user)
                withContext(Dispatchers.Main){
                    signupComplete.value = true
                }
            }
        }
    }
```

```
}
```

- **중요한것은 데이터베이스 연산의 경우 Dispatchers.IO 로 처리하고, 화면을 업데이트 해야 하는 경우 withContext(Dispatchers.Main) 으로 작업을 해준다는 점이다.**
- Login State 는 withContext 안에서 작업하지 않아도 되는데, 정확한 이유를 모르겠다.