A. Explain the updates made in the Model Classes and the dependencies added for this thing to work [Glide] –Explain the registration and login

B. Setup the [Authentication , Fire Store and Storage] in cloud with all the authorizations and authentications.

B. Update the Repo

fun observeProjects(): Flow<List<Project>> = *callbackFlow* **{** val snapShotListener = projectCollectionRef  
  **.whereEqualTo("userId", FirebaseAuth.getInstance().*currentUser*?.*uid*)**  
 .addSnapshotListener **{** values, err **->**

**………..**

**}**

**2.**

suspend fun uploadPhoto(photoUri: Uri): String {  
 var timeStamp = SimpleDateFormat("yyyymmdd\_HHmmss").format(Date())  
 var imageName = "Image\_$timeStamp.png"  
  
 var storageReference = FirebaseStorage.getInstance()  
 .*reference*.child("images").child(imageName)  
 storageReference.putFile(photoUri).await()  
 return storageReference.*downloadUrl*.await().toString()  
}

3. ADD PROJECT WITH IMAGE

suspend fun addProject(project: Project, imageUri: Uri?) {  
 **if (imageUri != null)  
 project.imageURL = uploadPhoto(imageUri)** **project.userId = FirebaseAuth.getInstance().*currentUser*?.*uid*.*toString*()**  
 projectCollectionRef.add(project)  
}

//TODO

citiesRef.where('state', '==', 'CO').where('name', '==', 'Denver');

<https://itsmevikash.medium.com/perform-simple-and-compound-queries-in-cloud-firestore-android-java-f435d016ec7f>

Code the sign In view Model

val context = application  
private var auth = FirebaseAuth.getInstance()  
private val \_user: MutableStateFlow<User?> = *MutableStateFlow*(null)  
val user: StateFlow<User?> = \_user  
var userRegistratedSuccessfully = *MutableStateFlow*(false)  
  
init {  
 if (auth.*currentUser* != null) \_user.value = User(auth.*currentUser*?.*email*!!, "")  
}

fun **registerUser**(email: String, password: String) = *viewModelScope*.*launch* **{** auth.createUserWithEmailAndPassword(email, password)  
 .addOnCompleteListener **{** task **->** if (task.*isSuccessful*) {  
 *// Sign in success, update UI with the signed-in user's information* userRegistratedSuccessfully.value = true  
 Toast.makeText(  
 context,  
 " Successfully registered user",  
 Toast.*LENGTH\_SHORT*,  
 ).show()  
 } else {  
 *// If sign in fails, display a message to the user.* Log.w("TAG", "createUserWithEmail:failure", task.*exception*)  
 Toast.makeText(  
 context,  
 " Failed to register user.",  
 Toast.*LENGTH\_SHORT*,  
 ).show()  
 }  
 **}  
}**fun signIn(email: String, password: String) = *viewModelScope*.*launch* **{** auth.signInWithEmailAndPassword(email, password)  
 .addOnCompleteListener **{** task **->** if (task.*isSuccessful*) {  
 *// Sign in success, update UI with the signed-in user's information* Log.d("TAG", "signInWithEmail:success")  
 \_user.value = User(auth.*currentUser*?.*email*!!, "")  
  
 } else {  
 *// If sign in fails, display a message to the user.* Log.w("TAG", "signInWithEmail:failure", task.*exception*)  
 \_user.value = null  
 }  
 **}  
}**fun signOut() = *viewModelScope*.*launch* **{** auth.signOut()  
 \_user.value = null  
**}**override fun onCleared() {  
 super.onCleared()  
 auth.signOut()

\_user.value = null  
  
}

**Remaining code just explain**

**Navigation checks**

val loggedInUser = signInViewModel.user.collectAsStateWithLifecycle().value  
  
NavHost(  
 startDestination =

**if (loggedInUser != null)**

NavDestinations.ProjectScreen.route

**else**

NavDestinations.LoginScreen.route)

**Explain the Login and Registration Screen**

4. CHOOSING AN IMAGE

var imageUri by remember **{** *mutableStateOf*<Uri?>(null) **}**var imageChooser = rememberLauncherForActivityResult(ActivityResultContracts.GetContent())**{** imageUri = **it  
}**

ElevatedButton(onClick = **{** imageChooser.launch("image/\*")  
**}**) **{** Text(text = "Choose Image") **}**

GlideImage(model = imageUri, contentDescription = "")