/// LokalApp: Job Listing App Structure

// File: build.gradle (app)

dependencies {

implementation 'androidx.core:core-ktx:1.10.1'

implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.6.2'

implementation 'androidx.activity:activity-compose:1.7.2'

implementation 'androidx.compose.ui:ui:1.5.0'

implementation 'androidx.compose.material:material:1.5.0'

implementation 'androidx.navigation:navigation-compose:2.7.5'

implementation 'com.squareup.retrofit2:retrofit:2.9.0'

implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

implementation 'androidx.room:room-runtime:2.6.1'

kapt 'androidx.room:room-compiler:2.6.1'

implementation 'androidx.room:room-ktx:2.6.1'

implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.1'

}

// File: Job.kt (Model)

data class Job(

val id: Int,

val title: String,

val location: String,

val salary: String?,

val phone: String?,

val description: String?

)

// File: BookmarkedJob.kt (Entity)

@Entity(tableName = "bookmarked\_jobs")

data class BookmarkedJob(

@PrimaryKey val id: Int,

val title: String,

val location: String,

val salary: String?,

val phone: String?,

val description: String?

)

// File: JobApiService.kt

interface JobApiService {

@GET("common/jobs")

suspend fun getJobs(@Query("page") page: Int): Response<JobResponse>

}

data class JobResponse(val data: List<Job>)

// File: JobDao.kt

@Dao

interface JobDao {

@Query("SELECT \* FROM bookmarked\_jobs")

fun getAllBookmarked(): Flow<List<BookmarkedJob>>

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertBookmarked(job: BookmarkedJob)

@Query("DELETE FROM bookmarked\_jobs WHERE id = :id")

suspend fun deleteById(id: Int)

}

// File: JobRepository.kt

class JobRepository(private val api: JobApiService, private val dao: JobDao) {

suspend fun fetchJobs(page: Int): List<Job> = api.getJobs(page).body()?.data ?: emptyList()

fun getBookmarkedJobs(): Flow<List<BookmarkedJob>> = dao.getAllBookmarked()

suspend fun bookmarkJob(job: Job) {

dao.insertBookmarked(BookmarkedJob(job.id, job.title, job.location, job.salary, job.phone, job.description))

}

suspend fun removeBookmark(id: Int) = dao.deleteById(id)

}

// File: JobViewModel.kt

class JobViewModel(private val repository: JobRepository) : ViewModel() {

var jobs = mutableStateListOf<Job>()

var page = 1

var isLoading by mutableStateOf(false)

var error by mutableStateOf<String?>(null)

fun loadJobs() {

viewModelScope.launch {

try {

isLoading = true

val newJobs = repository.fetchJobs(page++)

jobs.addAll(newJobs)

} catch (e: Exception) {

error = e.message

} finally {

isLoading = false

}

}

}

val bookmarks = repository.getBookmarkedJobs().stateIn(viewModelScope, SharingStarted.Lazily, emptyList())

fun bookmark(job: Job) {

viewModelScope.launch { repository.bookmarkJob(job) }

}

fun removeBookmark(id: Int) {

viewModelScope.launch { repository.removeBookmark(id) }

}

}

// File: Navigation.kt

@Composable

fun AppNavGraph(viewModel: JobViewModel) {

val navController = rememberNavController()

Scaffold(bottomBar = {

BottomNavigation {

BottomNavigationItem(icon = { Icon(Icons.Default.List, null) }, selected = false, onClick = {

navController.navigate("jobs")

})

BottomNavigationItem(icon = { Icon(Icons.Default.Favorite, null) }, selected = false, onClick = {

navController.navigate("bookmarks")

})

}

}) {

NavHost(navController, startDestination = "jobs") {

composable("jobs") { JobListScreen(viewModel, navController) }

composable("bookmarks") { BookmarkScreen(viewModel) }

composable("details/{jobId}") { backStackEntry ->

val jobId = backStackEntry.arguments?.getString("jobId")?.toInt()

JobDetailScreen(jobId = jobId, viewModel = viewModel)

}

}

}

}

// File: JobListScreen.kt

@Composable

fun JobListScreen(viewModel: JobViewModel, navController: NavController) {

val jobs = viewModel.jobs

val context = LocalContext.current

LazyColumn {

items(jobs) { job ->

JobCard(job = job, onClick = {

navController.navigate("details/${job.id}")

}, onBookmark = {

viewModel.bookmark(job)

})

}

item {

if (viewModel.isLoading) CircularProgressIndicator(modifier = Modifier.align(Alignment.CenterHorizontally))

else Button(onClick = { viewModel.loadJobs() }) { Text("Load More") }

}

}

}

// File: JobCard.kt

@Composable

fun JobCard(job: Job, onClick: () -> Unit, onBookmark: () -> Unit) {

Card(

modifier = Modifier.fillMaxWidth().padding(8.dp).clickable { onClick() },

elevation = 4.dp

) {

Column(modifier = Modifier.padding(16.dp)) {

Text(job.title, fontWeight = FontWeight.Bold)

Text("Location: ${job.location}")

job.salary?.let { Text("Salary: $it") }

job.phone?.let { Text("Phone: $it") }

Button(onClick = onBookmark) {

Text("Bookmark")

}

}

}

}

// File: BookmarkScreen.kt

@Composable

fun BookmarkScreen(viewModel: JobViewModel) {

val bookmarks by viewModel.bookmarks.collectAsState()

LazyColumn {

items(bookmarks) { job ->

JobCard(job = job.toJob(), onClick = {}, onBookmark = {

viewModel.removeBookmark(job.id)

})

}

}

}

// Extension function

fun BookmarkedJob.toJob() = Job(id, title, location, salary, phone, description)

// File: JobDetailScreen.kt

@Composable

fun JobDetailScreen(jobId: Int?, viewModel: JobViewModel) {

val job = viewModel.jobs.find { it.id == jobId }

if (job != null) {

Column(modifier = Modifier.padding(16.dp)) {

Text(job.title, fontWeight = FontWeight.Bold, fontSize = 20.sp)

Text("Location: ${job.location}")

Text("Salary: ${job.salary ?: "N/A"}")

Text("Phone: ${job.phone ?: "N/A"}")

Spacer(modifier = Modifier.height(8.dp))

Text(job.description ?: "No description available")

}

} else {

Text("Job not found")

}

}