Kotlin Productivity Tracker App (Jetpack Compose + MVVM)

This Kotlin sample project demonstrates advanced modern Android development practices. It comprises Jetpack Compose, MVVM design, Kotlin Coroutines, Flow, Room Database, Dependency Injection using Hilt, and Clean design concepts. It is intended to impress technical recruiters and HR teams from leading technology companies such as Google, Meta, and Microsoft.

Project Structure Overview

```
com.maria.productivitytracker
├── data
│   ├── local
│   │   ├── TaskDao.kt
│   │   ├── TaskEntity.kt
│   │   └── AppDatabase.kt
├── domain
│   ├── model
│   │   └── Task.kt
│   ├── repository
│   │   └── TaskRepository.kt
├── presentation
│   ├── ui
│   │   └── components/
│   │   └── screens/
│   ├── viewmodel
│   │   └── TaskViewModel.kt
├── di
│   └── AppModule.kt
└── MainActivity.kt
```

1. Room Entity (TaskEntity.kt)

```
@Entity(tableName = "tasks")
data class TaskEntity(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val title: String,
    val description: String,
```

```
    val isCompleted: Boolean = false,
    val timestamp: Long = System.currentTimeMillis()
)
```

2. DAO Interface (TaskDao.kt)

```
@Dao
interface TaskDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertTask(task: TaskEntity)

    @Update
    suspend fun updateTask(task: TaskEntity)

    @Delete
    suspend fun deleteTask(task: TaskEntity)

    @Query("SELECT * FROM tasks ORDER BY timestamp DESC")
    fun getAllTasks(): Flow<List<TaskEntity>>
}
```

3. Room Database (AppDatabase.kt)

```
@Database(entities = [TaskEntity::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    abstract fun taskDao(): TaskDao
}
```

4. Domain Layer Model (Task.kt)

```
data class Task(
    val id: Int = 0,
    val title: String,
    val description: String,
    val isCompleted: Boolean,
    val timestamp: Long
)
```

5. Repository Interface + Implementation (TaskRepository.kt)

```
interface TaskRepository {
    suspend fun addTask(task: Task)
    suspend fun updateTask(task: Task)
    suspend fun deleteTask(task: Task)
    fun getTasks(): Flow<List<Task>>
```

```
}

class TaskRepositoryImpl @Inject constructor(
    private val dao: TaskDao
) : TaskRepository {

    override suspend fun addTask(task: Task) {
        dao.insertTask(task.toEntity())
    }

    override suspend fun updateTask(task: Task) {
        dao.updateTask(task.toEntity())
    }

    override suspend fun deleteTask(task: Task) {
        dao.deleteTask(task.toEntity())
    }

    override fun getTasks(): Flow<List<Task>> {
        return dao.getAllTasks().map { entities -> entities.map { it.toDomain() } }
    }
}
```

6. Mapping Extensions

```
fun TaskEntity.toDomain(): Task = Task(id, title, description, isCompleted, timestamp)
fun Task.toEntity(): TaskEntity = TaskEntity(id, title, description, isCompleted,
timestamp)
```

7. ViewModel (TaskViewModel.kt)

```
@HiltViewModel
class TaskViewModel @Inject constructor(
    private val repository: TaskRepository
) : ViewModel() {

    private val _tasks = MutableStateFlow<List<Task>>(emptyList())
    val tasks: StateFlow<List<Task>> = _tasks.asStateFlow()

    init {
        viewModelScope.launch {
            repository.getTasks().collectLatest { _tasks.value = it }
        }
```

```
    }

    fun addTask(task: Task) = viewModelScope.launch {
        repository.addTask(task)
    }

    fun toggleCompletion(task: Task) = viewModelScope.launch {
        val updated = task.copy(isCompleted = !task.isCompleted)
        repository.updateTask(updated)
    }

    fun deleteTask(task: Task) = viewModelScope.launch {
        repository.deleteTask(task)
    }
}
```

8. Jetpack Compose UI (TaskListScreen.kt)

```
@Composable
fun TaskListScreen(viewModel: TaskViewModel = hiltViewModel()) {
    val tasks by viewModel.tasks.collectAsState()

    LazyColumn {
        items(tasks) { task ->
            TaskCard(task, onToggle = { viewModel.toggleCompletion(task) }, onDelete = {
                viewModel.deleteTask(task)
            })
        }
    }
}
```

9. Composable Task Card (TaskCard.kt)

```
@Composable
fun TaskCard(task: Task, onToggle: () -> Unit, onDelete: () -> Unit) {
    Card(
        modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth(),
        elevation = CardDefaults.cardElevation(8.dp)
    ) {
        Row(
            modifier = Modifier
```

```
            .padding(16.dp)
            .clickable { onToggle() },
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        Column {
            Text(text = task.title, fontWeight = FontWeight.Bold)
            Text(text = task.description, fontSize = 14.sp)
        }
        IconButton(onClick = onDelete) {
            Icon(Icons.Default.Delete, contentDescription = "Delete Task")
        }
    }
  }
}
```

10. Hilt DI Module (AppModule.kt)

```
@Module
@InstallIn(SingletonComponent::class)
object AppModule {

  @Provides
  fun provideDatabase(@ApplicationContext context: Context): AppDatabase =
    Room.databaseBuilder(context, AppDatabase::class.java, "tasks_db").build()

  @Provides
  fun provideDao(db: AppDatabase): TaskDao = db.taskDao()

  @Provides
  fun provideRepository(dao: TaskDao): TaskRepository = TaskRepositoryImpl(dao)
}
```

11. Main Activity (MainActivity.kt)

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      MaterialTheme {
        Surface(modifier = Modifier.fillMaxSize()) {
          TaskListScreen()
        }
```

```
        }
      }
    }
  }
```