

Android labs – 2022/23



| | |
|---|----------|
| Introduction..... | 2 |
| Classes plan | 2 |
| Learning resources | 2 |
| Lab #1- Introduction to the development workflow and tools | 2 |
| A) Kotlin basics | 2 |
| B) Introduction to Android native apps development | 2 |
| Alternative track (Java) | 3 |
| Week #2- UI, Navigation and Fragments | 3 |
| C) Fragments, navigation and lifecycles..... | 3 |
| Homework assignment #1 (HW1) | 3 |
| Week #3- Application architecture..... | 3 |
| D) RecyclerView | 3 |
| E) App architecture (ViewModel, LiveData, Room) | 3 |
| Week #4- Connecting to backend services..... | 4 |
| F) Access network services | 4 |
| G) Integrate Mobile Backend-as-a-Service (Firebase)..... | 5 |
| Lab #4- Context-awareness: sensors and location. | 5 |
| H) Location awareness | 5 |
| I) Using (built-in) sensors | 6 |
| Lab #5: Extending the mobile app with AI | 6 |
| J) Extending the app with AI | 6 |
| Lab #6: Introduction to declarative layouts (Composable)..... | 8 |
| K) Declarative layouts with Jetpack Compose | 8 |

Introduction

Classes plan

See the overall [course schedule](#), in Moodle.

Learning resources

Selected Android learning resources:

- Official [Android developers' documentation](#): tutorials, API documentation, tools, best practices,...
- **Main resource** for CM classes:
 - [“Android Basics with Compose”](#)
 - Parts from [“Android Development with Kotlin: Classroom course”](#) (with [lecture slides](#))
- Other Android training resources:
 - More [Courses by Google](#) with codelabs.
 - Google’s “Android Training Program” (ATP-PT) held in Portuguese (in the Fall of 2020) and with the Kotlin programming language. All lessons are online. Resources: [video lessons](#), [slide decks](#), [source code](#) for examples.
 - Android [Courses at Udacity](#) (by Google staff members)
- Books
 - [“Professional Android”](#) (2018) by Reto Meier [Available from O'Reilly]
 - [“Programming Android with Kotlin”](#)

Lab #1- Introduction to the development workflow and tools

A) Kotlin basics



Slides for Kotlin concepts:

- lesson 1 [Kotlin Basics](#) → Lesson 2 [Functions](#) → Lesson 3 [Classes and Objects](#)

See also:

- Kotlin [online playground](#) (run code in the browser)

Hands-on exercises (with code labs):

-  Take the [codelab for lesson 1](#), Activity #2 “Kotlin basics”.
-  Take [codelab for Lesson 3](#), activity #1 “Using Classes and Objects in Kotlin”



B) Introduction to Android native apps development

Slides for lesson 4 [“Build your first Android app”](#)

Slides for lesson 5 [“Layouts”](#) [except RecyclerView]

See also: [API levels](#)

Hands-on exercises & code labs

-  Complete [Lesson 4 activities](#).
-  Complete [Lesson 5 activities](#) up to Activity #2 “Calculate the tip.” Introduces the use of [“view binding”](#). [Exclude activity 3 RecyclerView]

Alternative track (Java)

- First Android app: [Java edition](#)





Week #2- UI, Navigation and Fragments

C) Fragments, navigation and lifecycles

- Slides: [Lesson 6](#) – App Navigation
- Slides: [Lesson 7](#) – Activity and Fragment Lifecycle

Hands-on exercises (with code labs):

Note: the starter code provided in some exercises may not work on recent JDK versions. You may need to ensure you are [using the “Embedded JDK”](#) option for extended compatibility.

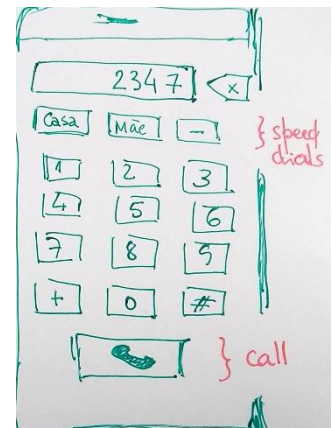
-  Complete [Lesson 6 – activity 1](#): Create a Fragment
-  Complete Lesson 6 – activity 2: Define Navigation paths [except step 10 Drawer menu]
-  Complete Lesson 6 – activity 3: Start an external Activity [be sure to implement the sharing implicit intent]
-  Complete [Lesson 7 – activity 1](#): Lifecycles and logging

Homework assignment #1 (HW1)

HW1: build an app that acts as a dialer, with a “keypad” to enter the number to call. Start with the simplest approach possible.

When you press the dial button, a call should be started (just [hand-over the number to the built-in dialer](#)).

You should add a set of 3 “speed dial” buttons (memories); when the user long presses one of these “speed dials”/memories, a secondary activity is offered to allow the user to update the speed dial details (define a label and associate a phone number). Suggestion: consider [calling a second activity for a result](#). (Alternatively, instead of two activities, you may consider using a hosting activity and two fragments with [navigation](#)).



Week #3- Application architecture

D) RecyclerView

- **Slides:** lesson 5 “[Layouts](#)” [RecyclerView slide 43+]
- [Optional]: slides for more [detailed RecyclerView use cases](#)

Hands-on exercises (with code labs):

-  Complete [Lesson 5. Activity 3](#) – Use RecyclerView

E) App architecture (ViewModel, LiveData, Room)

- **Slides:** [lesson 8](#) “App Architecture (UI)”
- **Slides:** [lesson 9](#) “App Architecture (Persistence)”

- [Recommendations](#) for Android Architecture
- [Android JetPack](#) overview and [short video](#) introduction [Architectural Components](#)
- Documentation guide: “[Save data in a local database using Room](#)”
- See also: simple (key-value) data storage using [Shared Preferences](#)

Hands-on exercises (with code labs):

 Complete [Lesson 8 activities](#).

 Complete [Lesson 9 activities](#).

 Or, alternatively, “[Room with a view](#)” code lab.

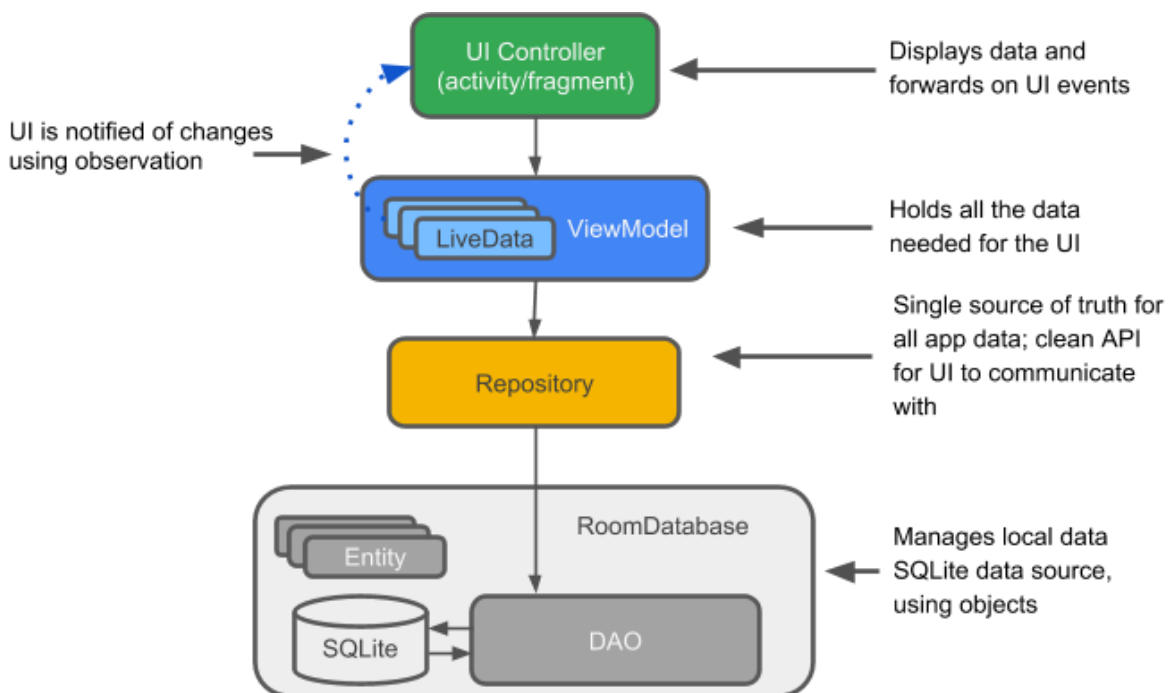


Figure 1- [Android architectural components](#): decouple UI controller from models and data sources.

Using The [Room persistence API](#) :

- Create entity(ies) to model the data. The **@Entity** will be mapped into a SQL table.
- Define an API to interact with the data in a **@Dao** object. Map methods into SQL statements (you may omit SQL for default @Insert, @Delete, @Update).
- If you wrap the results in a **LiveData**, the changes to the result set become observable (tracked), and data will be accessed in a separate thread automatically.
- Create a **@Database** object, declaring the entities to be managed and exposing required DAOs; implement a method to return a database singleton.

Week #4- Connecting to backend services

F) Access network services

- [Slides for Lesson 11](#): connect to the internet

Hands-on exercises (code lab):

 Complete [activities for Unit3 – lesson 11](#) [RecyclerView integration is optional]

Notes:

- Common web services use a [REST](#) architecture. Web services that offer REST architecture are known as *RESTful* services. The [Retrofit](#) library is a client library that enables your app to make requests to a REST web service. Retrofit works with conversion libraries to parse the JSON response from the web service into your app's live data (e.g.: Moshi).
- To simplify the process of managing images, use the [Glide library](#) to download, buffer, decode, and cache images in your app.

G) Integrate Mobile Backend-as-a-Service (Firebase)

- Starting with FB in Android video: [EN](#) | [PT](#)
- Lecture on [Firestore data](#) structuring.
- [Setup an Android project](#) to use Firebase.

Hands-on exercises (code lab):

 Complete Build Friendly Chat [code lab](#).

Suggested:

 Cloud [Firestore code lab](#).

Lab #4- Context-awareness: sensors and location.

H) Location awareness

Reading & resources:

- Concepts, API, and best practices to build [location-aware applications](#)
- Suggested: Meier's [Professional Android](#), chaps. 15
- Documentation on the [Service](#) application component (specially: foreground Services).

The "[Location and Context API](#)" by Google: harness the sensors and signals of mobile devices to provide awareness of user actions and their environment. The overall goal is to enhance accuracy (by combining data/signal sources), reduce battery draining and enhance privacy standards.

Requires [Google Play Services](#).

Hands-on exercises (code lab):

- Complete the "[Receive location updates in Kotlin](#)".
- Add and [interact with a \(Google\) map](#)

Location and Context APIs

English

| Where you are | What you're doing | What's nearby |
|---|--|---|
| <h3>Places API</h3> <p>Give your users contextual information about where they are, when they're there. Access detailed information about 100 million places across a wide range of categories.</p> | <h3>Google Fit Platform</h3> <p>Enable your users to record their fitness activity and track their fitness and health goals. Fit is a universal platform that lets users access their fitness data across multiple apps.</p> | <h3>Nearby Messages</h3> <p>Allow your users to find nearby devices and share messages in a way that's as frictionless as a conversation. Enable rich, collaborative group interactions.</p> |
| <h3>Geofencing</h3> <p>Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest.</p> | <h3>Activity Recognition API</h3> <p>The Activity Recognition API processes low power signals from multiple sensors in the device to accurately detect your users' current activity.</p> | <h3>Nearby Connections</h3> <p>Discover other devices nearby and create connections that enable real-time cross-device experiences.</p> |
| <h3>Fused Location Provider API</h3> <p>Get location data for your app based on combined signals from the device sensors using a battery-efficient API.</p> | <h3>Sensors API</h3> <p>Access raw data from all device sensors, as well as fused information from multiple sensors.</p> | <h3>Nearby Notifications</h3> <p>Nearby Notifications is an upcoming feature for contextual discovery. Associate your website or app with beacons, to provide low-priority notifications when scanned by devices that are nearby.</p> |
| | <h3>Sleep API</h3> <p>Determine when the user goes to sleep and wakes up, to help users understand their sleep habits.</p> | |

I) Using (built-in) sensors

Reading & resources:

- Sensors framework: [slides for #3.1 and #3.2](#) in Android Advanced Course
- Concepts, API and best practices to use [Sensors](#); [return values](#).

Hands-on exercises :

- complete the "[Getting Started with CameraX](#)" [up to (including) step 5- image capture use case]
- Optional: Create a [motion sensors](#) demonstrator: read and display, in real time, information from motion sensors.

Lab #5: Extending the mobile app with AI

J) Extending the app with AI

You can extend your mobile application with AI. A common use case is taking advantage of the device camera for image recognition/classification.

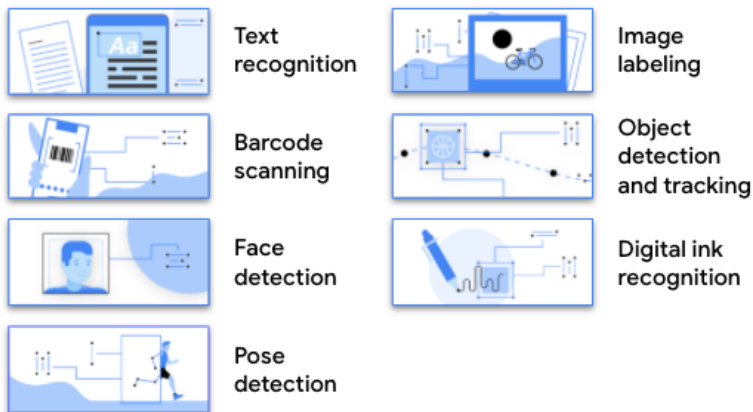
Google offers the [ML Kit mobile SDK](#) that brings Google's machine learning expertise to Android and iOS apps. Includes Vision + Natural Language APIs to solve common challenges in your apps, using (already made) models available from Google. The **ML Kit APIs run on-device**, allowing for real-time use cases (some updates may depend on the cloud).

For some use cases, instead of the generic ready-to-use models, you may want to **develop your**

own classification model. The [TensorFlow Lite](#) framework makes it easy to apply ML in your app. Works offline and you do not need to exchange data with the cloud.

- There is a [gallery of sample applications](#) that illustrate the use of TensorFlow Lite in Android
- There is an open collaborative space to share Tensor Flow models: [TensorFlow hub](#)
- Tensor Flow models can be prepared in friendly environments, such as [Colab notebooks](#) and [TeachableMachine](#) (web based).

Vision



Natural Language



Hands-on exercises:

1/

Start by running the [demonstrator app for ML-Kit Vision support in Android](#).

Try different ML use cases: track your own face, detect your posture; or simply read this QR Code



Note: you may get the “Error: The apk for your currently selected variant (vision-quickstart-proguard.apk) is not signed. Please specify a signing configuration for this variant (proguard).”


To change the build variant Android Studio uses, select Build > Select Build Variant in the menu bar (set to Debug).



2/

 [Create simple app to read QR-codes](#) (alternative [tutorial](#))

3/

 Complete the coding lab [Detect objects in images](#) [to build a visual product search] with ML Kit. This covers how to use on-device object detection in ML Kit to detect objects in images. Note that you just use the available classifiers as a “turnkey” API (no need to train your ML model).

Explore:

- Google I/O 2021 update on [ML Kit: Turnkey APIs to use on-device ML in mobile apps](#).
- [AppIntro](#): a library to facilitate the development of animated on-boarding interactions.



Lab #6: Introduction to declarative layouts (Composable)

K) Declarative layouts with Jetpack Compose

Resources:

- [“Compose” quick tutorial](#).
- [Thinking in Compose](#): theory and principles [must read]
- [Jetpack Compose course](#) (from basics to advanced)
- elements for [designing layouts](#) in Compose

Hands-on exercises (from codelabs):

-  [Compose basics](#) (with video)
-  [Handling state in compose](#) and design patterns (with video)

Key points:

- Describe *what* you want in the UI using Kotlin → fully declarative framework.
- UI elements are [functions that emit UI](#), not Views (thus no findViewById). UI is built from [Layout composables](#) (such as Box, Column, Row, and BoxWithConstraints) and [Modifiers constraints](#) (such as aspectRatio, offset, padding, size, and wrapContentSize).
- Data → Composable functions → UI. Apps provide data into composable functions, so they emit the required UI.
- Composable function can execute frequently and in any order → should be “light” and idempotent.
- Recomposition: if your (observable data) changes, implicated Composable functions are re-executed with new data to update the Composition. Not all Composable functions are used, if their associated state was not affected. Unlike with views, widgets are not exposed as objects, so you can’t change their properties.
- State tracking: API support to schedule recompositions for composables that depend on a particular state (signal what to look for and react) → MutableState wrapper.
- Pattern “state hoisting”: Composables should be stateless, moving the state to “upper” scopes. The “upper” scope passes the required state (data) as parameter to the Composable as well as the callback functions to receive any events of interest → [unidirectional data flow](#).