

LAB MANUAL

**Course: CSC303 Mobile Application Development**



**Department of Computer Science**

**COMSATS University Islamabad, Abbottabad Campus**

Table of Contents

[LAB 01: Environment setup & understanding 5](#_Toc85293728)

[Objective 5](#_Toc85293729)

[Scope 5](#_Toc85293730)

[Useful Concepts 5](#_Toc85293731)

[Exercises 6](#_Toc85293732)

[Exercises 7](#_Toc85293733)

[LAB 02: Basic Application Development 9](#_Toc85293734)

[Objective 9](#_Toc85293735)

[Scope 9](#_Toc85293736)

[Useful Concepts 9](#_Toc85293737)

[Lab Tasks 9](#_Toc85293738)

[Exercises 9](#_Toc85293739)

[LAB 03: Basic UI components and widget 10](#_Toc85293740)

[Objective 10](#_Toc85293741)

[Scope 10](#_Toc85293742)

[Useful Concepts 10](#_Toc85293743)

[Lab Tasks 10](#_Toc85293744)

[Exercises 10](#_Toc85293745)

[LAB 04: Activity, Intent, and Intent filters 11](#_Toc85293746)

[Objective 11](#_Toc85293747)

[Scope 11](#_Toc85293748)

[Useful Concepts 11](#_Toc85293749)

[Lab Tasks 11](#_Toc85293750)

[Exercises 11](#_Toc85293751)

[LAB 05: UI Layouts and Advanced UI Components 12](#_Toc85293752)

[Objective 12](#_Toc85293753)

[Scope 12](#_Toc85293754)

[Useful Concepts 12](#_Toc85293755)

[Lab Tasks 12](#_Toc85293756)

[Exercises 12](#_Toc85293757)

[LAB 06: Sessional 1 Exam 13](#_Toc85293758)

[LAB 07: Fragments 14](#_Toc85293759)

[Objective 14](#_Toc85293760)

[Scope 14](#_Toc85293761)

[Useful Concepts 14](#_Toc85293762)

[Lab Tasks 14](#_Toc85293763)

[Exercises 14](#_Toc85293764)

[LAB 08: Application Security and Permissions 15](#_Toc85293765)

[Objective 15](#_Toc85293766)

[Scope 15](#_Toc85293767)

[Useful Concepts 15](#_Toc85293768)

[Lab Tasks 15](#_Toc85293769)

[Exercises 15](#_Toc85293770)

[LAB 9: Data Storage & Content Providers 16](#_Toc85293771)

[Objective 16](#_Toc85293772)

[Scope 17](#_Toc85293773)

[Useful Concepts 17](#_Toc85293774)

[Lab Tasks 34](#_Toc85293775)

[Exercises 34](#_Toc85293776)

[LAB 10: Multithreading 35](#_Toc85293777)

[Objective 35](#_Toc85293778)

[Scope 35](#_Toc85293779)

[Useful Concepts 35](#_Toc85293780)

[Lab Tasks 35](#_Toc85293781)

[Exercises 35](#_Toc85293782)

[LAB 11: Broadcast Receivers 36](#_Toc85293783)

[Objective 36](#_Toc85293784)

[Scope 36](#_Toc85293785)

[Useful Concepts 36](#_Toc85293786)

[Lab Tasks 36](#_Toc85293787)

[Exercises 36](#_Toc85293788)

[LAB 12: Sessional 2 Exam 37](#_Toc85293789)

[LAB 13: Services 38](#_Toc85293790)

[Objective 38](#_Toc85293791)

[Scope 38](#_Toc85293792)

[Useful Concepts 38](#_Toc85293793)

[Lab Tasks 38](#_Toc85293794)

[Exercises 38](#_Toc85293795)

[LAB 14 Sensors and Third-party APIs 39](#_Toc85293796)

[Objective 39](#_Toc85293797)

[Scope 39](#_Toc85293798)

[Useful Concepts 39](#_Toc85293799)

[Lab Tasks 39](#_Toc85293800)

[Exercises 39](#_Toc85293801)

[LAB 15 Cross-Platform Development 40](#_Toc85293802)

[Objective 40](#_Toc85293803)

[Scope 40](#_Toc85293804)

[Useful Concepts 40](#_Toc85293805)

[Lab Tasks 40](#_Toc85293806)

[Exercises 40](#_Toc85293807)

[LAB 16: Final Exam 41](#_Toc85293808)

[Objectives 41](#_Toc85293809)

# LAB 01: Environment setup & understanding

## Objective

The objective of this lab is to understand how you can attach additional responsibilities to an object dynamically via restructuring its representation using the Decorator design pattern. In second session of the lab we learn about Factory Method that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

## Scope

The scope of this lab activity is the student’s ability to:

* Restructure the representation of an object with the help of Decorator to attach additional responsibility dynamically to an object.
* Provides an interface for creating objects in a superclass and allow subclasses to alter the type of the object.

## Useful Concepts

### Decorator Implementation Guidelines

1. Make sure your business domain can be represented as a primary component with multiple optional layers over it.
2. Figure out what methods are common to both the primary component and the optional layers. Create a component interface and declare those methods there.
3. Create a concrete component class and define the base behavior in it.
4. Create a base decorator class. It should have a field for storing a reference to a wrapped object. The field should be declared with the component interface type to allow linking to concrete components as well as decorators. The base decorator must delegate all work to the wrapped object.
5. Make sure all classes implement the component interface.
6. Create concrete decorators by extending them from the base decorator. A concrete decorator must execute its behavior before or after the call to the parent method (which always delegates to the wrapped object).
7. The client code must be responsible for creating decorators and composing them in the way the client needs.

### Factory Method Implementation Guidelines

1. Make all products follow the same interface. This interface should declare methods that make sense in every product.
2. Add an empty factory method inside the creator class. The return type of the method should match the common product interface.
3. In the creator’s code find all references to product constructors. One by one, replace them with calls to the factory method, while extracting the product creation code into the factory method.
4. You might need to add a temporary parameter to the factory method to control the type of returned product.
5. At this point, the code of the factory method may look pretty ugly. It may have a large switch operator that picks which product class to instantiate. But don’t worry, we’ll fix it soon enough.
6. Now, create a set of creator subclasses for each type of product listed in the factory method. Override the factory method in the subclasses and extract the appropriate bits of construction code from the base method.
7. If there are too many product types and it doesn’t make sense to create subclasses for all of them, you can reuse the control parameter from the base class in subclasses.
8. For instance, imagine that you have the following hierarchy of classes: the base Mail class with a couple of subclasses: AirMail and GroundMail; the Transport classes are Plane, Truck and Train. While the AirMail class only uses Plane objects, GroundMail may work with both Truck and Train objects. You can create a new subclass (say TrainMail) to handle both cases, but there’s another option. The client code can pass an argument to the factory method of the GroundMail class to control which product it wants to receive.
9. If, after all of the extractions, the base factory method has become empty, you can make it abstract. If there’s something left, you can make it a default behavior of the method.

## Exercises

### Activity 1 (Remote teaching)

Login in to subexpert.com with your student account and then watch the following interactive videos that demonstrates the Decorator and Factory Method.

* For Decorator Pattern
  + https://www.subexpert.com/CourseLectures/OnTopic/Design-Patterns/Decoratoy
* For Factory Method Pattern
  + https://www.subexpert.com/CourseLectures/OnTopic/Design-Patterns/Factory-Method

**Note:** In case of physical labs this activity is optional but recommended.

### Activity 2

The text book motivational code example for window/scrolling scenario is available on the following link. You need to run the version in Java which is quite simple.

* + https://en.wikipedia.org/wiki/ Decorator\_pattern

### Activity 3

Run one another very simple example about the Decorator pattern related to decorating geometrical shapes from the following link:

* + https://www.tutorialspoint.com/design\_pattern/decorator\_pattern.htm

### Activity 4

Imagine that you’re working on a notification library which lets other programs notify their users about important events. You need to first focus on the problem in this context and then understand its solution using the Decorator pattern from the following link:

* + <https://refactoring.guru/design-patterns/decorator>

**For Factory Method**

### Activity 5

Run the example on

* + https://github.com/sshpk/dp\_fall20/tree/master/FactoryMethodFA20

### Activity 6

Also run the following examples of factory method:

* + [Design Patterns: Factory Method in Java (refactoring.guru)](https://refactoring.guru/design-patterns/factory-method/java/example)
  + [Factory method pattern - Wikipedia](https://en.wikipedia.org/wiki/Factory_method_pattern) Modify the C# example into Java

## Exercises

Update the example with following additions:

Home Work 1:

Modify the example in Activity 3 to convert the output to "Shape : Circle with Red color" when it is decorated and "Shape: Circle" when not decorated.

Home Work 2:

For example, in Activity 3, Add one more decorator ThickBorderDecorator, which will decorate the shape with thick border.

Home Work 3:

Print shapes without decoration, with red color, with thick border only and then with both red color and thick border decoration in the Demo class.

Home Work 4:

For Activity 5, Add one another TriangleGeometry class to provide the Triangle factory method.

Home Work 5:

Instead of using four classifiers i.e. (Geometry, SquareGeometry, CircleGeometry etc.), We can implement factory method with a single method with any polymorphic behavior by passing it the type of object we need and then the method returns the desired object. Provide this single factory method implementation for the same example.

. Home Work 6:

Think on how can you utilize the Factory Method in your final year project and provide its implementation. (Optional)

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 02: Basic Application Development

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 03: Basic UI components and widget

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 04: Activity, Intent, and Intent filters

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 05: UI Layouts and Advanced UI Components

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 06: Sessional 1 Exam

**Purpose**

The purpose of this lab is to conduct the first sessional exam based on the activities conducted so far.

**Tasks**

The tasks will be decided by the respective course instructor/lab tutor.

# LAB 07: Fragments

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 08: Application Security and Permissions

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 9: Data Storage & Content Providers

## Objective

The objective of this lab is to understand how you can use multiple storage options provided by the Android to manage your data.

#### ****File based Storage:****

Android uses a file system that's like disk-based file systems on other platforms. The system provides several options for you to save your app data:

**App-specific storage:** Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.

**Shared storage:** Store files that your app intends to share with other apps, including media, documents, and other files.

**Preferences:** Store private, primitive data in key-value pairs.

**Databases:** Store structured data in a private database using the Room persistence library.

#### Cloud based Storage:

Other than the file storage options android provides APIs to use cloud based storage using Firebase. Firebase offers two cloud-based, client-accessible database solutions that support real time data syncing:

**Realtime:** Database is Firebase's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in realtime.

**Cloud Firestore:** is Firebase's newest database for mobile app development. It builds on the successes of the Realtime Database with a new, more intuitive data model. Cloud Firestore also features richer, faster queries and scales further than the Realtime Database.

#### Content Providers:

A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data.

Typically, you work with content providers in one of two scenarios; you may want to implement code to access an existing content provider in another application, or you may want to create a new content provider in your application to share data with other applications.

## Scope

The scope of this lab activity is the student’s ability to:

* Understand what storage options are suitable in which scenarios.
* How to use file-based storage options.
* How to use Firebase Dashboard to manage real-time databases.
* How to use content providers.

## Useful Concepts

### Permission and access to external storage:

Android defines the following storage-related permissions: READ\_EXTERNAL\_STORAGE, WRITE\_EXTERNAL\_STORAGE, and MANAGE\_EXTERNAL\_STORAGE.

### App-specific storage usage guidelines:

The system provides the following locations for storing such app-specific files:

**Internal storage directories:** These directories include both a dedicated location for storing persistent files, and another location for storing cache data.

**External storage directories:** These directories include both a dedicated location for storing persistent files, and another location for storing cache data.

#### Access from Internal Storage:

For each app, the system provides directories within internal storage where an app can organize its files. One directory is designed for your app's persistent files, and another contains your app's cached files. Your app doesn't require any system permissions to read and write to files in these directories.

#### Access persistent files:

Your app's ordinary, persistent files reside in a directory that you can access using the filesDir property of a context object. The framework provides several methods to help you access and store files in this directory.

#### Access and store files:

You can use the File API to access and store files.

To help maintain your app's performance, don't open and close the same file multiple times.

The following code snippet demonstrates how to use the File API:



#### Store a file using a stream:

As an alternative to using the File API, you can call openFileOutput() to get a FileOutputStream that writes to a file within the filesDir directory.

The following code snippet shows how to write some text to a file:

A picture containing text

Description automatically generated

To allow other apps to access files stored in this directory within internal storage, use a FileProvider with the FLAG\_GRANT\_READ\_URI\_PERMISSION attribute.

#### Access a file using stream:

To read a file as a stream, use openFileInput():

Graphical user interface, text, application, email

Description automatically generated

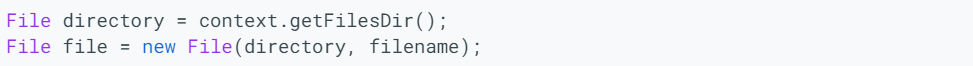
#### View list of files:

You can get an array containing the names of all files within the filesDir directory by calling fileList(), as shown in the following code snippet:



#### Create nested directories:

You can also create nested directories, or open an inner directory, by calling getDir() in Kotlin-based code or by passing the root directory and a new directory name into a File constructor in Java-based code:



#### Create cache files:

To create a cached file, call File.createTempFile():



Your app accesses a file in this directory using the cacheDir property of a context object and the File API:



#### Remove cache files:

To remove a file from the cache directory within internal storage, use one of the following methods:

The delete() method on a File object that represents the file:



The deleteFile() method of the app's context, passing in the name of the file:



#### Access from external storage:

If internal storage doesn't provide enough space to store app-specific files, consider using external storage instead. The system provides directories within external storage where an app can organize files that provide value to the user only within your app. One directory is designed for your app's persistent files, and another contains your app's cached files.

#### Verify that storage is available:

You can query the volume's state by calling Environment.getExternalStorageState(). If the returned state is MEDIA\_MOUNTED, then you can read and write app-specific files within external storage. If it's MEDIA\_MOUNTED\_READ\_ONLY, you can only read these files.

For example, the following methods are useful to determine the storage availability:

Graphical user interface, text, application, email

Description automatically generated

#### Select a physical storage location:

To access the different locations, call ContextCompat.getExternalFilesDirs().



#### Access persistent files:

The following code snippet demonstrates how to call getExternalFilesDir():



#### Create cache files:

To add an app-specific file to the cache within external storage, get a reference to the externalCacheDir:



#### Remove cache files:

To remove a file from the external cache directory, use the delete() method on a File object that represents the file:



#### Media Content:

If your app works with media files that provide value to the user only within your app, it's best to store them in app-specific directories within external storage, as demonstrated in the following code snippet:

Graphical user interface, text, application

Description automatically generated

#### Example Codes:

[Ibtisam/InternalStorageExample (github.com)](https://github.com/Ibtisam/InternalStorageExample)

[Ibtisam/ExternalStorageExample (github.com)](https://github.com/Ibtisam/ExternalStorageExample)

### Preferences:

Interface for accessing and modifying preference data returned by Context.getSharedPreferences(String, int). For any particular set of preferences, there is a single instance of this class that all clients share.

If you have a relatively small collection of key-values that you'd like to save, you should use the SharedPreferences APIs. A SharedPreferences object points to a file containing key-value pairs and provides simple methods to read and write them. Each SharedPreferences file is managed by the framework and can be private or shared.

#### Get a handle to shared preferences:

You can create a new shared preference file or access an existing one by calling one of these methods: getSharedPreferences() OR getPreferences()

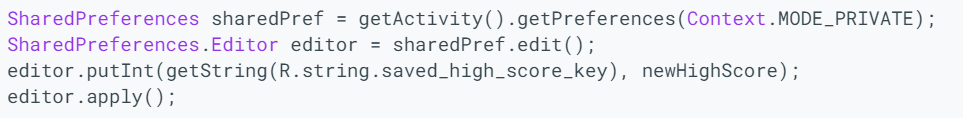




#### Write to shared preferences:

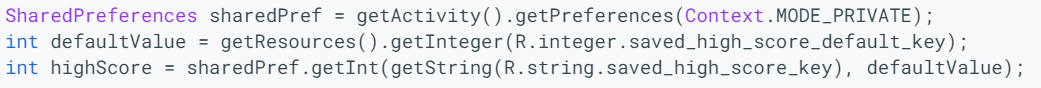
To write to a shared preferences file, create a SharedPreferences.Editor by calling edit() on your SharedPreferences.

Pass the keys and values you want to write with methods such as putInt() and putString(). Then call apply() or commit() to save the changes. For example:



#### Read from shared preferences:

To retrieve values from a shared preferences file, call methods such as getInt() and getString(), providing the key for the value you want, and optionally a default value to return if the key isn't present. For example:



#### Example Code:

[Ibtisam/SharedPreferencesExample (github.com)](https://github.com/Ibtisam/SharedPreferencesExample)

### Databases:

Apps that handle non-trivial amounts of structured data can benefit greatly from persisting that data locally. The most common use case is to cache relevant pieces of data so that when the device cannot access the network, the user can still browse that content while they are offline.

#### Save data in local database using Room API:

The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.

#### Setup:

To use Room in your app, add the following dependencies to your app's build.gradle file:

Text

Description automatically generated

#### Primary Components:

There are three major components in Room:

* The database class that holds the database and serves as the main access point for the underlying connection to your app's persisted data.
* Data entities that represent tables in your app's database.
* Data access objects (DAOs) that provide methods that your app can use to query, update, insert, and delete data in the database.

Graphical user interface, text, application, chat or text message

Description automatically generated

#### Sample Implementation:

This section presents an example implementation of a Room database with a single data entity and a single DAO.

#### Data Entry:

The following code defines a User data entity. Each instance of User represents a row in a user table in the app's database.

Graphical user interface, text, application

Description automatically generated

#### Data Access Object (DAO):

The following code defines a DAO called UserDao. UserDao provides the methods that the rest of the app uses to interact with data in the user table.

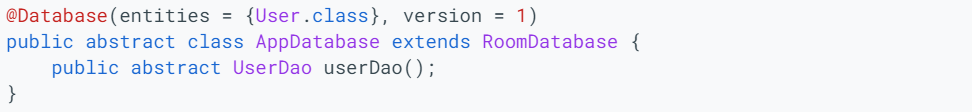
Graphical user interface, text, application

Description automatically generated

#### Database:

The following code defines an AppDatabase class to hold the database. AppDatabase defines the database configuration and serves as the app's main access point to the persisted data. The database class must satisfy the following conditions:

* The class must be annotated with a @Database annotation that includes an entities array that lists all of the data entities associated with the database.
* The class must be an abstract class that extends RoomDatabase.
* For each DAO class that is associated with the database, the database class must define an abstract method that has zero arguments and returns an instance of the DAO class.



#### Usage:

After you have defined the data entity, the DAO, and the database object, you can use the following code to create an instance of the database:



You can then use the abstract methods from the AppDatabase to get an instance of the DAO. In turn, you can use the methods from the DAO instance to interact with the database:



#### Code Example:

[Ibtisam/RoomAPIExample (github.com)](https://github.com/Ibtisam/RoomAPIExample)

### Cloud based Storage – Firebase:

Firebase provides several services for cloud-based storage like Realtime database and Firestore.

#### Add Firebase to your Android Project:

Adding Firebase to your app involves tasks both in the [Firebase console](https://console.firebase.google.com/) and in your open Android project (for example, you download Firebase config files from the console, then move them into your Android project).

#### Step 1: Create a Firebase Project:

1. In the Firebase console, click Add project.
   1. To add Firebase resources to an existing Google Cloud project, enter its project name or select it from the dropdown menu.
   2. To create a new project, enter the desired project name. You can also optionally edit the project ID displayed below the project name.
2. If prompted, review and accept the Firebase terms.
3. Click Continue.
4. Click Create project (or Add Firebase, if you're using an existing Google Cloud project).

Firebase automatically provisions resources for your Firebase project. When the process completes, you'll be taken to the overview page for your Firebase project in the Firebase console.

#### Step 2: Register your app with Firebase:

To use Firebase in your Android app, you need to register your app with your Firebase project. Registering your app is often called "adding" your app to your project.

1. Go to the Firebase console.
2. In the center of the project overview page, click the Android icon () or Add app to launch the setup workflow.
3. Enter your app's package name in the Android package name field.
4. Click Register

#### Step 3: Add a Firebase configuration file:

1. Add the Firebase Android configuration file to your app:
   1. Click **Download google-services.json** to obtain your Firebase Android config file (google-services.json).
   2. Move your config file into the module (app-level) directory of your app.

**Notes:**

* The Firebase config file contains unique, but non-secret identifiers for your project. To learn more about this config file, visit Understand Firebase Projects.
* You can download your Firebase config file again at any time.
* Make sure the config file name is not appended with additional characters, like (2).

1. To enable Firebase products in your app, add the google-services plugin to your Gradle files.
   1. In your root-level (project-level) Gradle file (build.gradle), add rules to include the Google Services Gradle plugin. Check that you have Google's Maven repository, as well.

Graphical user interface, text, application, email

Description automatically generated

* 1. In your module (app-level) Gradle file (usually app/build.gradle), apply the Google Services Gradle plugin:

Graphical user interface, text

Description automatically generated with medium confidence

#### Step 4: Add Firebase SDKs to your app:

1. Using the Firebase Android BoM, declare the dependencies for the Firebase products that you want to use in your app. Declare them in your module (app-level) Gradle file (usually app/build.gradle).

Graphical user interface, text, application

Description automatically generated

1. Sync your app to ensure that all dependencies have the necessary versions.

#### Create a Realtime Database:

1. Navigate to the Realtime Database section of the Firebase console. You'll be prompted to select an existing Firebase project. Follow the database creation workflow.
2. Select a starting mode for your Firebase Security Rules:
   1. Test Mode: Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data. After testing, make sure to review the Understand Firebase Realtime Database Rules section.
   2. Locked Mode: Denies all reads and writes from mobile and web clients. Your authenticated application servers can still access your database.
3. Choose a region for the database.
4. Click Done.

When you enable Realtime Database, it also enables the API in the Cloud API Manager.

#### Add Realtime database SDK to your app:

Using the Firebase Android BoM, declare the dependency for the Realtime Database Android library in your module (app-level) Gradle file (usually app/build.gradle).

Text

Description automatically generated

#### Configure Realtime database rules:

The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to.

#### Write to your Database:

Retrieve an instance of your database using getInstance() and reference the location you want to write to.

Graphical user interface, text, application, email

Description automatically generated

You can save a range of data types to the database this way, including Java objects. When you save an object the responses from any getters will be saved as children of this location.

#### Read from your Database:

Graphical user interface, text, application

Description automatically generated

#### Important Links:

[Installation & Setup on Android  |  Firebase Documentation (google.com)](https://firebase.google.com/docs/database/android/start)

#### Create a Cloud Firestore Database:

1. If you haven't already, create a Firebase project: In the Firebase console, click Add project, then follow the on-screen instructions to create a Firebase project or to add Firebase services to an existing GCP project.
2. Navigate to the Cloud Firestore section of the Firebase console. You'll be prompted to select an existing Firebase project. Follow the database creation workflow.
3. Select a starting mode for your Cloud Firestore Security Rules:
   1. Test mode: Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data. After testing, make sure to review the Secure your data section.
   2. Locked mode: Denies all reads and writes from mobile and web clients. Your authenticated application servers (C#, Go, Java, Node.js, PHP, Python, or Ruby) can still access your database.
4. Select a location for your database.
5. Click Done.

#### Setup your development environment:

Add the required dependencies and client libraries to your app.

#### Text Description automatically generated

#### Initialize Cloud Firestore:

Initialize an instance of Cloud Firestore:

Text

Description automatically generated with low confidence

#### Add Data:

Cloud Firestore stores data in Documents, which are stored in Collections. Cloud Firestore creates collections and documents implicitly the first time you add data to the document. You do not need to explicitly create collections or documents.

Create a new collection and a document using the following example code.

Text, application

Description automatically generated

Now add another document to the users collection. Notice that this document includes a key-value pair (middle name) that does not appear in the first document. Documents in a collection can contain different sets of information.

Text

Description automatically generated

#### Read Data:

To quickly verify that you've added data to Cloud Firestore, use the data viewer in the Firebase console.

You can also use the "get" method to retrieve the entire collection.

Graphical user interface, text, application, email

Description automatically generated

#### Secure your data:

If you're using the Web, Android, or iOS SDK, use Firebase Authentication and Cloud Firestore Security Rules to secure your data in Cloud Firestore.

Here are some basic rule sets you can use to get started. You can modify your security rules in the Rules tab of the console.

Graphical user interface, text, application

Description automatically generated

#### Important Links:

[Getting Started With Cloud Firestore on Android - Firecasts - YouTube](https://www.youtube.com/watch?v=kDZYIhNkQoM)

[Get started with Cloud Firestore  |  Firebase Documentation (google.com)](https://firebase.google.com/docs/firestore/quickstart#android)

#### Add Firebase Authentication to your app:

Using the Firebase Android BoM, declare the dependency for the Firebase Authentication Android library in your module (app-level) Gradle file (usually app/build.gradle).

Text

Description automatically generated

To use an authentication provider, you need to enable it in the Firebase console. Go to the Sign-in Method page in the Firebase Authentication section to enable Email/Password sign-in and any other identity providers you want for your app.

#### Check Current Authentication state:

1. Declare an instance of FirebaseAuth.



1. In the onCreate() method, initialize the FirebaseAuth instance.



1. When initializing your Activity, check to see if the user is currently signed in.

Graphical user interface, text, application

Description automatically generated

#### Signup new user:

Create a new createAccount method that takes in an email address and password, validates them, and then creates a new user with the createUserWithEmailAndPassword method.

Graphical user interface, text, application

Description automatically generated

Add a form to register new users with their email and password and call this new method when it is submitted.

#### Sign in existing users:

Create a new signIn method which takes in an email address and password, validates them, and then signs a user in with the signInWithEmailAndPassword method.

**Graphical user interface, text, application

Description automatically generated**

#### Access User Authentication:

If a user has signed in successfully you can get their account data at any point with the getCurrentUser method.

Graphical user interface, text, application

Description automatically generated

#### Code Example:

[quickstart-android/auth/app/src/main at master · firebase/quickstart-android (github.com)](https://github.com/firebase/quickstart-android/tree/master/auth/app/src/main)

#### Important Links:

[Authenticate Using Google Sign-In on Android  |  Firebase Documentation](https://firebase.google.com/docs/auth/android/google-signin)

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 10: Multithreading

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

* Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 11: Broadcast Receivers

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

* Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 12: Sessional 2 Exam

**Objectives**

The purpose of this lab is to conduct the second sessional exam based on the activities conducted so far.

**Tasks**

The tasks will be decided by the respective course instructor/lab tutor.

# LAB 13: Services

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 14 Sensors and Third-party APIs

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 15 Cross-Platform Development

## Objective

## Scope

## Useful Concepts

## Lab Tasks

### Activity 1

### Activity 2

## Exercises

### Assignment Deliverables

Create a one-minute video demonstration of your home activities, upload it to a cloud storage and then share the link with your class teacher.

# LAB 16: Final Exam

# Objectives

The purpose of this lab is to conduct the final exam based on the activities conducted throughout the semester.

**Lab Tasks**

The tasks will be decided by the respective course instructor/lab tutor.