Rakamin Academy

mandiri

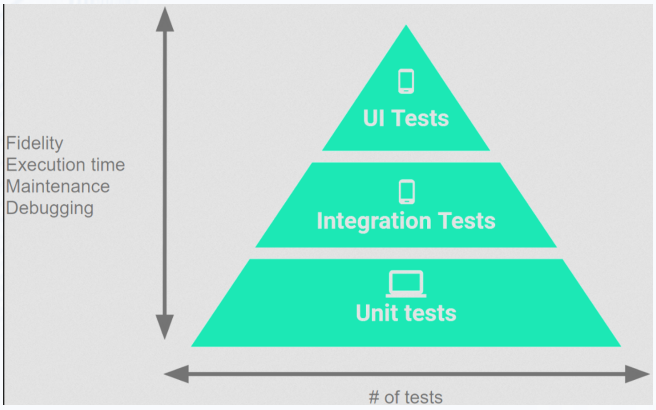
Project Based Internship

Unit Testing

Principles of Unit Testing, Unit Testing Tools, and Unit Testing Instrumentation

Daftar Isi

A. Principles of Unit Testing



Piramida Pengujian, yang ditunjukkan pada gambar di atas mengilustrasikan bagaimana aplikasi yang dikembangkan harus menyertakan tiga kategori pengujian: kecil, sedang, dan besar:

- Pengujian kecil (Unit Tests) adalah pengujian unit yang memvalidasi perilaku aplikasi, satu class pada satu waktu.
- Pengujian sedang (Integration Tests) adalah pengujian integrasi yang memvalidasi interaksi antara level tumpukan dalam modul, atau interaksi antara modul terkait.
- Pengujian besar (UI Tests) adalah pengujian end-to-end yang memvalidasi perjalanan user yang mencakup beberapa modul aplikasi.

Saat kita membuat piramida, dari pengujian kecil hingga pengujian besar, setiap pengujian tidak hanya meningkat fidelitasnya, tetapi juga meningkat waktu eksekusi dan upaya untuk mengelola dan men-debug. Oleh karena itu, kita harus menulis lebih banyak pengujian unit daripada pengujian integrasi, dan lebih banyak pengujian integrasi daripada pengujian end-to-end.

Unit testing adalah sebuah langkah pengujian terhadap perangkat lunak atau komponen dari sebuah perangkat lunak. Biasanya, unit testing dilakukan

disaat masa development atau pengembangan dari sebuah aplikasi yang dilakukan oleh developer. Pengujian unit testing ini meliputi dari function, method, procedure, module, serta object.

Tujuan dari unit testing adalah untuk menguji serta memastikan kalau kode-kode yang kita buat sudah berjalan sesuai spesifikasi nya serta mendeteksi jika terdapat bug dari sebuah program. Sedangkan manfaat dari Unit Testing sendiri adalah membantu mengurangi error di sebuah program dan membantu kita menguji, apakah program yang kita buat sudah layak dirilis ke pasar atau tidak.

B. Unit Testing Tools

Ada beberapa tools otomatis yang tersedia untuk melakukan unit testing pada pengembangan aplikasi Android, diantaranya :

1. **JUnit**: Junit adalah tool gratis untuk pengujian pada bahasa pemrograman java. Junit menyediakan assertions untuk mengidentifikasi test method.
2. **Mockito** : Mockito merupakan mocking framework yang dapat digunakan secara gratis oleh semua developer yang membutuhkan sebuah mocking framework. Mockito ini bekerja dengan memberikan nilai balikan true pada setiap kelas yang di mock. Sehingga tidak peduli apakah kelas tersebut berfungsi atau tidak, akan dianggap tetap berjalan semestinya.
3. **MockK** : MockK merupakan mocking framework yang dibuat untuk Kotlin. Seperti Mockito, Mockk memungkinkan kita membuat dan mematikan objek dalam kode pengujian. Mockito adalah framework yang sering digunakan oleh developer Java dan sangat powerful. Tapi memang ada

beberapa gangguan saat digunakan dengan Kotlin. Mockk, yang dirancang khusus untuk Kotlin dengan pengalaman yang lebih menyenangkan.

Sementara tools unit testing pada pengembangan aplikasi iOS yang biasanya digunakan oleh developer iOS adalah XCTest. XCTest merupakan official testing tools dari XCode yang dimana dikhususkan untuk IOS development, dan test juga ditulis menggunakan bahasa yang sama dengan IOS Development, yaitu Obj-C atau Swift.

C. Unit Testing Instrumentation

Untuk melakukan unit testing, pengembang menulis code untuk menguji sebuah function yang spesifik dalam perangkat lunak. Pengembang pada umumnya menggunakan UnitTest Framework untuk mengembangkan pengujian otomatis untuk unit testing.

Unit testing memiliki dua tipe yaitu manual dan automated. Unit testing biasanya automated, meski demikian unit testing masih dapat dilakukan secara manual.

Teknik unit testing dikategorikan menjadi tiga bagian, yaitu Black box testing yang menguji user interface, beserta input dan output, White box testing yang menguji functional behaviour dari perangkat lunak, dan Gray box testing yang digunakan untuk menjalankan test suites, test methods, test cases, dan melakukan risk analysis.

Sekarang mari kita membahas karakteristik pengujian unit yang baik. Prinsip-prinsip pengujian unit yang baik yaitu sebagai berikut :

1. **Sederhana untuk ditulis**

   Developer biasanya melakukan pendekatan dengan sejumlah pengujian unit untuk memenuhi beberapa kasus dan bagian dari kinerja aplikasi, jadi seharusnya jelas untuk mengkodekan semua rutin pengujian tanpa usaha besar.

2. **Dapat dibaca**

   Tujuan unit test harus transparan. Tes unit yang baik menggambarkan cerita tentang beberapa karakteristik perilaku aplikasi kita. Jadi seharusnya hanya mengetahui keadaan mana yang sedang diuji dan jika pengujian gagal, mudah untuk menemukan cara mengatasi masalah tersebut. Dengan pengujian unit konvensional, kita dapat membuat bug tanpa benar-benar men-debug kode.

3. **Dapat dikitalkan**

   Tes unit harus gagal terutama jika ada bug dalam sistem di bawah tes. Kelihatannya cukup sederhana, tetapi developer sering mengalami masalah saat pengujian mereka mencapai titik impas saat tidak ada bug yang disisipkan. Misalnya, pengujian mungkin memenuhi syarat saat berangkat satu per satu, tetapi gagal saat mengoperasikan seluruh rangkaian pengujian, atau meneruskan mesin komunitas dan tidak menyukai server integrasi konstan. Kondisi ini merupakan karakteristik dari ketidaksempurnaan desain. Tes unit yang baik harus dapat direproduksi dan otonom dari faktor luar seperti latar belakang atau urutan berjalan.

4. **Cepat**

   Developer mengatasi pengujian unit sehingga mereka dapat menjalankannya secara teratur dan memeriksa apakah tidak ada

kekurangan yang ditambahkan. Jika pengujian unit ditunda, pengembang lebih cenderung untuk langsung menjalankannya di mesin mereka sendiri. Satu tes lambat tidak akan membuat perbedaan yang berarti; tambahkan seribu lagi dan kita pasti akan menunggu sebentar. Tes unit yang lebih lambat dapat menyebabkan ketergantungan pada lingkungan.

D. Unit Testing in Android

Kode yang harus diuji bergantung pada faktor-faktor seperti jenis aplikasi, tim pengembangan, jumlah kode lama, dan arsitektur yang digunakan. Bagian berikut menguraikan apa yang mungkin ingin dipertimbangkan oleh pemula saat merencanakan apa yang akan diuji di aplikasi mereka.

Organisasi direktori pengujian

Proyek tipikal di Android Studio berisi dua direktori yang menyediakan pengujian bergantung dengan lingkungan eksekusinya. Atur pengujian kita di direktori berikut seperti yang dijelaskan:

- Direktori androidTest harus berisi pengujian yang dijalankan pada perangkat nyata atau virtual. Pengujian tersebut mencakup pengujian integrasi, pengujian end-to-end, dan pengujian lainnya di mana JVM saja tidak dapat memvalidasi fungsionalitas aplikasi kita.
- Direktori pengujian harus berisi pengujian yang berjalan di mesin lokal kita, seperti pengujian unit (unit testing). Berbeda dengan yang di atas, ini bisa berupa pengujian yang berjalan di JVM lokal.

Unit Tests Essential

Saat mengimplementasi best practice, kita harus memastikan bahwa kita menggunakan pengujian unit dalam kasus berikut:

1. Pengujian unit untuk ViewModels, atau Presenters.
2. Pengujian unit untuk data layer, khususnya repository. Sebagian besar data layer harus platform-independent. Melakukan hal itu memungkinkan pengujian gkita untuk menggantikan modul database dan sumber data jarak jauh dalam pengujian.
3. Pengujian unit untuk platform-independen layer lainnya seperti Domain layer, seperti use cases dan interactors.
4. Tes unit untuk kelas utilitas seperti manipulasi string dan perhitungan matematika.

Menguji Kasus Tepi (Edge Cases)

Tes unit harus fokus pada kasus normal dan edge. Kasus tepi adalah skenario yang tidak biasa yang tidak mungkin ditangkap oleh penguji manusia dan pengujian yang lebih besar. Contohnya termasuk yang berikut:

- Operasi matematika menggunakan bilangan negatif, nol, dan syarat batas. Semua kemungkinan kesalahan koneksi jaringan.
- Data rusak, seperti format JSON yang salah.
- Mensimulasikan penyimpanan penuh saat menyimpan ke file.
- Objek dibuat ulang di tengah proses (seperti aktivitas saat perangkat diputar).

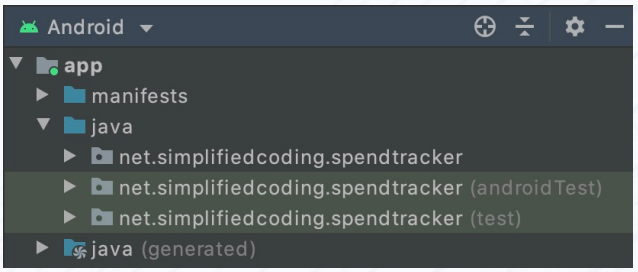
Tes Unit yang Harus Dihindari

Beberapa tes unit harus dihindari karena nilainya yang rendah:

- Pengujian yang memverifikasi operasi kerangka kerja atau pustaka yang benar, bukan kode yang kita buat.
- Entry points framework seperti activity, fragment, atau service tidak boleh memiliki logika bisnis sehingga pengujian unit tidak boleh menjadi prioritas. Tes unit untuk aktivitas memiliki nilai kecil, karena sebagian besar akan mencakup kode kerangka kerja dan memerlukan penyiapan yang lebih rumit. Pengujian berinstrumen seperti pengujian UI dapat mencakup kelas-kelas ini.

E. Implementasi Unit Testing di Android dengan JUnit4

Sekarang mari kita bahas topik utama kita yaitu Android Unit Test. Setiap kali kita membuat proyek Android, kita akan melihat ketiga paket ini.



Kita dapat melihat bahwa kita memiliki main package (yang pertama), dan di dalam package ini, kita memasukkan semua kode aplikasi kita. Tetapi dengan package ini, kita memiliki dua package lagi. Kita dapat membedakan packages ini dengan petunjuk yang ditampilkan androidTest & test. kita tidak akan membahas androidTest sekarang. Tujuan kita hari ini hanyalah package

uji coba (test). Di dalam package ini, kita menulis semua kode untuk Unit Test. Sekarang buka file build.gradle level aplikasi dan lihat blok dependensi.

```
dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'

    testImplementation 'junit:junit:4.13.2'

    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```

Kita dapat melihat di sini kita memiliki testImplementation dan androidTestImplementation . testImplementation adalah library yang tersedia di dalam test package. Dan dengan cara yang sama jika kita ingin library tersedia di dalam paket androidTest, kita perlu menggunakan androidTestImplementation.

Sekarang, frametowk unit test JUnit ditambahkan secara default di sini. Setiap kali kita membuat proyek apa pun, itu ditambahkan secara default. Kita akan menggunakan JUnit untuk menguji unit kode kita.

Tetapi dengan JUnit kita akan menggunakan Google Truth, untuk menyederhanakan assertions. Karena tidak tersedia secara default; kita perlu menambahkannya di dalam blok dependensi file build.gradle level aplikasi.

```
testImplementation "com.google.truth:truth:1.1.2"
```

Kode yang akan Dilakukan Unit Testing

Untuk contoh ini, kita memiliki fungsi yang sangat sederhana yang akan kita uji unit.

```
object Validator {
    fun validateInput(amount: Int, desc: String): Boolean {
        return !(amount <= 0 || desc.isEmpty())
    }
}
```

Ini adalah kode yang sangat sederhana. Fungsi ini akan memvalidasi parameter yang diberikan. Aturannya adalah sebagai berikut :

1. Jumlah yang diberikan harus lebih besar dari nol, dan.
2. Desc yang diberikan tidak boleh kosong.

Jika parameter yang diberikan memenuhi syarat, kita akan mengembalikan true, yang berarti input valid. Untuk kasus lain, kita akan mengembalikan false, yang berarti input yang diberikan tidak valid.

Sekarang kita perlu menguji fungsi ini untuk memastikan fungsi ini berfungsi seperti yang diharapkan. Dan untuk menguji fungsi ini, kita akan membuat file pengujian di dalam paket pengujian. kita dapat membuat file secara manual, atau kita dapat mengikuti langkah-langkah ini.

1. Klik kanan pada kelas yang perlu diuji.
2. Klik menu Generate dan kemudian pilih Test.
3. Pilih JUnit4 dan tekan ok.

Pembuatan Unit Test

Sekarang akhirnya kita akan menguji unit fungsi validateInput kita.

```
@RunWith(JUnit4::class)
class ValidatorTest{

    @Test
    fun whenInputIsValid(){
        val amount = 100
        val desc = "Some random desc"
        val result = Validator.validateInput(amount, desc)
        assertThat(result).isEqualTo(true)
    }

    @Test
    fun whenInputIsInvalid(){
        val amount = 0
        val desc = ""
        val result = Validator.validateInput(amount, desc)
        assertThat(result).isEqualTo(false)
    }

}
```

Sekarang di sini kita memiliki kelas; yang akan berisi semua fungsi untuk menguji Validator . Setiap kelas pengujian unit perlu dianotasi dengan @RunWith(JUnit4::class) (JUnit4::class karena kita menggunakan JUnit4, kita dapat mengubahnya dengan versi apa pun yang ingin kita gunakan). Di dalam kelas ini, kita akan menulis fungsi pengujian untuk menguji semua skenario. Karena ini hanyalah sebuah contoh, kita hanya memiliki dua test case. Satu untuk input yang valid, dan satu lagi untuk input yang tidak valid.

Hal lain yang perlu diperhatikan adalah, kita perlu memberi anotasi pada semua fungsi pengujian kita dengan @Test . Sekarang di sini kita dapat melihat saya memiliki dua fungsi.

1. whenInputIsValid() : fungsi ini untuk menguji kasus, ketika kita akan memberikan input yang valid ke fungsi kita. Jadi yang kita lakukan di sini adalah, kita hanya memanggil fungsi validateInput() dan mengirimkan

input yang valid. Sekarang kita menegaskan bahwa nilai yang dikembalikan adalah benar.

2. whenInputIsInvalid() : di sini juga kita melakukan hal yang sama seperti di atas; satu-satunya perbedaan adalah kita memberikan input yang tidak valid kali ini dan kita menyatakan bahwa hasilnya sama dengan salah.

Untuk menjalankan pengujian, kita dapat menggunakan tombol play di Android Studio. kita akan melihat tombol play sebelum setiap fungsi dan kelas test.

Jalankan saja tesnya, dan kita akan melihat bahwa test berhasil dan sesuai. Jika pengujian gagal, kita perlu melakukan perubahan pada fungsi kita. Jadi setiap kali kita mengubah sesuatu dalam fungsi, kita perlu mengujinya lagi. Dan dengan cara ini, kode kita tidak akan rusak dengan perubahan di masa mendatang karena harus selalu lulus ujian terlebih dahulu.

References

https://codepolitan.com/blog/apa-itu-unit-testing-yuk-kenalan

https://developer.android.com/training/testing/fundamentals

https://muklasr.medium.com/unit-testing-45fbf725e780

https://medium.com/@mlutern/mobile-testing-tools-8c77de14569

https://www.iroidtechnologies.com/blog/unit-testing-principles

https://www.simplifiedcoding.net/android-unit-test-tutorial/