Testing

Testing (Parte I)

{desafío} latam_

Implementar tests unitarios y de instrumentación para la verificación del buen funcionamiento de los componentes de un proyecto Android.

- Unidad 1:
  Acceso a datos en Android
- Unidad 2:
  Consumo de API REST
- Unidad 3:
  Testing
- Unidad 4:
  Distribución del aplicativo Android

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

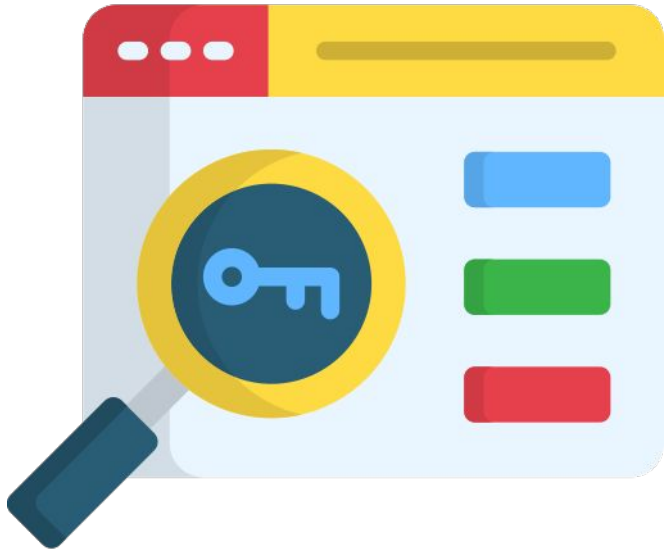
- *Testing*

¿Cómo crees que se puede probar una tabla en una base de datos?

/* Anotaciones */

Anotaciones

Las anotaciones en las pruebas de Android son palabras clave especiales que se utilizan para proporcionar información adicional al marco de prueba. Se utilizan para marcar métodos de prueba, especificar opciones de configuración y realizar otras tareas relacionadas con las pruebas.



Algunas de las anotaciones más utilizadas en las pruebas de Android son:

Anotaciones más utilizadas

- **@Test:** esta anotación se utiliza para marcar un método de prueba. El marco de prueba ejecutará todos los métodos que están anotados con @Test.

- **@Before:** esta anotación se usa para marcar un método que debe ejecutarse antes de cada método de prueba. Por lo general, se usa para configurar cualquier objeto o dato que se necesite para las pruebas.

- **@After:** esta anotación se usa para marcar un método que debe ejecutarse después de cada método de prueba. Por lo general, se usa para limpiar cualquier objeto o dato que se haya creado durante las pruebas.

Anotaciones más utilizadas

- **@RunWith:** esta anotación se usa para especificar el corredor de prueba que se debe usar para ejecutar las pruebas. Hay varios ejecutores de pruebas disponibles para Android, como AndroidJUnitRunner y RobolectricTestRunner.

- **@Rule:** esta anotación se utiliza para especificar reglas de prueba que deben aplicarse al método de prueba. Las reglas de prueba se utilizan para personalizar el comportamiento del marco de prueba, como deshabilitar animaciones o simular objetos.

- **@Ignore:** esta anotación se usa para deshabilitar temporalmente un método de prueba. El marco de prueba omitirá cualquier método que esté anotado con @Ignore.

/* Truth(Assert library) */

Truth

Es una biblioteca de aserciones popular para Java y Kotlin que está diseñada para facilitar la escritura de pruebas legibles y mantenibles. Estas son algunas de las principales características:

- **Sintaxis fluida y expresiva:** Truth proporciona una sintaxis fluida y expresiva para escribir afirmaciones que se leen como una oración de lenguaje natural. Esto facilita la escritura y la comprensión de las pruebas y puede ayudar a reducir los errores y mejorar la capacidad de mantenimiento.

- **Amplio conjunto de comparadores integrados:** Truth viene con un amplio conjunto de comparadores integrados que le permiten comparar fácilmente objetos, colecciones y otros tipos de datos. Estos comparadores están diseñados para ser altamente personalizables y flexibles, por lo que puede escribir pruebas que se adapten a sus necesidades específicas.

Truth

- **API extensible:** Truth proporciona una API extensible que le permite crear comparadores personalizados y objetos asertivos, que se pueden usar para encapsular una lógica de aserción compleja y hacer que sus pruebas sean más expresivas y modulares.

- **Falla rápida:** Truth utiliza un enfoque de prueba de falla rápida, lo que significa que deja de evaluar una prueba tan pronto como encuentra la primera falla de afirmación. Esto puede ayudar a acelerar la ejecución de la prueba y reducir el ruido en la salida de la prueba.

- **Integración con marcos de prueba:** Truth se integra bien con marcos de prueba populares como JUnit y TestNG, y se puede usar con otros marcos de prueba que admitan bibliotecas de aserción personalizadas.

Truth - Ejemplo

```
String string = "awesome";

assertThat(string).startsWith("awe");

assertWithMessage("Without me, it's just aweso")
    .that(string)
    .contains("me");
```

Otro ejemplo:

```
Iterable<Color> googleColors = googleLogo.getColors();

assertThat(googleColors)
    .containsExactly(BLUE, RED, YELLOW, BLUE, GREEN, RED)
    .inOrder();
```

/* Test de integración */

Test de integración

Las pruebas de integración en Android son un tipo de prueba que se enfoca en verificar que las diferentes partes de una aplicación funcionan correctamente cuando se integran entre sí. Este tipo de prueba generalmente se realiza después de los Unit Test y antes de que la aplicación se lance a producción.

Las pruebas de integración implican probar las interacciones entre diferentes módulos, componentes y sistemas dentro de una aplicación. Esto puede incluir probar las interacciones entre la interfaz de usuario y la lógica de negocio, entre la aplicación y backend, o entre diferentes partes de la aplicación que dependen unas de otras.

Test de integración

Las pruebas de integración se pueden realizar manualmente o utilizando herramientas de prueba automatizadas. A menudo implica configurar entornos de prueba que se asemejan mucho al entorno de producción y ejecutar una serie de pruebas que simulan diferentes acciones y escenarios del usuario.

El objetivo de las pruebas de integración es identificar y abordar cualquier problema o error que pueda surgir de las interacciones entre las diferentes partes de la aplicación. Al realizar pruebas de integración, los desarrolladores pueden asegurarse de que la aplicación funcione según lo previsto y brinde una experiencia de usuario perfecta.

/* Room Test */

Room Testing - Ejemplo

¿Cómo probar una tabla en Room?

Primero debemos asegurarnos de tener las siguientes dependencias en nuestro proyecto:

```
dependencies {
    // Room
    implementation "androidx.room:room-runtime:$room_version"
    kapt "androidx.room:room-compiler:$room_version"

    // Testing
    testImplementation "androidx.arch.core:core-testing:$arch_version"
    testImplementation "org.mockito:mockito-core:$mockito_version"
}
```

Crea una base de datos de prueba utilizando una base de datos en memoria, que le permite probar sus consultas sin datos persistentes:

```
@RunWith(AndroidJUnit4::class)
class MyRoomDatabaseTest {

    private lateinit var database: MyRoomDatabase

    @Before
    fun setup() {
        database = Room.inMemoryDatabaseBuilder(
            InstrumentationRegistry.getInstrumentation().context,
            MyRoomDatabase::class.java
        ).build()
    }

    @After
    fun tearDown() {
        database.close()
    }

    // A continuación puedes testear tus queries
}
```

Escribe las pruebas para tus consultas de Room. Por ejemplo, puede probar una consulta simple como esta:

```
@Test
fun testInsertAndRetrieveUser() {
    // Crea un usuario
    val user = User("John", "Doe")

    // Inserta el usuario en la base de datos
    database.userDao().insert(user)

    // Trea el usuario recién insertado
    val retrievedUser = database.userDao().getUser(user.id)

    // Verifica que el usuario que insertaste sea el mismo que el que viene desde la db
    assertEquals(user, retrievedUser)
}
```

Room Testing - Explicación del ejemplo

- En este ejemplo, creamos un usuario, lo insertamos en la base de datos, lo recuperamos y luego verificamos que el usuario recuperado coincida con el usuario original.

- Al probar sus consultas de sala de esta manera, puede asegurarse de que funcionan como se esperaba y que su base de datos está configurada correctamente para almacenar y recuperar datos.

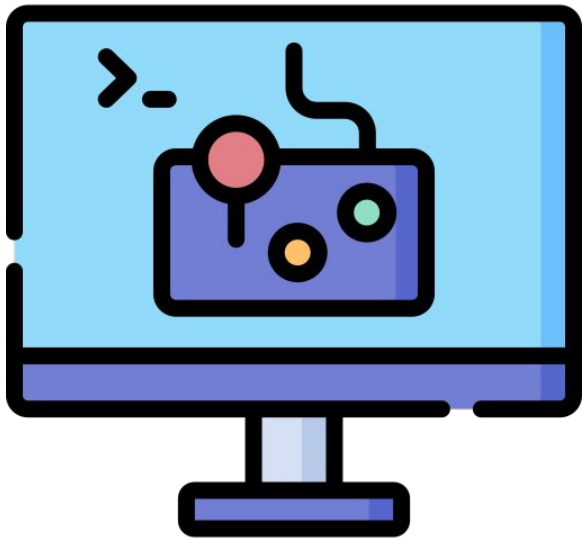
/* Espresso */

Espresso

Espresso es un marco de pruebas de interfaz de usuario para Android que se utiliza para escribir pruebas automatizadas para aplicaciones de Android. Proporciona un conjunto de API que permiten a los desarrolladores simular las interacciones del usuario con la aplicación y verificar que la aplicación se comporte correctamente en respuesta.

Espresso está diseñado para ser un marco de prueba ligero y fácil de usar que se puede integrar perfectamente en un proyecto de aplicación de Android. Proporciona una API fluida que permite escribir las pruebas de manera concisa y expresiva.

Espresso



Espresso es especialmente útil para probar los flujos de usuario y las interacciones de la interfaz de usuario en las aplicaciones de Android. Se puede usar para probar una amplia gama de elementos de la interfaz de usuario, como botones, campos de texto, vistas de lista y más. También admite funciones avanzadas como emparejadores personalizados e IdlingResources para manejar operaciones asincrónicas.

Las pruebas de espresso se ejecutan en el dispositivo o emulador, lo que las hace más confiables y precisas que las pruebas que se ejecutan en una máquina de escritorio. También brindan comentarios valiosos sobre el rendimiento y la usabilidad de una aplicación que se está probando.

/* Espresso IdlingResources */

Espresso IdlingResources

- Espresso IdlingResources es un mecanismo proporcionado por el marco de prueba de Espresso en Android para manejar operaciones asincrónicas durante las pruebas de IU.

- En Android, muchas operaciones, como las llamadas de red o las consultas a la base de datos, se ejecutan de forma asincrónica, y Espresso debe ser informado cuando tales operaciones están en curso para que pueda esperar a que se completen antes de realizar otras acciones. Si Espresso no está informado sobre el estado de estas operaciones asincrónicas, puede realizar acciones en la interfaz de usuario antes de que se completen las operaciones, lo que puede generar pruebas irregulares.

Espresso IdlingResources

- Para evitar esto, proporciona la interfaz IdlingResource, que permite a los desarrolladores declarar y administrar los recursos que están ocupados durante la ejecución de la prueba. Un IdlingResource simplemente representa una operación que está en curso y notifica a Espresso cuando se vuelve inactivo.

- Por ejemplo, si tiene una solicitud de red que necesita realizar durante una prueba de IU, puede crear un IdlingResource que monitoreará el estado de la solicitud y notificará a Espresso cuando se complete. Esto asegurará que Espresso no continúe ejecutando la prueba hasta que finalice la solicitud.

Espresso IdlingResources

Espresso proporciona un conjunto de IdlingResources preconstruidos para operaciones asíncronas comunes, como solicitudes de red o animaciones, pero también puede crear sus propios IdlingResources personalizados para otros tipos de operaciones asíncronas. Al administrar adecuadamente estos recursos, puede escribir pruebas de IU más confiables y sólidas.

De los tipos de tests mencionados anteriormente ¿cuál crees que es el más fácil de implementar?

Próxima sesión...

- *MockWebServer*
- *Roboelectric*
- *Cómo testear coroutinas*
- *Cómo testear ROOM y LiveData/StateFlow*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales