BACKEND

El **back end** es la parte de la aplicación que se encarga de toda la lógica para que la misma funcione. Algunas de las funciones que se gestionan en esta parte son:

* Las peticiones del **front end.**
* Lógica de negocio.
* Conexión con **bases de datos** (relacionales y no relacionales).
* Logueo de errores, para encontrar luego, más rápidamente las soluciones.
* Uso de librerías del servidor web, por ejemplo, para implementar temas de caché o para comprimir las imágenes de la web.
* La seguridad de los sitios web que gestiona
* Optimización de los recursos para que las páginas sean performantes.

Un **back end** debe ser capaz de tener una capa de servicios para que el **front end** pueda consumirla y así poder realizar peticiones. En el desarrollo de esta capa hay que conectarse a una base de datos y definir que le es permitido mostrar al **front end.**

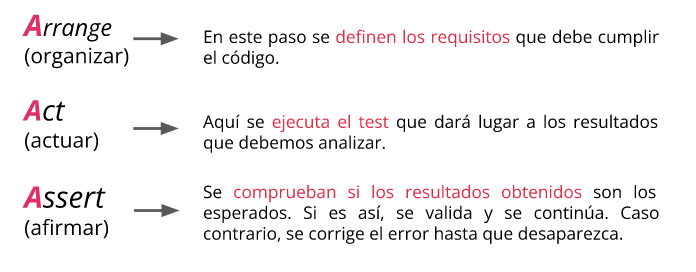
**TEST UNITARIOS vs de INTEGRACION**

**Estos test se usan para comprobar que nuestro código funciona en cualquiera de los escenarios. Para esto se usa JUnit**

**TEST UNITARIO**

**Testea de manera específica, de manera parcial el código. Toma una pequeña parte del software para determinar si funciona como esperábamos.**

**Antes de aplicar un test unitario se debe realizar el siguiente proceso (las 3ª):**

****

**Ventajas de los test Unitarios:**

**- Facilitan cambios en el código**

**- Encontrar Bugs**

**- Proveen documentación**

**- Mejoran el diseño y la calidad del código**

**Principio FIRST: características que deben tener los test unitarios para ser test de calidad:**

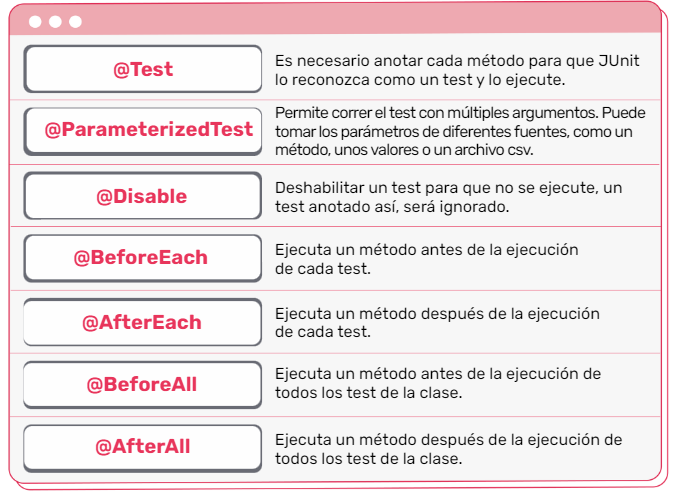
* **Fast(rapidos), un proyecto tiene miles de test, estos deben ser rapidos de correr.**
* **Isolated/Independent(aislados/independientes), cada test debe ser independiente y no debe necesitar de otro test para correrse**
* **Repeatable(repetibles), los resultados no deben depender de datos del ambiente mientras están corriendo, (ej, horario del sistema)**
* **Self-Validating(auto-validados), no debe ser requerida una inspección manual para validar los resultados**
* **Thorough(completos), deben cubrir cada escenario de uso y no solo buscar una cobertura del 100%**

**TEST DE INTEGRACION**

**Test de manera global, cubren un modulo deL test. Las unidades individuales se integran para formar componentes mas grandes. (ej: dos unidades ya probadas se combinan en un componente integrado y se prueba la interfaz entre ellas. Este test tiene por objetivo validar la interaccion entre los modulos de software. Con estos test** podemos asegurarnos que los diferentes flujos del código funcionan correctamente.

Estos test son importantes cuando necesitamos saber si el sistema cumple con las especificaciones, y cuando necesitamos actualizar una parte del código o mejorar la performance. Hay que estar atentos al diseño del test para no obtener falsos positivos (resultados).

**Debemos aplicar las siguientes anotaciones en JUnit:**

****

**@BeforeAll**, se ejecuta solo una vez antes de todos los test unitarios. En este momento es bueno inicializar variables.

**@BeforeEach**, se ejecuta antes de cada test, sirve para inicializar variables comunes a todos los test

**@Test**, sirve para generar un test unitario

**@Disable**, lo usamos cuando queremos que un test NO se ejecute

**@AfterEach**, corre después de cada test

**@AfterAll**, corre después de que terminen todos los test unitarios

**@Tag**, permite lanzar conjuntos de test en función de las etiquetas que especifiquemos.

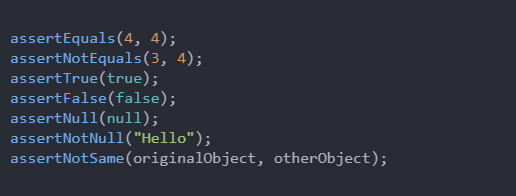
Ademas de las anotaciones, realizamos validaciones con las siguientes funcionalidades:

* assertEquals, se utiliza para validar si 2 resultados son iguales
* assertTrue, para saber si el resultado es verdadero o falso
* thrown.expect, nos aseguramos que recibimos una excepción (error) por ejemplo nos aseguraríamos que una división por cero dé una exception o error

Los test unitarios son aplicables a todo el código

**Assertions de JUnit** (ubicados en: org.junit.jupirter.api.Assertions)

* assertArrayEquals
* assertEquals
* assertTrue and assertFalse
* assertNull and assertNotNull
* assertSame and assertNotSame
* assertAll
* assertNotEquals
* assertIterableEquals
* assertThrows
* assertTimeout and assertTimeoutPreemptively
* assertLinesMatch



**JUnit**

Es el framework open source de testing para Java mas utilizado.

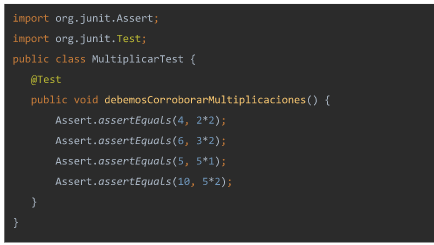
Permite escribir y ejecutar test automatizados

Es soportado por todas las IDEs, build tolos, y por frameworks como Spring

Configuracion de librerías en Intellij 🡪 [enlace](file:///C:\Users\SOL\Dropbox\CTD\Backend%20I\Configuración%20de%20la%20libería%20.pdf)

**Testeo Parametrizado y Test Suite**

Testeo Parametrizado: se utiliza para comprobar diferentes casos que nos hagan repetir código



Para construir un test parametrizado JUnit utiliza un custom runner que es **Parametrized**

