DIVE IN: MOBILE TESTING

Balanzino, Conti, Zanino

REPLY
IRISCUBE

SUMMARY

LETS START...

“Pay attention to zeros. If there is a zero, someone will divide by it” – Dr. Cem Kaner

WHO & WHY [1/4]

WHO'S IRISCUBE REPLY

IRISCUBE is a company of **Reply** group providing consultancy services and developing omnichannel **innovative solutions for the finance industry**, covering topics such as web, **NATIVE MOBILE**, architectures, microservices, backend development, strategical consultancy.

A FEW NUMBERS:

Avg profit growth y2y: **+40%**

Number of developers: **~250**

Avg developers' age: **25,8 YO**

OUR MAIN CUSTOMERS:

INTESA SANPAOLO

CREDEM

UBI Banca

nexi

BANCA mediolanum

FIDEURAM

GRUPPO BANCARIO Credito Valtellinese

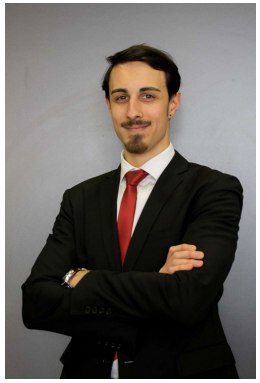
WHO & WHY [2/4]

WHO WE ARE

Marco Balanzino

- Team leader
- Android developer
- Android enthusiast

Marco Zanino



- Android Architect
- Team Leader
- Senior Android developer

Alessandro Conti

- Google Associate Android Developer
- Project Manager
- Senior Consultant

WHO & WHY [3/4]

WHO WE ARE

droidcon
Italy // Torino

https://www.youtube.com/watch?v=9N8Lctn1IQM&t=

WHO & WHY [4/4]

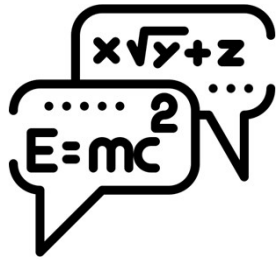
WHY WE'RE HERE

WE KNOW (ANDROID) THINGS!

We believe in **personal growth** and we strongly encourage our colleagues to constantly keep up with the **latest news** in the Android development world.

WE DO (ANDROID) STUFF!

In Iriscube Reply every year we have to study a specific argument, this is called «**Centro di competenza**». This year we are focusing in **Mobile testing** and we want to share this knowledge with you!



WE WILL HELP YOU WITH THE EXAM!

Considering that **we know things** and **we do stuff**, we will help you develop the final android application required to **pass the exam.**

INTRO TO TESTING [1/2]

CATEGORIES OF ANDROID TESTS

To ensure that you application works well, it is important to write software tests. This helps you to enhance and maintain the Android application.
Unit testing for Android can be classified into:

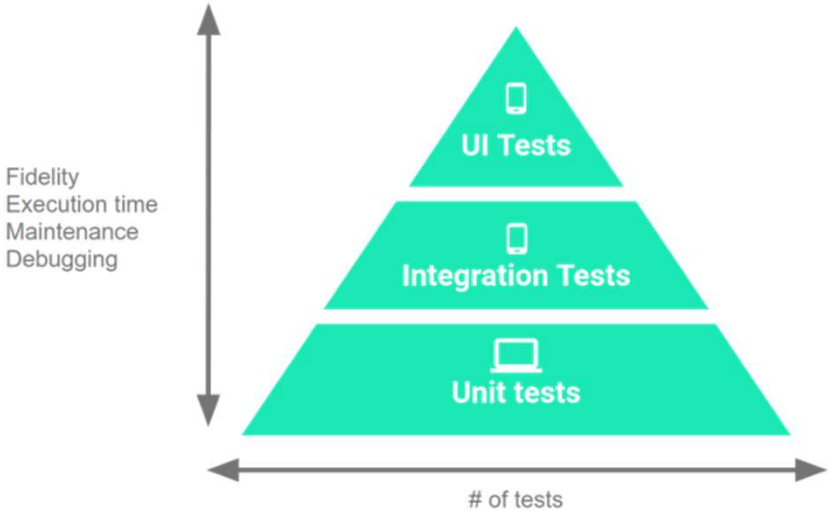
- **Local unit tests** - tests which can run on the JVM.
- **Instrumented unit tests** - tests which require the Android system

Local unit tests **run much faster** compared to the time required to **deploy and run** tests on an Android device. **PREFER WRITING LOCAL UNIT TESTS AND ONLY RUN TESTS ON ANDROID**, if you require a real Android system.

If you write local unit test and have dependencies to **Android API**, you need to replace them, e.g., via a **MOCKING FRAMEWORK LIKE MOCKITO**

INTRO TO TESTING [2/2]

THE TESTING PYRAMID



- **10% - LARGE TESTS** are **integration and UI tests** that run by completing a UI workflow. They ensure that key end-user tasks work as expected on emulators or real devices.
- **20% - MEDIUM TESTS** are **integration tests** that sit in between small tests and large tests. They integrate several components, and they run on emulators or real devices.
- **70% - SMALL TESTS** are **unit tests** that you can run in isolation from production systems. They typically mock every major component and should run quickly on your machine.

UNIT TEST

UNIT TESTS [1/11]

BASICS

A unit test verifies in **ISOLATION** the functionality of a certain component. For example, assume you have a **conversion method** that is **widely used by components** of your application, you should test the correctness of the conversion and not how the values are used in latter functions.

To use JUnit tests for your Android application, you need to **ADD IT AS A DEPENDENCY** to your Gradle build file:

```
dependencies {
    // Unit testing dependencies
    testCompile 'junit:junit:4.12'
    // Set this dependency if you want to use the Hamcrest matcher library
    testCompile 'org.hamcrest:hamcrest-library:1.3'
    // more stuff, e.g., Mockito
}
```

UNIT TESTS [2/11]
BASICS & ANNOTATIONS

In Junit there are some basic **ANNOTATIONS** that you should understand **before anything else**:

- **@BeforeClass** – Run once before any of the test methods in the class, public static void
- **@AfterClass** – Run once after all the tests in the class have been run, public static void
- **@Before** – Run before @Test, public void
- **@After** – Run after @Test, public void
- **@Test** – This is the test method to run, public void

UNIT TESTS [3/11]

BASICS & ANNOTATIONS

```
// Run once, e.g. Database connection, connection pool
@BeforeClass
public static void runOnceBeforeClass() {
    System.out.println("@BeforeClass - runOnceBeforeClass");
}

// Run once, e.g close connection, cleanup
@AfterClass
public static void runOnceAfterClass() {
    System.out.println("@AfterClass - runOnceAfterClass");
}

// Should rename to @BeforeTestMethod
// e.g. Creating an similar object and share for all @Test
@Before
public void runBeforeTestMethod() {
    System.out.println("@Before - runBeforeTestMethod");
}
```

```
// Should rename to @AfterTestMethod
@After
public void runAfterTestMethod() {
    System.out.println("@After - runAfterTestMethod");
}

@Test
public void test_method_1() {
    System.out.println("@Test - test_method_1");
}

@Test
public void test_method_2() {
    System.out.println("@Test - test_method_2");
}
```

UNIT TESTS [4/11]
BASICS & ANNOTATIONS

Output:

```
@BeforeClass - runOnceBeforeClass

@Before - runBeforeTestMethod
@Test - test_method_1
@After - runAfterTestMethod

@Before - runBeforeTestMethod
@Test - test_method_2
@After - runAfterTestMethod

@AfterClass - runOnceAfterClass
```

UNIT TESTS [5/11]

ASSERTIONS

ASSERT is a method useful in determining **Pass** or **Fail** status of a test case, the assert methods are provided by the class **org.junit.Assert** which extends **java.lang.Object** class.

There are **various types of assertions** like Boolean, Null, Identical etc.

Junit provides a class named **Assert**, which provides a bunch of assertion methods useful in writing test cases and to detect test failure

UNIT TESTS [6/11]
ASSERTIONS

- **Boolean:**
  If you want to test the boolean conditions (true or false), you can use following assert methods

  assertTrue(condition)
  assertFalse(condition)

  Here the condition is a boolean value.

- **Null object:**
  If you want to check the nullability of an object/variable, you have the following methods:

  assertNull(object)
  assertNotNull(object)

  Here object is Java/Kotlin object

UNIT TESTS [7/11]
ASSERTIONS

- **Identical:**
  If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.

  assertSame(expected, actual), It will return true if **expected == actual**
  assertNotSame(expected, actual)

- **Assert equals:**
  If you want to test equality of two objects, you have the following methods

  assertEquals(expected, actual)

  It will return true if: **expected.equals(actual)** returns true.

UNIT TESTS [8/11]
ASSERTIONS

- **Assert equals (EXTENDED):**
  You have **assertEquals(a,b)** which relies on the equals() method of the Object class.

  Here it will be evaluated as **a.equals(b).**
  Here the class under test is used to determine a suitable equality relation.
  If a class does not override the **equals()** method of **Object** class, it will get the default behaviour of **equals()** method, i.e. object identity.
  If **a** and **b** are primitives such as **byte, int, boolean**, etc. then the following will be done for assertEquals(a,b) :

  a and **b** will be converted to their equivalent wrapper object type (**Byte,Integer, Boolean**, etc.), and then **a.equals(b)** will be evaluated.

UNIT TESTS [9/11]
ASSERTIONS

- **Floating point assertions:**
  When you want to compare floating point types (e.g. **double** or **float**), you need an additional required parameter **delta** to avoid problems with round-off errors while doing floating point comparisons.
  The assertion evaluates as given below:

  Math.abs(expected – actual) <= delta

  For example:

  assertEquals(aDoubleValue, anotherDoubleValue, 0.001)

UNIT TESTS [10/11]
ASSERTIONS

- **Assert array equals:**
  If you want to test equality of arrays, you have the following methods as given below:

  assertArrayEquals(expected, actual)

  Above method must be used if arrays have the same length, for each valid value for **i**, you can check it as given below:

  assertEquals(expected[i],actual[i])
  assertArrayEquals(expected[i],actual[i])

UNIT TESTS [11/11]

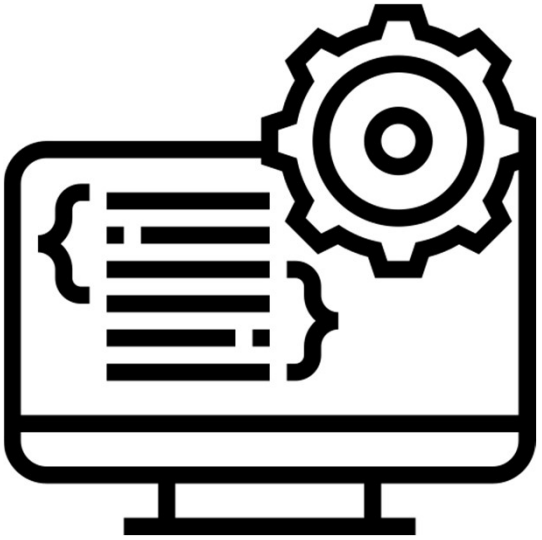
SIMPLE EXAMPLES

```java
@Test
public void testConvertFahrenheitToCelsius() {
    float actual = ConverterUtil.convertFahrenheitToCelsius(100);
    // expected value is 212
    float expected = 212;
    // use this method because float is not precise
    assertEquals("Conversion from fahrenheit to celsius failed", expected, actual, 0.001);
}
```

```java
@Test
public void testConvertCelsiusToFahrenheit() {
    float actual = ConverterUtil.convertCelsiusToFahrenheit(212);
    // expected value is 100
    float expected = 100;
    // use this method because float is not precise
    assertEquals("Conversion from celsius to fahrenheit failed", expected, actual, 0.001);
}
```

LIVE CODING

3... 2... 1... GO!



INTEGRATION TEST

INTEGRATION TESTS [1/3]

WHY DO WE NEED THEM?

- Integration tests **verify how** different units **COLLABORATE WITH ONE ANOTHER**.
- With only single-class tests, **the test suite may pass** but the feature **MAY BE BROKEN**, if a failure occurs in the interface between modules. Integration tests will verify end-to-end feature behaviour and **catch these bugs**.
- Attempting to refactor a system that only has single-class tests is often painful, because developers usually have to **completely refactor the test suite at the same time**, invalidating the safety net. The integration tests are the **SAFETY NET FOR THE REFACTORING**.
- Although **small tests are fast and focused**, allowing you to address failures quickly, they're also **LOW-FIDELITY** and **SELF-CONTAINED**, making it difficult to have confidence that a passing test **allows your app to work**.

INTEGRATION TESTS [2/3]

WHY DO WE NEED THEM?

- An integration test typically covers a **larger volume** of your system than a single-class test, so it's **MORE EFFICIENT**.
- Writing both unit and integration tests **INCREASES THE COVERAGE** faster and improves the **reliability of the tests**.

INTEGRATION TESTS [3/3]

WHAT ABOUT SERVICES? ANSWER: MOCKWEBSERVER!

Your applications will probably **be connected** to some HTTP web server that provides vital information for the user and therefore you will expect some precise response, **responses you need to verify in your tests...**
In order to do so, **MOCKWEBSERVER** comes really handy, let's see why:

- It lets you **specify which responses to return** and then verify that **requests were made as expected**
- You can **set a custom body** with a string, input stream or byte array. Also **add headers** with a fluent builder API
- Can be used to **simulate a slow network**. This is useful for testing timeouts and interactive testing

https://github.com/square/okhttp/tree/master/mockwebserver

UNIT VS INTEGRATION TEST

LETS SEE THE DIFFERENCES...

Pretty clear, right?

UI TEST

UI TESTS [1/7]
UI TEST ANATOMY

How would you expect a UI test anatomy should be? Let's say we want to test a very simple application that changes it's view after a button is pressed. Probably a human being would think about **these simple steps**:

- **FIND** the view
- **PERFORM** an action
- **CHECK** the results

As we said, this is pretty easy to understand for us, but how do we translate these steps into our application's code?

UI TESTS [2/7]
ESPRESSO

Use **Espresso** to write **CONCISE, BEAUTIFUL, AND RELIABLE** Android UI tests.

Espresso tests **STATE EXPECTATIONS, INTERACTIONS, AND ASSERTIONS** clearly **without the distraction** of boilerplate content, custom infrastructure, or messy implementation details getting in the way.

Espresso tests **run optimally fast**! It lets you **LEAVE YOUR WAITS, SYNCS, SLEEPS, AND POLLS BEHIND** while it manipulates and asserts on the application UI when it is at rest.

UI TESTS [3/7]

ESPRESSO

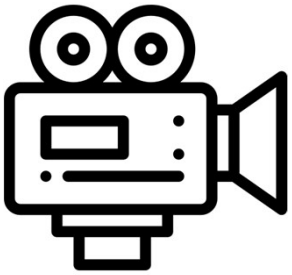
2. PERFORM an action

1. FIND the view

```
@Test
public void greet sHello() {
    onView(withId(R.id.name_field)).perform(typeText("Steve"));
    onView(withId(R.id.greet_button)).perform(click());
    onView(withText("Hello Steve!")).check(matches(isDisplayed()));
}
```

3. CHECK the results

UI TESTS [4/7]
ESPRESSO TEST RECORDER



What if we don't even want to think about those simple steps... The **ESPRESSO TEST RECORDER** is here for us!

The Espresso Test Recorder tool lets you create UI tests for your app **without writing any test code**. By recording a test scenario, you can record your interactions with a device and add assertions to verify UI elements in particular snapshots of your app.

Espresso Test Recorder then takes the saved recording and automatically **generates a corresponding UI test** that you can run to test your app.

Espresso Test Recorder is definitely an interesting and promising tool. Unfortunately, mainly due to the limitations associated with a small number of assertions to be used, at the moment it does not allow for creating complex tests. **This means we really have to USE OUR BRAIN then...**

UI TESTS [5/7]
ASSERTIONS

Espresso UI tests depend on **complex assertions** that are constructed from **VIEWMATCHER**, **VIEWACTION**, and **VIEWASSERTION** components.
At first this system can seem rather complex but really they combine to create a **fluid syntax** that is quite **elegant and concise**.
For example a basic assertion in Espresso to verify the text "**Hello World**!" is **displayed** when the main activity loads might look like the following:

```
@Test
public void shouldDisplayHelloWorld() {
  onView(withText("Hello World!")).check(matches(isDisplayed()));
}
```

This one line assertion contains **five distinct components** including **onView(), withText(), check(), matches(),** and **isDisplayed().**

UI TESTS [6/7]

ESPRESSO CHEAT SHEET

ViewMatcher

USER PROPERTIES

```
withId(...)
withText(...)
withTagKey(...)
withTagValue(...)
hasContentDescription(...)
withContentDescription(...)
withHint(...)
withSpinnerText(...)
hasLinks()
hasEllipsizedText()
hasMultilineTest()
```

UI PROPERTIES

```
isDisplayed()
isCompletelyDisplayed()
isEnabled()
hasFocus()
isClickable()
isChecked()
isNotChecked()
withEffectiveVisibility(...)
isSelected()
```

HIERARCHY

```
withParent(Matcher)
withChild(Matcher)
hasDescendant(Matcher)
isDescendantOfA(Matcher)
hasSibling(Matcher)
isRoot()
```

and many more...

```
onView(ViewMatcher)
      .perform(ViewAction)
      .check(ViewAssertion)
```

ViewAction

CLICK/PRESS

```
click()
doubleClick()
longClick()
pressBack()
pressIMEActionButton()
pressKey([int/EspressoKey])
pressMenuKey()
closeSoftKeyboard()
openLink()
```

GESTURES

```
scrollTo()
swipeLeft()
swipeRight()
swipeUp()
swipeDown()
```

TEXT

```
clearText()
typeText(String)
typeTextIntoFocusedView(String)
replaceText(String)
```

ViewAssertion

POSITION ASSERTIONS

```
isLeftOf(Matcher)
isRightOf(Matcher)
isLeftAlignedWith(Matcher)
isRightAlignedWith(Matcher)
isAbove(Matcher)
isBelow(Matcher)
isBottomAlignedWith(Matcher)
isTopAlignedWith(Matcher)
```

```
matches(Matcher)
doesNotExist()
selectedDescendantsMatch(...)
```

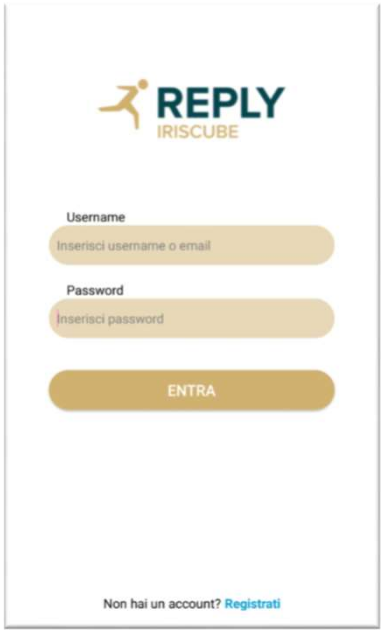
LAYOUT ASSERTIONS

```
noEllipseizedText(Matcher)
noMultilineButtons()
noOverlaps([Matcher])
```

https://developer.android.com/training/testing/espresso/cheat-sheet

UI TESTS [7/7]

ESPRESSO, MORE EXAMPLES USING WHAT WE KNOW...

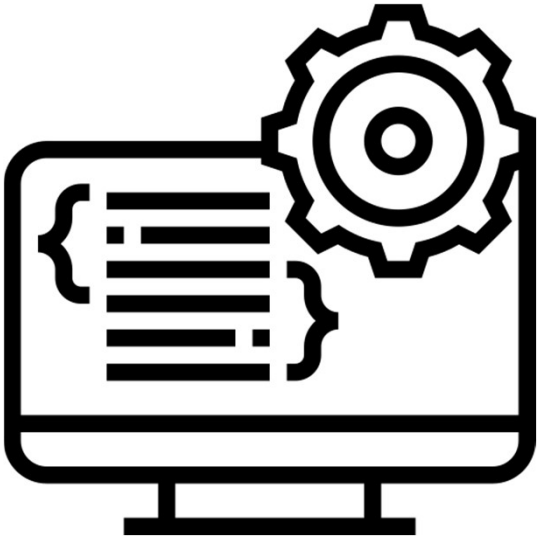


Lets consider this example:

The app **OPENS** the login screen.
If the username or the **password is EMPTY**, the screen should **SHOW an error**.
Also, we need to have tests that will make sure for given incorrect auth an error **is being SHOWN**, and for given correct auth, the **main screen is OPEN**

LIVE CODING

3... 2... 1... GO!



LIVE CODING

- Test 1: the app opens in the login page
- Test 2: if username or pw are empty an error view should be shown
  - Test 2.1: giocare con ViewMatchers -> è presente l'HINT all'interno del campo password?
  - Test 2.2: controllare se i campi sono cliccabili/editabili
  - Test 2.3: controllare se gli oggetti sono posizionati nel posto giusto (giocare con ViewAssertions)
- Test 3: if username or pw are incorrect an error view should be shown
  - Test 3.0: controllare visualizzazione softkeyboard
  - Test 3.1: download risorsa static da internet con url sbagliato non si vede, fixare issue e rieseguire test
  - Test 3.2: download risorsa static, viene visualizzata una progress nel frattempo, verificare che al termine del download la risorsa sia visualizzata e la progress sia gone (idling resources)
- Test 4: with correct input the main screen is open
- Test 5: this test returns a KO, students will have to fix the issue and run the test again
- Implementation of new small feature:
  - Implementare la paginetta di post-login con le tre sezioni, controllare che al tap su ognuna venga lanciato l'intent corretto
- Versione difficile: scrivere una viewAssertion Custom

EXAM APPLICATIONS

EXAM APPLICATIONS [1/7]

GENERIC GUIDELINES

The applications will have to:

- Use the **COMMON ANDROID PROPERTIES** (e.g. Shared Preferences)
- Use **FIREBASE** remote database
- Provide **LOCAL STORAGE** when the connection is not available
- Use components like **LISTVIEW AND RECYCLERVIEW TO REPRESENT DATA SETS** properly
- Use **PUBLIC APIs** to retrieve data
- Use **TRANSITIONS APIs** to get a good UX

EXAM APPLICATIONS [2/7]

BASKET SCOUT COACH

ABSTRACT:

The application will have to generate a basketball's scout. By using this app, you'll be able to manage all of basketball stats and generate complete team and player reports after the game.

BASE REQUIREMENTS:

Develop an app that allows you to manage the statistics related to a single basketball game.

IMPROVEMENTS:

- Provide a dashboard in which it will be possible to see the recap for each player of each registered team
- Include settings menu in which it will be possible to select only the stats desired by the user
- Generate an excel report for each match to share with different channels (e.g. email, facebook, whatsapp, telegram, ecc…)
- We accept other improvement proposals…

EXAM APPLICATIONS [3/7]

BOOK YOUR GAMES

ABSTRACT:

The application will have to book your basketball, football, volleyball, tennis and other playgrounds. By using this app, you'll be able to manage your playground reservations.

BASE REQUIREMENTS:

Develop an app that allows you to manage playground reservations for each registered user and distinguish between two kind of users:

- Administrator: will be able to manage the reservations on all the fields (insertion, cancellation, edit, addition/removal playgrounds, etc.)
- User: can only manage his reservations

IMPROVEMENTS:

- Remind the user reservation via notifications
- Admin statistics with the most frequently booked playgrounds
- Generate an excel report with the timesheet to share with different channels (e.g. email, facebook, whatsapp, telegram, ecc…)

EXAM APPLICATIONS [4/7]

TASK MAKER

ABSTRACT:

The application will have to manage your daily tasks and notify you about the upcoming deadlines.

BASE REQUIREMENTS:

Develop an app that allows you to show, create, delete and complete tasks. Each task will have title, description, start date, end date and an image (optional). Specifically, the cancellation and completion can be effected by swipe on the object itself, instead the creation can be effected by FAB.

IMPROVEMENTS:

- Use the transition APIs to improve the UX
- Notify the user when a task is close to expiration
- Advanced notification settings with customized email notifications (Google Calendar style)
- We accept other improvement proposals…

EXAM APPLICATIONS [5/7]

TRACK MY WORKOUT

ABSTRACT:

The application will have to organize, track and follow your workout improvements.

BASE REQUIREMENTS:

The application will have a series of exercises to be performed, organized by muscle group. The exercises will be shown on a list of at least 25 items. When the user select one item, he will lead to a detail page that allows you to save the weight used, track the training history and also track the monthly/weekly improvements.

IMPROVEMENTS:

- Notify the user when he/she has a performance decrease (understood as a decrease in the training frequency) or an unbalance of the trained muscle groups.
- Develop a section where users can challenge each other
- Allow the addition of videos, vocal notes, etc. for each workout to store or analyze the performances

EXAM APPLICATIONS [6/7]

METEODROID

ABSTRACT:

The application will have to monitor the weather forecast.

BASE REQUIREMENTS:

The application will have to connect to a weather forecast service and expose a list with some more important details, additional information will be provided when a list item is pressed.
The application will consist of an initial screen showing the weather forecast list based on the user's current position, with a significant icon for the type of forecast (time, sun, etc.), the minimum and maximum temperature, the reference day and time.
There will also be a detail screen showing additional information, such as wind speed.

IMPROVEMENTS:

- Allow the user to set a favourite position, which will take priority over the current user position.
- Provide some charts with temperatures trend
- Notify the user in case of extreme weather in their position (hailstorm / strong wind / etc ...)

EXAM APPLICATIONS [7/7]

COURSERA APP

ABSTRACT:

The application will have to organize, track and follow your courses improvements

BASE REQUIREMENTS:

The application must contain a series of online courses to which the user can register and on which he can define specific deadlines (for example start date and end date). The users can book their lessons through a appropriately configured calendar and, on the basis of their planning, he/she will receive notifications that remind him the lesson beginning.

IMPROVEMENTS:

- On the basis of the dates entered by the user in each course, provide daily notifications that will indicate if it will be online, late or in advance with the lesson program.
- Share the results achieved through other platforms like LinkedIn, Facebook, ecc…
- Detail the level of single lesson progress. For example if there is a video of 30 min and 10 were displayed, automatically track 33%, if there are 10,000 lines of explanation and have been viewed / viewed on screen 1000, trace 10%.

USEFUL LINKS

USEFUL LINKS [1/1]

- **GITHUB REPOSITORY:** https://github.com/zanyx/unito_testing
- **HAMCREST DOCUMENTATION:** http://hamcrest.org/JavaHamcrest/index
- **MOCKITO DOCUMENTATION:** https://site.mockito.org/
- **POWERMOCK DOCUMENTATION:** https://github.com/powermock/powermock

USEFUL LIBRARIES

USEFUL LIBRARIES [1/1]

You might find these libraries really helpful in order to complete your exams applications:

- **LOTTIE:** library that renders After Effects animations in real time, allowing apps to use animations as easily as they use static images
- **RETROFIT:** Type-safe HTTP client for Android
- **ROOM:** provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite
- **MPCHART:** powerful Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, dragging and animations
- **GSON:** serialization/deserialization library to convert Java Objects into JSON and back
- **PICASSO/GLIDE/FRESCO:** allows for hassle-free image loading in your application—often in one line of code
- **TIMBER:** A logger with a small, extensible API which provides utility on top of **Android's** normal Log class
- **OKHTTP:** open source project designed to be an efficient HTTP client. It supports the SPDY protocol. SPDY is the basis for HTTP 2.0 and allows multiple HTTP requests to be multiplexed over one socket connection.
- **ESPRESSO/KAKAO:** you know what it does....

CREDITS

- Icons from www.flaticon.com
- https://proandroiddev.com/testing-android-ui-with-pleasure-e7d795308821
- http://www.vogella.com/tutorials/AndroidTesting/article.html#testing-android-applications
- https://medium.com/android-testing-daily/breaking-down-espresso-assertions-98746278024b
- http://michaelevans.org/blog/2015/09/15/testing-intents-with-espresso-intents/
- https://developer.android.com/training/testing/espresso/intents
- https://www.html.it/pag/62653/test-delle-interfacce-con-espresso/
- https://link.medium.com/qh5aEFqurT
- https://proandroiddev.com/kakao-how-to-make-ui-testing-great-again-19972cf13740
- https://proandroiddev.com/writing-ui-tests-with-espresso-and-kakao-409c95d325bf
- https://developer.android.com/distribute/best-practices/develop/in-app-a-b-testing
- https://proandroiddev.com/writing-integration-tests-in-android-b0436978ed7b
- https://developer.android.com/studio/test/espresso-test-recorder
- https://www.raywenderlich.com/7109-test-driven-development-tutorial-for-android-getting-started

CONTACT US

m.balanzino@reply.it

m.zanino@reply.it

a.conti@reply.it

THANK YOU

www.reply.com

REPLY
IRISCUBE