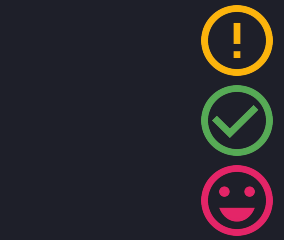
Testing Kotlin Coroutines

I need a time machine

maiatoday@gmail.com



civic

Maia Grotepass

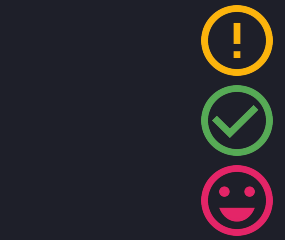
Dev

Outline

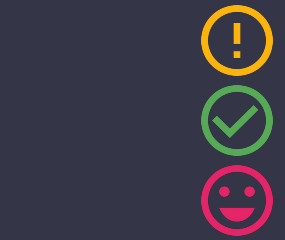
- Android and coroutines
- Problems
- What I want
- Fix
- Demo
  - Hello 8 Ball
  - Slow - Fast
  - Flakey - Predictable
  - ViewModel tests
  - API tests

What next?

Android and Coroutines



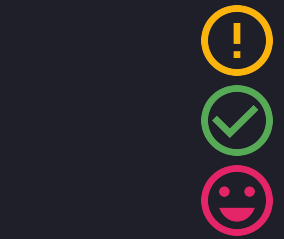
Problems



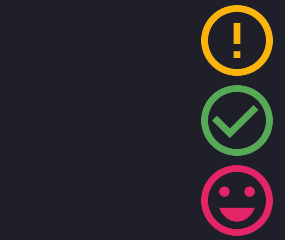
Well,

Code does **not** run on Main/UI thread

Lifecycle aware components check for Main/UI thread

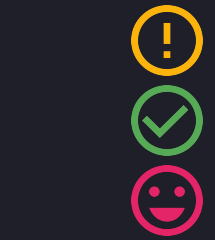


Tests finish **before** the code runs

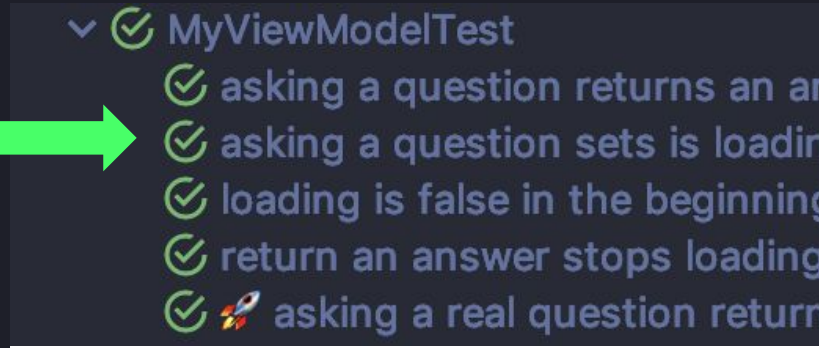


// don’t change … the test might fail ¯_(ツ)_/¯

delay(3000)

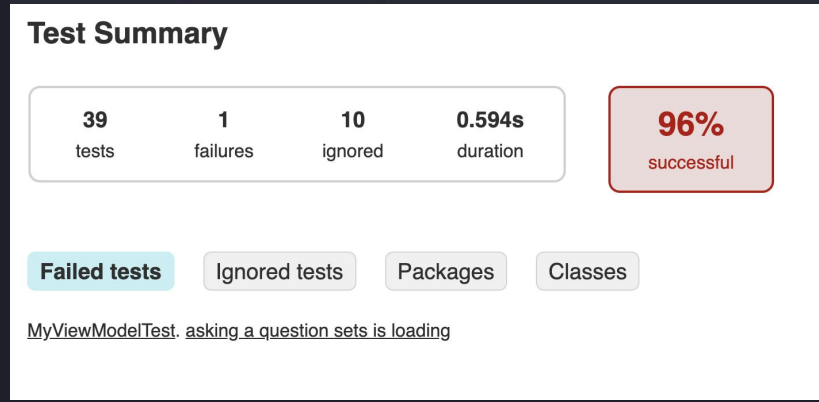


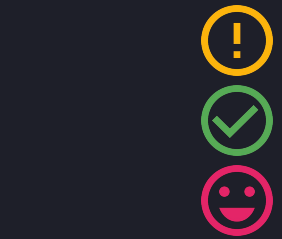
pass/fail locally



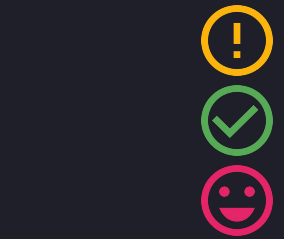
fail/pass on CI

BUILD FAIL

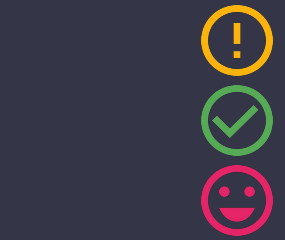




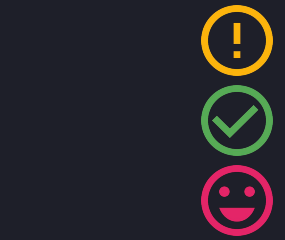
```
@Ignore("This test takes too long")

@Ignore("This test sometimes fails")
```



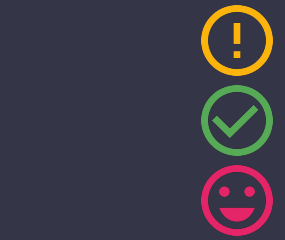
What I want



Fast
Predictable
CI



Fix



Fix - a time machine

- Run everything on **one** thread
  - runBlocking -> runBlockingTest [1]
  - Swap out UI/Main thread [1]
  - Inject
- **Control** the dispatchers [1]
- Architecture
- JVM

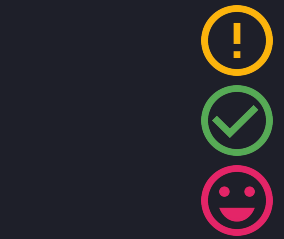
[1] experimental kotlinx-coroutines-test

kotlin-coroutines-test

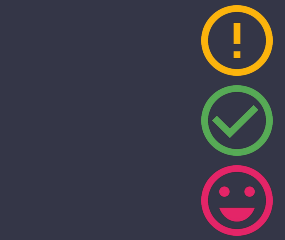
- Main delegation
  - setMain
  - resetMain

kotlin-coroutines-test

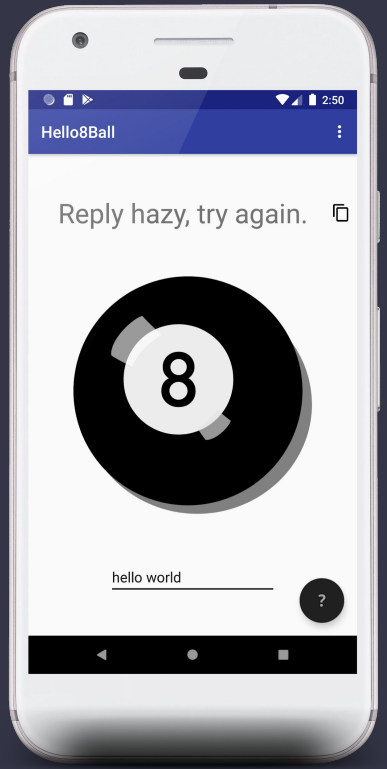
- TestCoroutineDispatcher
  - runBlockingTest
  - pauseDispatcher
  - advanceTimeBy

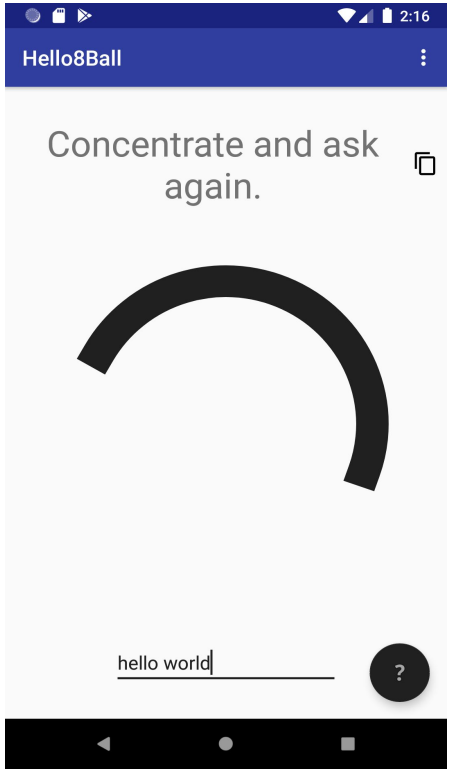


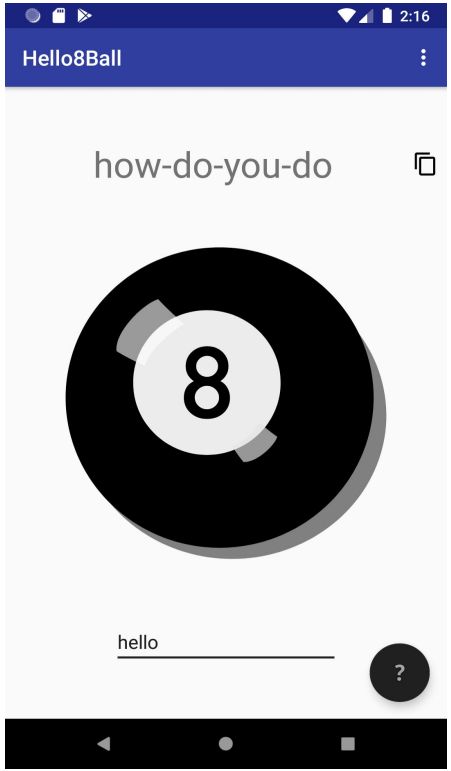
Demo

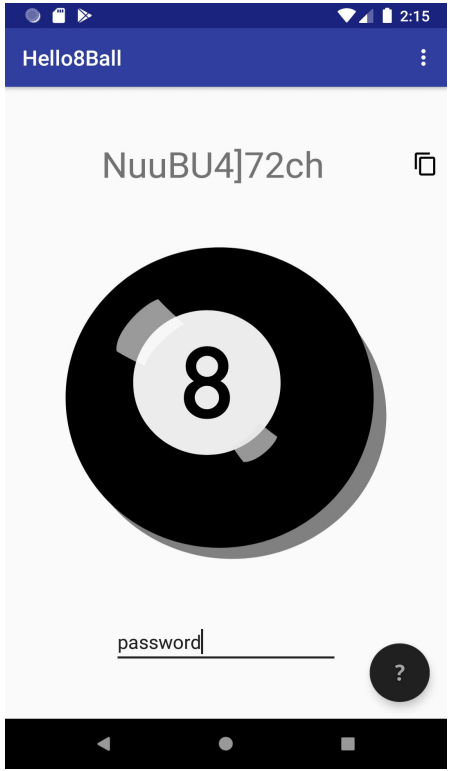


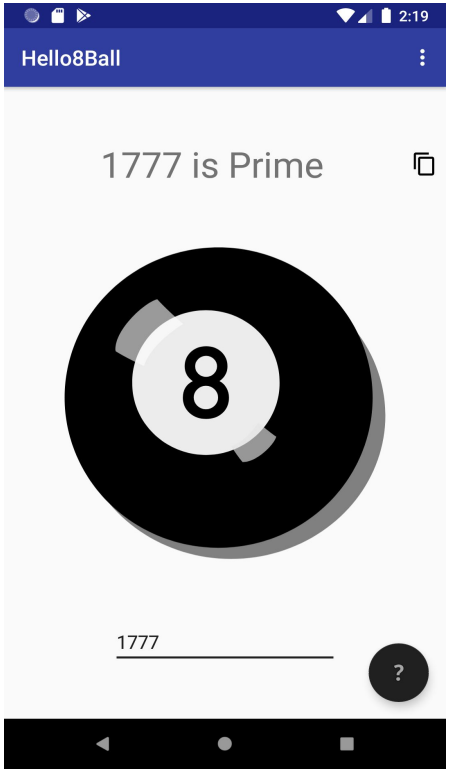
Hello 8 Ball

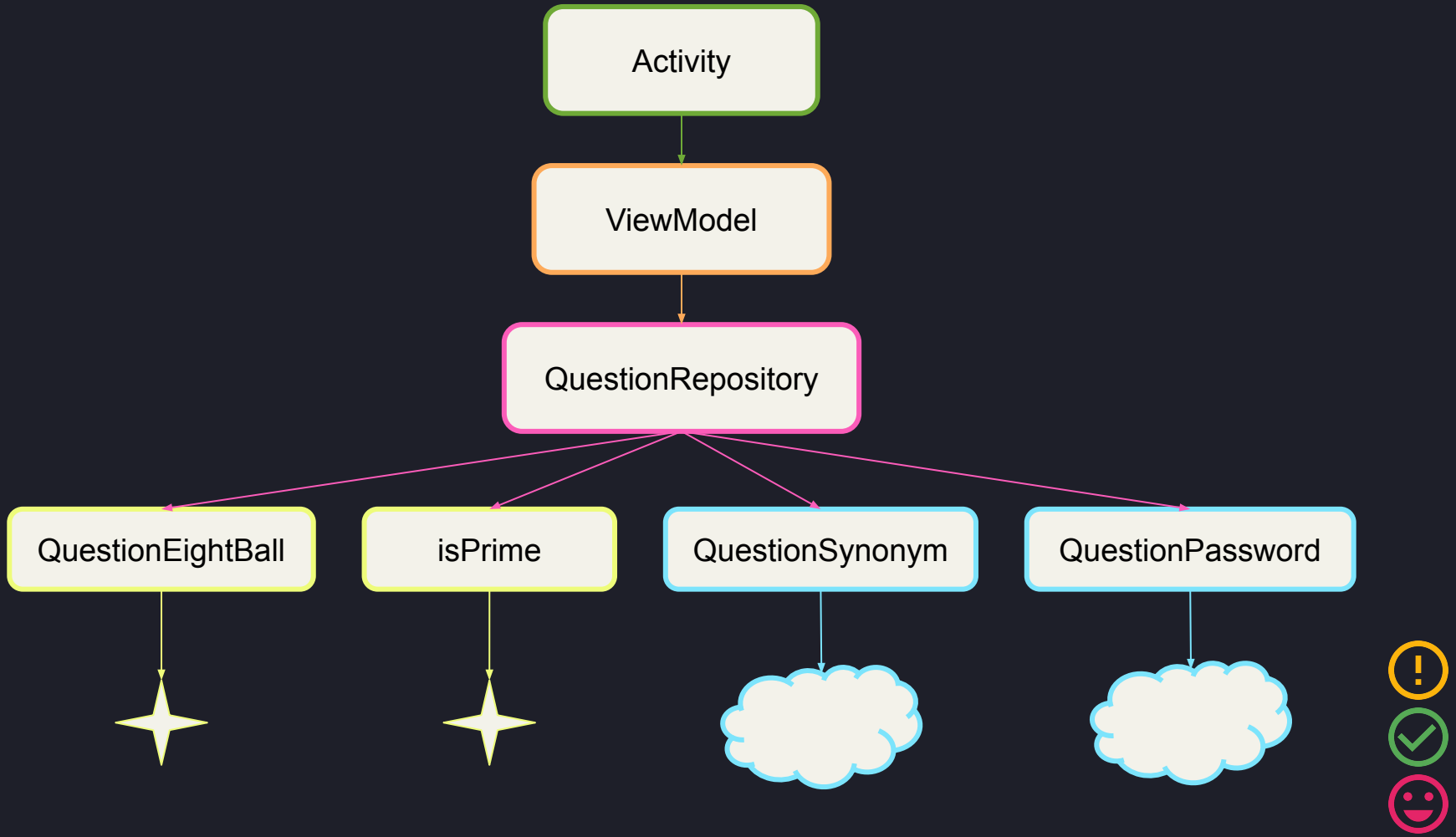


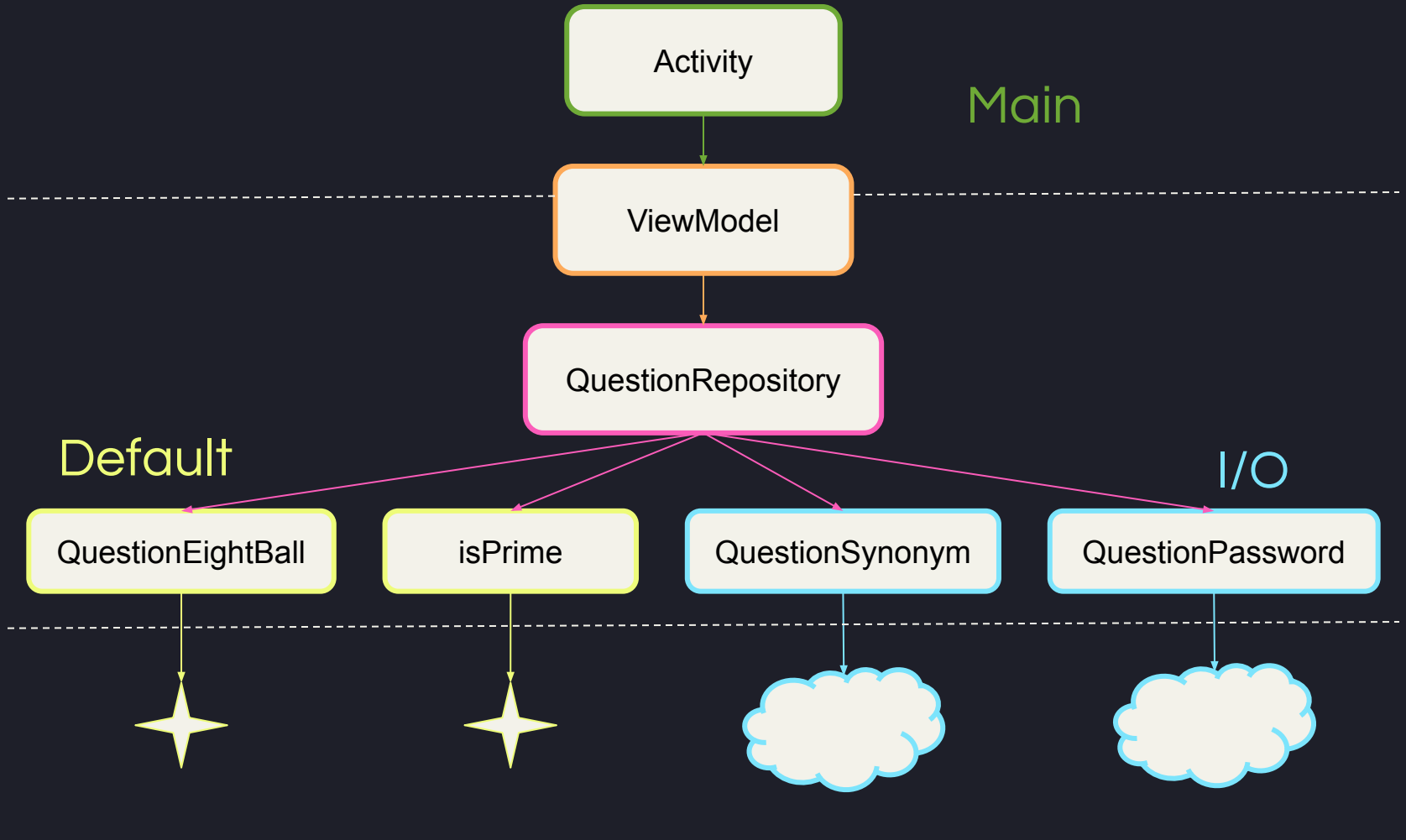

2:16
Hello8Ball
Concentrate and ask again.
hello world
?


2:16
Hello8Ball
how-do-you-do
8
hello
?

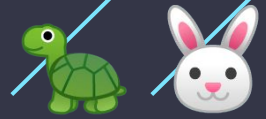

2:15
Hello8Ball
NuuBU4]72ch
8
password
?

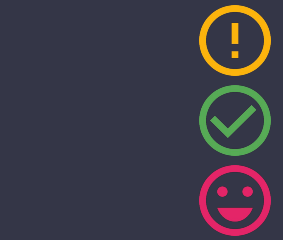

2:19
Hello8Ball
1777 is Prime
8
1777
?


Activity
ViewModel
QuestionRepository
QuestionEightBall
isPrime
QuestionSynonym
QuestionPassword


Activity
Main
ViewModel
QuestionRepository
Default
I/O
QuestionEightBall
isPrime
QuestionSynonym
QuestionPassword

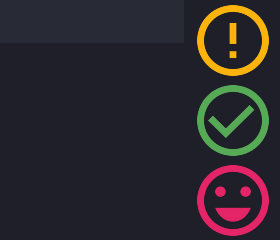
Slow vs Fast





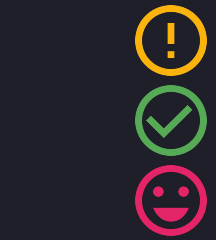
```
object QuestionEightBall : QuestionInterface {

    internal val answers: List<String> = listOf(...)

    override suspend fun getAnswer(question: String): String {
        // simulate a eightBall call here
        val randomMillis = (500 + 1000 * Math.random()).toLong()
        delay(randomMillis)
        return answers.shuffled().first()
    }
}
```

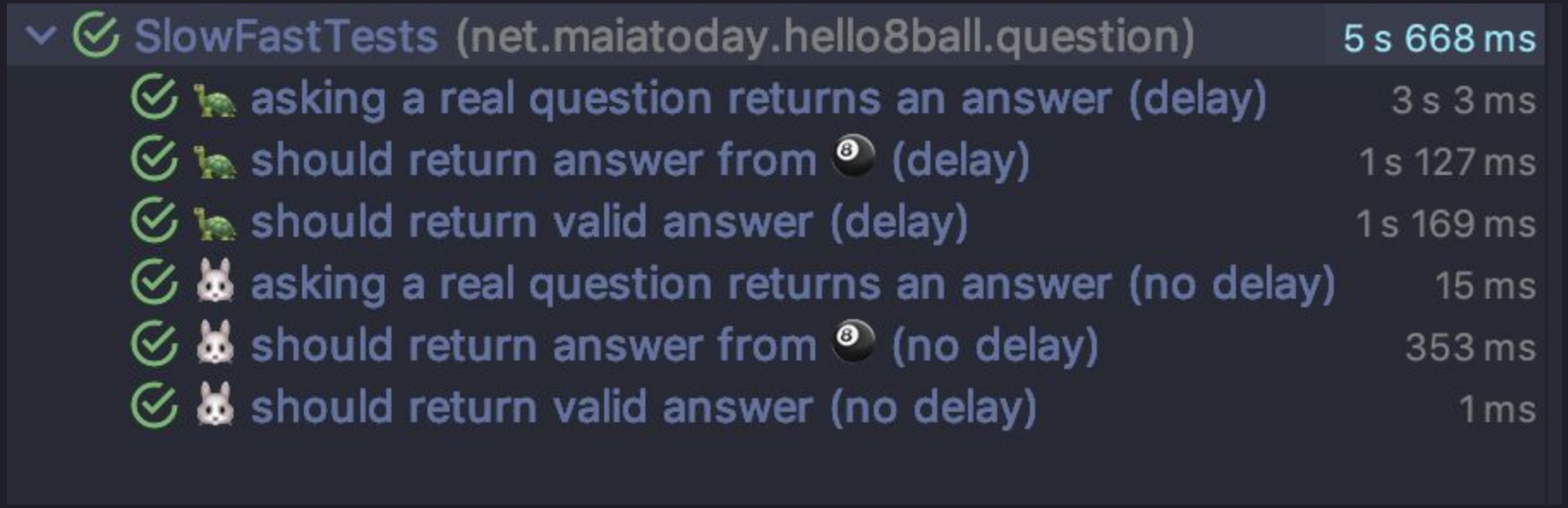
Slow vs Fast



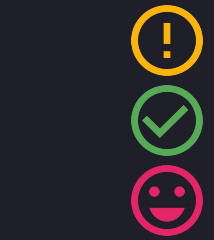
```
@Test
fun `🐌 should return valid answer (delay)`() = runBlocking { this: CoroutineScope
    val answer = QuestionEightBall.getAnswer()
    assertThat(answer).isIn(QuestionEightBall.answers)
}

@Test
fun `🐰 should return valid answer (no delay)`() = runBlockingTest { this: TestCoroutineScope
    val answer = QuestionEightBall.getAnswer()
    assertThat(answer).isIn(QuestionEightBall.answers)
}
```

Slow vs Fast




SlowFastTests (net.maiatoday.hello8ball.question) 5 s 668 ms
asking a real question returns an answer (delay) 3 s 3 ms
should return answer from (delay) 1 s 127 ms
should return valid answer (delay) 1 s 169 ms
asking a real question returns an answer (no delay) 15 ms
should return answer from (no delay) 353 ms
should return valid answer (no delay) 1 ms

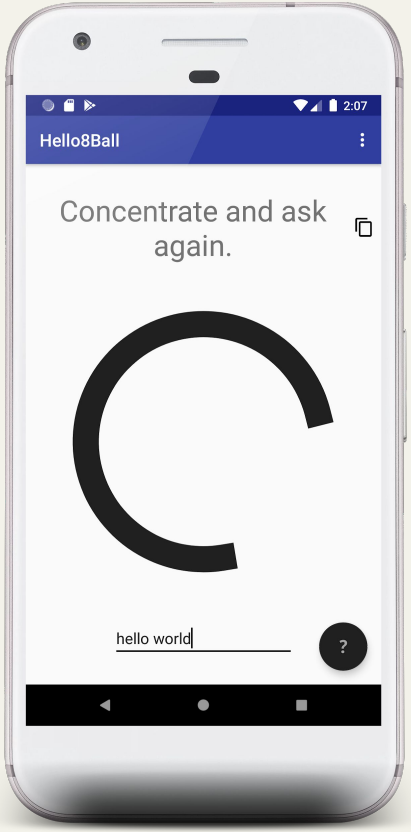
Slow vs Fast



Flakey vs Predictable

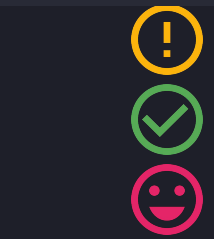
Flakey Predictable

Testing the progress bar

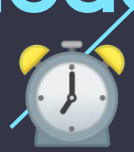


```kotlin
@Test
fun `asking a question sets is loading`() = runBlocking { this: CoroutineScope
    val mockQuestionInterface = Mockito.mock(QuestionInterface::class.java)
    val repository = QuestionRepository(mockQuestionInterface)
    val subject = MyViewModel(repository)

    subject.fetchAnswer( question: "hello world")
    delay( timeMillis: 1000) // … the test might fail ¯\_(ツ)_/¯

    Truth.assertThat(subject.isloading.getValueForTest()).isTrue()
}
```

Flakey

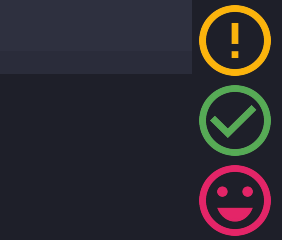


ViewModel tests



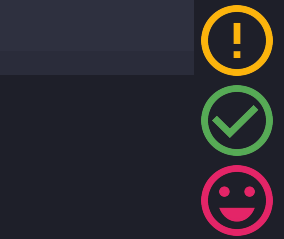
```kotlin
@ExperimentalCoroutinesApi
class MyViewModelTest {

    // Set the main coroutines dispatcher for unit testing.
    // We are setting the above-defined testDispatcher as the Main thread dispatcher.
    @get:Rule
    var coroutinesTestRule = CoroutinesTestRule()

    // Executes each task synchronously using Architecture Components.
    @get:Rule
    val instantTaskExecutorRule = InstantTaskExecutorRule()

    val testDispatcher = coroutinesTestRule.testDispatcher
    val contextProvider = TestDispatcherProvider(testDispatcher)
```

ViewModel - rule to control Main



```kotlin
@ExperimentalCoroutinesApi
class MyViewModelTest {

    // Set the main coroutines dispatcher for unit testing.
    // We are setting the above-defined testDispatcher as the Main thread dispatcher.
    @get:Rule
    var coroutinesTestRule = CoroutinesTestRule()

    // Executes each task synchronously using Architecture Components.
    @get:Rule
    val instantTaskExecutorRule = InstantTaskExecutorRule()

    val testDispatcher = coroutinesTestRule.testDispatcher
    val contextProvider = TestDispatcherProvider(testDispatcher)
```

ViewModel - rule to run synchronously



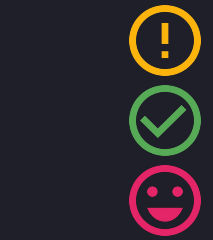
```kotlin
@ExperimentalCoroutinesApi
class MyViewModelTest {

    // Set the main coroutines dispatcher for unit testing.
    // We are setting the above-defined testDispatcher as the Main thread dispatcher.
    @get:Rule
    var coroutinesTestRule = CoroutinesTestRule()

    // Executes each task synchronously using Architecture Components.
    @get:Rule
    val instantTaskExecutorRule = InstantTaskExecutorRule()

    val testDispatcher = coroutinesTestRule.testDispatcher
    val contextProvider = TestDispatcherProvider(testDispatcher)
```

ViewModel - testDispatcher

```
@Test
fun `asking a question sets is loading ⚡🎱`() =
    testDispatcher.runBlockingTest { this: TestCoroutineScope
        pauseDispatcher {

            // setup fake that responds slowly
            val fakeInterface: QuestionInterface = SlowFakeAnswer( timeout: 5000)
            val repository = QuestionRepository(
                eightBall = fakeInterface,
                contextProvider = contextProvider
            )

            // setup subject
            val subject = MyViewModel(repository)
            subject.fetchAnswer( question: "hello world")

            // control time and test
            assertThat(subject.isloading.getValueForTest()).isFalse()
            advanceTimeBy( delayTimeMillis: 1)
```

```
            contextProvider = contextProvider
        )

        // setup subject
        val subject = MyViewModel(repository)
        subject.fetchAnswer( question: "hello world")

        // control time and test
        assertThat(subject.isloading.getValueForTest()).isFalse()
        advanceTimeBy( delayTimeMillis: 1)
        assertThat(subject.isloading.getValueForTest()).isTrue()
        advanceTimeBy( delayTimeMillis: 4998)
        assertThat(subject.isloading.getValueForTest()).isTrue()
        advanceTimeBy( delayTimeMillis: 1)
        assertThat(subject.isloading.getValueForTest()).isFalse()
    }
}
```

ViewModel

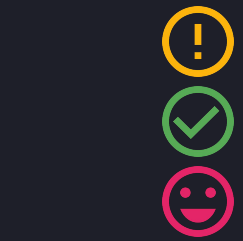


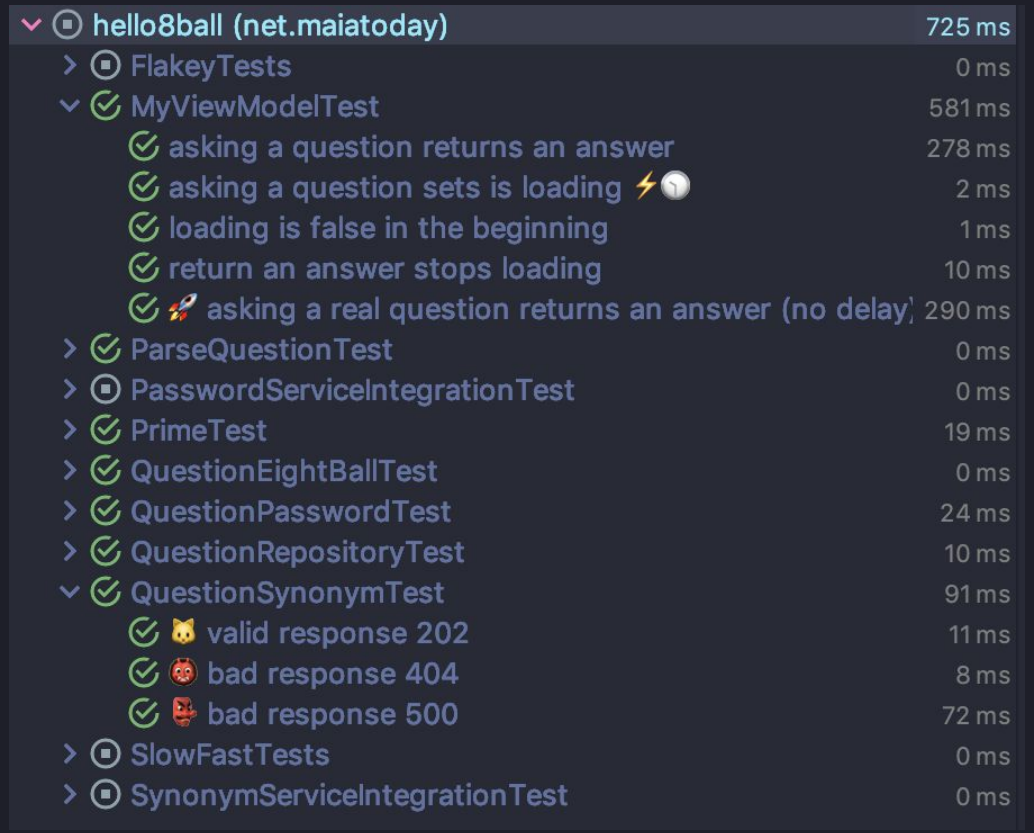
```
            contextProvider = contextProvider
        )

        // setup subject
        val subject = MyViewModel(repository)
        subject.fetchAnswer( question: "hello world")

        // control time and test
        assertThat(subject.isloading.getValueForTest()).isFalse()
        advanceTimeBy( delayTimeMillis: 1)
        assertThat(subject.isloading.getValueForTest()).isTrue()
        advanceTimeBy( delayTimeMillis: 4998)
        assertThat(subject.isloading.getValueForTest()).isTrue()
        advanceTimeBy( delayTimeMillis: 1)
        assertThat(subject.isloading.getValueForTest()).isFalse()
    }
}
```

ViewModel - step time and check loading

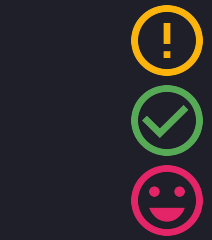

MyViewModelTest (net.maiatoday.hello8ball.view) 355 ms
asking a question sets is loading ⚡ 355 ms

ViewModel

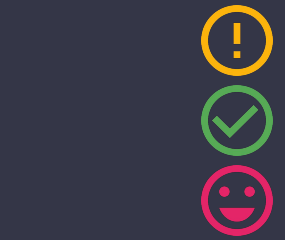



hello8ball (net.maiatoday) 725 ms
FlakeyTests 0 ms
MyViewModelTest 581 ms
asking a question returns an answer 278 ms
asking a question sets is loading ⚡ 2 ms
loading is false in the beginning 1 ms
return an answer stops loading 10 ms
🚀 asking a real question returns an answer (no delay) 290 ms
ParseQuestionTest 0 ms
PasswordServiceIntegrationTest 0 ms
PrimeTest 19 ms
QuestionEightBallTest 0 ms
QuestionPasswordTest 24 ms
QuestionRepositoryTest 10 ms
QuestionSynonymTest 91 ms
valid response 202 11 ms
bad response 404 8 ms
bad response 500 72 ms
SlowFastTests 0 ms
SynonymServiceIntegrationTest 0 ms

All tests - CI

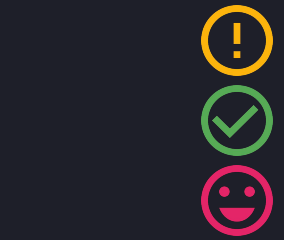


What Next?



What next?

1. Add one test and make it run on CI
2. Inject the dispatchers and/or add the kotlin testing library
3. Migrate architecture to separate Android/coroutine code/other code
4. Add a more coroutine tests



References

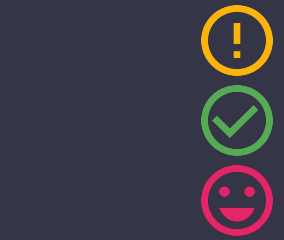
- Library: Kotlinx coroutines test
- Video: Coroutines +Testing = <3
- Video: Writing awesome tests
- Book: Learning Concurrency in Kotlin
- Repo: Codelab Kotlin Coroutines

Code - Slides - Bonus

https://github.com/maiatoday/Hello8Ball

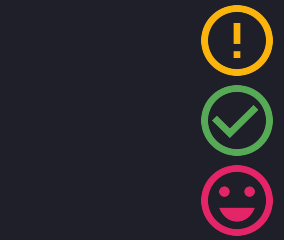
Slides in /slides

Bonus : lint, detekt, coverage, circle ci



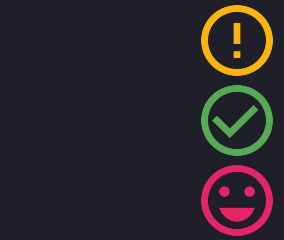
Questions 🎱

Reply hazy, try again.



```
override fun starting(description: Description?) {
    super.starting(description)
    Dispatchers.setMain(testDispatcher)
}

override fun finished(description: Description?) {
    super.finished(description)
    Dispatchers.resetMain()
    testDispatcher.cleanupTestCoroutines()
}
```

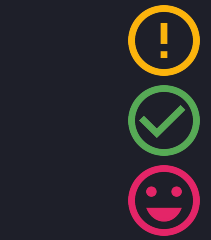
ViewModel Rule



Api tests

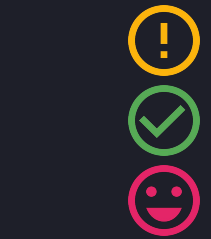
```
interface PasswordService {
    @GET( value: "query")
    fun getPasswordAsync(
        @Query( value: "command") command: String = "password",
        @Query( value: "format") format: String = "json",
        @Query( value: "count") count: Int = 1
    ): Deferred<PasswordResponse>
```

API



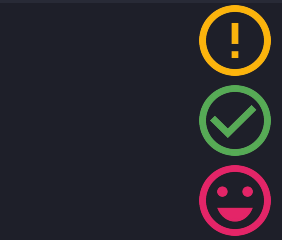
```
@Test
fun `success password service access`() = runBlocking { this: CoroutineScope
    val response = passwordService.getPasswordAsync().await()
    val passwords = response.char
    assertThat(passwords.size).isEqualTo( expected: 1)
    assertThat(passwords[0]).isNotEmpty()
}
```

API



```
class QuestionPassword(private val service: PasswordService = PasswordService.instance) :
    QuestionInterface {
    override suspend fun getAnswer(question: String): String {
        return try {
            val response = service.getPasswordAsync().await()
            val passwords = response.char
            passwords[0]
        } catch (e: HttpException) {
            "Oops no password"
        }
    }
}
```

API



```kotlin
@Before
fun setUp() {
    service = PasswordService.passwordService(server.url( path: "/"))
    subject = QuestionPassword(service)
}

@Test
fun `👹 bad response 404`() = runBlocking { this: CoroutineScope
    server.enqueue(MockResponse().setResponseCode(404))
    val answer = subject.getAnswer( question: "password")
    assertThat(answer).isEqualTo( expected: "Oops no password")
}
```

API