How to have Kotlin Coroutines in Production and sleep well

Disclaimer

Everything presented here is a product of production experience and research findings and provided AS IS without any guarantees.

About me

About me

— Lead Software Engineer

About me

— Lead Software Engineer

— Primary Skill: Android

About me

— Lead Software Engineer

— Primary Skill: Android

— Doing some Solution Architecture from time to time.

About me

— Lead Software Engineer
— Primary Skill: Android
— Doing some Solution Architecture from time to time.
— Certified Google Cloud Architect.

About me

- Lead Software Engineer
- Primary Skill: Android
- Doing some Solution Architecture from time to time.
- Certified Google Cloud Architect.

About you

About you

— Android developers?

About you

— Android developers?

— Did you use coroutines?

About you

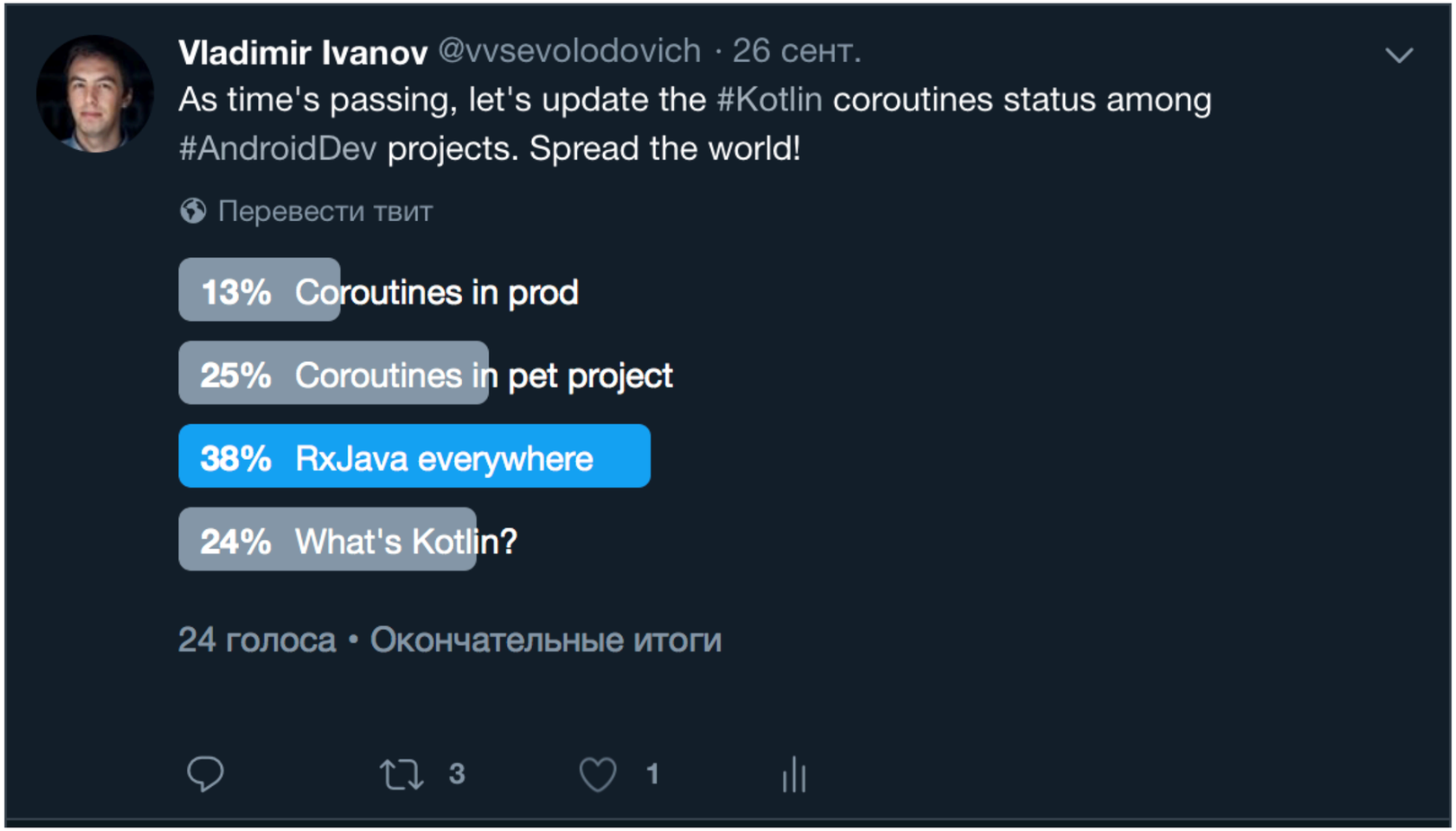
— Android developers?

— Did you use coroutines?

— Do you have them in prod?

KOTLIN 1.3 RC

Kotlin coroutines adoption



Earlier in the series

Earlier in the series

— RxJava is too complex for the most cases

Earlier in the series

— RxJava is too complex for the most cases

— Moving from RxJava to Kotlin Coroutines

Let's recall why coroutines in a first place...

RxJava 2 implementation

```
interface ApiClientRx {

    fun login(auth: Authorization)
        : Single<GithubUser>
    fun getRepositories
        (reposUrl: String, auth: Authorization)
        : Single<List<GithubRepository>>

}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    compositeDisposable.add(apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            ))
}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    compositeDisposable.add(apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            ))
}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    compositeDisposable.add(apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            ))
}
```

```
private fun attemptLoginRx() {
    showProgress(true)
    compositeDisposable.add(apiClient.login(auth)
            .flatMap {
                user -> apiClient.getRepositories(user.repos_url, auth)
            }
            .map {
                list -> list.map { it.full_name }
            }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .doFinally { showProgress(false) }
            .subscribe(
                    { list -> showRepositories(this, list)    },
                    { error -> Log.e("TAG", "Failed to show repos", error) }
            ))
}
```

Caveats

Caveats

— Complex

Caveats

— Complex

— Unrelated stack traces

Caveats

— Complex

— Unrelated stack traces

— Performance overhead(may be?)

Using coroutines

(before 0.26)

Base interface

```
interface ApiClient {

    suspend fun login(auth: Authorization) : GithubUser
    suspend fun getRepositories
        (reposUrl: String, auth: Authorization)
        : List<GithubRepository>

}
```

```
private fun attemptLogin() {
    launch(UI) {
        val auth = BasicAuthorization(login, pass)
        try {
            showProgress(true)
            val userInfo = async { apiClient.login(auth) }.await()
            val repoUrl = userInfo.repos_url
            val list = async { apiClient.getRepositories(repoUrl, auth) }.await()
            showRepositories(
                this,
                list.map { it -> it.full_name }
            )
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

```
private fun attemptLogin() {
    launch(UI) {
        val auth = BasicAuthorization(login, pass)
        try {
            showProgress(true)
            val userInfo = async { apiClient.login(auth) }.await()
            val repoUrl = userInfo.repos_url
            val list = async { apiClient.getRepositories(repoUrl, auth) }.await()
            showRepositories(
                this,
                list.map { it -> it.full_name }
            )
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

```
private fun attemptLogin() {
    launch(UI) {
        val auth = BasicAuthorization(login, pass)
        try {
            showProgress(true)
            val userInfo = async { apiClient.login(auth) }.await()
            val repoUrl = userInfo.repos_url
            val list = async { apiClient.getRepositories(repoUrl, auth) }.await()
            showRepositories(
                this,
                list.map { it -> it.full_name }
            )
        } catch (e: RuntimeException) {
            showToast("Oops!")
        } finally {
            showProgress(false)
        }
    }
}
```

Why coroutines?

Why coroutines?

— Simpler

Why coroutines?

— Simpler

— Same performance or better

Why coroutines?

— Simpler

— Same performance or better

— Tests are easy

Step aside...

withContext vs launch/async

withContext vs launch/async

— launch/async may create new Coroutine context

withContext vs launch/async

— launch/async may create new Coroutine context

— withContext reuses an existing one

[1] https://stackoverflow.com/questions/50230466/kotlin-withcontext-vs-async-await

— Use withContext whether is semantically fits you. Async/await - for parallel execuction. [1]

[1] https://stackoverflow.com/questions/50230466/kotlin-withcontext-vs-async-await

Step aside… again

Refactoring a really complex rx...

```
observable1.getSubject().zipWith(observable2.getSubject(), (t1, t2) -> {
        // side effects
        return true;
    }).doOnError {
        // handle errors
    }
    .zipWith(observable3.getSubject(), (t3, t4) -> {
        // side effects
        return true;
    }).doOnComplete {
        // gather data
    }
    .subscribe()
```

Basically with coroutines it becomes

```
try {
    val firstChunkJob = async { call1 }
    val secondChunkJob = async { call2 }
        val thirdChunkJob = async { call3 }
    return Result(
        firstChunkJob.await(),
        secondChunkJob.await(),
        thirdChunkJob.await())
} catch (e: Exception) {
    // handle errors
}
```

Why wouldn't you use coroutines in prod?


#KEYANDPEELE

— Lifecycle management

- Lifecycle management
- Lack of complex tasks experience

- Lifecycle management
- Lack of complex tasks experience
- Lack of tools

- Lifecycle management
- Lack of complex tasks experience
- Lack of tools
- No understanding of testing

1. Lifecycle management

1. Lifecycle management

1. Lifecycle management

— Coroutine can leak as well as disposable, AsyncTask

1. Lifecycle management

— Coroutine can leak as well as disposable, AsyncTask

— So we need to stop them

Stopping

— Thread.stop()

Stopping

— ~~Thread.stop()~~

Stopping

- ~~Thread.stop()~~
- How do we cancel with Rx?
- CompositeDisposable!

RxJava way

```
private val compositeDisposable = CompositeDisposable()
```

RxJava way

```
private val compositeDisposable = CompositeDisposable()

fun requestSmth() {
    compositeDisposable.add(
        apiClientRx.requestSomething()
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(result -> {})
}
```

RxJava way

```
private val compositeDisposable = CompositeDisposable()

fun requestSmth() {
  compositeDisposable.add(
    apiClientRx.requestSomething()
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(result -> {})
}

override fun onDestroy() {
  compositeDisposable.dispose()
}
```

Coroutines way

```
private val job: Job? = null
```

Coroutines way

```
private val job: Job? = null

fun requestSmth() {
    job = launch(UI) {
        val user = apiClient.requestSomething()
        ...
    }
}
```

Coroutines way

```
private val job: Job? = null

fun requestSmth() {
  job = launch(UI) {
    val user = apiClient.requestSomething()
    ...
  }
}

override fun onDestroy() {
  job?.cancel()
}
```

Issues?

Issues?

— Cloning fields for every possible job

Issues?

— Cloning fields for every possible job

— Sustaining a registry for jobs

Issues?

— Cloning fields for every possible job

— Sustaining a registry for jobs

— Duplicating code

Alternatives

Alternatives

— CompositeJob

Alternatives

— CompositeJob

— Lifecycle

CompositeJob

CompositeJob

```
private val job: CompositeJob = CompositeJob()

fun requestSmth() {
    job.add(launch(UI) {
        val user = apiClient.requestSomething()
        ...
        })
}

override fun onDestroy() {
    job.cancel()
}
```

CompositeJob

```
class CompositeJob {

    private val map = hashMapOf<String, Job>()

    fun add(job: Job, key: String = job.hashCode().toString())
        = map.put(key, job)?.cancel()

    fun cancel(key: String) = map[key]?.cancel()

    fun cancel() = map.forEach { _, u -> u.cancel() }
}
```

CompositeJob

```
class CompositeJob {

    private val map = hashMapOf<String, Job>()

    fun add(job: Job, key: String = job.hashCode().toString())
        = map.put(key, job)?.cancel()

    fun cancel(key: String) = map[key]?.cancel()

    fun cancel() = map.forEach { _, u -> u.cancel() }
}
```

CompositeJob

```
class CompositeJob {

    private val map = hashMapOf<String, Job>()

    fun add(job: Job, key: String = job.hashCode().toString())
        = map.put(key, job)?.cancel()

    fun cancel(key: String) = map[key]?.cancel()

    fun cancel() = map.forEach { _, u -> u.cancel() }
}
```

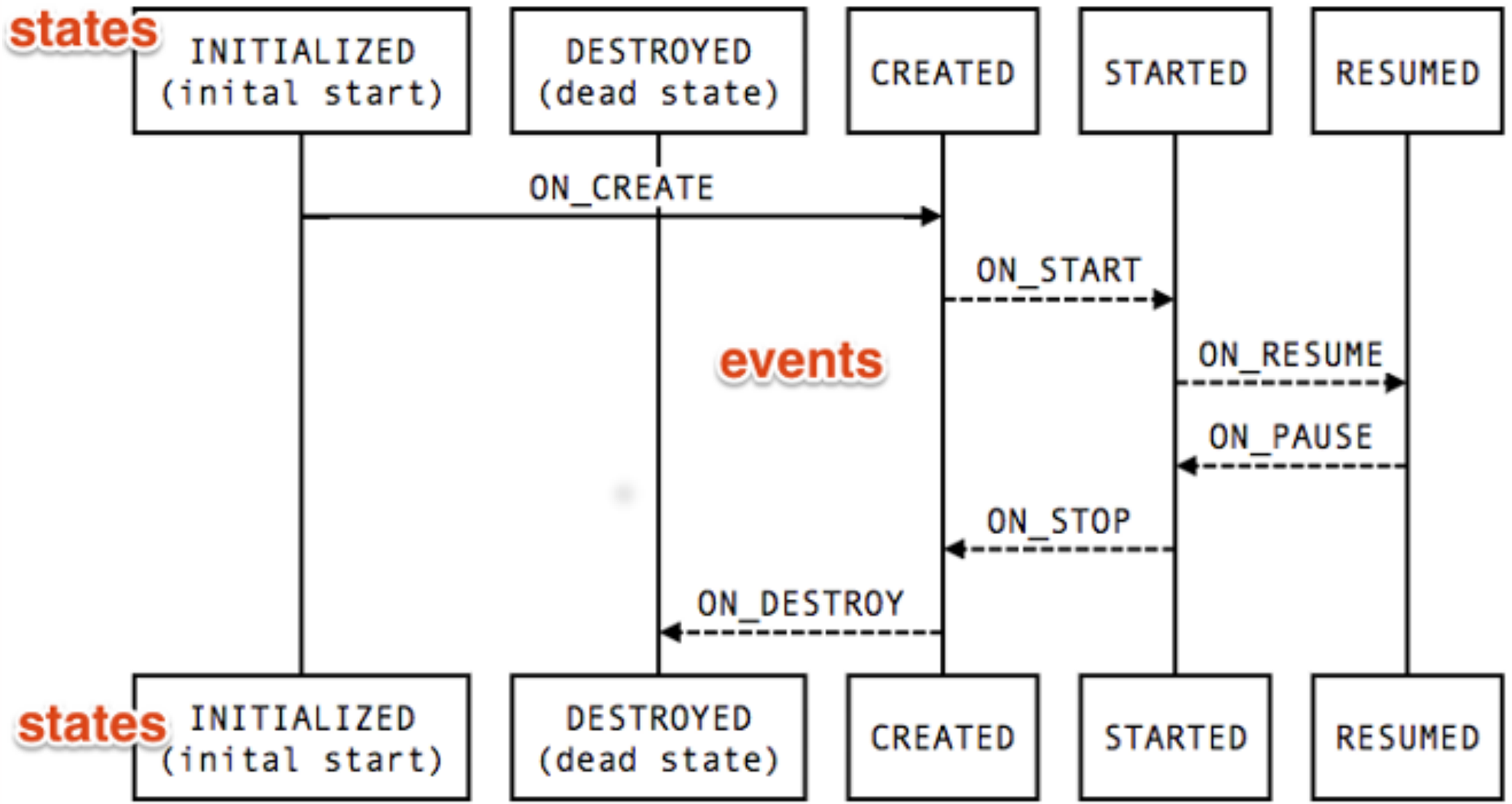
CompositeJob

```
class CompositeJob {

    private val map = hashMapOf<String, Job>()

    fun add(job: Job, key: String = job.hashCode().toString())
        = map.put(key, job)?.cancel()

    fun cancel(key: String) = map[key]?.cancel()

    fun cancel() = map.forEach { _, u -> u.cancel() }
}
```

CompositeJob

```
class CompositeJob {

    private val map = hashMapOf<String, Job>()

    fun add(job: Job, key: String = job.hashCode().toString())
        = map.put(key, job)?.cancel()

    fun cancel(key: String) = map[key]?.cancel()

    fun cancel() = map.forEach { _, u -> u.cancel() }
}
```

Lifecycle-aware job


states
INITIALIZED
(inital start)
DESTROYED
(dead state)
CREATED
STARTED
RESUMED
ON_CREATE
ON_START
events
ON_RESUME
ON_PAUSE
ON_STOP
ON_DESTROY
states
INITIALIZED
(inital start)
DESTROYED
(dead state)
CREATED
STARTED
RESUMED

```
public class MyObserver implements LifecycleObserver {
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    public void connectListener() {
        ...
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    public void disconnectListener() {
        ...
    }
}
```

```
class AndroidJob(lifecycle: Lifecycle) : Job by Job(), LifecycleObserver {

    init {
        lifecycle.addObserver(this)
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_DESTROY)
    fun destroy() {
        Log.d("AndroidJob", "Cancelling a coroutine")
        cancel()
    }
}
```

```
private var parentJob = AndroidJob(lifecycle)

fun do() {
    job = launch(UI, parent = parentJob) {
        // code
    }
}
```

2. Complex use-cases

Complex use-cases

Complex use-cases

— Operators

Complex use-cases

— Operators

— Error-handling

Complex use-cases

— Operators

— Error-handling

— Caching

Repeat - RxJava

`repeatWhen()`

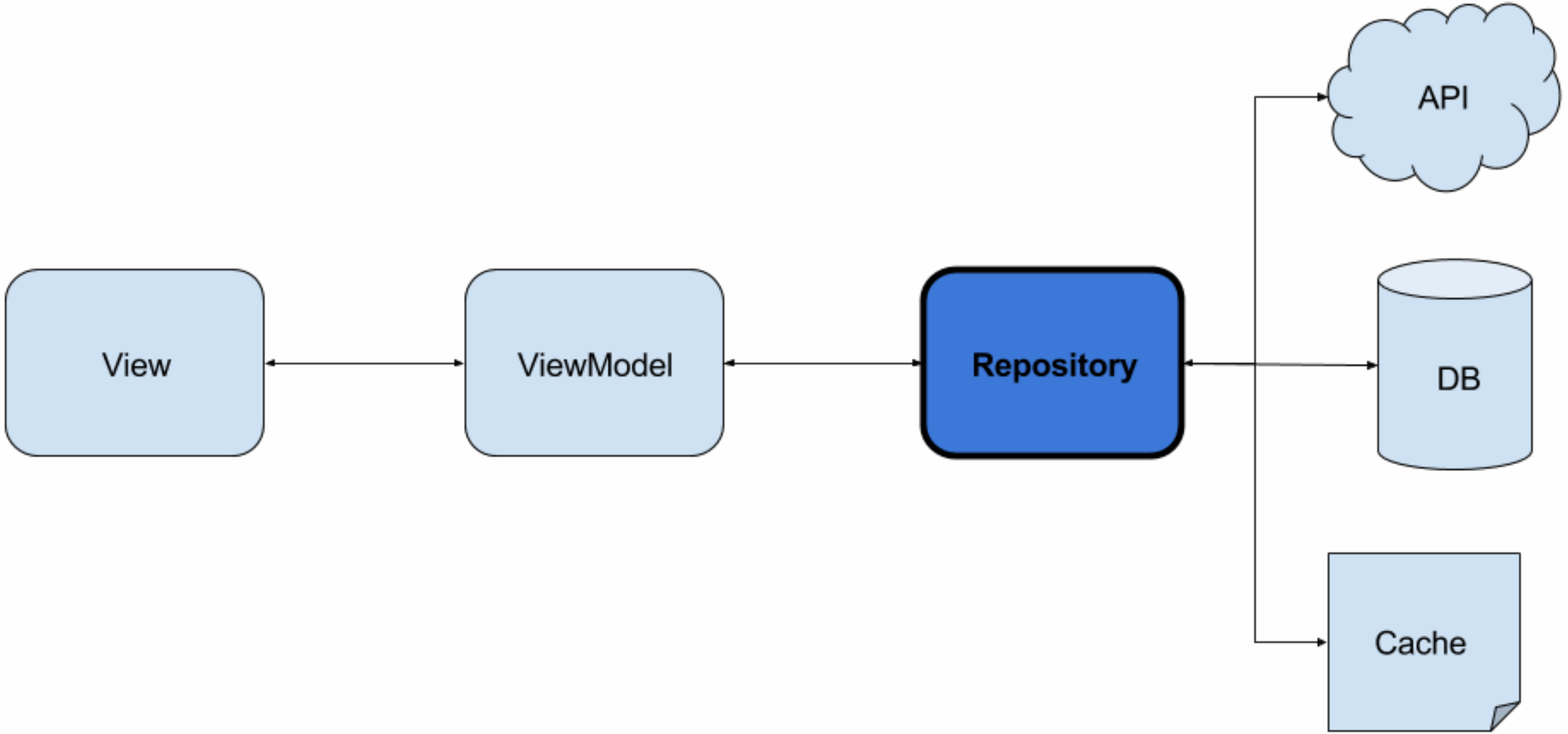
Repeat - Coroutines

```
suspend fun <T> retryDeferredWithDelay(
        deferred: () -> Deferred<T>,
        tries: Int = 3,
        timeDelay: Long = 1000L
    ): T {

    for (i in 1..tries) {
        try {
            return deferred().await()
        } catch (e: Exception) {
            if (i < tries) delay(timeDelay) else throw e
        }
    }
    throw UnsupportedOperationException()
}
```

Zip - coroutines

```
suspend fun <T1, T2, R> zip(
    source1: Deferred<T1>,
    source2: Deferred<T2>,
    zipper: BiFunction<T1, T2, R>): R {
    return zipper.apply(source1.await(), source2.await())
}

suspend fun <T1, T2, R> Deferred<T1>.zipWith(
    other: Deferred<T2>,
    zipper: BiFunction<T1, T2, R>): R {
    return zip(this, other, zipper)
}
```

Cache


API
View
ViewModel
Repository
DB
Cache

Cache

Cache

— Network Source

Cache

— Network Source

— In-memory cache

Cache

— Network Source

— In-memory cache

— Persistent cache(with expiration)

Cache

```
launch(UI) {
    var data = withContext(dispatcher) { persistence.getData() }
    if (data == null) {
        data = withContext(dispatcher) { memory.getData() }
        if (data == null) {
            data = withContext(dispatcher) { network.getData() }
            memory.cache(url, data)
            persistence.cache(url, data)
        }
    }
}
```

Rx has RxCache

Declaration

```
public interface FeatureConfigCacheProvider {

    @ProviderKey("features")
    @LifeCache(duration = 15, timeUnit = TimeUnit.MINUTES)
    fun getFeatures(
            result: Observable<Features>,
            cacheName: DynamicKey
        ): Observable<Reply<Features>>

}
```

Usage

```
val restObservable = configServiceRestApi.getFeatures()
val features =
    featureConfigCacheProvider.getFeatures(
        restObservable,
        DynamicKey(CACHE_KEY)
    )
```

Usage

```
val restObservable = configServiceRestApi.getFeatures()
val features =
    featureConfigCacheProvider.getFeatures(
        restObservable,
        DynamicKey(CACHE_KEY)
    )
```

Coroutine Cache in development!

Will look smth like that:

```
val restFunction = configServiceRestApi.getFeatures()
val features = withCache(CACHE_KEY) { restFunction() }
```


toptal
EXCEPTION!

Error handling

Error handling

— Simple error handling is done with try-catch-finally

Error handling

— Simple error handling is done with try-catch-finally

— In production you have nested try-catches

What to do?

What to do?

— CoroutineExceptionHandler

What to do?

— CoroutineExceptionHandler

— Result classes

Coroutine Exception Handler

Handler example

```
val handler = CoroutineExceptionHandler(handler = { , error ->
    hideProgressDialog()
    val defaultErrorMsg = "Something went wrong"
    val errorMsg = when (error) {
        is ConnectionException ->
            userFriendlyErrorMessage(error, defaultErrorMsg)
        is HttpResponseException ->
            userFriendlyErrorMessage(Endpoint.EndpointType.ENDPOINT_SYNCPLICITY, error)
        is EncodingException ->
            "Failed to decode data, please try again"
        else -> defaultErrorMsg
    }
    Toast.makeText(context, errorMsg, Toast.LENGTH_SHORT).show()
})
```

Handler example

```
val handler = CoroutineExceptionHandler(handler = { , error ->
    hideProgressDialog()
    val defaultErrorMsg = "Something went wrong"
    val errorMsg = when (error) {
        is ConnectionException ->
            userFriendlyErrorMessage(error, defaultErrorMsg)
        is HttpResponseException ->
            userFriendlyErrorMessage(Endpoint.EndpointType.ENDPOINT_SYNCPLICITY, error)
        is EncodingException ->
            "Failed to decode data, please try again"
        else -> defaultErrorMsg
    }
    Toast.makeText(context, errorMsg, Toast.LENGTH_SHORT).show()
})
```

Handler example

```
val handler = CoroutineExceptionHandler(handler = { , error ->
    hideProgressDialog()
    val defaultErrorMsg = "Something went wrong"
    val errorMsg = when (error) {
        is ConnectionException ->
            userFriendlyErrorMessage(error, defaultErrorMsg)
        is HttpResponseException ->
            userFriendlyErrorMessage(Endpoint.EndpointType.ENDPOINT_SYNCPLICITY, error)
        is EncodingException ->
            "Failed to decode data, please try again"
        else -> defaultErrorMsg
    }
    Toast.makeText(context, errorMsg, Toast.LENGTH_SHORT).show()
})
```

Usage

```
launch(uiDispatcher + handler) {
    ...
}
```

Result approach

```
sealed class Result {

    data class Success(val payload: String)
        : Result()

    data class Error(val exception: Exception)
        : Result()
}
```

Result usage

```
override suspend fun doTask(): Result = withContext(CommonPool) {
    if ( !isSessionValidForTask() ) {
        return@withContext Result.Error(Exception())
    }
        ...

    try {
        Result.Success(restApi.call())
    } catch (e: Exception) {
        Result.Error(e)
    }
}
```

3. Testing

3. Testing

— Replacing context

3. Testing

— Replacing context

— Mocking coroutines

Your presenter

```
val login() {
    launch(UI) {
        ...
    }
}
```

Your better presenter

```
val login(val coroutineContext = UI) {
    launch(coroutineContext) {
        ...
    }
}
```

Presenter test

```
fun testLogin() {
    val presenter = LoginPresenter()
    presenter.login(Unconfined)

}
```

Mocking coroutines

Mockk

```
coEvery {
    apiClient.login(any())
} returns githubUser
```

Mockito-kotlin

```
given {
    runBlocking {
        apiClient.login(any())
    }
}.willReturn (githubUser)
```

Presenter test

```
fun testLogin() {
    val githubUser = GithubUser('login')
    val presenter = LoginPresenter(mockApi)
    presenter.login(Unconfined)
    assertEquals(githubUser, presenter.user())
}
```

Summary

Summary

— You can refactor Rx easily

Summary

— You can refactor Rx easily

— You can cover your code with tests easily

Summary

— You can refactor Rx easily

— You can cover your code with tests easily

— The coroutines don't generate crashes

Summary

— You can refactor Rx easily

— You can cover your code with tests easily

— The coroutines don't generate crashes

— Just do it!

Useful links - 1

Useful links - 1

— Coroutine receipts: https://proandroiddev.com/android-coroutine-recipes-33467a4302e9

Useful links - 1

— Coroutine receipts: https://proandroiddev.com/android-coroutine-recipes-33467a4302e9

— Coroutines guide: https://github.com/Kotlin/kotlinx.coroutines/blob/master/coroutines-guide.md

Useful links - 1

— Coroutine receipts: https://proandroiddev.com/android-coroutine-recipes-33467a4302e9

— Coroutines guide: https://github.com/Kotlin/kotlinx.coroutines/blob/master/coroutines-guide.md

— Codelab: https://codelabs.developers.google.com/codelabs/kotlin-coroutines/index.html#0

Useful links - 2

Useful links - 2

— https://twitter.com/vvsevolodovich 🦤

Useful links - 2

— https://twitter.com/vvsevolodovich 🦤

— https://medium.com/@dzigorium 

Useful links - 2

— https://twitter.com/vvsevolodovich 🐦

— https://medium.com/@dzigorium 

— https://mobiusconf.com/

KOTLIN 1.3 RC