Milan Community Dev Tech Talk

Milan - 19 September 2023

Who are we?

We are former exGDG Milano members, **software engineers**, who have decided to continue the extraordinary **adventure** of building a **developer community** in Milan.

What we want to do?

- Expand the local developers community (not only Android)
- Share technical knowledge
- Learn new things together
- Hang out and have fun!

How we want to achieve that?

- Tech Talks
- Soft Skill Talks
- Short tech courses
- Roundtable about various topics
- Online community to keep in touch and discuss
- Mentoring program
- CV Reviews and tips
- Informal meetup (aka pizza and beer :))

All things have to be Community driven!

Who hosts us today?

Autogenerate analytics documentation in a KMM project: a case study

Cristiano Munaro

Who I am

Cristiano Munaro, Android Developer

Working for 

Chiasso, Switzerland

Github: github.com/cmunaro

What we will see

- Analytics 101
- Problem overview
- KMP 101
- How to: Events spider
- How to: Check events implementation
- How to: Generate and publish the doc

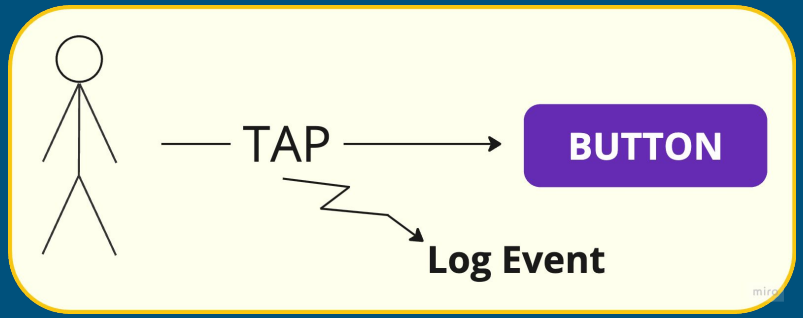
Analytics

- Definition: collection, interpretation, and presentation of data to gain insights, make informed decisions, and improve processes.

- Purpose: help organizations understand patterns, trends, and correlations in data to drive data-driven decision, optimize performance, and enhance user experiences.

Analytics of interest

- Usage Analytics: Focus on user behavior, engagement, and interaction with the app. Based on events. (tools: Firebase Analytics, etc..)
  - User Engagement: Daily/monthly active users, retention rate.
  - User Behavior: Most used features/screens, conversion funnels.
  - Geographical Data: User locations, language preferences.
  - Demographics: Age, gender, device preferences.



Our requirement(s)

- Tracks as many events you can.
  - have fun!

Trouble points

- We really need to track everything (?)
- We need to be aligned between iOS and Android
- We need to keep track of every analytics event we create
- We should not quit to sell coconut at the beach

What we can do

- We can have a board, list of features to implements for each analytics
  - Easy, we can leverage on GitHub as for other features

- We must keep track of all analytics to make them available to other stakeholders
  - Need to generate documentations
  - At every release we Must update the analytics documentation for both platforms

How we can do it

- Writing the documentation for every single event, and keeping it up-to-date manually is not an option
- Track bucket of events implementation in Github issues
- Selling coconut at the beach is better than become amanuensis

- Make some magic to compile the documentation
- Make some other magic to track the implementation
- Do not lose too much time on this

What I wanna archive

Autogenerated analytics events

Class name	Event name	Description	Parameters	Android	iOS
BalanceDecreasedEvent	**balance_decreased**	The user has decreased his money amount	delta Amount of money removed from the account	YES	NO
BalanceIncreasedEvent	**balance_increased**	The user increased his money amount	delta Amount of money added to the account	YES	NO
EasterEggShownEvent	**easter_egg_saw_event**	The user saw the super duper easter egg	crocodileSound Super secret crocodile sound elephantHappiness How much an elephant is happy from 5 to 42	NO	NO

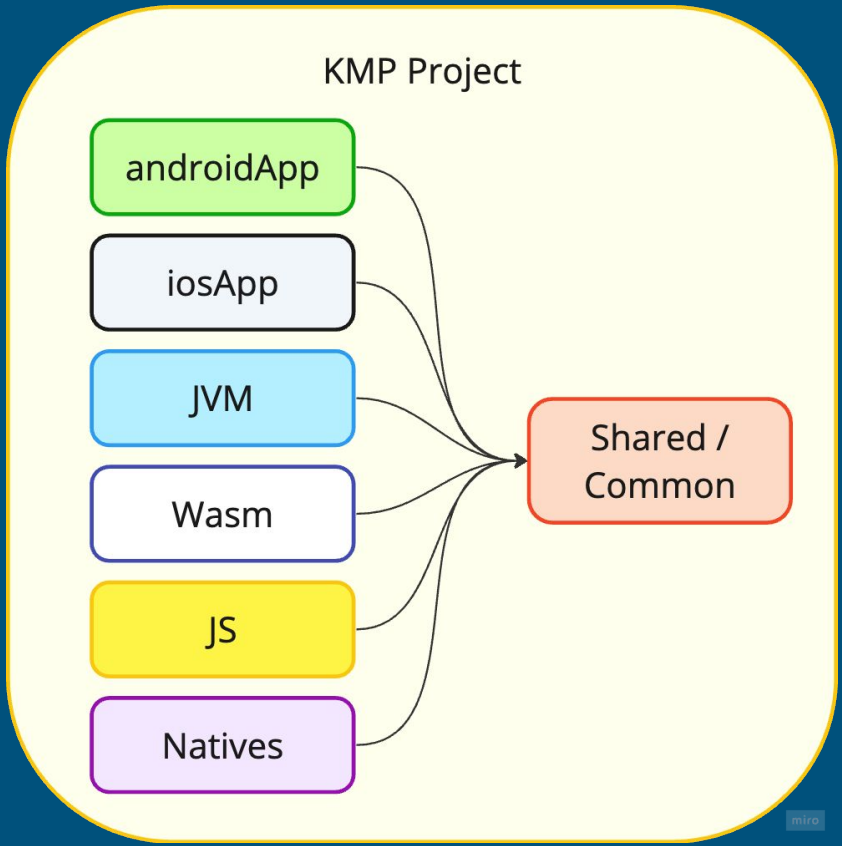
Project structure

- KMP project with iOS and Android targets
- Mono-repo project
- Shared modules (with domain and data layers)

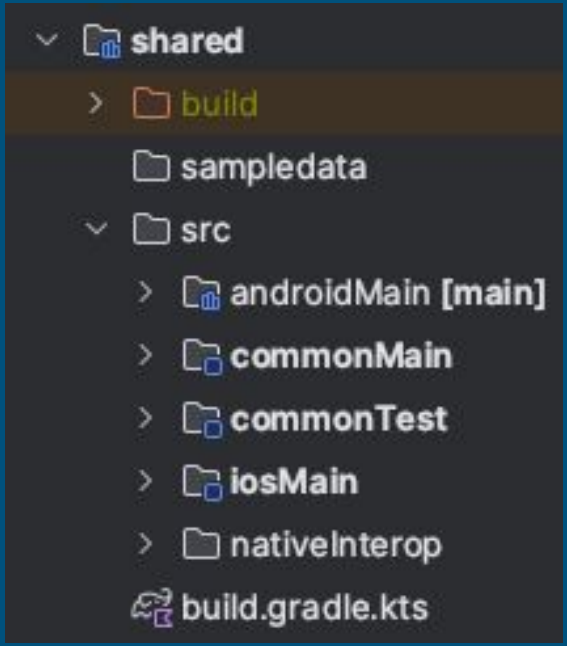
https://kotlinlang.org/docs/multiplatform.html

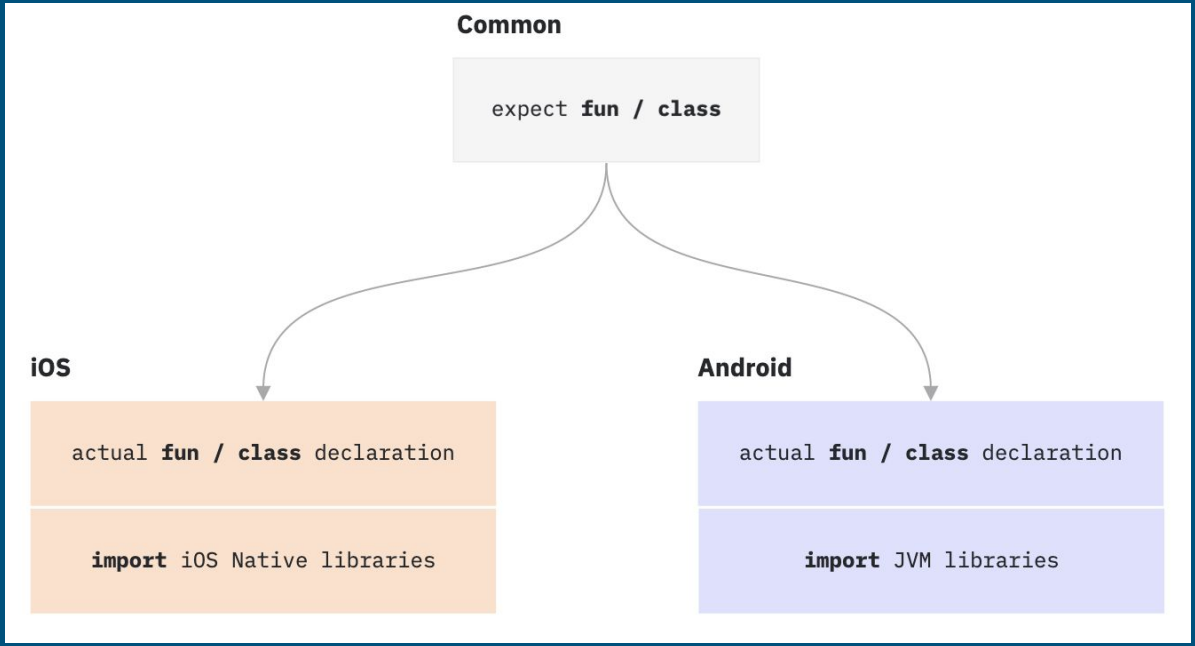
KMP 101

- Kotlin Multiplatform is a technology designed to simplify the development of cross-platform projects
- You can share common code among your different targets



One micro example





shared - common

```
interface AnalyticsRepository {
    /**
     Log an event into the analytics bucket
     */
    cmunaro
    fun log(event: [redacted]Event)
}
```

```
internal expect class GetSocialAnalyticsRepository() : AnalyticsRepository {
    cmunaro
    override fun log(event: [redacted]Event)
}
```

```
single<AnalyticsRepository>(named(AnalyticsImplementation.GET_SOCIAL)) { GetSocialAnalyticsRepository() }
```

shared - androidMain

```
import im.getsocial.sdk.Analytics

cmunaro
internal actual class GetSocialAnalyticsRepository : AnalyticsRepository {
    cmunaro
    actual override fun log(event: [redacted]Event) {
        Analytics.trackCustomEvent(event.name, event.eventParameters.toMutableMap())
    }
}
```

What we can do till now

- One new shared analytics module
  - Event definitions
  - Utility to send analytics to online services

- Apps import that module and log the event they want to send

What we miss

- Auto detect events
- Track implementation
- Generate documentation

How can we automatically list all the analytics?

How can we write (and read) the description of an event?

Annotations & Reflection

Example of an event

```
@Target(AnnotationTarget.CLASS, AnnotationTarget.PROPERTY, AnnotationTarget.VALUE_PARAMETER)
@Retention(AnnotationRetention.RUNTIME)
annotation class Description(val value: String)
```

```
abstract class AnalyticsEvent(val name: String)

@Description("Some super useful event")
data class SomeEvent(
    @Description("Some parameter") val parameter: String
) : AnalyticsEvent(name = "some_event")
```

Our goal

```
data class EventRepresentation(
    val className: String,
    val eventName: String,
    val description: String,
    val parameters: List<ParameterRepresentation>,
    val eventImplementation: EventImplementation
)
```

```
data class ParameterRepresentation(
    val parameterName: String,
    val description: String,
    val typeName: String
)

data class EventImplementation(
    val implementedByAndroid: Boolean,
    val implementedByIOS: Boolean
)
```

Reflection 101

- Framework to enable the introspection of a program at runtime, allows to access (and also alter) classes, fields and methods at runtime
- Kotlin Reflect library extends the basic capabilities
  It Adds supports to:
  - invoke functions
  - get subclasses of sealed class
  - access annotations
  - access parameters
  - create instances
  - and more..

Downside:

- Need to add metadata (+app size)
- Slower (need to interrogate JVM)
- If we shrink the code, we lose names
- **Not multiplatform**

Get the subclasses

```
sealed class Base
data object A: Base()
data object B: Base()

fun main(args: Array<String>) {
    Base::class.sealedSubclasses List<KClass<out Base>>
        .map { it.simpleName } List<String?>
        .run(::println)
}
```

```
> Task :run
[A, B]

BUILD SUCCESSFUL in 1s
```

Get description from annotations

```
annotation class MustBePrinted(val value: String)
@MustBePrinted("Hello world!")
data object A

fun main(args: Array<String>) {
    A::class.annotations List<Annotation>
        .filterIsInstance<MustBePrinted>() List<MustBePrinted>
        .map(MustBePrinted::value) List<String>
        .run(::println)
}
```

```
> Task :run
[Hello world!]

BUILD SUCCESSFUL in 1s
```

Get parameters annotations and type

```kotlin
annotation class MustBePrinted(val value: String)

data class A(
    @MustBePrinted("Hello world!") val value: String
)

fun main(args: Array<String>) {
    A::class.primaryConstructor KFunction<A>?
        ?.parameters List<KParameter>?
        ?.map { kParameter ->
            val annotationValues = kParameter.annotations List<Annotation>
                .filterIsInstance<MustBePrinted>() List<MustBePrinted>
                .map(MustBePrinted::value)
            kParameter.type to annotationValues ^map
        } List<Pair<KType, List<String>>>?
        ?.run(::println)
}
```

```
> Task :run
[(kotlin.String, [Hello world!])]

BUILD SUCCESSFUL in 1s
```

Get event name

```
@Description("The user increased his money amount")
data class BalanceIncreasedEvent(
    @Description("Amount of money added to the account") val delta: Double
) : AnalyticsEvent(name = "balance_increased")
```

```
sealed class AnalyticsEvent(val name: String)
```

Name is available only at runtime 🧐

How to get the name

```
val KClass<out AnalyticsEvent>.eventName: String
    get() = generateDummyInstance().name
```

```
fun <T : Any> KClass<T>.generateDummyInstance(): T {
    val constructorParameters = primaryConstructor!!.parameters
        .associateWith { param -> getValueFor(param) }
    return primaryConstructor!!.callBy(constructorParameters)
}
```

How to get a dummy value for each type?

Get a value

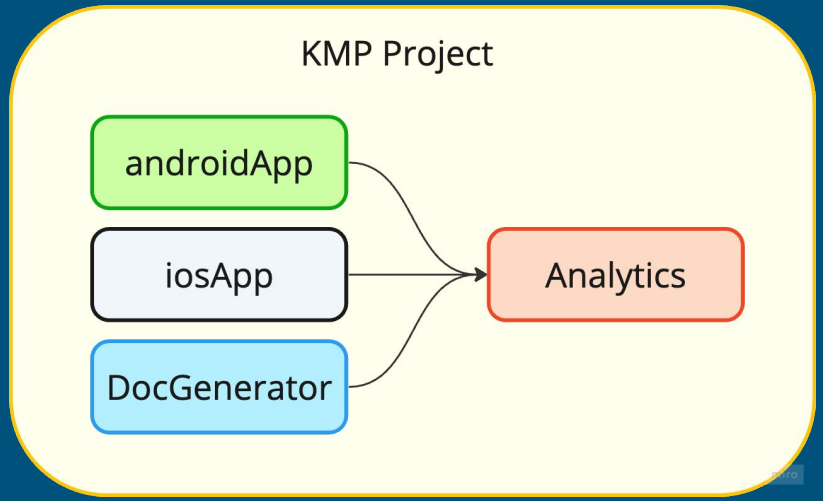
- Nullable types -> null
- Primitive types -> dummy value
- Enum -> First entry
- Others -> we try to instantiate it from its primary constructor

```
fun getValueFor(kParameter: KParameter): Any? {
    if (kParameter.type.isMarkedNullable) {
        return null
    }
    return when (kParameter.type.classifier) {
        Boolean::class -> false
        Char::class -> ' '
        Short::class -> 0
        Int::class -> 0
        Float::class -> 0f
        Double::class -> 0.0
        String::class -> ""
        else -> {
            val clazz = kParameter.type.javaType as Class<*>
            when {
                clazz.isEnum -> clazz.enumConstants.first()

                else -> runCatching { clazz.kotlin.generateDummyInstance() }
                    .onFailure { throw Exception("Unhandled default type for $kParameter") }
            }
        }
    }
}
```

We have a working event spider!

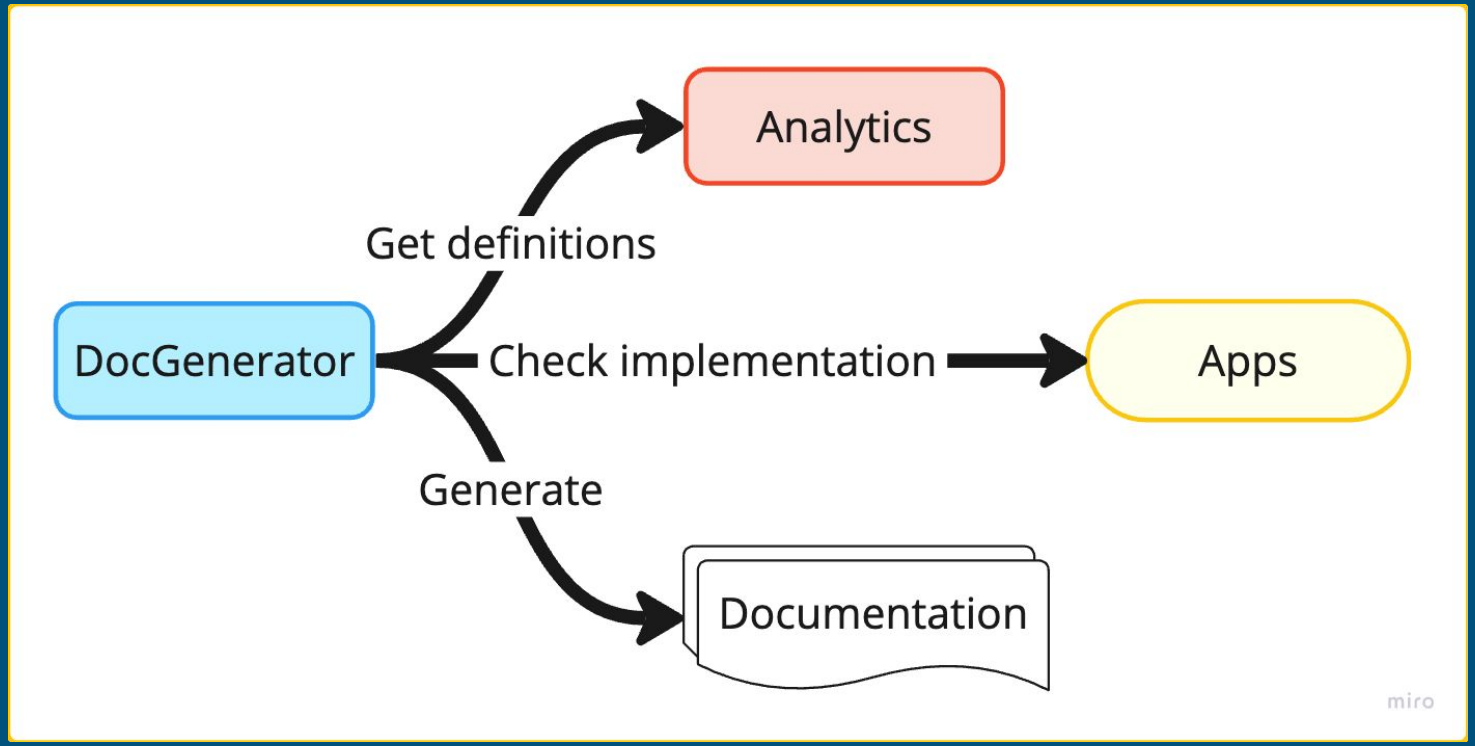
```
data class EventRepresentation(
  ✅ val className: String,
  ✅ val eventName: String,
  ✅ val description: String,
  ✅ val parameters: List<ParameterRepresentation>,
  ❌ val eventImplementation: EventImplementation
)

data class ParameterRepresentation(
  ✅ val parameterName: String,
  ✅ val description: String,
  ✅ val typeName: String
)
```

What we have so far



- 3 Targets: Android, iOS, JVM
- Android and iOS trigger the log of events
- Analytics KMP shared module where the events and the utility to log them are defined
- DocGenerator module which uses reflection to get every defined event and its description

From another point of view



Check implementation

We want to know if a platform has implemented some feature.. From its code

=> We need to parse the code!

2 ways:

- Manually search inside the code for event instantiation
  (fast, simple but unstable)
- Use something more appropriate: AST (generate an ast for each codebase and find the event initialization in the tree)

What we did

Manually search into source code, fast to implement with the aim to be substituted in the future (never changed 😅)

- Get the events definitions
- For each .kt and .swift file, search for the name of every events
- Minimal logic to detect if the name of the event found is commented or inside a comment block
- Keep track of the founded events

Generate doc

Objective: create something to make the event intermediate representation useful for other non technical stakeholders

- Web page? ✅
- Writing in HTML or JS? 🥥❌
- Kotlin? 👍

So we need to generate a webpage with Kotlin

Kotlinx.html

- Generate HTML with a Kotlin DSL 🚀
- Available for JVM and JS
- Easy to use

https://github.com/Kotlin/kotlinx.html

```
override fun createEventDocumentationPage(
    events: List<EventRepresentation>
): String {
    return createHTML().html { this: HTML
        pageHead()
        pageBody(events)
    }
}

private fun HTML.pageHead() {
    head { this: HEAD
        style { +webPageStyleDataSource.style }
        title(PAGE_TITLE)
    }
}

private fun HTML.pageBody(events: List<EventRepresentation>) {
    body { this: BODY
        h2 { +LONG_PAGE_TITLE }
        eventsTable(events)
    }
}
```

Deliver the doc: Github Pages

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at
https://cmunaro.github.io/AutogenerateAnalyticsDocumentationSample/
Last deployed by cmunaro 17 minutes ago

Visit site ···

Build and deployment

Source

Deploy from a branch

Branch

Your GitHub Pages site is currently being built from the `/docs` folder in the `master` branch. Learn more about configuring the publishing source for your site.

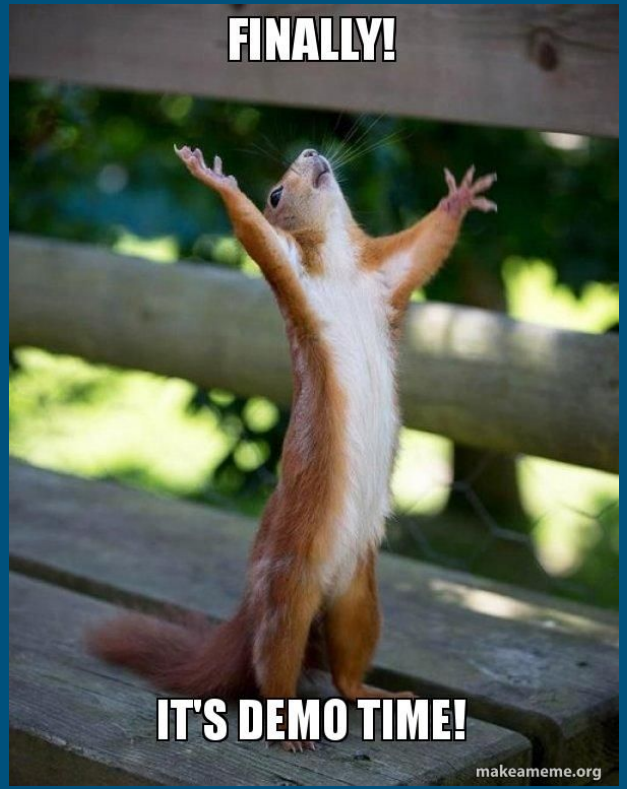
master /docs Save

Learn how to add a Jekyll theme to your site.

- Every push in master triggers the deploy of the /docs folder
- Every push in develop triggers the generation of the webpage

```yaml
name: Update analytics documentation

on:
  push:
    branches: ["develop"]

permissions:
  contents: write
  pull-requests: write

jobs:
  update_analytics_documentation:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Setup java 17
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'
          cache: 'gradle'
      - name: Create analytics documentation
        run: ./gradlew clean :docGenerator:run --args $(pwd)
      - name: Create Pull Request
        uses: peter-evans/create-pull-request@v4
        with:
          commit-message: "docs: Update analytics documentation"
          branch: "docs/autoUpdateAnalytics"
          title: "docs: Update analytics documentation"
          body: ""
          delete-branch: true
```

Demo time



That’s all, thank you!

Feel free to ask questions!

Find the sample here:
github.com/cmunaro/AutogenerateAnalyticsDocumentationSample