Kommentaire client 

Malt Academy - 2020

L’instant promo

@martinbonnin

GraphQL Summit 🚀

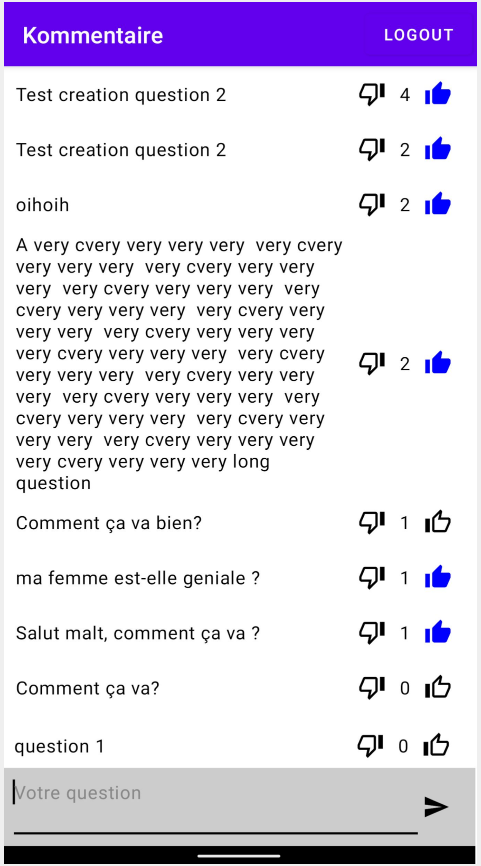
30 - 31 Juillet & 6 - 7 Août
Online
https://summit.graphql.com/

Le client Kommentaire

https://github.com/kommentaire/kommentaire-app

https://bit.ly/kommentaire-apk

Le client Kommentaire



- S’identifier
- Poser une question
- Upvote/Downvote

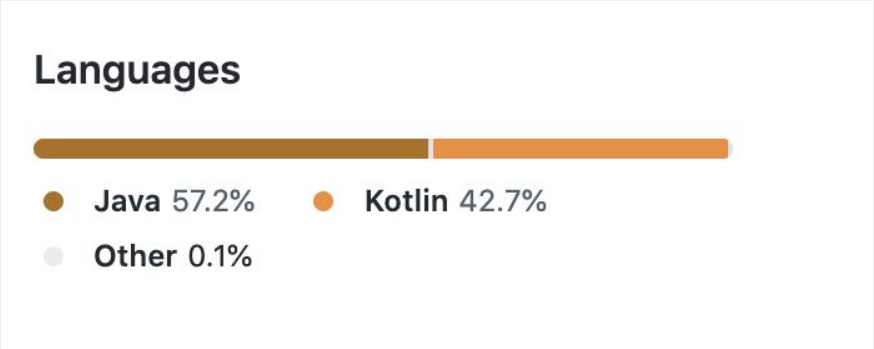
Apollo-Android
Jetpack Compose

Apollo-Android
Jetpack Compose

🥐 - Demain matin 9h - 🐸

Apollo Android: Présentation

- https://github.com/apollographql/apollo-android
- Démarré en 2016
- Kotlin depuis 2019
- Apollo-jvm



Apollo Android: fonctionnalités

- Modèles fortement typés
- Parser Json sans reflexion
- Query/Mutations/Subscription
- Cache
- RxJava2/RxJava3/Coroutines

Questions.graphql

```
query Questions {
  questions {
    id
    content
    votes
  }
}
```

Compiler

Schema.json

```
"data": {
   "__schema": {
     ...
  }
}
```

QuestionsQuery.kt

```
data class Data(
  val questions: List<Question>
) {
  data class Question(...)
}
```

Apollo Android: vue d'ensemble

Gradle Plugin

Compiler

Api

Runtime

Normalized Cache

HTTP Cache

RxJava support

Coroutines support

Espresso support

...

Apollo Android: installation

```
plugins {
    // ...
    id("com.apollographql.apollo").version("2.2.0")
}
```

Apollo Android: installation

```
plugins {
    // ...
    id("com.apollographql.apollo").version("2.2.0")
}


dependencies {
    // ...
    implementation 'com.apollographql.apollo:apollo-runtime:2.2.0'
    implementation 'com.apollographql.apollo:apollo-coroutines-support:2.2.0'
}
```

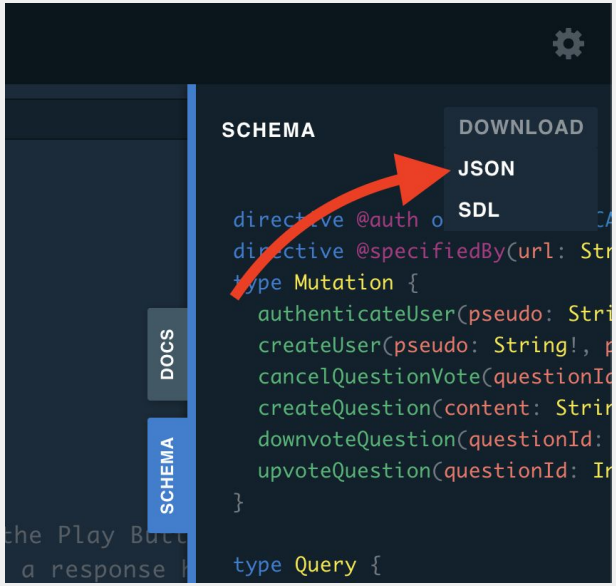
Apollo Android: installation

```
plugins {
    // ...
    id("com.apollographql.apollo").version("2.2.0")
}


dependencies {
    // ...
    implementation 'com.apollographql.apollo:apollo-runtime:2.2.0'
    implementation 'com.apollographql.apollo:apollo-coroutines-support:2.2.0'
}


apollo {
    generateKotlinModels.set(true)
}
```

Apollo Android: téléchargement d'un schema



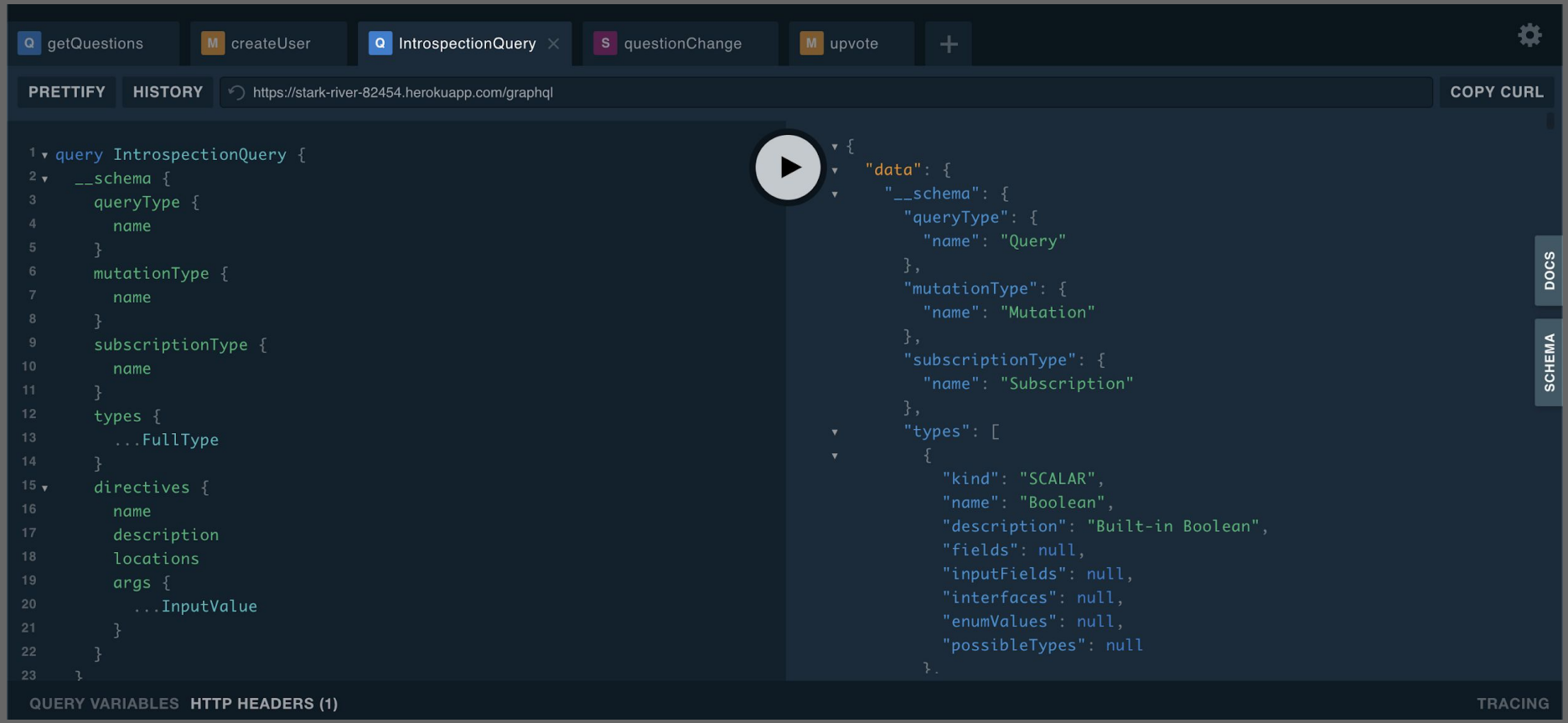
Avec playground

Apollo Android: téléchargement d'un schema

```
./gradlew downloadApolloSchema \
--endpoint='https://stark-river-82454.herokuapp.com/' \
--schema='app/src/main/graphql/fr/kommentaire/schema.json'
```

Avec le plugin Apollo

Introspection



Apollo Android: queries

```
// src/main/graphql/fr/.../questions.graphql

query getQuestions {
  questions {
    id
    content
    votes
  }
}
```

- On a src/main/graphql/.../schema.json
- On a src/main/graphql/.../questions.graphql

```
./gradlew assembleDebug 🚀
```

Questions.graphql

```
query Questions {
  questions {
    id
    content
    votes
  }
}
```

Compiler

Schema.json

```
"data": {
   "__schema": {
     ...
  }
}
```

QuestionsQuery.kt

```
data class Data(
  val questions: List<Question>
) {
  data class Question(...)
}
```

Apollo Android: modèle généré 😅

```
class GetQuestions2Query : Query<GetQuestions2Query.Data, GetQuestions2Query.Data,
    Operation.Variables> {
  override fun operationId(): String = OPERATION_ID
  override fun queryDocument(): String = QUERY_DOCUMENT
  override fun wrapData(data: Data?): Data? = data
  override fun variables(): Operation.Variables = Operation.EMPTY_VARIABLES
  override fun name(): OperationName = OPERATION_NAME
  override fun responseFieldMapper(): ResponseFieldMapper<Data> = ResponseFieldMapper.invoke {
    Data(it)
  }

  @Throws(IOException::class)
  override fun parse(source: BufferedSource, scalarTypeAdapters: ScalarTypeAdapters): Response<Data>
      = SimpleOperationResponseParser.parse(source, this, scalarTypeAdapters)

  @Throws(IOException::class)
  override fun parse(byteString: ByteString, scalarTypeAdapters: ScalarTypeAdapters): Response<Data>
      = parse(Buffer().write(byteString), scalarTypeAdapters)

  @Throws(IOException::class)
  override fun parse(source: BufferedSource): Response<Data> = parse(source, DEFAULT)

  @Throws(IOException::class)
  override fun parse(byteString: ByteString): Response<Data> = parse(byteString, DEFAULT)

  override fun composeRequestBody(scalarTypeAdapters: ScalarTypeAdapters): ByteString =
      OperationRequestBodyComposer.compose(
    operation = this,
    autoPersistQueries = false,
    withQueryDocument = true,
    scalarTypeAdapters = scalarTypeAdapters
  )

  override fun composeRequestBody(): ByteString = OperationRequestBodyComposer.compose(
    operation = this,
    autoPersistQueries = false,
    withQueryDocument = true,
    scalarTypeAdapters = DEFAULT
  )

  override fun composeRequestBody(
    autoPersistQueries: Boolean,
    withQueryDocument: Boolean,
    scalarTypeAdapters: ScalarTypeAdapters
  ): ByteString = OperationRequestBodyComposer.compose(
    operation = this,
    autoPersistQueries = autoPersistQueries,
    withQueryDocument = withQueryDocument,
    scalarTypeAdapters = scalarTypeAdapters
  )

  data class Question(
    val __typename: String = "Question",
    val id: Int,
    val content: String,
    val userVoteType: UpvoteType
  ) {
    fun marshaller(): ResponseFieldMarshaller = ResponseFieldMarshaller.invoke { writer ->
      writer.writeString(RESPONSE_FIELDS[0], this@Question.__typename)
      writer.writeInt(RESPONSE_FIELDS[1], this@Question.id)
      writer.writeString(RESPONSE_FIELDS[2], this@Question.content)
      writer.writeString(RESPONSE_FIELDS[3], this@Question.userVoteType.rawValue)
    }

    companion object {
      private val RESPONSE_FIELDS: Array<ResponseField> = arrayOf(
          ResponseField.forString("__typename", "__typename", null, false, null),
          ResponseField.forInt("id", "id", null, false, null),
          ResponseField.forString("content", "content", null, false, null),
          ResponseField.forEnum("userVoteType", "userVoteType", null, false, null)
          )

      operator fun invoke(reader: ResponseReader): Question = reader.run {
        val __typename = readString(RESPONSE_FIELDS[0])!!
```

Apollo Android: modèle généré

```
class GetQuestionsQuery : Query<> {
  data class Data(
    val questions: List<Question>
  ) : Operation.Data {
    // ...
  }

  data class Question(
    val __typename: String = "Question",
    val id: Int,
    val content: String,
    val votes: Int
  ) {
    // ...
  }
  //

}
```

Apollo Android: modèle généré

```
class GetQuestionsQuery : Query<> {

  // ...

  companion object {
    val QUERY_DOCUMENT: String = QueryDocumentMinifier.minify(
          """
          |query getQuestions2 {
          |  questions {
          |    __typename
          |    id
          |    content
          |    userVoteType
          |  }
          |}
          """.trimMargin()
        )
  }
}
```

Apollo Android: execution

```
val apolloClient = ApolloClient.builder()
    .serverUrl("https://stark-river-82454.herokuapp.com/graphql")
    .build()
```

Apollo Android: execution

```
val apolloClient = ApolloClient.builder()
    .serverUrl("https://stark-river-82454.herokuapp.com/graphql")
    .build()


val response = apolloClient.query(GetQuestionsQuery()).toDeferred().await()
```

Apollo Android: execution

```
val apolloClient = ApolloClient.builder()
    .serverUrl("https://stark-river-82454.herokuapp.com/graphql")
    .build()


val response = apolloClient.query(GetQuestionsQuery()).toDeferred().await()


response.data?.questions?.forEach {
    println("${it.id}: ${it.content}")
}


// 1: Est-ce que ça fonctionne ?
// 2: Est-ce que ça fonctionne pour de vrai ?
```

Apollo Android: execution

```
val apolloClient = ApolloClient.builder()
    .serverUrl("https://stark-river-82454.herokuapp.com/graphql")
    .build()


val response = apolloClient.query(GetQuestionsQuery()).toDeferred().await()


response.data?.questions?.forEach {
    println("${it.id}: ${it.content}")
    // it.id is non-nullable
    // same for it.content
}
```

Apollo Android: mutations

```
// GraphQL
mutation createQuestion($content: String!) {
    createQuestion(content: $content) {
        id
    }
}

// Kotlin
val mutation = CreateQuestionMutation("Est-ce que ça fonctionne ?")
val response = apolloClient.mutate(mutation)
        .toDeferred()
        .await()
```

Apollo Android: subscriptions

```
// GraphQL
subscription QuestionChange {
    questionChange {
        id
    }
}

// Kotlin
val response = apolloClient.subscribe(QuestionChangeSubscription())
        .toFlow()
        .collect {
           println("received ${it.data?.questionChanges.size} changes")
        }
```

Le client Kommentaire

Pour aller plus loin

Authentification

```
private class AuthorizationInterceptor(val context: Context): Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val request = chain.request().newBuilder()
            .addHeader("Authorization", User.getToken(context) ?: "")
            .build()
        return chain.proceed(request)
    }
}
```

Authentification

```
private class AuthorizationInterceptor(val context: Context): Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val request = chain.request().newBuilder()
            .addHeader("Authorization", User.getToken(context) ?: "")
            .build()
        return chain.proceed(request)
    }
}

ApolloClient.builder()
        .serverUrl("https://apollo-fullstack-tutorial.herokuapp.com/")
        .okHttpClient(OkHttpClient.Builder()
            .addInterceptor(AuthorizationInterceptor(context))
            .build()
        )
        .build()
```

Fragments

```
// GraphQL
fragment questionFragment on Question {
    content
    votes
}
query getQuestions {
    questions {
        ...questionFragment
    }
}

// Kotlin
response.data?.questions?.forEach {
    println(it.fragments.questionFragment.content)
}
```

Fragments

```
// GraphQL
subscription QuestionChange {
    questionChange {
        ...questionFragment
    }
}
```

```
// Kotlin
response.data?.questions?.forEach {
    println(it.fragments.questionFragment.content)
}
```

Inline Fragments

```
// GraphQL
query GetUser {
    … on iOSUser {
        iPhoneModel
    }
    … on AndroidUser {
        deviceBrand
        deviceModel
    }
}

// Kotlin
data.asIOSUser?.iPhoneModel
data.asAndroidUser?.deviceBrand
```

Depreciation

```
query {
  pinnedRepositories {
    nodes {
      id
    }
  }
}
```

```
  /**
   * A list of repositories this user
has pinned to their profile
   */
  @Deprecated(message =
      "pinnedRepositories will be
removed Use ProfileOwner.pinnedItems
instead. Removal on 2019-10-01 UTC.")
  val pinnedRepositories:
                  PinnedRepositories
```

Persisted queries

```
query {
  viewer {
    login
    bio
    status {
      emoji
      message
    }
  }
}
```

```
“OperationId”: viewerQuery
```

Cache

- Basé sur SqlDelight
- Cache normalisé
- Offline-ready

Compilateur

- Basé sur un parser Antlr
- Utilise javapoet et kotlinpoet

Multiplatform (experimental)

Gradle Plugin

Compiler

Api

Runtime

Normalized Cache

HTTP Cache

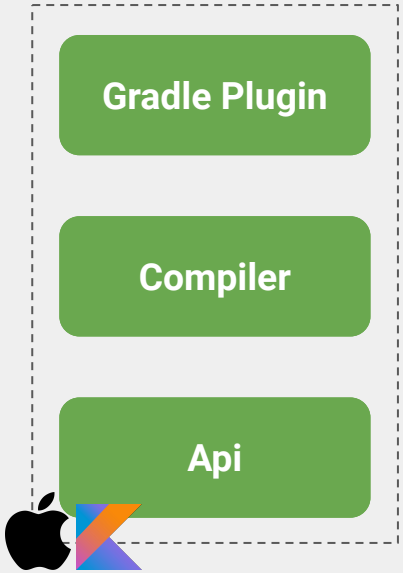
RxJava support

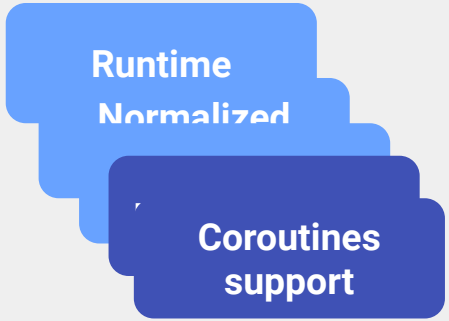
Coroutines support

Espresso support

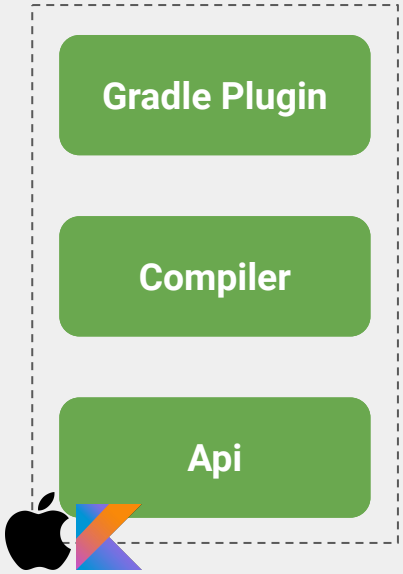
...

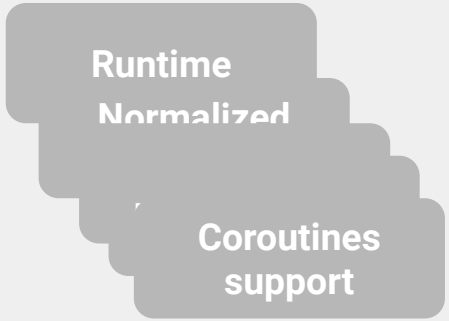
Multiplatform (experimental)





Multiplatform (experimental)

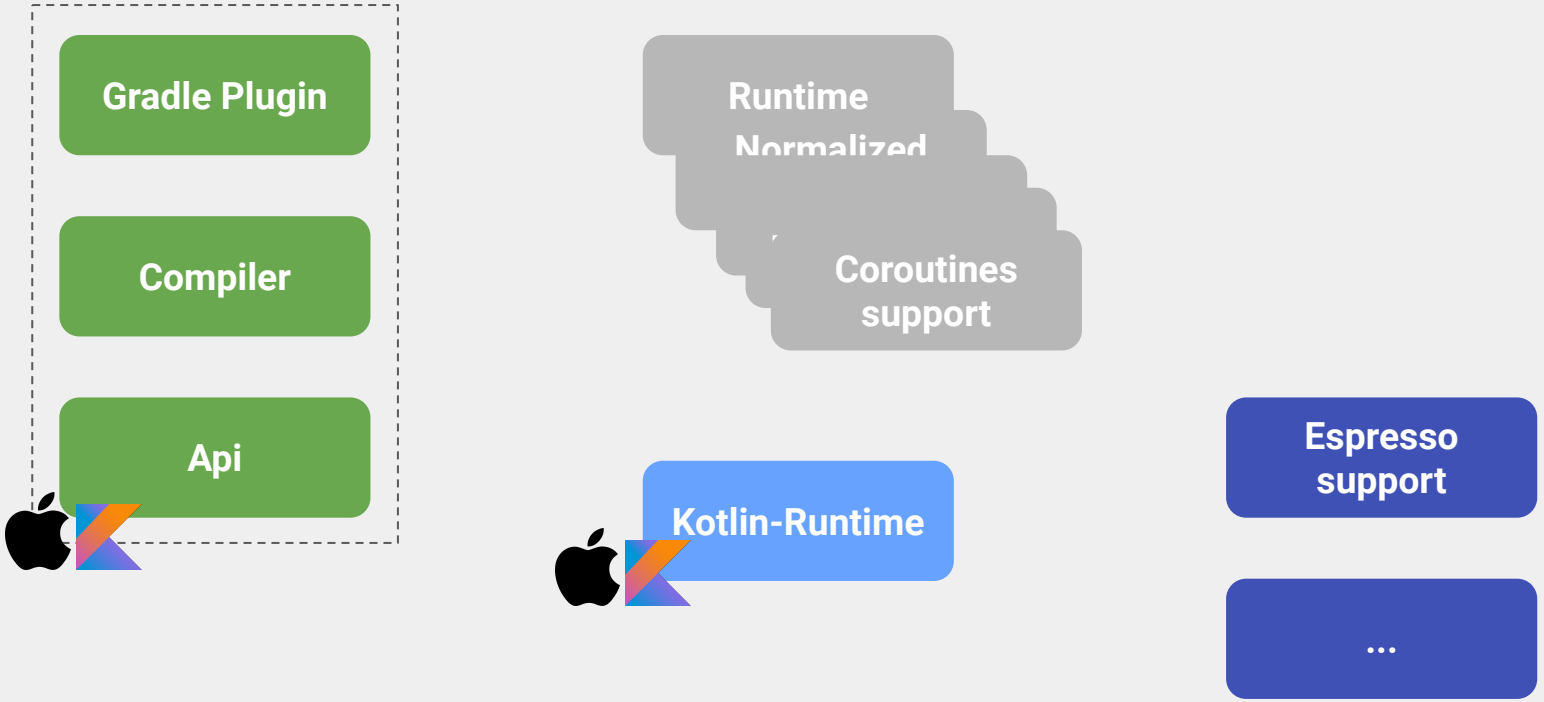




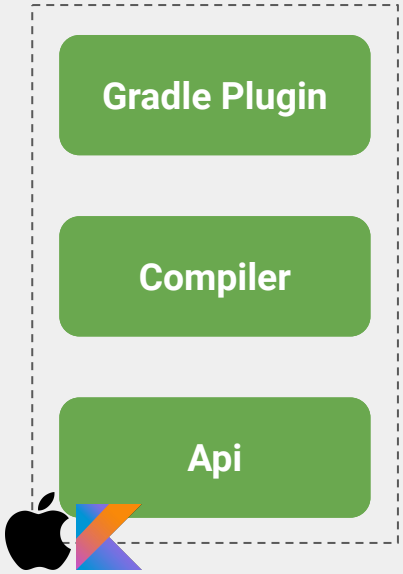
Espresso support

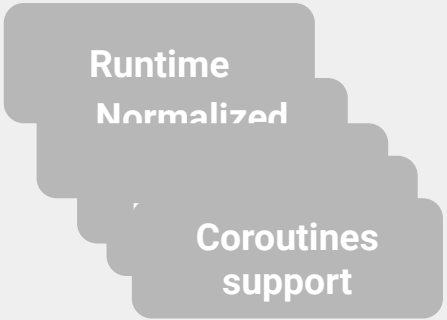
...

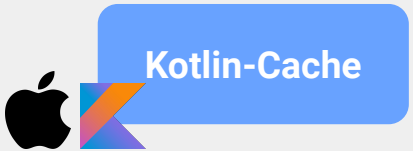
Multiplatform (experimental)



Multiplatform (experimental)







Merci !

@martinbonnin