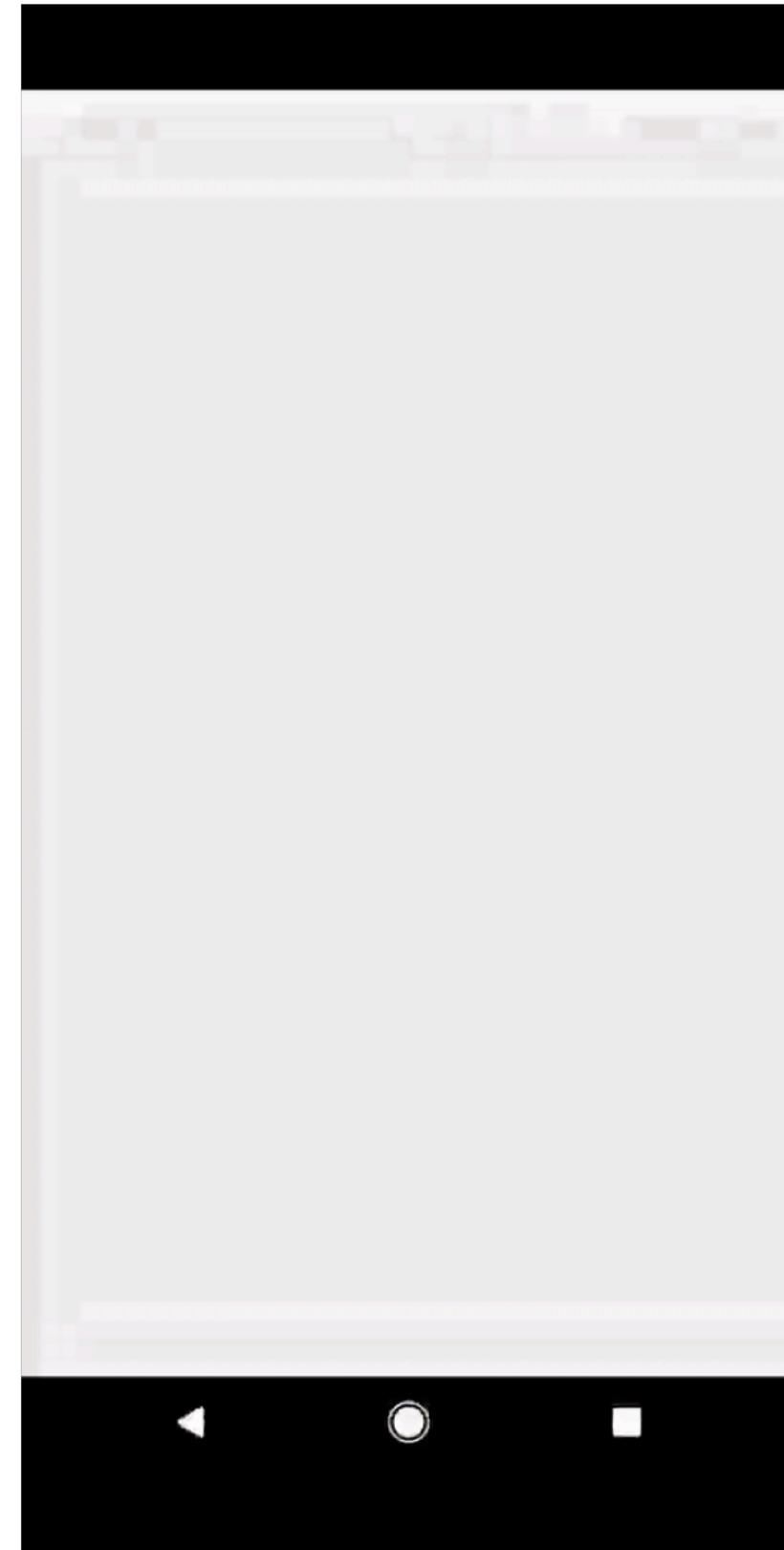


```
Query DroidConNYC{ slide(id: "1")
  {
    Title
    Authors
    Company
  }
}

{
  Title: "Intro to GraphQL on Android",
  Authors: ["Brian Plummer", "Mike Nakhimovich"],
  Company: "The New York Times"
}
```

We work at The
New York Times

We do **a lot** of data
loading



Recently, our team moved from using RESTful APIs to GraphQL for our data loading architecture

What's GraphQL?

- A query language for APIs and a runtime for fulfilling those queries with your existing data
- Alternative to RESTful API
- Client driven - get only data you need
- Works on iOS, Android, Web

GraphQL was created by Facebook as a new standard for server/client data transfer

- Give front end developers an efficient way to ask for minimal data
- Give server-side developers a robust way to get their data out to their users

GraphQL is As Easy as 1-2-3

- Describe your data
- Ask for what you want
- Get predictable results

Describe Your Data in a Schema

```
type Character {  
    name: String!  
    appearsIn: [Episode]!  
}
```

Describe Your Data in a Schema

```
type Character {  
    name: String!  
    appearsIn: [Episode]!  
}
```

Character is a GraphQL Object Type, meaning it's a type with some fields. Most of the types in your schema will be object types.

Describe Your Data in a Schema

```
type Character {  
    name: String!  
    appearsIn: [Episode]!  
}
```

name and **appearsIn** are fields on the Character type. That means that name and appearsIn are the only fields that can appear in any part of a GraphQL query that operates on the Character type.

Describe Your Data in a Schema

```
type Character {  
  name: String!  
  appearsIn: [Episode]!  
}
```

String is one of the built-in scalar types. These are types that resolve to a single scalar object and can't have sub-selections in the query.

GraphQL Example Schema

```
type Character {  
    name: String!  
    appearsIn: [Episode]!  
}
```

String! means that the field is non-nullable, meaning that the GraphQL service promises to always give you a value when you query this field.

GraphQL Example Schema

```
type Character {  
    name: String!  
    appearsIn: [Episode]!  
}
```

[Episode]! represents an array of Episode objects. Since it is also non-nullable, you can always expect an array (with zero or more items) when you query the appearsIn field.

Ask for what you need

```
{  
  hero {  
    name  
    heig█  
  }  
}  
  
{  
  "hero": {  
    "name": "Luke Skywalker"  
  }  
}
```

get predictable results

Combine Resources into One Request

```
{  
  hero {  
    name  
    # Queries can have comments  
    friends {  
      name  
    }  
  }  
}
```

Combine Resources into One Request

```
{  
  hero {  
    name  
    # Queries can have comments  
    friends {  
      name  
    }  
  }  
}  
  
{  
  "data": {  
    "hero": {  
      "name": "R2-D2",  
      "friends": [  
        {  
          "name": "Luke Skywalker"  
        },  
        {  
          "name": "Han Solo"  
        },  
        {  
          "name": "Leia Organa"  
        }  
      ]  
    }  
  }  
}
```

Reuse Fields in a Fragment

```
{  
  leftColumn: hero(episode: EMPIRE) {  
    ...comparisonFields  
  }  
  rightColumn: hero(episode: JEDI) {  
    ...comparisonFields  
  }  
}  
  
fragment comparisonFields on Character {  
  name  
  appearsIn  
}
```

Reuse Fields in a Fragment

```
{  
  leftColumn: hero(episode: EMPIRE) {  
    ...comparisonFields  
  }  
  rightColumn: hero(episode: JEDI) {  
    ...comparisonFields  
  }  
}  
  
fragment comparisonFields on Character {  
  name  
  appearsIn  
}
```

```
{  
  "data": {  
    "leftColumn": {  
      "name": "Luke Skywalker",  
      "appearsIn": [  
        "NEWHOPE",  
        "EMPIRE",  
        "JEDI"  
      ]  
    },  
    "rightColumn": {  
      "name": "R2-D2",  
      "appearsIn": [  
        "NEWHOPE",  
        "EMPIRE",  
        "JEDI"  
      ]  
    }  
  }  
}
```



Example:

Loading Data from
Github Using REST
vs GraphQL

On any platform, data loading consists of these steps:

1. Model Data
2. Network
3. Transform
4. Persist

How does REST look on Android?

...It Looks Like a lot of Dependencies

Data Modeling Networking Storage Transform

Immutables OKhttp Store Gson

Curl Retrofit SqlDelight RxJava

Yes, those are all needed 

```
,  
{  
  "url": "https://api.github.com/repos/vmg/redcarpet/issues/607",  
  "repository_url": "https://api.github.com/repos/vmg/redcarpet",  
  "labels_url": "https://api.github.com/repos/vmg/redcarpet/issues/607/labels{/name}",  
  "comments_url": "https://api.github.com/repos/vmg/redcarpet/issues/607/comments",  
  "events_url": "https://api.github.com/repos/vmg/redcarpet/issues/607/events",  
  "html_url": "https://github.com/vmg/redcarpet/issues/607",  
  "id": 211546203,  
  "number": 607,  
  "title": "How to even rendering `entities`? What how when?",  
  "user": {  
    "login": "dgsan",  
    "id": 313910,  
    "avatar_url": "https://avatars2.githubusercontent.com/u/313910?v=4",  
    "gravatar_id": "",  
    "url": "https://api.github.com/users/dgsan",  
    "html_url": "https://github.com/dgsan",  
    "followers_url": "https://api.github.com/users/dgsan/followers",  
    "following_url": "https://api.github.com/users/dgsan/following{/other_user}",  
    "gists_url": "https://api.github.com/users/dgsan/gists{/gist_id}",  
    "starred_url": "https://api.github.com/users/dgsan/starred{/owner}{/repo}",  
    "subscriptions_url": "https://api.github.com/users/dgsan/subscriptions",  
    "organizations_url": "https://api.github.com/users/dgsan/orgs",  
    "repos_url": "https://api.github.com/users/dgsan/repos",  
    "events_url": "https://api.github.com/users/dgsan/events{/privacy}",  
    "received_events_url": "https://api.github.com/users/dgsan/received_events",  
    "type": "User",  
    "site_admin": false  
},  
}
```

Start with Inspection

`curl -i "https://api.github.com/repos/vmg/redcarpet/issues?state=closed"`

Model Your Data

```
interface Issue {  
    User user();  
    String url();  
}
```

```
interface User {  
    long id();  
    String name();  
}  
}
```

Error prone even with code generation

Data Modeling with Immutables

```
@Value.Immutable
interface Issue {
    User user();
    String url();

    @Value.Immutable
    interface User {
        long id();
        String name();
    }
}
```

Error Prone even with Code Generation

Data Parsing with Gson

```
@Gson.TypeAdapters  
@Value.Immutable  
interface Issue {  
    User user();  
    String url();  
  
    @Value.Immutable  
    interface User {  
        long id();  
        String name();  
    }  
}
```

Networking

```
open fun provideRetrofit(gson: Gson, okHttpClient: OkHttpClient): GithubApi {  
    return Retrofit.Builder()  
        .client(okHttpClient)  
        .baseUrl(BuildConfig.BASE_URL)  
        .addConverterFactory(GsonConverterFactory.create(gson))  
        .addCallAdapterFactory(RxJavaCallAdapterFactory.create())  
        .build()  
        .create(GithubApi::class.java!!)}
```

Storage

```
CREATE TABLE issue (
    _id LONG PRIMARY KEY AUTOINCREMENT,
    id LONG NOT NULL,
    url STRING,
    title STRING,
    comments INT NOT NULL
}
```

Storage

```
public abstract class Issue implements IssueModel {  
    public static final Mapper<Issue> MAPPER =  
        new Mapper<>((Mapper.Creator<Issue>)  
            ImmutableIssue::of);  
  
    public static final class Marshal extends IssueMarshal {  
    }  
}
```

Storage

```
long insertIssue(Issue issue) {
    if (recordExists(Issue.TABLE_NAME, Issue.ID, String.valueOf(issue.id()))) {
        return 0;
    }

    return db.insert(Issue.TABLE_NAME, new Issue.Marshal()
        .url(issue.url())
        .id(issue.id())));
}
```

Storage - Memory

```
StoreBuilder.parsedWithKey<GitHubOrgId, BufferedSource, Issues>()
    .fetcher(fetcher)
    .persister(persister)
    .parser(parser)
    .memoryPolicy(MemoryPolicy
        .builder()
        .setMemorySize(11L)
        .setExpireAfterWrite(TimeUnit.HOURS.toSeconds(24))
        .setExpireAfterTimeUnit(TimeUnit.SECONDS)
        .build())
    .networkBeforeStale()
    .open()
```

A grayscale photograph of a person from the chest up. They are wearing a light-colored hoodie over a dark shirt and glasses. Their hands are on a laptop keyboard, and they are looking down at the screen. The background is plain and light.

That's a Good Architecture
It's also not something we can
expect a beginner to know

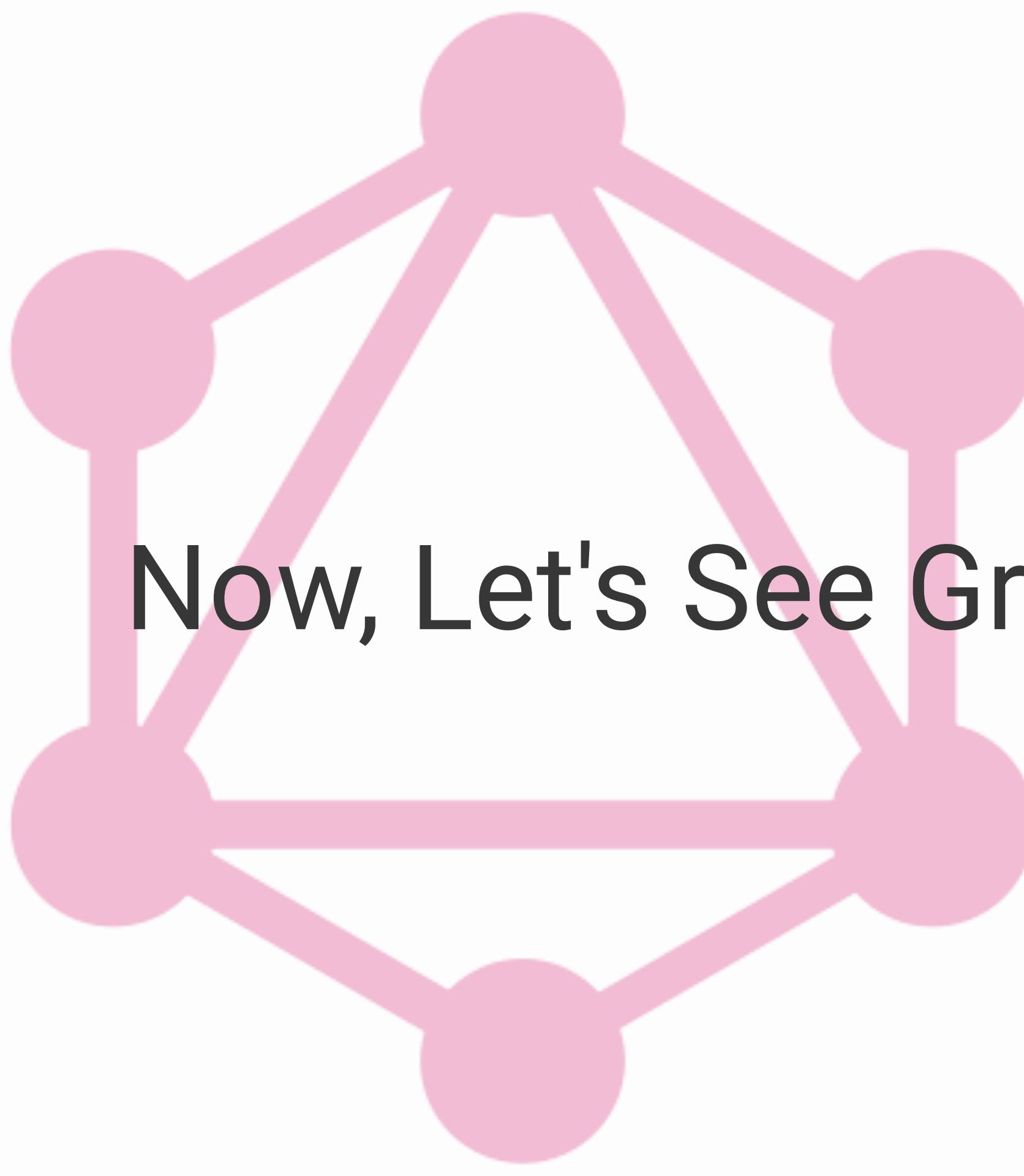
REST Feels Like Legacy Tech

Well-established and with great tools, but
cumbersome to work with



Thanks to Facebook, there's a new kid on the block

GraphQL



Now, Let's See GraphQL on Android

About 361,000 results (0.61 seconds)

Calling the Graph API - Android SDK - Facebook for Developers

<https://developers.facebook.com/docs/android/graph/> ▾

The Android SDK has support for integrating with Facebook's Graph API. With the Graph Request and Graph Response classes you can make requests and get responses.

GraphQL: A data query language | Engineering at Facebook | Code...
<https://code.facebook.com/posts/1611550941094/graphql-a-query-language/> ▾

This year we've begun the process of open-sourcing GraphQL by drafting a ... At the time, our iOS and
Android apps were thin wrappers around views of our ...

for your time and goodnight 😞

Android SDK - Facebook for Developers

<https://developers.facebook.com/docs/android/> ▾

Facebook SDK for Android ... Requires Android API 15. ... Graph API ... The Audience Network allows
you to monetize your Android apps with Facebook ads.

Getting Started · Android - Facebook Login · Changelog · Downloads

GraphQL Clients for iOS or Android · Issue #180 · facebook/graphql ...

<https://github.com/facebook/graphql/issues/180> ▾

May 24, 2016 - graphql - GraphQL is a query language and execution engine tied to any backend service.

All

News

Videos

Images

Shopping

More

Settings

Tools

About 361,000 results (0.61 seconds)

Calling the Graph API - Android SDK - Facebook for Developers

<https://developers.facebook.com/docs/android/graph/> ▾

The Android SDK has support for integrating with **Facebook Graph API**. With the GraphRequest and GraphResponse classes, you can make requests and get ...

Just Kiddin, Community to the Rescue!

This year we've begun the process of open-sourcing **GraphQL** by drafting a ... At the time, our iOS and Android apps were thin wrappers around views of our ...

Android SDK - Facebook for Developers

<https://developers.facebook.com/docs/android/> ▾

Facebook SDK for Android ... Requires **Android API 15**. ... **Graph API** ... The Audience Network allows you to monetize your **Android** apps with **Facebook ads**.

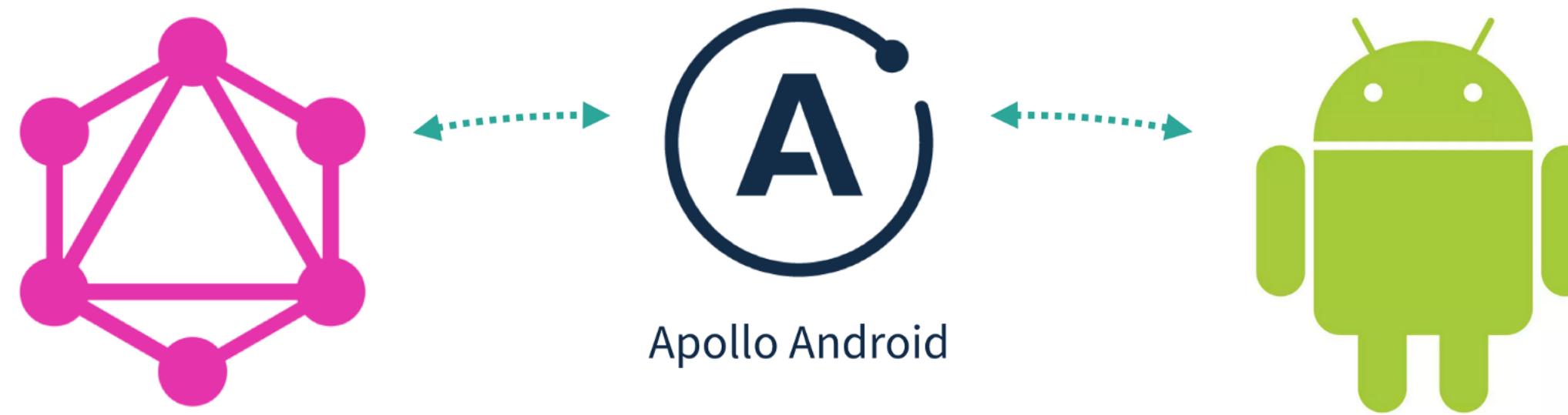
[Getting Started](#) · [Android - Facebook Login](#) · [Changelog](#) · [Downloads](#)

GraphQL Clients for iOS or Android · Issue #180 · facebook/graphql ...

<https://github.com/facebook/graphql/issues/180> ▾

May 24, 2016 - **graphql** - **GraphQL** is a query language and execution engine tied to any backend service.

Introducing Apollo-Android GraphQL



Apollo Android was developed by Shopify, New York Times, & Airbnb as an **Open Source** GraphQL solution.

Apollo-Android

- Built by Android devs for Android devs
- A strongly-typed, caching GraphQL client for Android
- Rich support for types and type mappings
- Code generation for the messy parts
- Query validation at compilation

Meets Facebook's GraphQL Spec

- Works with any GraphQL Query
- Fragments
- Union Types
- Nullability
- Deprecation

Apollo Reduces Setup to Work with a Backend

Data Modeling	Networking	Storage	Transform
Github Explorer	OKhttp	Apollo	RxJava
Apollo	Apollo	Apollo	Apollo

You Ain't Gonna Need It

~~Retrofit | Immutables| Gson | Guava | SqlDelight/Brite | Store | Curl
| JsonViewer.hu~~

Apollo-Android Has 2 Main Parts

- **Gradle Plugin** Apollo code generation plugin
- **Runtime** Apollo client for executing operations

A close-up photograph of a young child with blonde hair, wearing a green shirt. The child is looking slightly to the side with a thoughtful expression. The background is blurred.

Using Apollo-Android like a boss

Add Apollo Dependencies

```
build.gradle:  
dependencies {  
    classpath 'com.apollographql.apollo:gradle-plugin:0.4.1'  
}  
  
app/build.gradle:  
apply plugin: 'com.apollographql.android'  
....  
//optional RxSupport  
compile 'com.apollographql.apollo:apollo-rx-support:0.4.1'
```

Create a Standard GraphQL Query

Queries have params and define shape of response

```
organization(login:"nyTimes"){  
  repositories(first:6) {  
    Name  
  }  
}
```

Leave Your CURL at Home

Most GraphQL Servers have a GUI (GraphiQL)

<https://developer.github.com/v4/explorer/>

GraphiQL: Explore Schema and Build Queries

- Shape of Response
- Nullability Rules
- Enum values
- Types

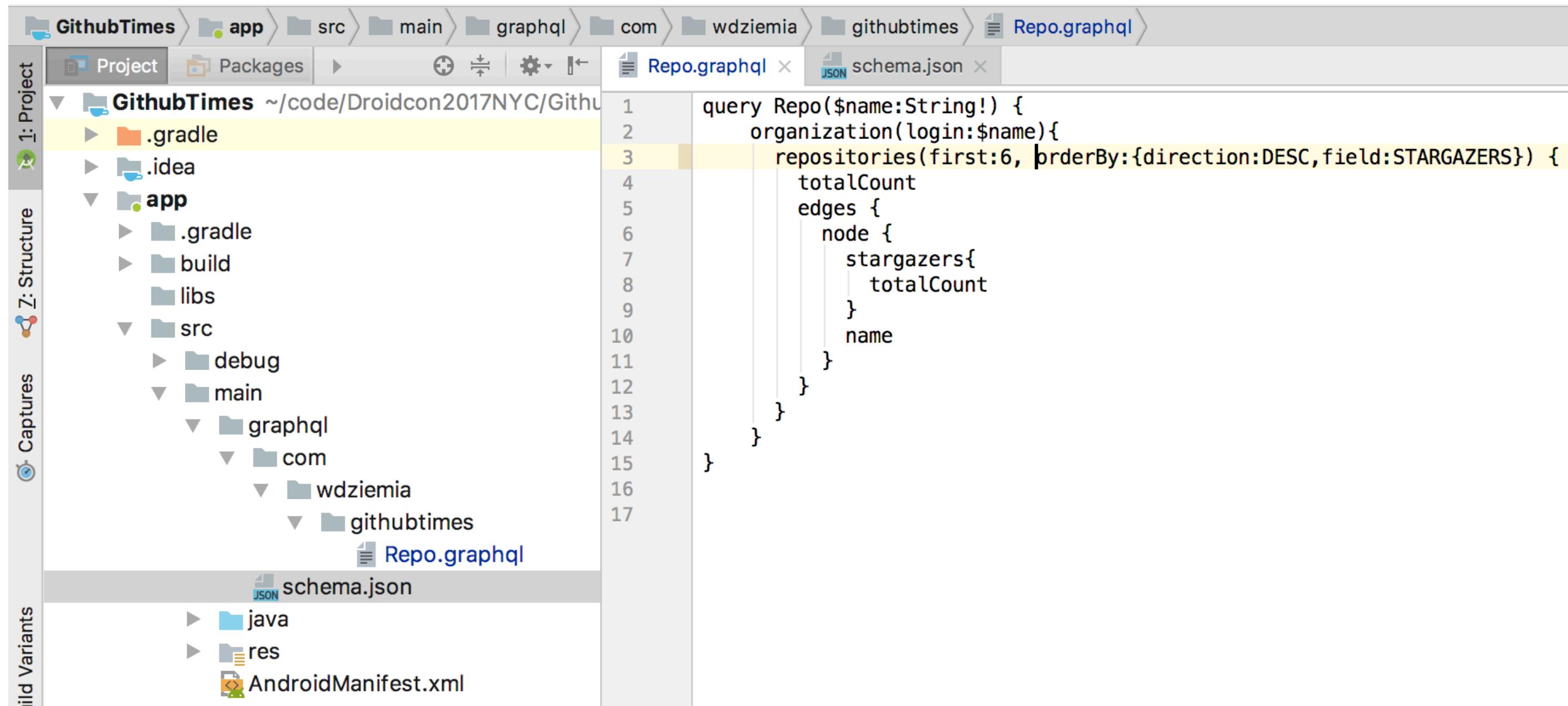
GraphiQL is easy!

The screenshot shows the GraphiQL interface with the following layout:

- Header:** "GraphiQL" button, a play button icon, "Prettify" button, and a "Docs" link.
- Left Panel:** A text input field containing the number "1".
- Right Panel:** A JSON response with syntax highlighting. The response is:

```
{  
  "data": null,  
  "errors": [  
    {  
      "message": "A query attribute must be specified and  
      must be a string."  
    }  
  ]  
}
```

Add Schema & RepoQuery.graphql to project & compile



A close-up, profile view of a person's head and shoulders. The person has short, dark brown hair and is wearing dark sunglasses. They are wearing a light-colored, possibly grey or blue, hoodie. The background is plain and light-colored.

Apollo Writes Code So You Don't Have To

```
private fun CodeGenerationIR.writeJavaFiles(context: CodeGenerationContext, outputDir: File,  
    outputPackageName: String?) {  
    fragments.forEach {  
        val typeSpec = it.toTypeSpec(context.copy())  
        JavaFile.builder(context.fragmentsPackage, typeSpec).build().writeTo(outputDir)  
    }  
  
    typesUsed.supportedTypeDeclarations().forEach {  
        val typeSpec = it.toTypeSpec(context.copy())  
        JavaFile.builder(context.typesPackage, typeSpec).build().writeTo(outputDir)  
    }  
  
    if (context.customTypeMap.isNotEmpty()) {  
        val typeSpec = CustomEnumTypeSpecBuilder(context.copy()).build()  
        JavaFile.builder(context.typesPackage, typeSpec).build().writeTo(outputDir)  
    }  
  
    operations.map { OperationTypeSpecBuilder(it, fragments, context.useSemanticNaming) }  
        .forEach {  
            val packageName = outputPackageName ?: it.operation.filePath.formatPackageName()  
            val typeSpec = it.toTypeSpec(context.copy())  
            JavaFile.builder(packageName, typeSpec).build().writeTo(outputDir)  
        }  
}
```

Actually Ivan(sav007) Does (He's Awesome)

Builder - For Creating Your Request Instance

```
//api
val query = RepoQuery.builder.name("nytimes").build()

//Generated Code
public static final class Builder {
    private @Nonnull String name;

    Builder() {
    }

    public Builder name(@Nonnull String name) {
        this.name = name;
        return this;
    }

    public RepoQuery build() {
        if (name == null) throw new IllegalStateException("name can't be null");
        return new RepoQuery(name);
    }
}
```

Notice How Our Request Param name is Validated

```
//api
val query = RepoQuery.builder.name("nytimes").build()

//Generated Code
public static final class Builder {
    private @Nonnull String name;

    Builder() {}

    public Builder name(@Nonnull String name) {
        this.name = name;
        return this;
    }

    public RepoQuery build() {
        if (name == null) throw new IllegalStateException("name can't be null");
        return new RepoQuery(name);
    }
}
```

Response Models

```
public static class Repositories {  
    final @Nonnull String __typename;  
    final int totalCount;  
    final @Nullable List<Edge> edges;  
    private volatile String $toString;  
    private volatile int $hashCode;  
    private volatile boolean $hashCodeMemoized;  
  
    public @Nonnull String __typename() { return this.__typename; }  
  
    //Identifies the total count of items in the connection.  
    public int totalCount() {return this.totalCount;}  
  
    //A list of edges.  
    public @Nullable List<Edge> edges() {return this.edges;}  
  
    @Override  
    public String toString() {...}  
  
    @Override  
    public boolean equals(Object o) { ... }  
  
    @Override  
    public int hashCode() {...}
```

Mapper - Reflection-Free Parser

```
public static final class Mapper implements ResponseFieldMapper<Repositories> {
    final Edge.Mapper edgeFieldMapper = new Edge.Mapper();

    @Override
    public Repositories map(ResponseReader reader) {
        final String __typename = reader.readString($responseFields[0]);
        final int totalCount = reader.readInt($responseFields[1]);
        final List<Edge> edges = reader.readList($responseFields[2], new ResponseReader.ListReader<Edge>() {
            @Override
            public Edge read(ResponseReader.ListItemReader reader) {
                return reader.readObject(new ResponseReader.ObjectReader<Edge>() {
                    @Override
                    public Edge read(ResponseReader reader) {
                        return edgeFieldMapper.map(reader);
                    }
                });
            }
        });
        return new Repositories(__typename, totalCount, edges);
    }
}
```

Can parse 20MB Response w/o OOM

Querying Github's API With Apollo Client

Building an Apollo Client

```
ApolloClient.builder()  
    .serverUrl("https://api.github.com/graphql")  
    .okHttpClient(okhttp)  
    .build();
```

Querying a Backend

```
query = RepoQuery.builder().name("nytimes").build()
```

Querying a Backend

```
query = RepoQuery.builder().name("nytimes").build()
```

```
ApolloQueryCall githubCall = apolloClient.query(query);
```

Querying a Backend

```
query = RepoQuery.builder().name("nytimes").build()

ApolloQueryCall githubCall = apolloClient.query(query);

githubCall.enqueue(new ApolloCall.Callback<>() {
    @Override
    public void onResponse(@Nonnull Response<> response) {
        handleResponse(response);
    }

    @Override
    public void onFailure(@Nonnull ApolloException e) {
        handleFailure(e);
    }
});
```

Apollo also Handles Storage

Storage with Apollo is done through Caches

- HTTP
- Normalized

HTTP Caching

- Similar to OKHTTP Cache (LRU)
- Streams response to cache same time as parsing
- Can Set Max Cache Size
- Useful for background updating to prefill cache
- Prefetching

```
apolloClient.prefetch(new RepoQuery("nytimes"));
```

HTTP Caching - as well as you can do in REST
Apollo Introduces a Normalized Cache

Apollo Store

Apollo Store

- Allows multiple queries to share same cached values
- Great for things like master/detail
- Caching is done post-parsing
- Each field is cached individually
- Apollo ships with both an in memory and a disk implementation of an Apollo Store
- You can even use both at same time

How Does Apollo Store Work?

- Each Object in Response will have its own record with ID
- All Scalars/Members will be merged together as fields
- When we are reading from Apollo, it will seamlessly read from Apollo Store or network

Setup Bi-Level Caching with Apollo Store

```
//Create DB
ApolloSqlHelper apolloSqlHelper = ApolloSqlHelper.create(context, "db_name");
//Create NormalizedCacheFactory
NormalizedCacheFactory normalizedCacheFactory = new LruNormalizedCacheFactory(EvictionPolicy.NO_EVICT)
                                         .chain(new SqlNormalizedCacheFactory(apolloSqlHelper));
```

Create a Cache Key Resolver

```
//Create the cache key resolver
CacheKeyResolver cacheKeyResolver = new CacheKeyResolver() {
    @Nonnull @Override
    public CacheKey fromFieldRecordSet(@Nonnull ResponseField field, @Nonnull Map<String, Object> recordSet) {
        String typeName = (String) recordSet.get("__typename");
        if (recordSet.containsKey("id")) {
            String typeNameAndIDKey = recordSet.get("__typename") + "." + recordSet.get("id");
            return CacheKey.from(typeNameAndIDKey);
        }
        return CacheKey.NO_KEY;
    }

    @Nonnull @Override
    public CacheKey fromFieldArguments(@Nonnull ResponseField field, @Nonnull Operation.Variables variables) {
        return CacheKey.NO_KEY;
    }
};
```

Init Apollo Client With a Cache

```
//Build the Apollo Client
ApolloClient apolloClient = ApolloClient.builder()
    .serverUrl("/")
    .normalizedCache(cacheFactory, resolver)
    .okHttpClient(okHttpClient)
    .build();
```

Don't Like Our Cache? BYO Cache

```
public abstract class NormalizedCache {  
  
    @Nullable public abstract Record loadRecord(@Nonnull String key, @Nonnull CacheHeaders cacheHeaders)  
  
    @Nonnull public Collection<Record> loadRecords(@Nonnull Collection<String> keys, @Nonnull CacheHeaders cacheHeaders)  
  
    @Nonnull public abstract Set<String> merge(@Nonnull Record record, @Nonnull CacheHeaders cacheHeaders)  
  
    public abstract void clearAll()  
  
    public abstract boolean remove(@Nonnull CacheKey cacheKey)
```

Apollo Is Reactive

QueryWatcher is a listener for changes to any fields in a query.

- When a dependent field changes in the Apollo Store, QueryWatcher will re-emit the response.
- QueryWatcher ~ endless subscription to a particular query.

Insert Example of a query watcher

Bonus: Includes RxJava Bindings

```
RxApollo.from(apolloClient.query(RepoQuery.builder().name("nytimes").build()))  
    .map(dataResponse -> dataResponse  
        .data()  
        .organization()  
        .repositories())  
    .subscribe(view::showRepositories, view::showError)
```

RxApollo response can be transformed into
LiveData

Version 1.0 ships soon!

- 380 commits
- 1000s of tests
- 18 contributors including devs from Shopify, Airbnb, NY Times
- Come join us at <https://github.com/apollographql/apollo-android>