

GROUND UP SERIES <https://graphql.org>

# GraphQL



## Syed Awase Khirni

RESEARCHER | ENTREPRENEUR | TECHNOLOGY COACH

@sak008 | [sak@sycliq.com](mailto:sak@sycliq.com) / [sak@territorialprescience.com](mailto:sak@territorialprescience.com) | +91. 9035433124

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project ([www.geo-spirit.org](http://www.geo-spirit.org)). He currently provides consulting services through his startup [www.territorialprescience.com](http://www.territorialprescience.com) and [www.sycliq.com](http://www.sycliq.com). He empowers the ecosystem by sharing his technical skills worldwide, since 2008. He provides training in Java Technology Stack, .Net Technology Stack, R, DataScience, Client Side frameworks (Angular, KnockOut, Aurelia, Vue, Ember, Backbone etc..), Node Stack, Machine Learning, Python Stack, Php Stack.

[www.territorialprescience.com](http://www.territorialprescience.com)

[www.sycliq.com](http://www.sycliq.com)



# Terms of Use

You shall not circulate these slides without written permission from Territorial Prescience Research I Pvt Ltd.

If you use any material, graphics or code or notes from these slides, you shall seek written permission from TPRI and acknowledge the author Dr. Syed Awase Khirni

If you have not received this material, post-training session, you shall destroy it immediately and not use it for unauthorized usage of the material. If any of the material, that has been shared is further used for any unauthorized training by the recipient, he shall be liable to be prosecuted for the damages. Any supporting material that has been provided by the author, shall not be used directly or indirectly without permission.

If this material, has been shared to any organization prior to the training and the organization does not award the contract to TPRI, it should not use the training material internally. If by any chance, the organization is using this training material without written permission, the organization is liable to pay for the damages to TPRI and is subjected to legal action, jurisdiction being Bangalore.

Any unauthorized usage, TPRI has right to claim damages ranging from USD 50000 to USD 10,0000 dollars as damages.

Any organization, which does not intend to go ahead with training or does not agree with the terms and conditions, should destroy the material from its network immediately. The burden of proof lies on the client, with whom this material has been shared.

Recovery of the damages and legal fees will be born by the client organization/candidate/party, which has violated the terms and conditions.

Only candidates who have attended the training session in person from Dr. Syed Awase Khirni, TPRI are entitled to hold this training material. They cannot further circulate it, or use it or morph it, or change it to provide trainings.

TPRI reserves all the rights to this material and code plays and right to modify them as and when it deems fit.

If you agree with the terms and conditions, please go ahead with using the training material. Else please close and destroy the slide and inform TPRI immediately.



# Slide Version Updates

Last Updated	Version	Release Date	Updated by	Code Plays Done @

Updated on March 2019

Please read terms of use for authorized access

Original Series



# GraphQL

- A query language for APIs and a runtime for fulfilling those queries with your existing data.
- An open-source data query and manipulation language for APIs; released by Facebook in 2015.
- Moved to GraphQL Foundation, hosted by the non-profit Linux Foundation on 7<sup>th</sup> Nov 2018.
- Created by Lee Byron, with a goal to make GraphQL omnipresent across web platforms.
- An approach to developing web APIs and has been compared and contrasted with REST and web service architectures.

Updated on March 2019

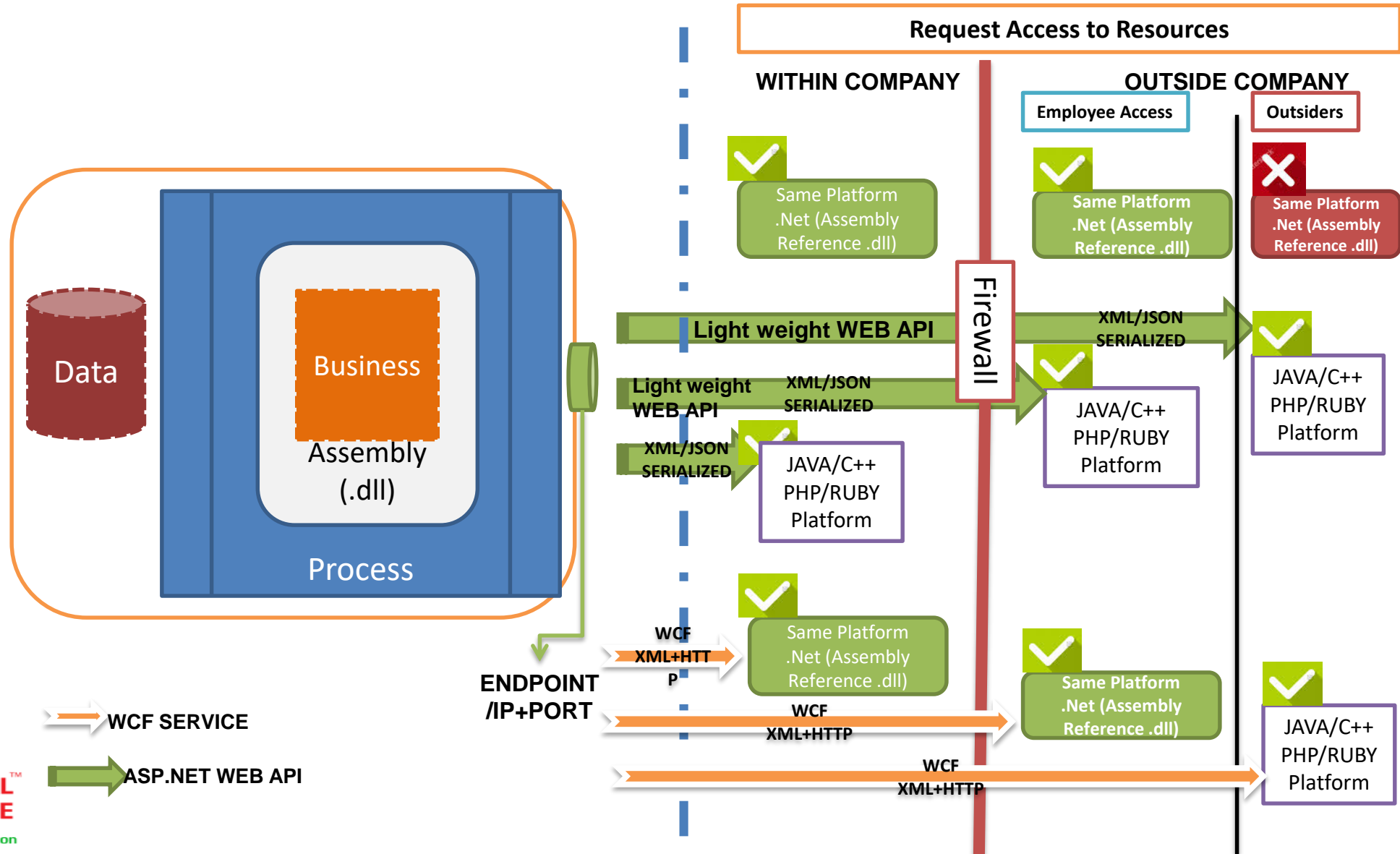


# GraphQL

- It allows clients to define the structure of the data required and the same structure of the data is returned from the server.
- It prevents from rendering excessive amount of data returned from the server.
- It has implications for how effective web caching of query results can be.
- It promises faster application and improved developer experience.
- It consists of a type system, query language and execution semantics, static validation and type introspection.
- GraphQL supports reading, writing(mutating) and subscribing to changes of data ( realtime updates-most commonly implemented using webhooks)
- GraphQL server are available for multiple languages: Haskell, JavaScript, Perl, Python, Ruby, Java, C++,C#, Scala, Go, Elixir, PHP,R and Clojure

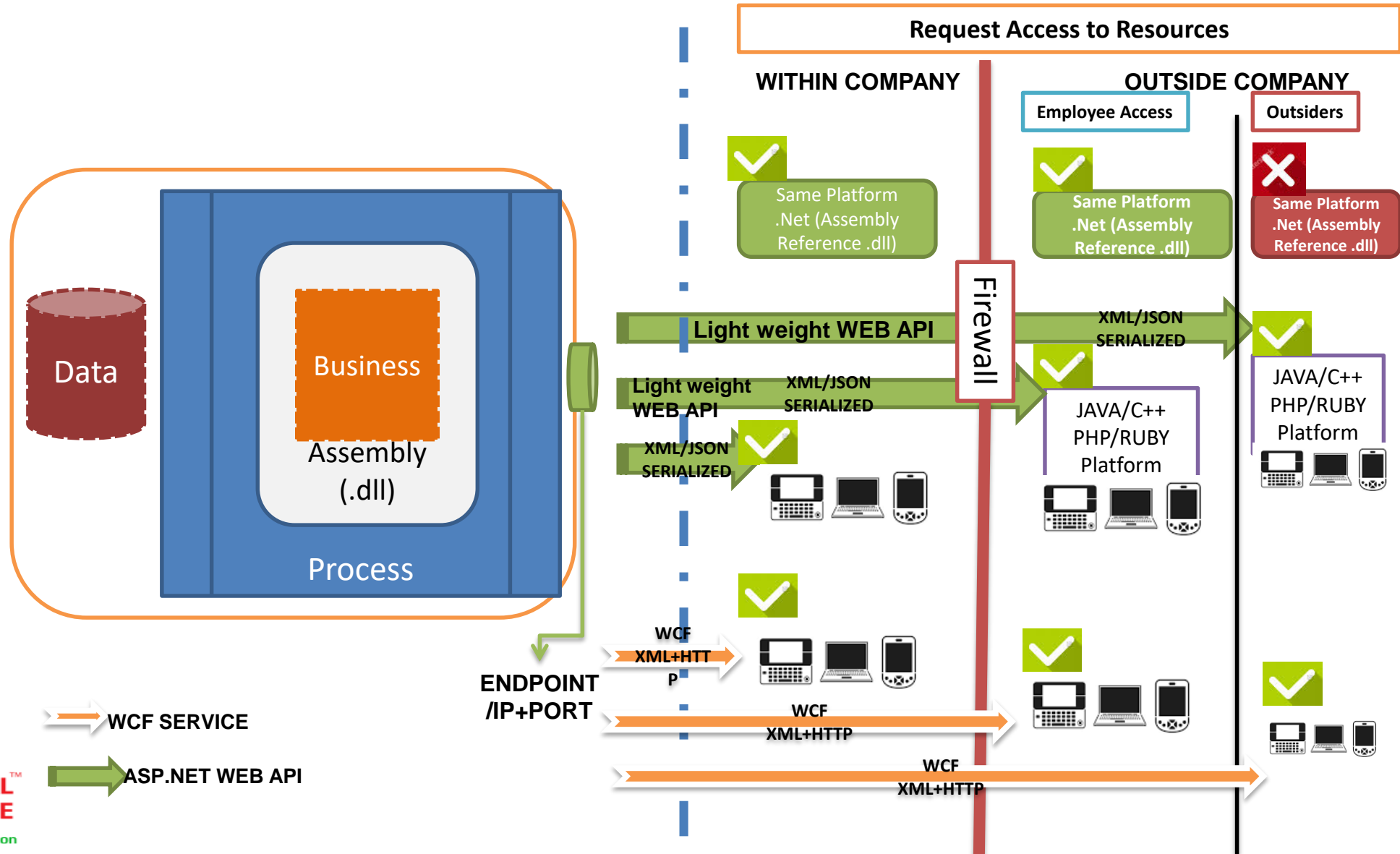


# ? NEED FOR WEB API/WCF





# ? NEED FOR WEB API/WCF





# REST Architectures/API Centric Architectures

- REST Architectures are valid solution when building API centric architectures.
- SOAP- Simple Object Access Protocol web service required complex tooling, hours of configuration, and very large manual.
- It required writing a WSDL(Web Service Description Language).
- REST invented in 2000, but saw wide adoption only in 2006/2007..
- Request and Response were easy to build and parse.
- REST being stateless and cacheable was the preferred approach for API centric architectures because of its lightweight.
- REST was an ideal choice for interoperability services across platforms that offered productivity, performance and maintainability.





# Changed Internet Landscape

- 2012-15 saw a jump in mobile internet consumption and has progressed steadily across middle-east, india and china.
- Q4-2014, 29% of total search volumes were from mobiles in USA according to Comscore.
- **Streaming of data over mobile bandwidth was higher than streaming data onto desktops over WiFi**
- Many IOT Connected car interface were streaming data over radio communications.
- 4G network latency overhead of radio communication oscillates between 100ms -600ms.
- 3G network latency overhead of radio communication oscillates upto 3500ms
- The maximum no of HTTP requests a mobile app can make without losing the users attention is one (Jakob Nielsen- UX Expert)
- Many IOT devices were added to the industrial network for real-time monitoring.



# Mobile Internet Landscape

- Constrained by Resource : Network/Radio Communications, Computing/Processing, Memory/Storage, Power.
- Radio communication demand more battery power and constantly ping to the nearest radio tower.
- API centric application suffer from latency and battery life on mobile.
- REST backends expose **one resource per endpoint. Each app must send multiple request to several endpoints.**
- Steps involved in establishing a radio communication to make a new HTTP request:
  - Wake up the radio
  - Request some bandwidth to the radio tower.
  - Demands more battery power.
  - Check if the HTTP request has been successful or failure.
  - Keep a count of HTTP request.



**REST backends expose ONE RESOURCE PER ENDPOINT.**

**A MOBILE APP MUST SEND MULTIPLE REQUESTS (HTTP) TO SEVERAL ENDPOINTS.**



# SERVICE-ORIENTED ARCHITECTURE

- Rest backends expose one resource per endpoint.
- For rendering a single mobile page view, a mobile app may make something between 6-12 HTTP request for fetching data from backend.
- Mobile browsers tend to make 4-6 concurrent HTTP requests.
- Applications rely on different REST providers for authentication, analytics, security, data etc...
- Latency is one of the key challenges here.
- **REST API Architectural style becomes less efficient and ideal for Mobile Applications.**



# Bundle and Minify

- These are two distinct performance optimization techniques that are applied in a web application.
- Bundling and minification improve performance by reducing the number of server requests and reducing the size of the requested static assets.
- Improves browser cache for rendering static assets.
- Widely adopted in ASP.Net MVC, AngularJS, Angular, ReactJS, JAX-RS Frameworks.
- With a single request, all related resources are rendered from server to the client.
- Minifying renders the data in compressed form from the server to the client, thereby improving the user experience and application performance.



# The Batch EndPoint.

<https://developers.facebook.com/docs/graph-api/making-multiple-requests>

- Batching allows developers to pass instructions for several operations in a single HTTP request and specify dependencies between related operations.
- All independent operations are processed in a parallel while the dependent operations are processed sequentially.
- Once all operations are complete, a consolidated response is passed back to you and the HTTP connection is closed.
- Limitations
  - Batch requests are limited to 50 requests per batch. However, each request within the batch is counted as a single request for rate limiting purposes.
  - Batch request cannot include multiple Adsets under the same Campaign.

Updated on March 2019



# Simple Batched Requests

<https://developers.facebook.com/docs/graph-api/making-multiple-requests>

- The Batch API takes in an JSON object that is an array of your HTTP requests.
- Each of the request has a method (corresponding to HTTP method GET/PUT/POST/DELETE), a relative URL , optional headers array and an optional body(for POST and PUT requests)
- The Batch API returns an array of logical HTTP responses represented as JSON arrays.
- Each response has a status code, an optional headers array, and an optional body (which is a JSON encoded string).
- To make a batched request, send the JSON object in a POST request to the endpoint

```
POST /{endpoint}?batch=[{json-object}]
```



# Simple Batched Request

<https://developers.facebook.com/docs/graph-api/making-multiple-requests>

## Sample Request

```
curl -i -X POST 'https://graph.facebook.com/me?batch=' // Formatted for clarity
[
  {
    "method": "GET",
    "relative_url": "{page-a-id}"
  },
  {
    "method": "GET",
    "relative_url": "{page-b-id}"
  }
]
&include_headers=false // Included to remove header information
&access_token={access-token}'
```

## Sample Response

```
[
  {
    "code": 200,
    "body": "{
      \"name\": \"Page A Name\",
      \"id\": \"{page-a-id}\"
    }"
  },
  {
    "code": 200,
    "body": "{
      \"name\": \"Page A Name\",
      \"id\": \"{page-b-id}\"
    }"
  }
]
```





# Complex Batch Requests

<https://developers.facebook.com/docs/graph-api/making-multiple-requests>

## Sample Request

```
curl \
-F 'access_token=...' \
-F 'batch=[{ "method":"POST","relative_url":"{page-id}/feed","body":"message=Test status update&link=http://developers.facebook.com/},{
"method":"GET","relative_url":"{page-id}/feed"}]' \
https://graph.facebook.com
```

## Sample Response >>>

```
[
  { "code": 200,
    "headers": [
      { "name": "Content-Type",
        "value": "text/javascript; charset=UTF-8" }
    ],
    "body": "{ \"id\": \"...\" }"
  },
  { "code": 200,
    "headers": [
      { "name": "Content-Type",
        "value": "text/javascript; charset=UTF-8" },
      { "name": "ETag",
        "value": "..." }
    ],
    "body": "{ \"data\": [{...}] }"
  }
]
```



Batch Endpoint is not **RESTFUL**,  
as it **DOES NOT EXPOSE A RESOURCE**.

### MANY REST Standards:

1. Open API <https://swagger.io/specification/>
2. ODATA <https://www.odata.org>
3. Hydra <http://www.hydra-cg.com>
4. RAML <https://raml.org>
5. RESTful Service Description Language (RSDL)

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture that keeps the RESTful style architecture unique from most other network application architectures



# Limitations of REST API Architectural Style

- Resources mapped to each endpoint demand multiple HTTP requests.
- Excessive response information from the backend.
- No common REST API standards.
- Demands a contractual understanding between frontend and backend developers for developing efficient API
- REST is modelled after HTTP action verbs and requires HTTP transport and suffers from the same performance and security problems of HTTP.
- Lack request multiplexing or batch endpoints.
- **SOAP Protocol was heavy and complex, REST Provided some relief, but changing landscape demands new approach/standards.**



# Mobile Application Data Challenges

- Varied Mobile Device Configuration and RAM has different application performance.
- Data requirements vary across devices.
- Users want instant access to data.
- Short attention span (UX perspective)
- Continued connectivity @ low battery or low radio communication bandwidth.
- Most javascript application are single threaded
- Multi-modal interfaces with different screen resolutions demand efficient computing resources in constrained computing platforms(mobile/IOT/Connected Cars)



# Need for a better REST

- Increased mobile usage created a need for efficient data loading
- Variety of different frontend frameworks and platforms on the client-side
- Fast development speed and expectations for rapid feature development.

Updated on March 2019

Please read terms of use for authorized access

Original Series



# New REST API Standard Requirements

- Ability to perform multiple parallel request to improve application performance. HTTP/2 binary framing layer enables full request and response multiplexing, by allowing the client and server to break down an HTTP message into two independent frames, interleave them and then reassemble them at the end.  
<https://hpbnc.co/http2/#request-and-response-multiplexing>
- Ability to perform batch endpoints (<https://developers.facebook.com/docs/graph-api/making-multiple-requests> )
- Ability to request an endpoint return only specific field in the response on a per-type basis by including a fields[TYPE] parameter.
- A uniform OpenAPI Specification used by all industry.
- A uniform standard to help you design and document APIs at scale with standard toolset (<https://swagger.io>)



# New REST API Standard Requirements

- Easily Configurable and Highly secure HTTP Request and Response with end-to-end encryption integrated.
- Real-time integrated communications
- Lightweight explicit query operations on the backend.
- Query and aggregate resource across multiple domains.
- Open Protocol standards based framework (not tied to HTTP alone)
- Support Event Driven Technology or Publish/Subscribe
- Support for IOT/Connected Car Platforms
- Stateful and Stateless Data Transmission options



# EXISTING SOLUTIONS IN THE MARKET?

## API PLATFORM

<https://api-platform.com>

- REST and GraphQL framework to build modern API-driven projects

## Google Protocol Buffer

<https://developers.google.com/protocol-buffers/>

- Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.
- 2008.

## Apache Thrift

<https://thrift.apache.org>

- The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages.

## Falcor

<https://netflix.github.io/falcor/starter/what-is-falcor.html>

- A server middleware and client SDK tool.
- Javascript only.
- Allows you to model all your backend data as a single virtual JSON object on your Node Server.
- On the client you work with your remote JSON object
- One Model Everywhere.
- It does not offer a schema and static types.





GraphQL is a *specification*, with implementations in many languages.

A query language for APIs and a runtime for fulfilling those queries with your existing data. It is a specification similar to reactive extension for various languages.

SYED AWASE KHIRNI

# GraphQL

<https://graphql.org>



# GraphQL vs REST

## GraphQL

- A single endpoint for the **entire API**
- Body of the response is a normalized and JSON Object and the content is always under a **data** key.
- Parameters are passed between parantheses and inside the query.
- Only explicit query request sent across for limited or required information as response
- Uses special query language called graphql.
- Uses special context/content type: **“application/graphql”**
- **Tied to HTTP POST request ONLY.**

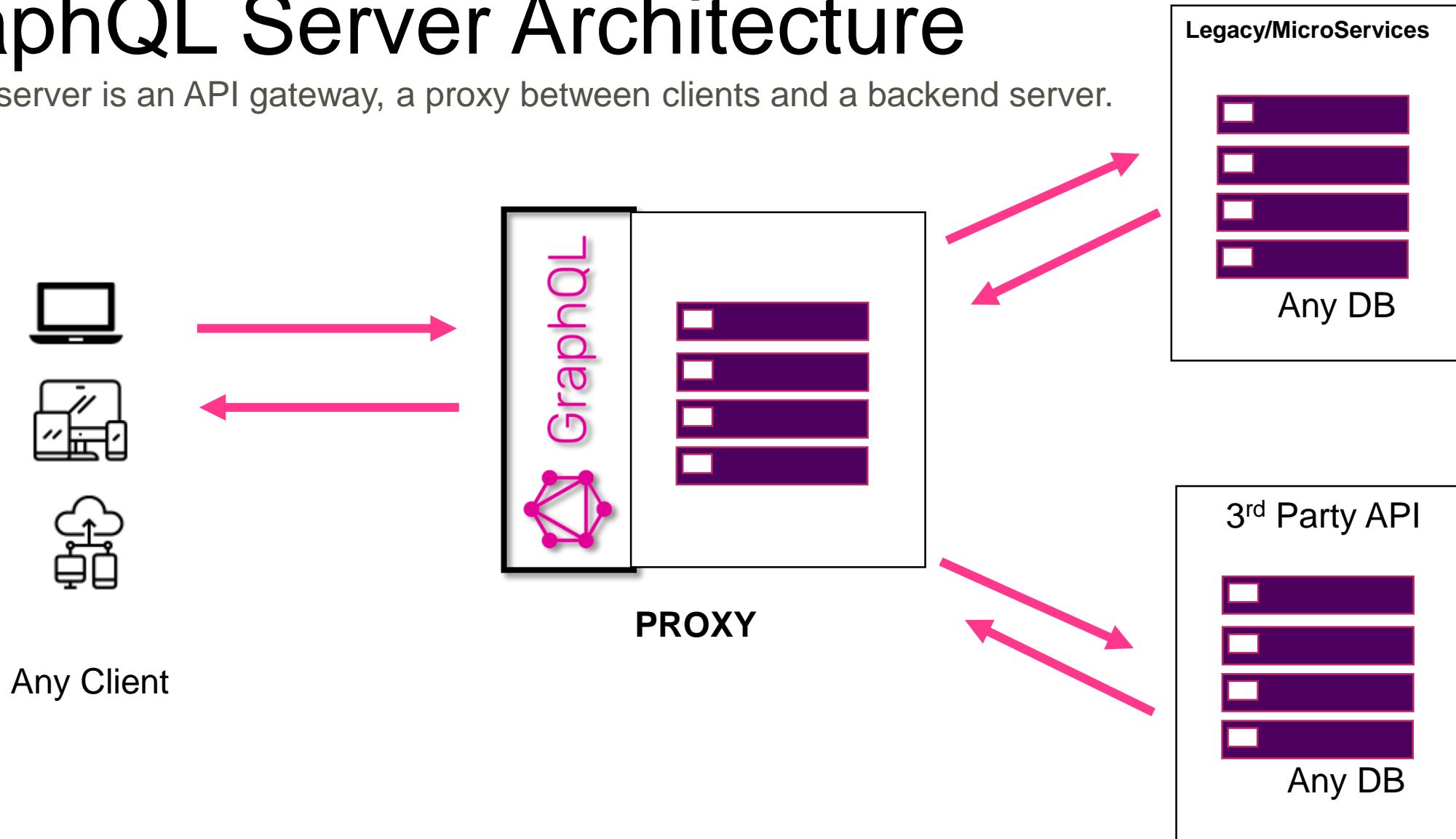
## REST

- The API exposes one endpoint per resource.
- Body of the response is a JSON object
- Response is usually complete resource from the endpoint.
- Uses context/content type: **“application/json”**
- **Tied to HTTP Semantics (action verbs: GET/PUT/POST/DELETE)**



# GraphQL Server Architecture

GraphQL server is an API gateway, a proxy between clients and a backend server.





# GraphQL

- It is a query language similar to SQL but for querying web APIs rather than databases.
- Solves the over-fetching of data from the server.
  - Ask for what you need, get exactly that – meaning the client has full control over which data it wants from the server.
    - REST endpoints are mapped on to the resource.
- Solves the under-fetching of the data from the server.
  - Get many resources in a single request.
    - Using REST API to fetch data from two different resource, we need to make two separate calls to the server.
    - GraphQL lets you get many resources in a single request.
- Describe what's possible with a type system
  - In GraphQL, we write a schema that fully describes your API
  - We can do something similar OpenAPI specification (Swagger API)
  - Schema first approach adopted by GraphQL that defines a clear contract between the server and client.
  - Provides an ecosystem for building powerful developer tools using the type system.
- Provides support for versionless API
- GraphQL is a layer that we can add on top of the existing code. It takes care of exposing an API, all the data and business logic can stay the same.
- It is available for many different languages.
- It is an Open Specification.



# <https://landscape.graphql.org>

The screenshot shows the GraphQL Landscape website. The header includes the GraphQL logo and the title "GraphQL Landscape". Below the header, there's a navigation bar with tabs for "Landscape" and "Card Mode". The main content area is divided into several sections:

- Projects:** A grid of project cards, including GraphQL Foundation, GraphQL, and others.
- Tools:** A grid of tool cards, including GraphQL, GraphQL Playground, and others.
- Services:** A grid of service cards, including GraphQL, GraphQL API, and others.
- Databases:** A grid of database cards, including GraphQL, GraphQL DB, and others.
- GraphQL Adopter:** A large grid of adopter cards, including Istio, Kubernetes, and many others.
- General:** A grid of general cards, including GraphQL, GraphQL, and others.

On the left side, there's a sidebar with filters and a "Reset Filters" button. The bottom of the page features a "TERRITORIAL PRESCIENCE" logo and the text "provoke thought, invoke action".

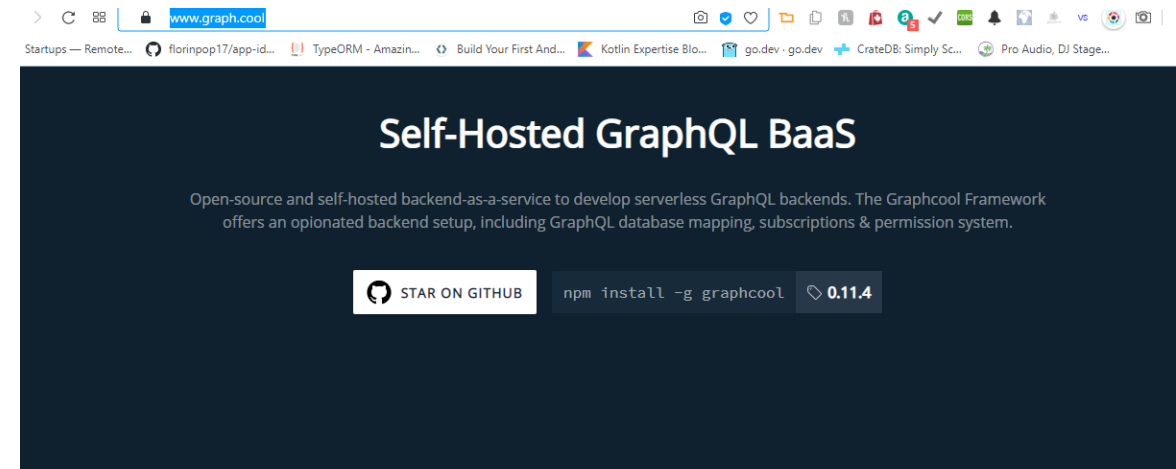


# GraphQL Server Libraries

C# / .NET	Clojure
Elixir	Erlang
Go	Groovy
Java	Javascript
Julia	Kotlin
Perl	PHP
Python	R
Ruby	Rust
Scala	Swift
Ocaml/Reason	

[www.graph.cool](http://www.graph.cool)

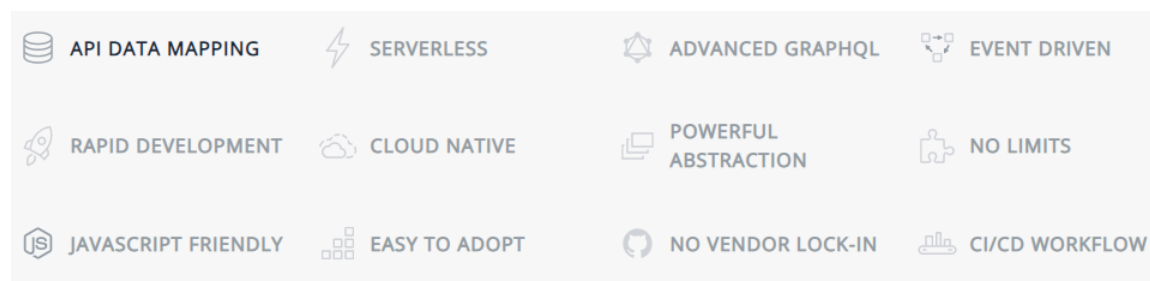
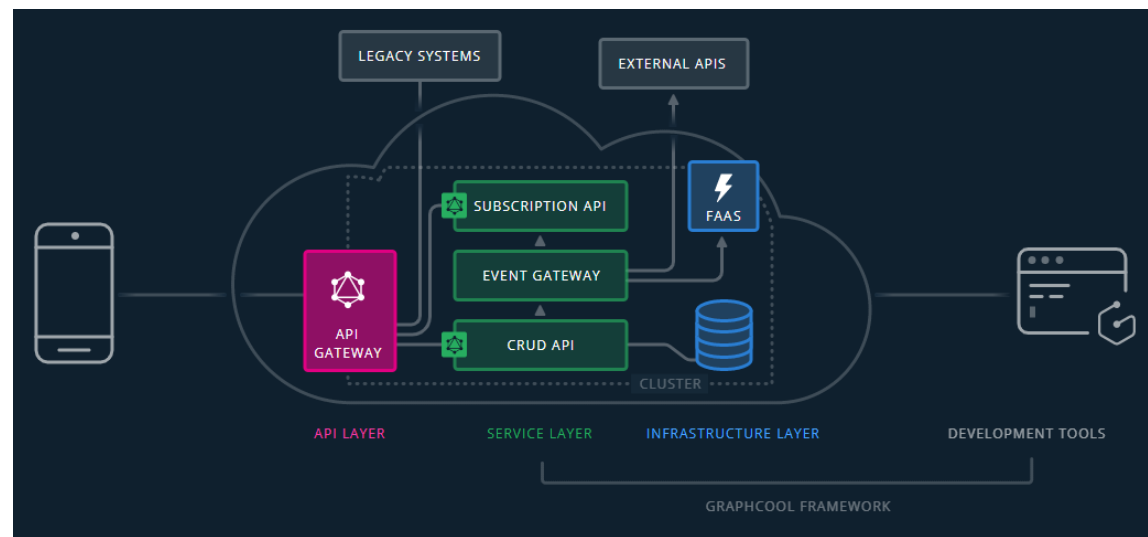
- Open-source and self-hosted backend-as-a service to develop serverless GraphQL backends.
- Offers opinionated backend setup, including GraphQL database mapping, subscription and permission system.





# Graphcool Framework Architecture

<https://www.graph.cool>





# GraphQL Features

- An API gateway, or a proxy server that sits in front of data backends.
- It is a DSL to express the business logic exposed to clients.
- It is a declarative query language. Clients get what they ask for.
- It is designed for Remote Procedure Call, and not limited to CRUD like REST API.
- It is a strong, statically typed language. Quick development and less error-prone
- It supports read, write and subscribe operations
- It moves the center of gravity towards the client. The client decides what fields they get in response, not the server.
- It is a specification for fetching and updating data that can be implemented in any protocol. It works on HTTP.
- It offers built-in API documentation.
- It supports API evolution and allows for deprecation warnings.
- Easy integration with legacy or existing backend API systems.

Updated on March 2019

Please read terms of use for authorized access

Original Series





# GraphQL Features

- Strongly typed
- Code generation and introspection
- Queries, mutations, subscriptions
- Transport agnostic
- Client-specified shape of response
- Efficient.
- Enables declarative data fetching by the client
- GraphQL server exposes single endpoint and responds to queries
- Single endpoint

Updated on March 2019

Please read terms of use for authorized access

Original Series



# When to use?

## GraphQL

- Client-driven development
- IOT/Mobile Architectural Support
- Data driven UI
  - Structured data
  - Complex data
  - Query-driven
  - Real-time/offline
- It is contract between client and server
- Compliance with OpenAPI Specification.
- Application clearly defines the relationship of bounded contexts.

## REST

- For Resources
- Web based Architectural or Service based Architectures
- Leveraging HTTP caching, HTTP Content Types, Hypermedia(HATEOAS)
- HTTP client.
- HTTP/2 Performance gains
- Compliance with OpenAPI Specification.



# GraphQL Playground



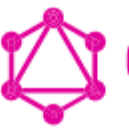












<https://github.com/prisma-labs/graphql-playground>

Please read terms of use for authorized access

Original Series



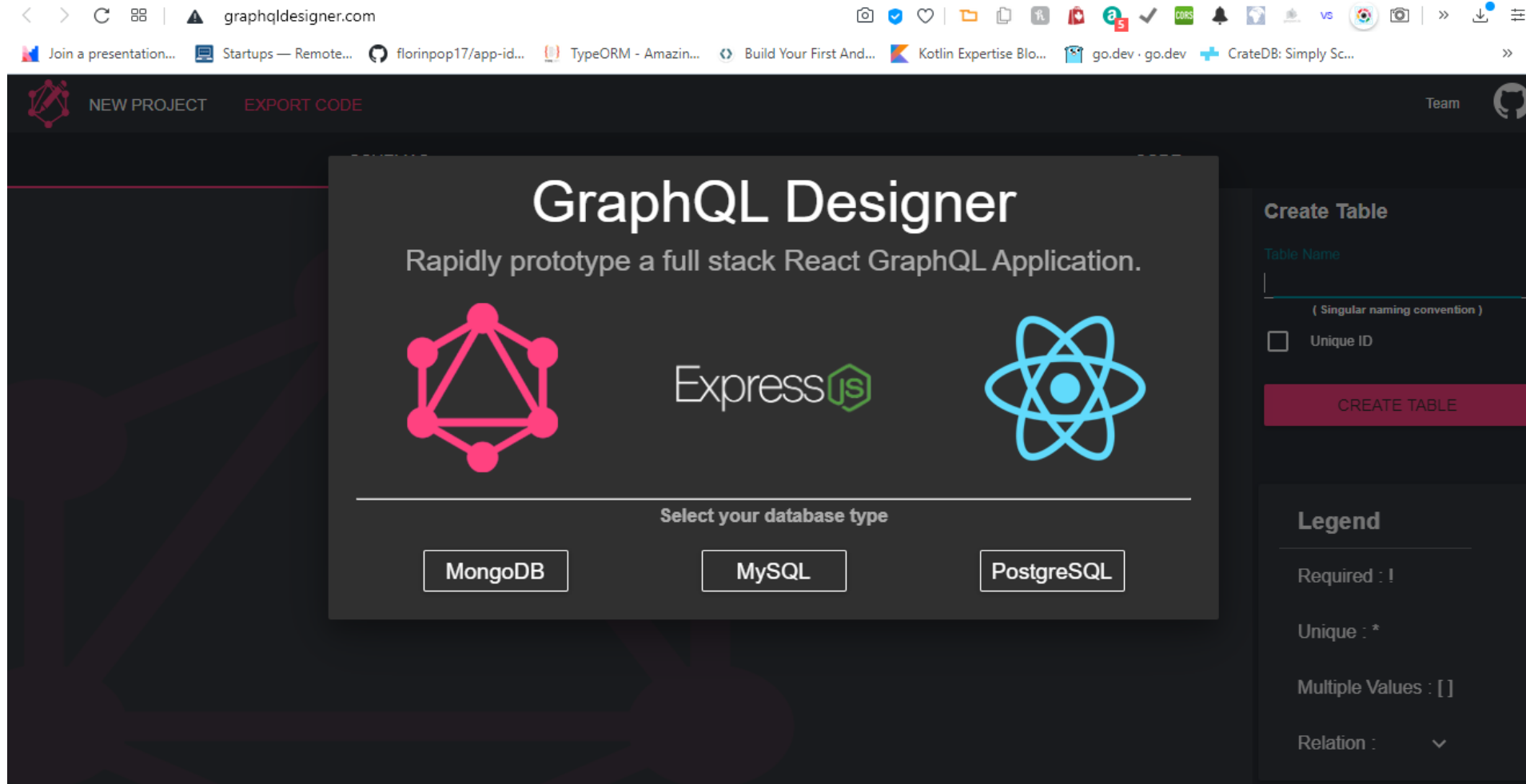
# GraphQL Tools

 <b>APOLLO ANGULAR</b>  Apollo Angular The Guild ★ 252	 <b>APOLLO CLI</b>  Apollo CLI Apollo ★ 2,564 Funding: \$52.2M	 <b>GraphiQL</b>  GraphQL GraphQL Foundation ★ 10,994	 <b>GRAPHQL CLI</b>  GraphQL CLI The Guild ★ 1,315	 <b>CODE GENERATOR</b>  GraphQL Code Generator The Guild ★ 2,125
 <b>GRAPHQL CONFIG</b>  GraphQL Config The Guild ★ 792	 <b>GRAPHQL INSPECTOR</b>  GraphQL Inspector The Guild ★ 669	 <b>GRAPHQL MESH</b>  GraphQL Mesh The Guild ★ 1,224	 <b>GRAPHQL MODULES</b>  GraphQL Modules The Guild ★ 656	 <b>GRAPHQL SCALARS</b>  GraphQL Scalars The Guild ★ 697
 <b>GRAPHQL TOOLS</b>  GraphQL Tools (by The Guild) The Guild ★ 2,491	 <b>GRAPHQL MERGE</b>  merge-graphql-schemas The Guild ★ 920	 <b>GRAPHQL SOFA</b>  SOFA The Guild ★ 521	 <b>tartiflette.</b>  Tartiflette Delymotion ★ 624 Funding: \$68.5M	 <b>TypeGraphQL</b>  TypeGraphQL TypeGraphQL ★ 4,785



<http://graphqldesigner.com>

# GraphQL Designer



Please read terms of use for authorized access

Original Series













Updated on March 2019



# Who uses GraphQL?

<https://graphql.org/users/>

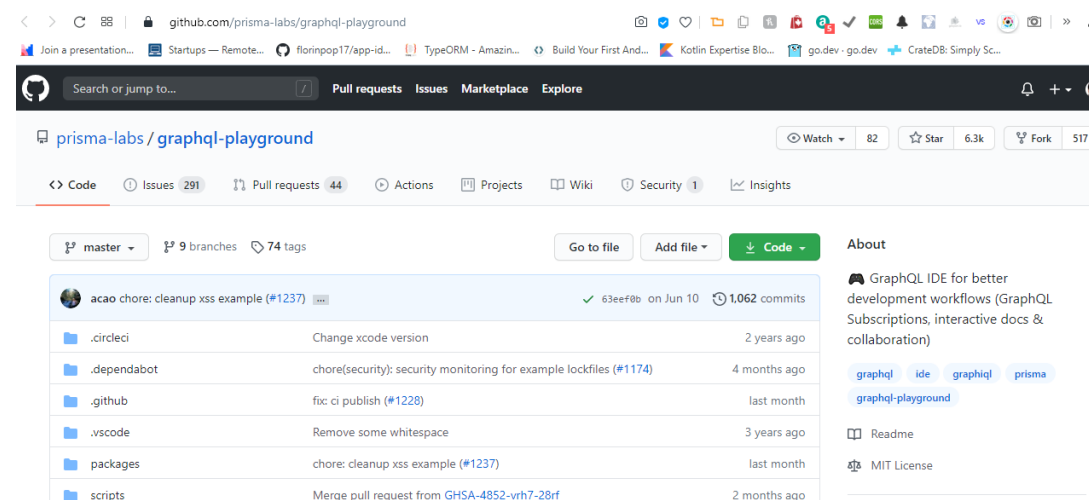
**Partial List Only**

 Lyft (adopter) Lyft MCap: \$9.25B	 Magento (adopter) Magento Commerce MCap: \$212.11B	 NashTech (adopter) NashTech	 NBC News Digital (adopter) NBCUniversal MCap: \$194.2B
 New Relic (adopter) New Relic MCap: \$3.9B	 PayPal (adopter) PayPal MCap: \$208.12B	 Pinterest (adopter) Pinterest MCap: \$15.08B	 Pluralsight (adopter) Pluralsight MCap: \$2.78B
 Rakuten Rakuten MCap: \$12.62B	 ResearchGate (adopter) ResearchGate Funding: \$87.6M	 Salsify (adopter) Salsify Funding: \$97.6M	 Satispay (adopter) Satispay Funding: \$50.59M



## GraphQL-Playground

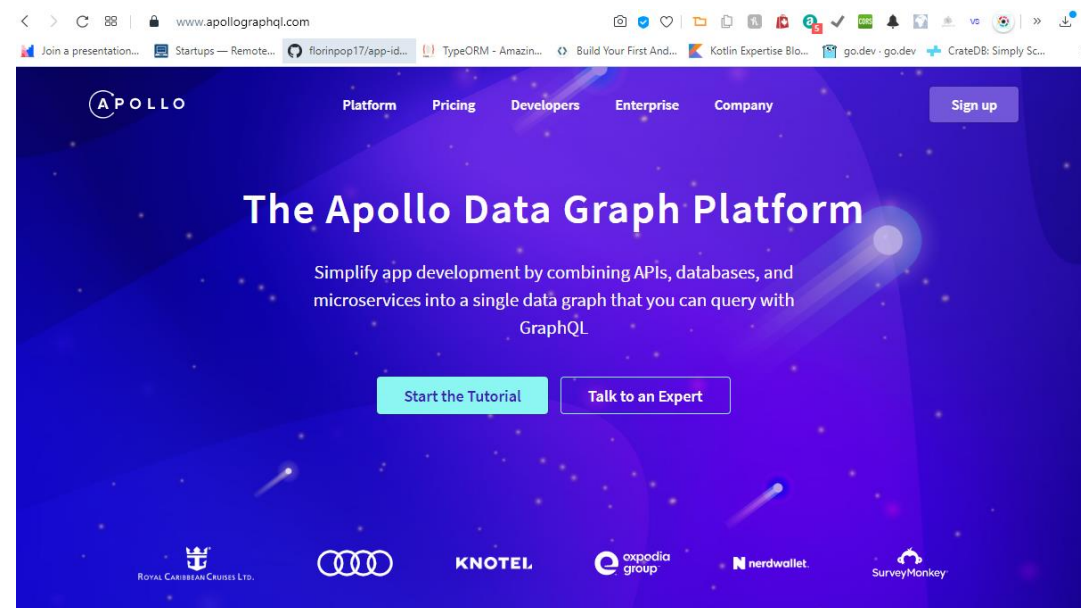
<https://github.com/prisma-labs/graphql-playground>



The screenshot shows the GitHub repository for 'prisma-labs/graphql-playground'. The repository is in the 'prisma-labs' organization. It has 82 watches, 6.3k stars, and 517 forks. The repository is currently on the 'master' branch, with 9 branches and 74 tags. The commit history shows a recent commit by 'acao' titled 'chore: cleanup xss example (#1237)' on June 10, 2019, with 1,062 commits in total. The file list includes '.circleci', '.dependabot', '.github', '.vscode', 'packages', and 'scripts'. The 'About' section describes GraphQL IDE for better development workflows, including GraphQL Subscriptions, interactive docs, and collaboration. It also mentions a Readme and MIT License.

## Apollo GraphQL

<https://www.apollographql.com>



The screenshot shows the Apollo GraphQL website. The header includes the Apollo logo and navigation links for Platform, Pricing, Developers, Enterprise, and Company, along with a Sign up button. The main content area features the title 'The Apollo Data Graph Platform' and a description: 'Simplify app development by combining APIs, databases, and microservices into a single data graph that you can query with GraphQL'. Below this, there are two buttons: 'Start the Tutorial' and 'Talk to an Expert'. The footer displays logos for various partners, including Royal Caribbean Cruises Ltd., Audi, Knottel, Expedia Group, Nerdwallet, and SurveyMonkey.



SYED AWASE KHIRNI

# Domain Driven Design

Updated on March 2019





SYED AWASE KHIRNI

# 1.GraphQL Server

Building a GraphQL server using Node.js and Express



# Building Blocks

- A GraphQL Server consists of the following:
  - **A Schema:** it defines our entities but also what we can query or call on a mutation on
  - **Resolvers:** resolver functions talks to a 3<sup>rd</sup> party API or our database and ends up returning data back to the user.

- Dependencies
  - Much needed dependencies for building a graphql server
    1. Expressjs – to create our web server
    2. Graphql – core libraries to enable us to leverage graphql
    3. Express-graphql – this enables us to bind together graphql and express.

```
npm init
```

```
npm install express express-graphql graphql --save
```

```
npm install graphql-tools --save
```



## Create a GraphQL Server

- Creating a GraphQL Server
  - Src/app.js
    - Schema
    - Resolver
    - Express Server
    - Endpoint
- Node src/app.js

src > app.js

ex1-basicgraphqlserver > src > JS app.js

```

1  var express = require('express');
2  var {graphqlHTTP}= require('express-graphql');
3  var {buildSchema} = require('graphql');
4  //GraphQL Schema Definition language
5  //(GraphQL Interface Definition Language)
6  var schema = buildSchema(`
7    type Query{
8      | message:String
9    }
10 `);
11 //Root Resolver
12 var root={
13   | message:()=>`Hello World! Welcome to GraphQL Server Session!`
14 };
15 //create an express server and a graphql endpoint
16 var app = express();
17 app.use('/graphql', graphqlHTTP({
18   | schema: schema,
19   | rootValue: root,
20   | graphiql: true
21 }));
22 app.listen(4545,()=>console.log('Express GraphQLServer Now running on localhost:4545/graphql'));

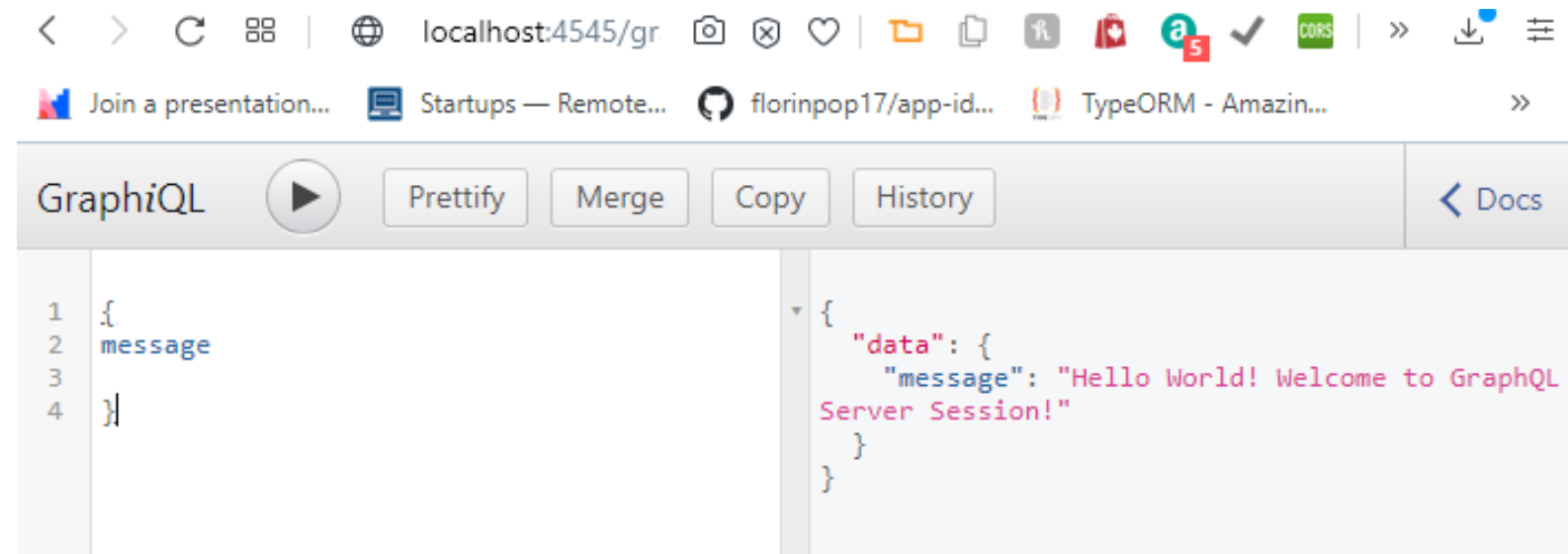
```



<http://localhost:4545/graphql>

## Running GraphQL Server

- Run graphql server
- Node src/app.js





SYED AWASE KHIRNI

# 3.GraphQL Server

Building a GraphQL server using Node.js and the Apollo Framework

Updated on March 2019



src &gt; index.js

## Create a GraphQL Server

- Creating a GraphQL Server
  - Src/app.js
    - Schema
    - Resolver
    - Express Server
    - Endpoint
- Node src/app.js

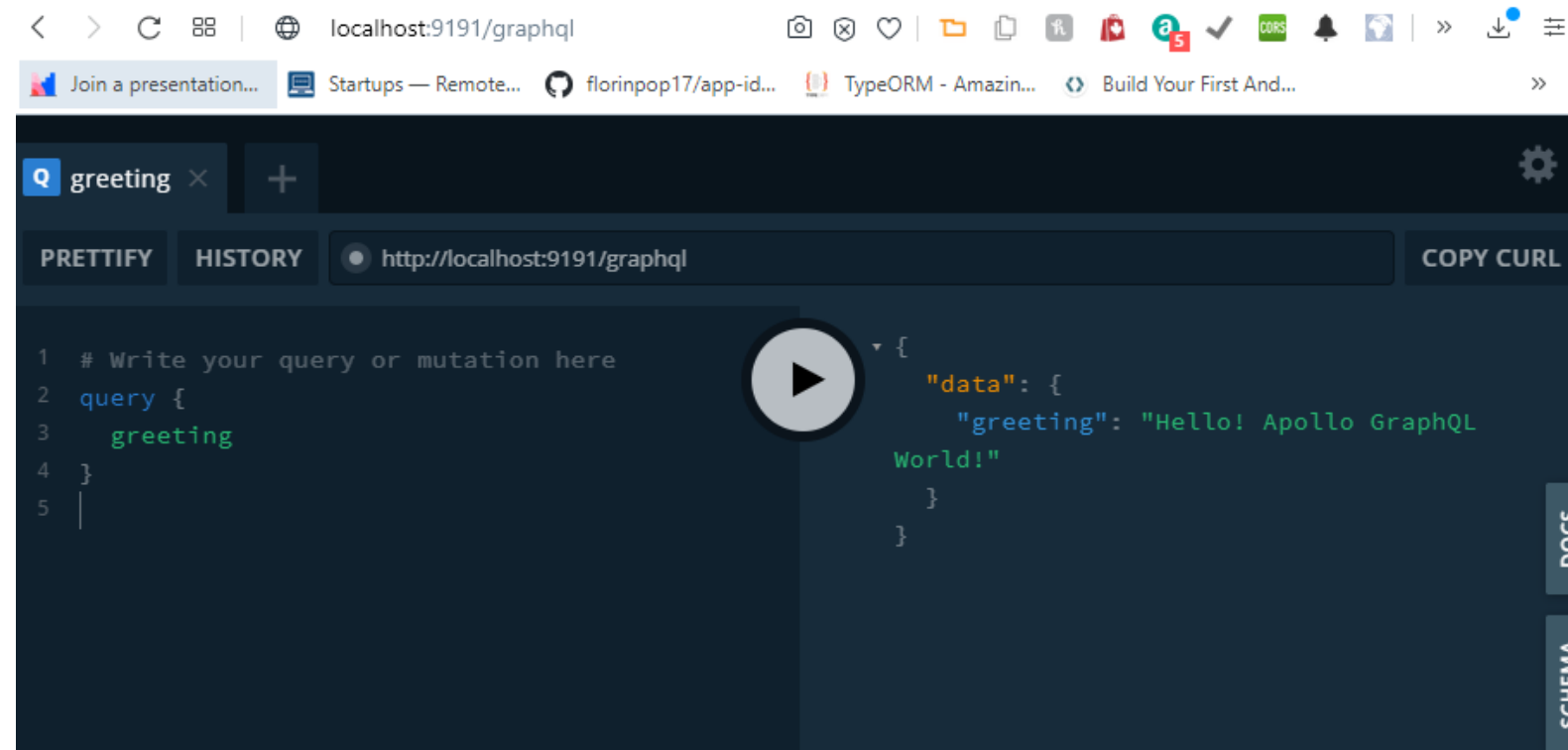
```
ex3-apollographqlsrvr > src > JS index.js
1
2  const {ApolloServer, gql} = require('apollo-server');
3  //schema = graphql schema definition language
4  const typeDefs=gql`
5    |   type Query{
6    |     |   greeting:String
7    |   }
8  `;
9  console.log(typeDefs);
10 //resolvers
11 const resolvers={
12   |   Query:{
13   |     |   greeting: () => 'Hello! Apollo GraphQL World!'
14   |   }
15 };
16
17 //spin off server
18 const server = new ApolloServer({typeDefs, resolvers});
19 server.listen({port:9191})
20   .then(({url})=> console.log(`Server running at ${url}`));
```



http://localhost:9191/graphql

## Running GraphQL Server

- Run graphql server
- Node src/app.js





SYED AWASE KHIRNI

# GraphQL Schema Definition Language