HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD

www.heig-vd.ch

GraphQL

The new standard for API development

What is an API?

An API is an interface that let programmers interact with data.
When you design an API, always think about **developer experience**.

REST APIs

REST is an architectural concept for network based software.

It has become a standard for designing web APIs.

- stateless servers
- structured access to resources
- support any type of resources
- language agnostic

How it feels to design a REST API?

As REST is just a **concept/style** we have a lot of freedom. A common mistake is to design an API according to the views of **one specific** client application.

To solve this we need to follow best practices and think ahead.

- Define a good structure for our endpoints
- Define what information to return for different type of resources
- Think about features (sorting, filtering, etc..)
- Maintain a good documentation and provide examples
- and more..

REST and Endpoints

A well structured endpoints provide a clear and consistent interface. Giants like Twitter, Facebook, Google and **Github** are often an inspiration for other companies.

```
/users
/users/:username
/users/:username/followers
/users/:username/repos
/repos/:owner/:repo/issues
/repos/:owner/:repo/issues/:number
...
```

REST and Granularity

We need to make choices about what fields to return for each type of resource.

```
GET https://api.github.com/repos/facebook/react/issues/14139
```

```
{
  "id": 378458607,
  "state": "open",
  "number": 14139,
  "title": "Tapping outside of an input on iOS does not fire...",
  "user": {
    // What should we include here ?
    // There is endpoint for this !
  },
  "body": "...",  // Can I have an excerpt instead ?
  "created_at": "2018-11-07T20:35:47Z",

  ... +10 more fields // Do I really need more fields ?
}
```

REST and HATEOAS

It is also a good practice to provides users with the means to explore the API more deeply.

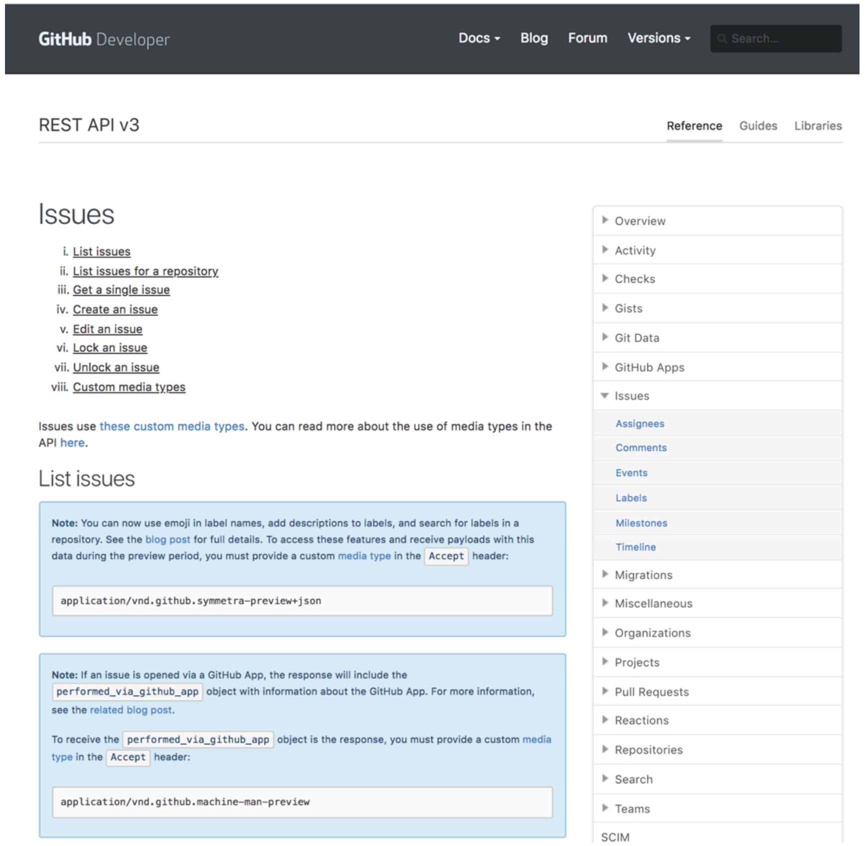
That's what HATEOAS (Hypermedia As The Engine Of Application State) does.

```
GET https://api.github.com/repos/facebook/react/issues/14139
```

```
{
  "state": "open",
  "id": 378458607,
  "number": 14139,
  "title": "Tapping outside of an input on iOS does not fire onBlur event",
  "user": { ... },
  "body": "...",
  "created_at": "2018-11-07T20:35:47Z",
  "url": "https://api.github.com/repos/facebook/react/issues/14139",
  "repository_url": "https://api.github.com/repos/facebook/react",
  "comments_url": "https://api.github.com/repos/facebook/react/issues/14139/comments",
  "events_url": "https://api.github.com/repos/facebook/react/issues/14139/events",

  ... +10 more fields
}
```

REST and Documentation



Issues

i. List issues
ii. List issues for a repository
iii. Get a single issue
iv. Create an issue
v. Edit an issue
vi. Lock an issue
vii. Unlock an issue
viii. Custom media types

Issues use these custom media types. You can read more about the use of media types in the API here.

List issues

Note: You can now use emoji in label names, add descriptions to labels, and search for labels in a repository. See the blog post for full details. To access these features and receive payloads with this data during the preview period, you must provide a custom media type in the `Accept` header:

```
application/vnd.github.symmetra-preview+json
```

Note: If an issue is opened via a GitHub App, the response will include the `performed_via_github_app` object with information about the GitHub App. For more information, see the related blog post.

To receive the `performed_via_github_app` object is the response, you must provide a custom media type in the `Accept` header:

```
application/vnd.github.machine-man-preview
```

- Document advanced features: sorting, filtering, pagination, HTTP Methods, etc..
- Provide responses example

How it feels to consume a REST API?

```
async function getOpenIssues() {
  const repo = await request('https://api.github.com/repos/facebook/react');
  const issues = await request('https://api.github.com/repos/facebook/react/issues');

  return {
    count: repo.open_issues_count,
    issues: await Promise.all(issues.map(async issue => {
      const comments = await request(issue.comments_url);

      return {
        title: issue.title,
        author: {
          login: issue.user.login,
          avatar: issue.user.avatar_url,
        },
        body: issue.body,
        state: issue.state,
        date: issue.created_at,
        comments: comments.map(comment => ({
          author: {
            login: comment.user.login,
            avatar: comment.user.avatar_url,
          },
          body: comment.body,
          date: comment.created_at,
        })),
      };
    })),
  };
}
```

How it feels to consume a REST API

In the previous example we used Github's REST API to fetch the last 30 issues from the React repository

This is what we can conclude:

- **Multiple requests** are necessary to get the data we need
    - We sent 32 requests in total
    - Pagination and rate limiting were out of the scope for this example. But we should handle that too 🤫
- As we were not interested in all data, **65%** of the data received by Github were **wasted** (~130 KB used, ~380 KB received) 😭
- Note that we also had to spend time reading the documentation

Convinced ?

Github's thoughts

> The REST API is responsible for over 60% of the requests made to our database tier. This is partly because, by its nature, **hypermedia navigation** requires a client to **repeatedly communicate** with a server so that it can get all the information it needs.

> We heard from integrators that our REST API also **wasn’t very flexible**.

> It seemed like our responses simultaneously sent **too much data** and **didn’t include data that consumers needed**.

> We wanted to be smarter about how our resources were **paginated**. We wanted assurances of **type-safety** for user-supplied parameters. We wanted to generate **documentation** from our code

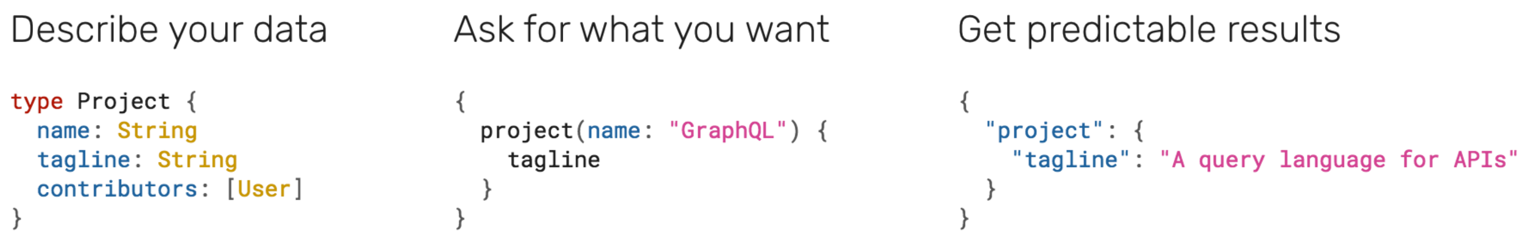
> We studied a variety of API specifications built to make some of this easier, but we found that **none of the standards totally matched our requirements**.

Github's thoughts

> ...And then we learned about **GraphQL**.

https://githubengineering.com/the-github-graphql-api/

GraphQL



Describe your data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

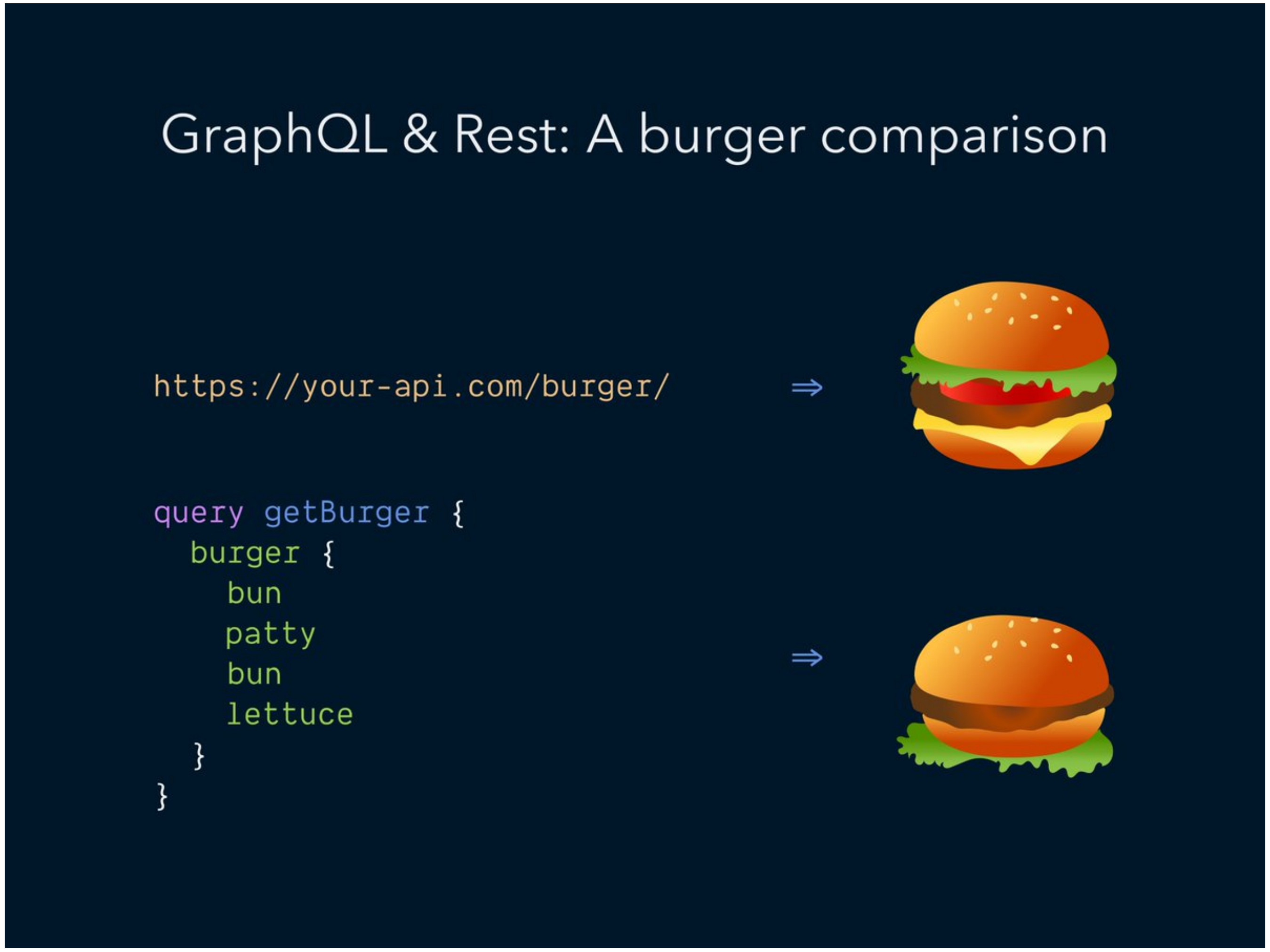
Get predictable results

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

- GraphQL is a new API standard that provides a more efficient, powerful and flexible alternative to REST
- It is a **Q**uery **L**anguage for **APIs** (not databases)
- It's not only for React developers
- It was developed and open-sourced by Facebook

https://www.graphql.org/

GraphQL



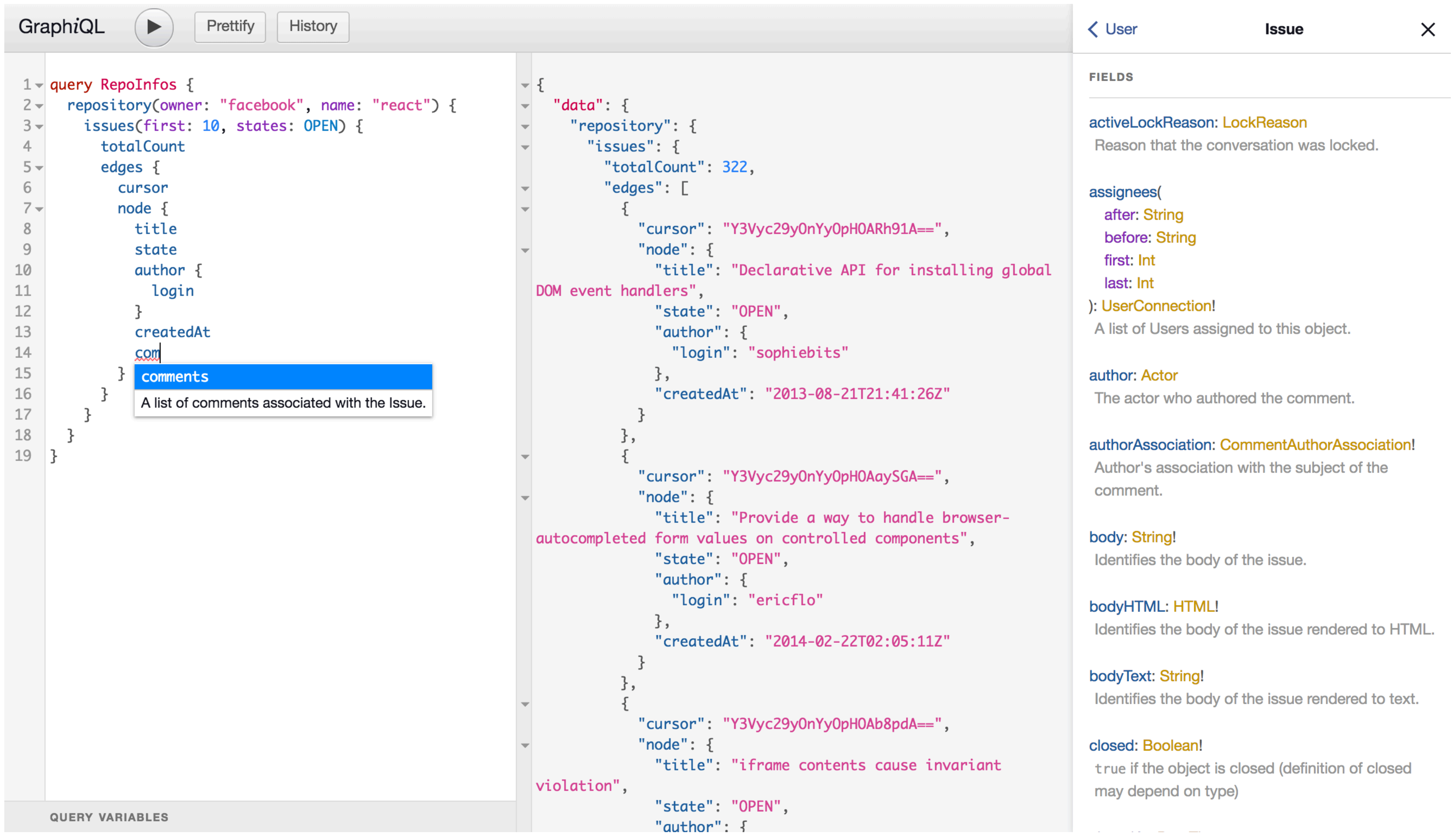
```
https://your-api.com/burger/        ⇒
```

```
query getBurger {
  burger {
    bun
    patty
    bun
    lettuce
  }
}
```

https://apievangelist.com/2018/06/29/rest-api-and-graphql-burger-king/

GraphiQL

GraphiQL is an in-browser tool for writing, validating, and testing GraphQL queries.



```
query RepoInfos {
  repository(owner: "facebook", name: "react") {
    issues(first: 10, states: OPEN) {
      totalCount
      edges {
        cursor
        node {
          title
          state
          author {
            login
          }
          createdAt
          com
        }
      }
    }
  }
}
```

comments
A list of comments associated with the Issue.

```
{
  "data": {
    "repository": {
      "issues": {
        "totalCount": 322,
        "edges": [
          {
            "cursor": "Y3Vyc29yOnYyOpHOARh91A==",
            "node": {
              "title": "Declarative API for installing global
DOM event handlers",
              "state": "OPEN",
              "author": {
                "login": "sophiebits"
              },
              "createdAt": "2013-08-21T21:41:26Z"
            }
          },
          {
            "cursor": "Y3Vyc29yOnYyOpHOAaySGA==",
            "node": {
              "title": "Provide a way to handle browser-
autocompleted form values on controlled components",
              "state": "OPEN",
              "author": {
                "login": "ericflo"
              },
              "createdAt": "2014-02-22T02:05:11Z"
            }
          },
          {
            "cursor": "Y3Vyc29yOnYyOpHOAb8pdA==",
            "node": {
              "title": "iframe contents cause invariant
violation",
              "state": "OPEN",
              "author": {
```

FIELDS

activeLockReason: LockReason
Reason that the conversation was locked.

assignees(
after: String
before: String
first: Int
last: Int
): UserConnection!
A list of Users assigned to this object.

author: Actor
The actor who authored the comment.

authorAssociation: CommentAuthorAssociation!
Author's association with the subject of the comment.

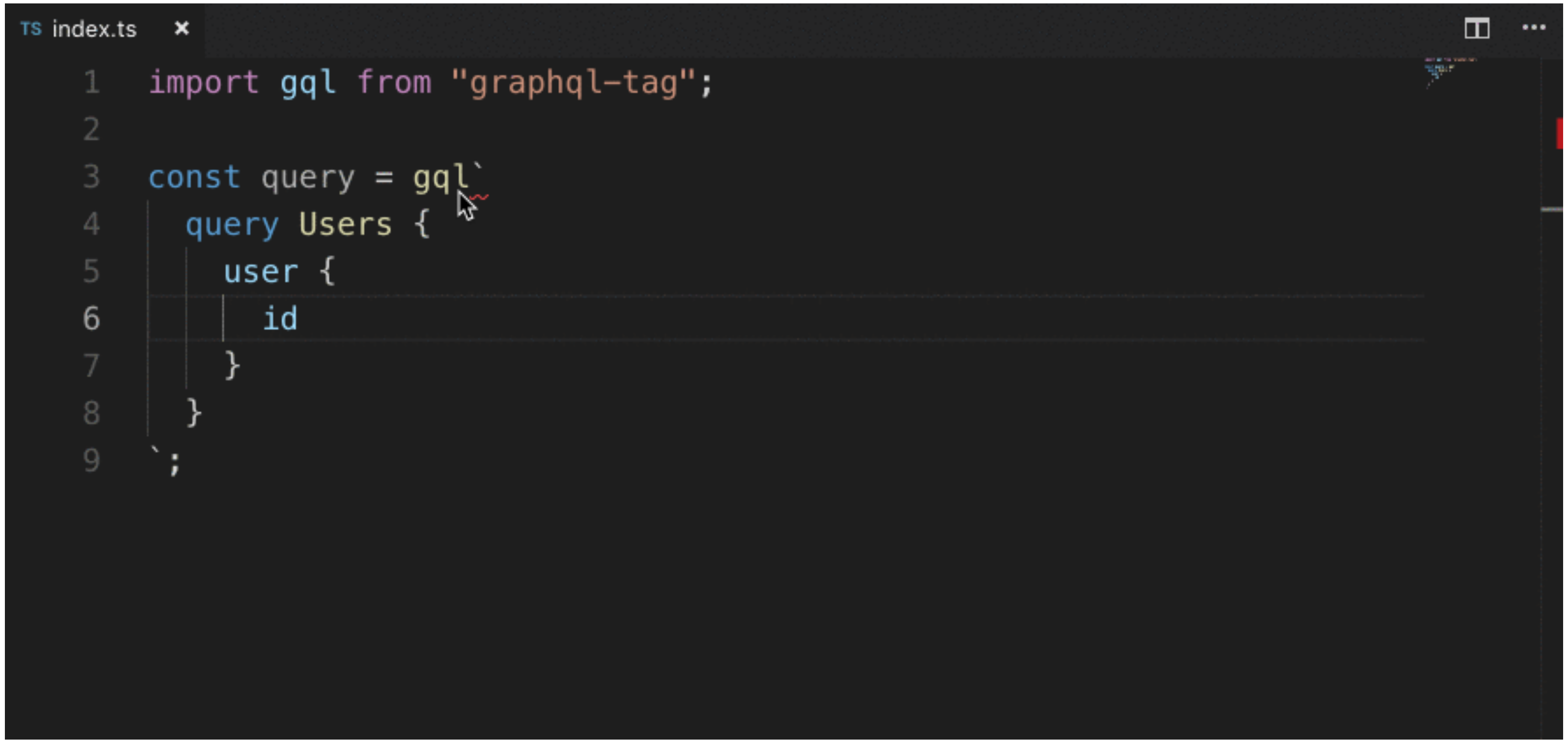
body: String!
Identifies the body of the issue.

bodyHTML: HTML!
Identifies the body of the issue rendered to HTML.

bodyText: String!
Identifies the body of the issue rendered to text.

closed: Boolean!
`true` if the object is closed (definition of closed may depend on type)

IDE integration


index.ts

```
import gql from "graphql-tag";

const query = gql`
  query Users {
    user {
      id
    }
  }
`;
```

https://www.prisma.io/blog/vscode-thieghu7shoo/

How it feels to consume a GraphQL API?

```
query {
  repository(owner: "facebook", name: "react") {
    issues(first: 10) {
      totalCount
      edges {
        cursor
        node {
          title
          state
          author { login avatarUrl }
          body
          createdAt
          comments(first: 3) {
            totalCount
            edges {
              node {
                author {
                  login
                  avatarUrl
                }
                body
                createdAt
              }
            }
          }
        }
      }
    }
  }
}
```

Copy this example in Github's Graphql Explorer

How it feels to consume a GraphQL API

The previous query uses GraphQL to ask exactly the same information as earlier in how it feels to consume a REST API.

Comparing to REST, we can conclude that GraphQL is:

- **self-documented** — we get predictable responses
- **less chatty** — only 1 request is need to get all the data we need
- **more efficient** — 0 bytes were wasted 🎉
- **more flexible** — we can ask exactly what we need and have multiple ways to paginate results
- **more robust** — it provide type safety

Convinced ?

...and this is just the beginning

GraphQL core concepts

Schema Definition – Types

As an API designer you first have to define the schema of your API. The syntax for writing schemas is called Schema Definition Language (SDL).

Here is a partial example of how Github defined the type `User`

```
type User {
  id: ID!
  login: String!
  email: String!
  avatarUrl(size: Int): URI
  repositories: [Repositories]
}
```

- This type `User` has 5 fields
- `ID`, `String`, `Int` and `URI` are called scalar types (equivalent of primitive types in many languages) because they don't have sub-fields
- The `!` following the type means that this field is required.
- `avatarUrl` is field that can take an argument `size`
- `repositories` is an array. This is how we create one-to-many relationship

Schema Definition – Types

Here is another example of a minimal representation of a repository

```
type Repository {
  id: ID!
  name: String!
  isPrivate: Boolean!
  issues: [Issue]
  owner: User
}
```

- We defined a one-to-may relationship between `Repository` and `Issue`
- We defined a relationship between the types `Repository` and `User`
- If went further, `Issue` would also define a relation with `User`
- There is no limit for relations and this is what makes possible to create a big Graph with our data

Schema Definition – Endpoints

Instead of having multiple endpoints that return fixed data structure, Graphql APIs typically expose a single/few endpoint.

```
type Query {
  user(login: String!): User
  repository(owner: String!, name: String!): Repository
}
```

The type `Query` is a special type for creating endpoints. Think of them as entry-points to our Graph. Here is an example of a query that a client could send to the server

```
{

  user(login: 'paulnta') {
    id
    login
    avatarUrl
  }
}
```

```
{
  "data": {
    "user": {
      "id": "MDQ6VXNlcjk1MzExODA=",
      "login": "paulnta",
      "avatarUrl": "https://avatars0.gith..."
    }
  }
```

Schema Definition

A schema is a simple collection of GraphQL types.

```
type User { ... }
type Repository { ... }

type Query { ... }
type Mutation { ... }
type Subscription { ... }
```

`Query` `Mutation` and `Subscription` are called root types because they act as entry points for requests sent by clients

- `Query` - define root queries
- `Mutation` - define queries for mutating data
- `Subscription` - define subscriptions to data changes (websockets)

Schema Definition - Queries

```
fragment simpleUser on User {
  login
  repositories { id name }
}

{
  paulnta: user(login: 'paulnta') {
    ...simpleUser
  }
  edri: user(login: 'edri') {
    ...simpleUser
  }
}
```

- We defined a `fragment` on the type `User`. Fragments are reusable units that lets you construct as set of fields
- GraphQL support request batching. We made two requests in a single query.
- It's possible to create aliases for any field `paulnta` and `edri` instead of `user`.

https://www.graphql.org/learn/queries/

Data fetching from client app

Data fetching from client app

There is generally to different approaches for fetching data from a client app

Using plain HTTP

- All you need to do is sending a POST request with your GraphQL query inside the body
- GET request are supported to (the query is placed in query string parameters)

Using a client library recommended

- Provides a good abstraction and lets you focused on the most important — your app !
- You don't have to worry about lower-level networking details
- It comes with really powerful features !

Client Libraries

Relay is Facebook's homegrown GraphQL client that they open-sourced alongside GraphQL in 2015.

Relay is heavily **optimized for performance**. It started out as a routing framework that got combined with data loading responsibilities. The performance benefits of Relay come at the cost of a **notable learning curve**

https://facebook.github.io/relay/

Client libraries

APOLLO

recommended

Apollo Client is a community-driven effort to build an **easy-to-understand**, flexible and powerful GraphQL client.

Right now there is a JavaScript client with bindings for popular frameworks like React, Angular, Ember or Vue as well as early versions of iOS and Android clients.

Apollo is **production-ready** and has handy features like caching, optimistic UI, subscription support and many more.

https://www.apollographql.com/client/

React Apollo basic concept

With React apollo you use `Query` components in order to fetch GraphQL data and attach result to your UI.

```
import gql from 'graphql-tag'
import { Query } from 'react-apollo'

const GET_CURRENT_USER = gql`
  viewer {
    login
    name
  }
`

const CurrentUser = () => (
  <Query query={GET_CURRENT_USER}>
    {({ loading, error, data }) => {
      if (loading) return "Loading..."
      if (error) return `Error! ${error.message}`

      const { viewer } = data;
      return (
        <div>
          {viewer.name} {viewer.login}
        </div>
      )
    }}
  </Query>
)
```

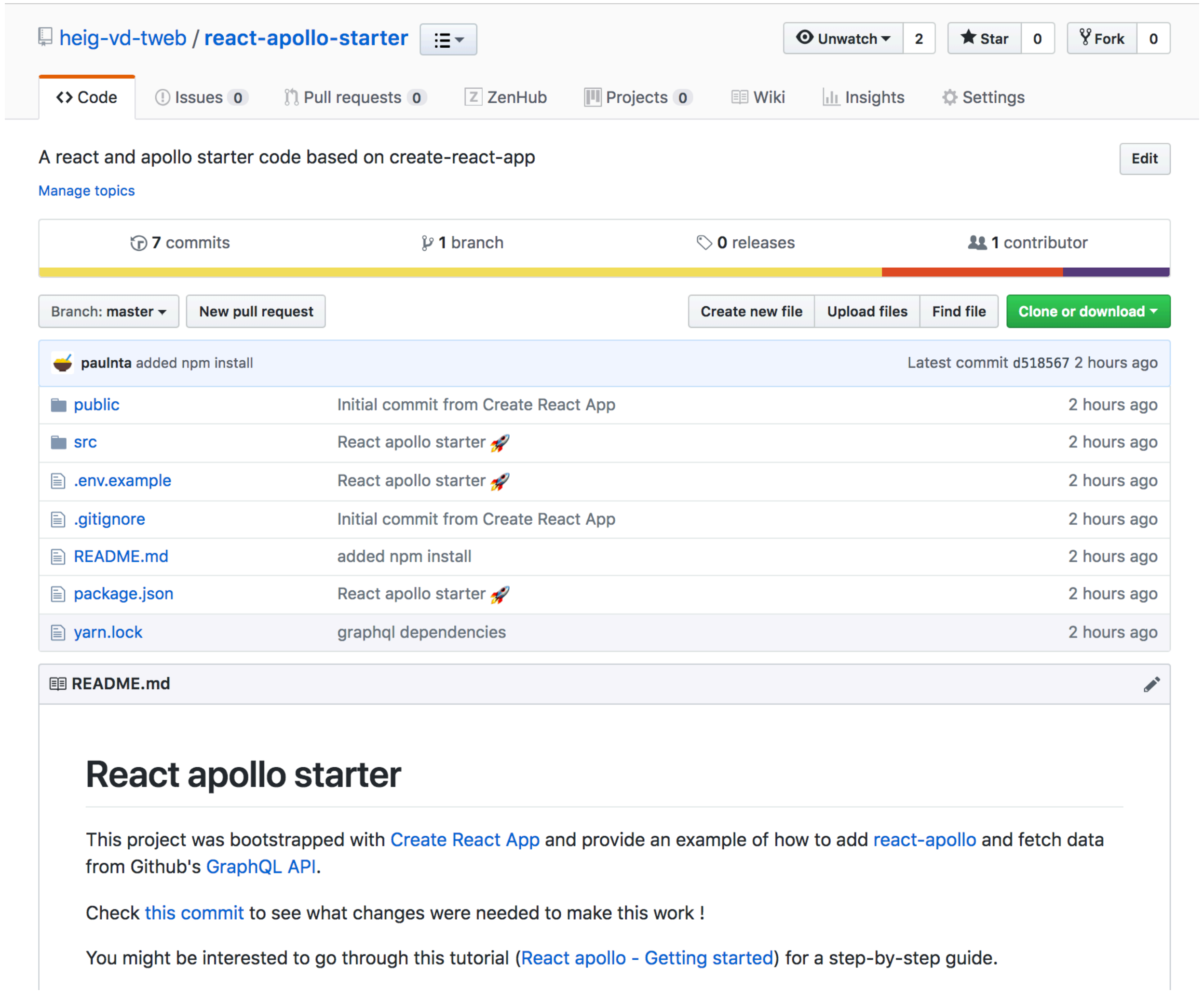
React Apollo advanced concepts

React apollo helps you implement first-class features

- caching
- mutations
- optimistic UI
- subscriptions
- pagination
- server-side rendering
- prefetching
- and more..

Data fetching from client app

React apollo starter



heig-vd-tweb / react-apollo-starter

A react and apollo starter code based on create-react-app

paulnta added npm install — Latest commit d518567 2 hours ago

public	Initial commit from Create React App	2 hours ago
src	React apollo starter 🚀	2 hours ago
.env.example	React apollo starter 🚀	2 hours ago
.gitignore	Initial commit from Create React App	2 hours ago
README.md	added npm install	2 hours ago
package.json	React apollo starter 🚀	2 hours ago
yarn.lock	graphql dependencies	2 hours ago

README.md

React apollo starter

This project was bootstrapped with Create React App and provide an example of how to add react-apollo and fetch data from Github's GraphQL API.

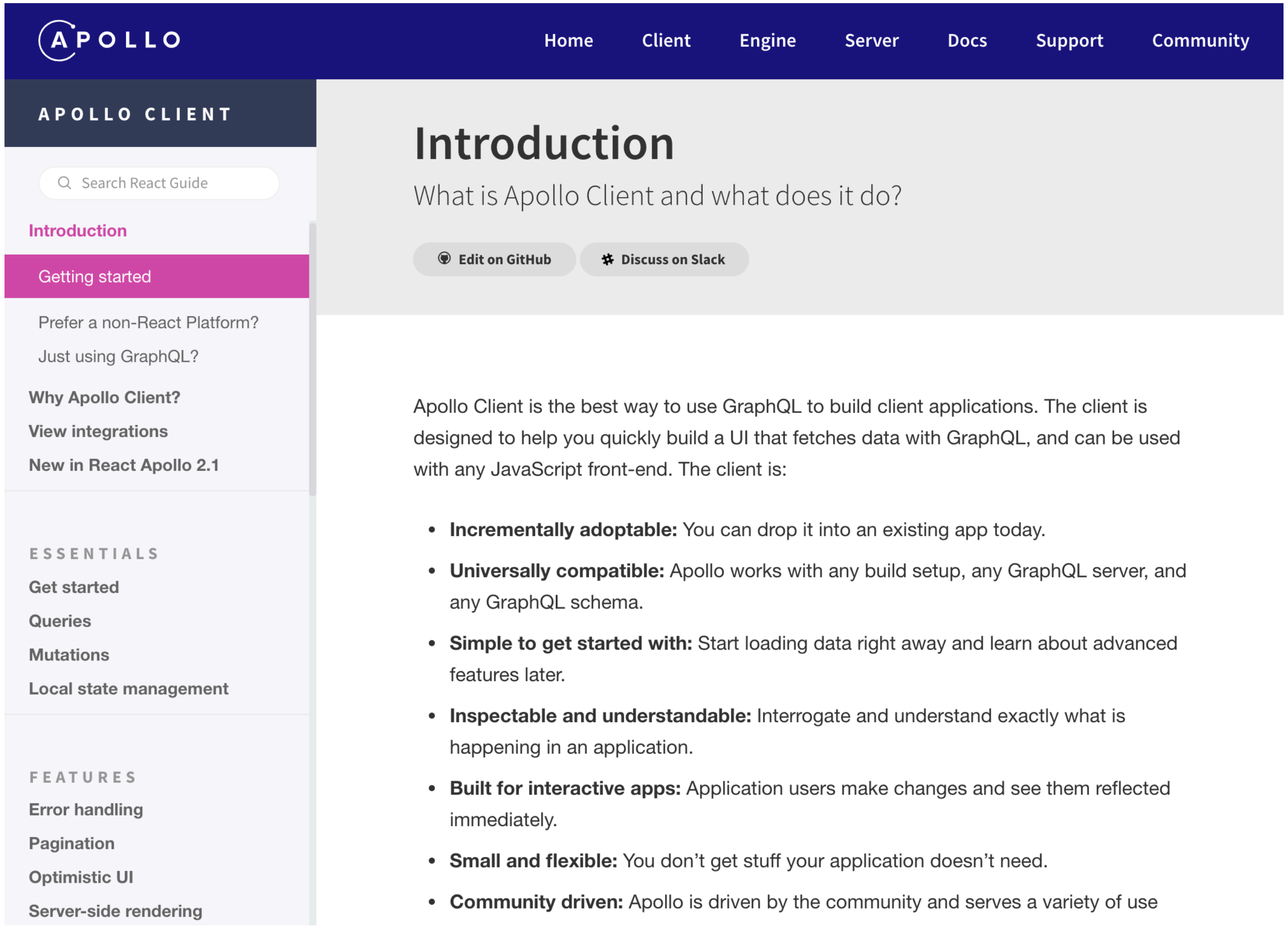
Check this commit to see what changes were needed to make this work !

You might be interested to go through this tutorial (React apollo - Getting started) for a step-by-step guide.

https://github.com/heig-vd-tweb/react-apollo-starter heig-vd

Data fetching from client app

React apollo documentation



Introduction

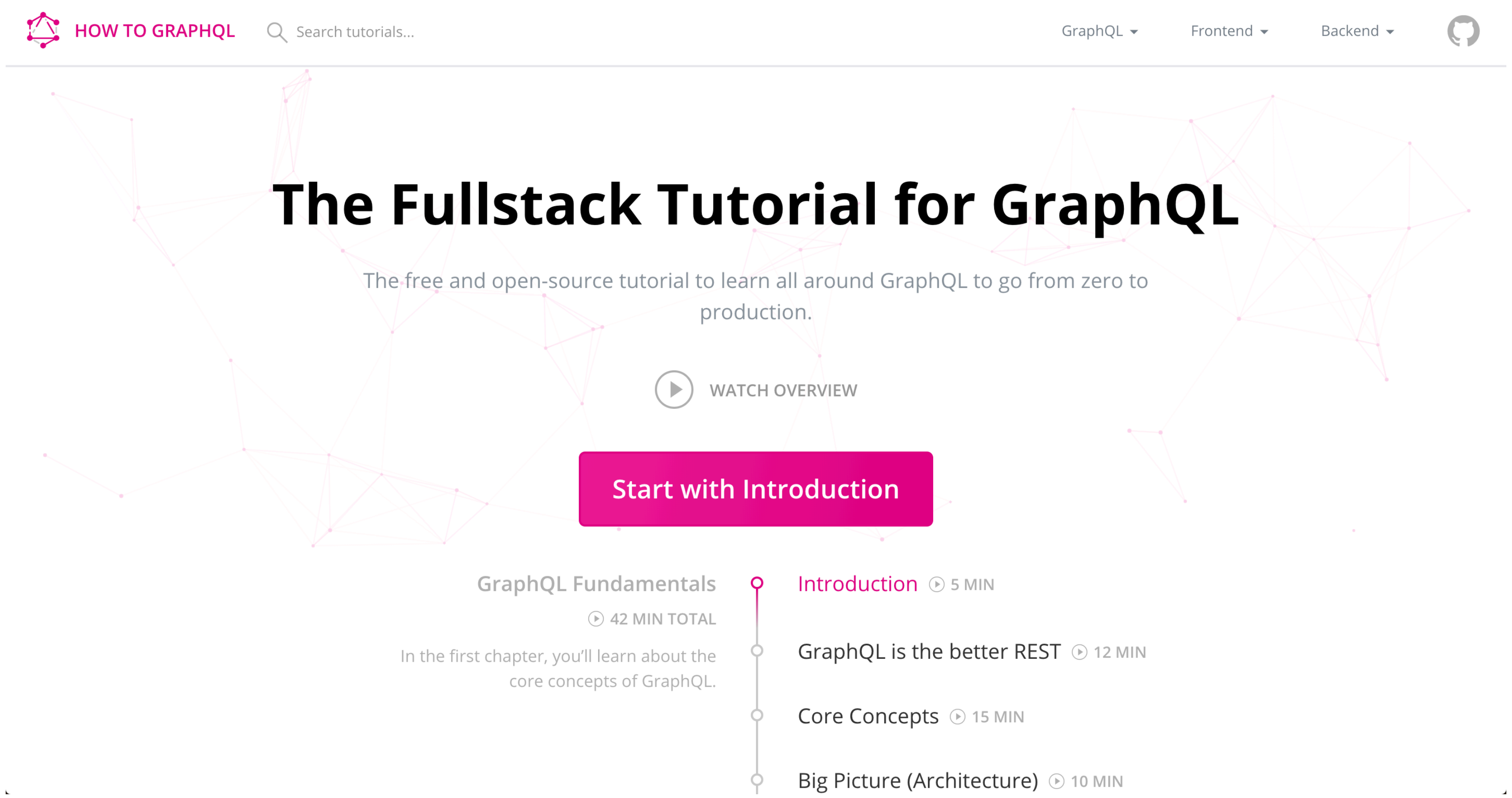
What is Apollo Client and what does it do?

Apollo Client is the best way to use GraphQL to build client applications. The client is designed to help you quickly build a UI that fetches data with GraphQL, and can be used with any JavaScript front-end. The client is:

- **Incrementally adoptable:** You can drop it into an existing app today.
- **Universally compatible:** Apollo works with any build setup, any GraphQL server, and any GraphQL schema.
- **Simple to get started with:** Start loading data right away and learn about advanced features later.
- **Inspectable and understandable:** Interrogate and understand exactly what is happening in an application.
- **Built for interactive apps:** Application users make changes and see them reflected immediately.
- **Small and flexible:** You don't get stuff your application doesn't need.
- **Community driven:** Apollo is driven by the community and serves a variety of use

https://www.apollographql.com/docs/react/

References - Graphql tutorials



HOW TO GRAPHQL Search tutorials... GraphQL Frontend Backend

The Fullstack Tutorial for GraphQL

The free and open-source tutorial to learn all around GraphQL to go from zero to production.

WATCH OVERVIEW

Start with Introduction

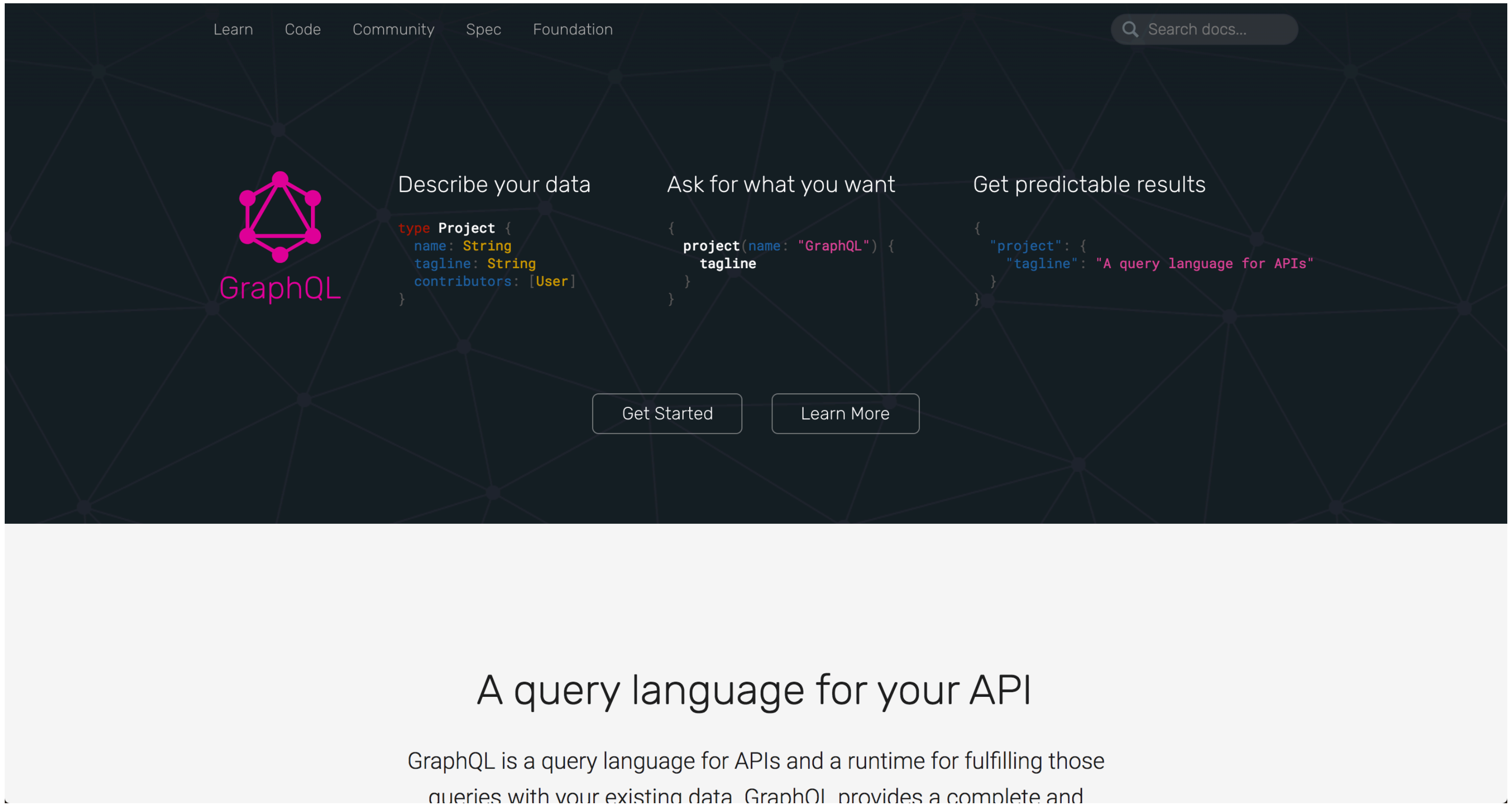
GraphQL Fundamentals
42 MIN TOTAL
In the first chapter, you'll learn about the core concepts of GraphQL.

- Introduction 5 MIN
- GraphQL is the better REST 12 MIN
- Core Concepts 15 MIN
- Big Picture (Architecture) 10 MIN

https://www.howtographql.com/

Data fetching from client app

References - Graphql documentation



Describe your data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Get predictable results

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

A query language for your API

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and

https://graphql.org/