Boost your API Development with **GraphQL and Prisma**

@nikolasburk

Nikolas Burk

Based in Berlin
Developer at @prisma

 @nikolasburk

 @nikolasburk

Agenda

1. GraphQL Introduction
2. Understanding GraphQL Servers
3. Building GraphQL Servers with Prisma

@nikolasburk

1 GraphQL Introduction

 @nikolasburk

What is GraphQL?

- **A query language for APIs** (alternative to REST, SOAP, OData, ...)
- Language-agnostic on backend and frontend
- Developed by Facebook, now led by GraphQL Foundation

GraphQL has become the new API standard

Benefits of GraphQL

- ✓ **"Query exactly the data you need"** (in a single request)
- ✓ **Declarative** & **strongly typed** schema
- ✓ Schema used as **cross-team** communication tool
- ✓ **Decouples teams**
- ✓ **Incrementally adoptable**
- ✓ Rich ecosystem & **very active community**

```
query {
  user(id: “user123”) {
    name
    posts {
      title
    }
  }
}
```


```
{
  "data" :{
    "user": {
      "name": "Sarah",
      "posts": [
        { "title": "Join us for GraphQL Conf 2019” },
        { "title": "GraphQL is the future of APIs" },
      ]
    }
  }
}
```

```
/user/123

/user/123/posts

/user/123/followers
```

REST

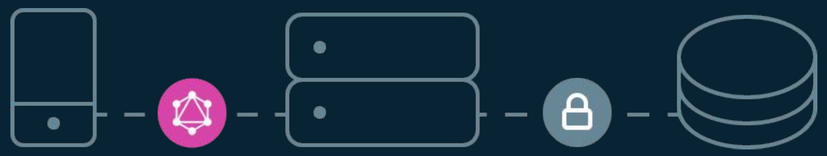
- **Multiple** endpoints
- **Server** decides what data is returned

```
query {
  user(id: 123) {
    id
    name
    posts {
      title
      thumbnailURL
    }
    followers(last: 3) {
      name
    }
```

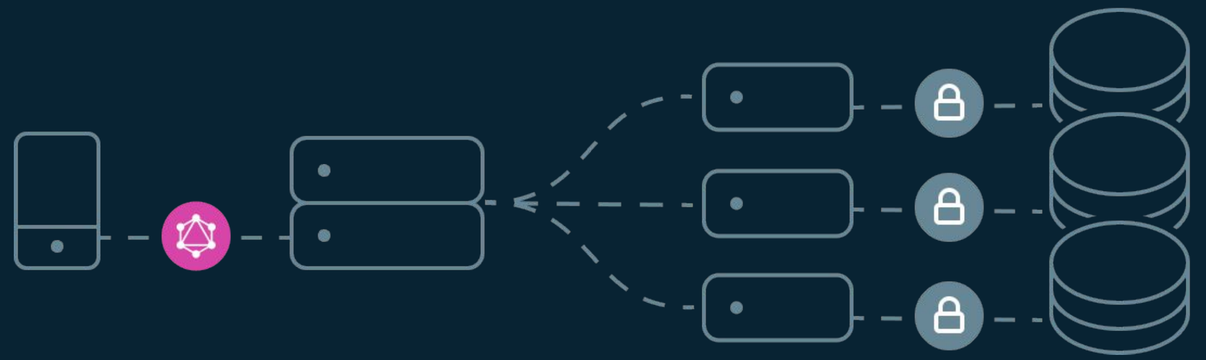
GraphQL

- **Single** endpoint
- **Client** decides what data is returned

Architectures / Use cases of GraphQL



GraphQL-Native Backend



GraphQL as API Gateway

Demo

2

Understanding GraphQL Servers

@nikolasburk

3 parts of a GraphQL server

1. API definition: **The GraphQL schema**
2. Implementation: **Resolver functions**
3. Server: **Network (HTTP), Middleware, ...**

Code-first

“Hello World”

index.ts

```
const Query = queryType({
 definition(t) {
   t.string('hello', {
     args: { name: stringArg() },
     resolve: (_, args) => {
       return `Hello ${args.name}`
     }
   })
 },
})

const schema = makeSchema({ types: [Query] })
const server = new GraphQLServer({ schema })
server.start(() => console.log(`Running on http://localhost:4000`))
```

“Hello World”

index.ts

```
const Query = queryType({
 definition(t) {
   t.string('hello', {
     args: { name: stringArg() },
     resolve: (_, args) => {
       return `Hello ${args.name}`
     }
   })
 },
})

const schema = makeSchema({ types: [Query] })
const server = new GraphQLServer({ schema })
server.start(() => console.log(`Running on http://localhost:4000`))
```

schema.graphql (generated)

```
type Query {
   hello(name: String!): String!
}
```

“Hello World”

index.ts

```
const Query = queryType({
 definition(t) {
   t.string('hello', {
     args: { name: stringArg() },
     resolve: (_, args) => {
       return `Hello ${args.name}`
     }
   })
 },
})

const schema = makeSchema({ types: [Query] })
const server = new GraphQLServer({ schema })
server.start(() => console.log(`Running on http://localhost:4000`))
```

schema.graphql (generated)

```
type Query {
   hello(name: String!): String!
}
```

`User` model: Query

index.ts

```
const User = objectType({
 name: 'User',
 definition(t) {
   t.id('id')
   t.string('name')
 }
})

const Query = queryType({
 definition(t) {
   t.field('users', {
     type: 'User',
     list: true,
     resolve: () => db.users.findAll()
   })
 },
})
```

`User` model: Query

index.ts

```
const User = objectType({
 name: 'User',
 definition(t) {
   t.id('id')
   t.string('name')
 }
})

const Query = queryType({
 definition(t) {
   t.field('users', {
     type: 'User',
     list: true,
     resolve: () => db.users.findAll()
   })
 },
})
```

`User` model: Query

index.ts

```
const User = objectType({
 name: 'User',
 definition(t) {
   t.id('id')
   t.string('name')
 }
})

const Query = queryType({
 definition(t) {
   t.field('users', {
     type: 'User',
     list: true,
     resolve: () => db.users.findAll()
   })
 },
})
```

schema.graphql (generated)

```
type User {
  id: ID!
  name: String!
}

type Query {
  users: [User!]!
}
```

`User` model: Mutation

index.ts

```
const User = objectType({
 name: 'User',
 definition(t) {
   t.id('id')
   t.string('name')
 }
})

const Mutation = mutationType({
 definition(t) {
   t.field('createUser', {
     type: 'User',
     args: { name: stringArg() },
     list: true,
     resolve: (_, args) => db.users.create({name: args.name})
   })
 },
})
```

`User` model: Mutation

index.ts

```
const User = objectType({
 name: 'User',
 definition(t) {
   t.id('id')
   t.string('name')
 }
})

const Mutation = mutationType({
 definition(t) {
   t.field('createUser', {
     type: 'User',
     args: { name: stringArg() },
     list: true,
     resolve: (_, args) => db.users.create({name: args.name})
   })
 },
})
```

schema.graphql (generated)

```
type User {
  id: ID!
  name: String!
}

type Mutation {
  createUser(name: String): User!
}
```

Demo

3

Building GraphQL Servers with Prisma

@nikolasburk

GraphQL resolvers are hard

- ✗ A lot of **CRUD boilerplate**
- ✗ **Deeply nested** queries
- ✗ **Performant database access** & N+1 problem
- ✗ Database **transactions**
- ✗ Difficult to achieve **full type-safety**
- ✗ Implementing **realtime operations**

What is Prisma?

Prisma replaces traditional ORMs and simplifies database workflows

Database Access (ORM)

Type-safe database access with the auto-generated Prisma client

Migrations

Declarative data modeling and schema migrations

Admin UI

Visual data management with Prisma Admin

Query Analytics

Quickly identify slow data access patterns

Prisma is the database-interface for application developers

GraphQL Nexus & `nexus-prisma`

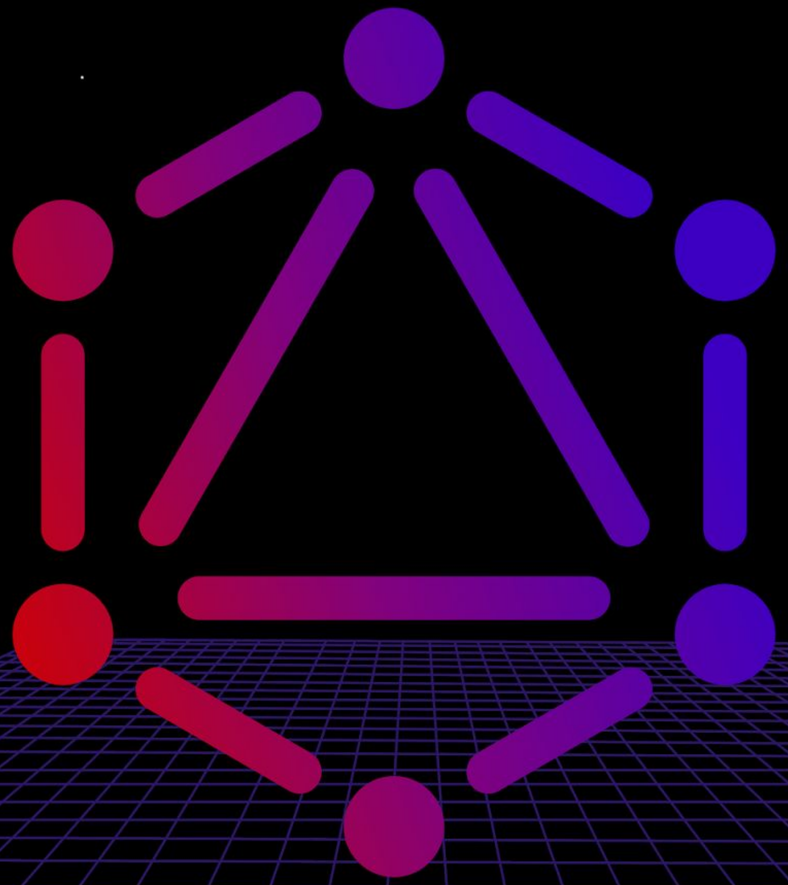
- GraphQL-native backend framework for Node.js
- Modern alternative to Ruby on Rails, Spring, Django, …
- Features:
  - ✓ Full **type-safety** (supporting JavaScript & TypeScript)
  - ✓ Deep **database integration** via Prisma
  - ✓ **Powerful CLI** (incl. scaffolding, hot-reload dev server, build, …)
  - ✓ **Compatible** with GraphQL ecosystem

Demo

prisma.io/blog 👀

GraphQL Conf Berlin 2019

www.graphqlconf.org →



Kosmos Berlin,
June 20-21

CFP Closed

Honeypot × Prisma

Thank you 🙏

 @nikolasburk

 @nikolasburk