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

GraphQL Introduction

 @nikolasburk

What is GraphQL?

- **New API standard** developed by Facebook
- Specification for **type system & query language**
- Core primitives: **Query, Mutation & Subscription**

Why use GraphQL?

- **Strongly typed schema** for your API
- Query exactly the **data you need**
- **Rich tooling ecosystem** & great community

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

HTTP POST

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

Learn more 💡

Top 5 Reasons To Use GraphQL

http://bit.ly/**top-5-reasons-for-graphql**

GraphQL Introduction

https://www.howtographql.com/**basics/0-introduction/**

 @nikolasburk

Demo

Understanding GraphQL Servers

@nikolasburk

3 parts of a GraphQL server

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

1 The GraphQL Schema

- **Strongly typed** & written in GraphQL Schema Definition Language (SDL)
- **Defines API** capabilities (contract for client-server communication)
- Special root types: **Query**, **Mutation**, **Subscription**

Example: Hello World

GRAPHQL SCHEMA

```
type Query {
  hello: String!
}
```

QUERY

```
query {
  hello
}
```

RESPONSE

```
{
  “hello”: “Hello World”
}
```

Example: CRUD for User type

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

type Query {
  user(id: ID!): User
  users: [User!]!
}

type Mutation {
  createUser(name: String!): User!
  updateUser(id: ID!, name: String!): User
  deleteUser(id: ID!): User
}
```

Example: CRUD for User type

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

type Query {
  user(id: ID!): User
  users: [User!]!
}

type Mutation {
  createUser(name: String!): User!
  updateUser(id: ID!, name: String!): User
  deleteUser(id: ID!): User
}
```

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

```
{
  “user”: {
    “name”: “Sarah”
  }
}
```

2 Resolver functions

- Concrete **implementation of the API**
- **One resolver function per field** in SDL schema
- **Query execution**: Invoke resolvers for all fields in query

Example: Hello World

GRAPHQL SCHEMA

```
type Query {
  hello: String!
}
```

RESOLVER FUNCTIONS

```
const resolvers = {
  Query: {
    hello: () => `Hello World`
  }
}
```

Example: CRUD for User type

GRAPHQL SCHEMA

```
type Query {
  user(id: ID!): User
  users: [User!]!
}

type Mutation {
  createUser(name: String!): User!
  updateUser(id: ID!, name: String!): User
  deleteUser(id: ID!): User
}

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

RESOLVER FUNCTIONS

```
const resolvers = {
  Query: {
    user: (root, args) => db.getUser(args.id),
    users: () => db.getUsers()
  },
  Mutation: {
    createUser: (root, args) =>
      db.createUser(args.name),
    updateUser: (root, args) =>
      db.updateUser(args.id, args.name),
    deleteUser: (root, args) =>
      db.deleteUser(args.id),
  },
  User: {
    id: (root) => root.id,
    name: (root) => root.name
  }
}
```

3 Setup

- "**GraphQL engine**" to orchestrate resolver invocations
- Network layer based on "**graphql-yoga**" *(network configuration: port, endpoints, CORS …)*
- **Middleware** *(analytics, logging, crash reporting …)*

…putting it all together

```
import { GraphQLServer } from 'graphql-yoga'

SCHEMA
const typeDefs = `
  type Query {
    hello: String!
  }
`

RESOLVERS
const resolvers = {
  Query: {
    hello: () => `Hello World`
  }
}

SETUP
const server = new GraphQLServer({
  typeDefs,
  resolvers
})

server.start(() => console.log(`Server is running on port 4000`))
```

…putting it all together

```
import { GraphQLServer } from 'graphql-yoga'

SCHEMA
const typeDefs = `
  type Query {
    hello: String!
  }
`

RESOLVERS
const resolvers = {
  Query: {
    hello: () => `Hello World`
  }
}

SETUP
const server = new GraphQLServer({
  typeDefs,
  resolvers
})

server.start(() => console.log(`Server is running on port 4000`))
```

…putting it all together

```
import { GraphQLServer } from 'graphql-yoga'

SCHEMA      const typeDefs = `
              type Query {
                hello: String!
              }
            `

RESOLVERS   const resolvers = {
              Query: {
                hello: () => `Hello World`
              }
            }

SETUP       const server = new GraphQLServer({
              typeDefs,
              resolvers
            })

            server.start(() => console.log(`Server is running on port 4000`))
```

…putting it all together

```
import { GraphQLServer } from 'graphql-yoga'

const typeDefs = `
  type Query {
    hello: String!
  }
`

const resolvers = {
  Query: {
    hello: () => `Hello World`
  }
}

const server = new GraphQLServer({
  typeDefs,
  resolvers
})

server.start(() => console.log(`Server is running on port 4000`))
```

SCHEMA

RESOLVERS

SETUP

Learn more 💡

GraphQL Server Basics - The Schema

http://bit.ly/**graphql-the-schema**

GraphQL Server Tutorial

https://www.howtographql.com/**graphql-js/0-introduction/**

 @nikolasburk

Demo

Building GraphQL Servers with Prisma

@nikolasburk

What is Prisma?

- **Strongly-typed access layer** for your database
- **Auto-generated** and **type-safe** database client
- **Declarative** data modelling and migrations

How does Prisma work?

1

Connect Database

Set up Prisma by connecting your database.

```
prisma
introspect
```

2

Define Datamodel

Define your datamodel using the simple and declarative SDL syntax.

3

Migrate Database

Migrate your database based on the defined datamodel.

```
prisma
generate
```

4

Generate Client

Generate the Prisma client for your preferred programming language.

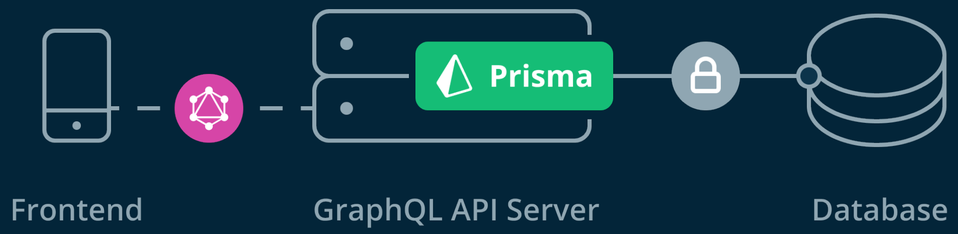
Start Building

Use the generated client to build your app.

GraphQL Architectures

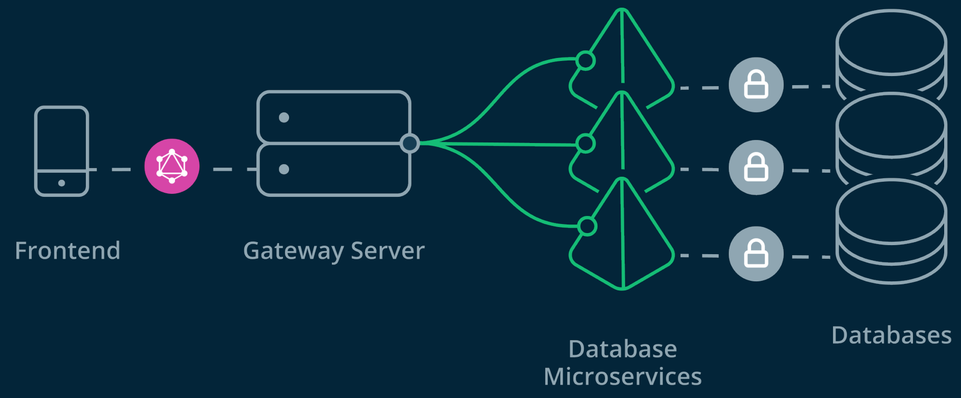
1

GraphQL "Monolith"



2

Microservices + GraphQL Gateway



GraphQL + Prisma = ❤️

- **Performant** query resolution (e.g. N+1 problem)
- Out-of-the-box **CRUD and realtime** operations
- **End-to-end** type-safety

github.com/**prisma/graphqlgen**

Learn more 💡

Prisma Quickstart

https://www.prisma.io/**docs/get-started**

What is Prisma?

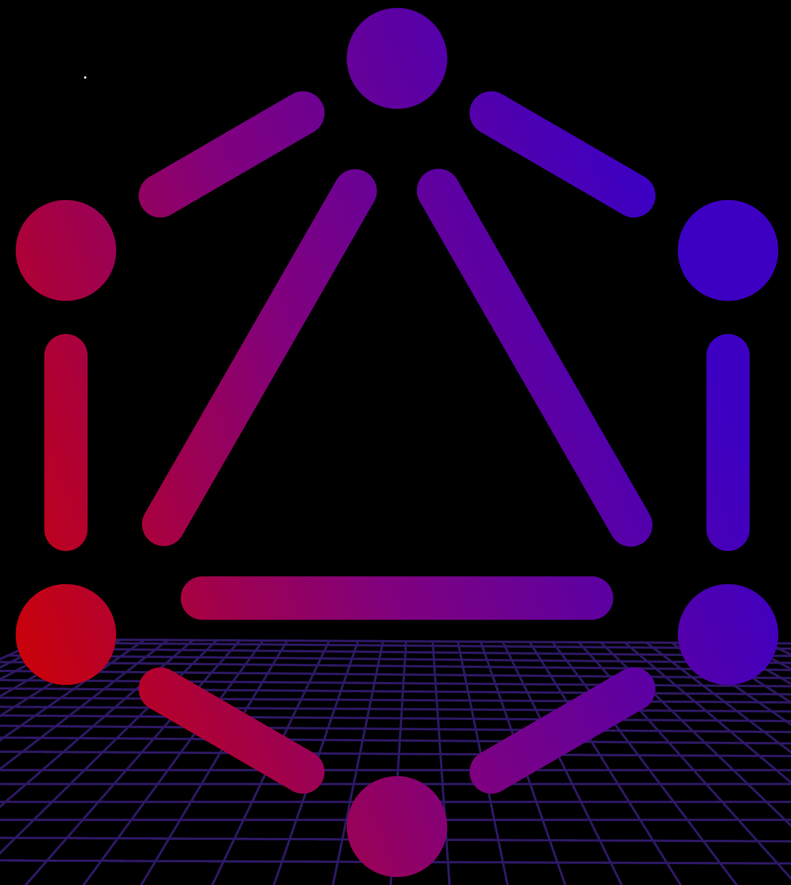
http://bit.ly/**prisma-introduction**

 @nikolasburk

Demo

GraphQL Conf Berlin 2019

www.graphqlconf.org →



Kosmos Berlin,
June 20-21

CfP Open

Honeypot × Prisma

Thank you 🙏

 @nikolasburk

 @nikolasburk