Code-first GraphQL Server Development with **GraphQL Nexus** and **Prisma**

Lukáš Huvar

Lukáš Huvar

- GraphQL Playground: core maintainer
- Prisma intern: working on Prisma Admin

@huv1k @huv1k

Agenda

1. **GraphQL servers** introduction
2. Code first approach with **GraphQL Nexus**
3. Building GraphQL servers with **Prisma** and **Yoga2**

1. GraphQL servers introduction

Parts of the GraphQL server

- API definition with **GraphQL schema**
- Resolver functions
- Server: Networking, Middleware, …

GraphQL schema

- Defines server’s **API**
- **Communication tool** between teams
- **Schema first / Code first**

Code first

- Schema is a **generated artifact** from the code
- Great for scaling in large projects
- **Type-safety** end to end
- No need for **extra tools**

Resolver functions

- Constructs data for each field
- resolve: (root, args, context, info) => {}

Server

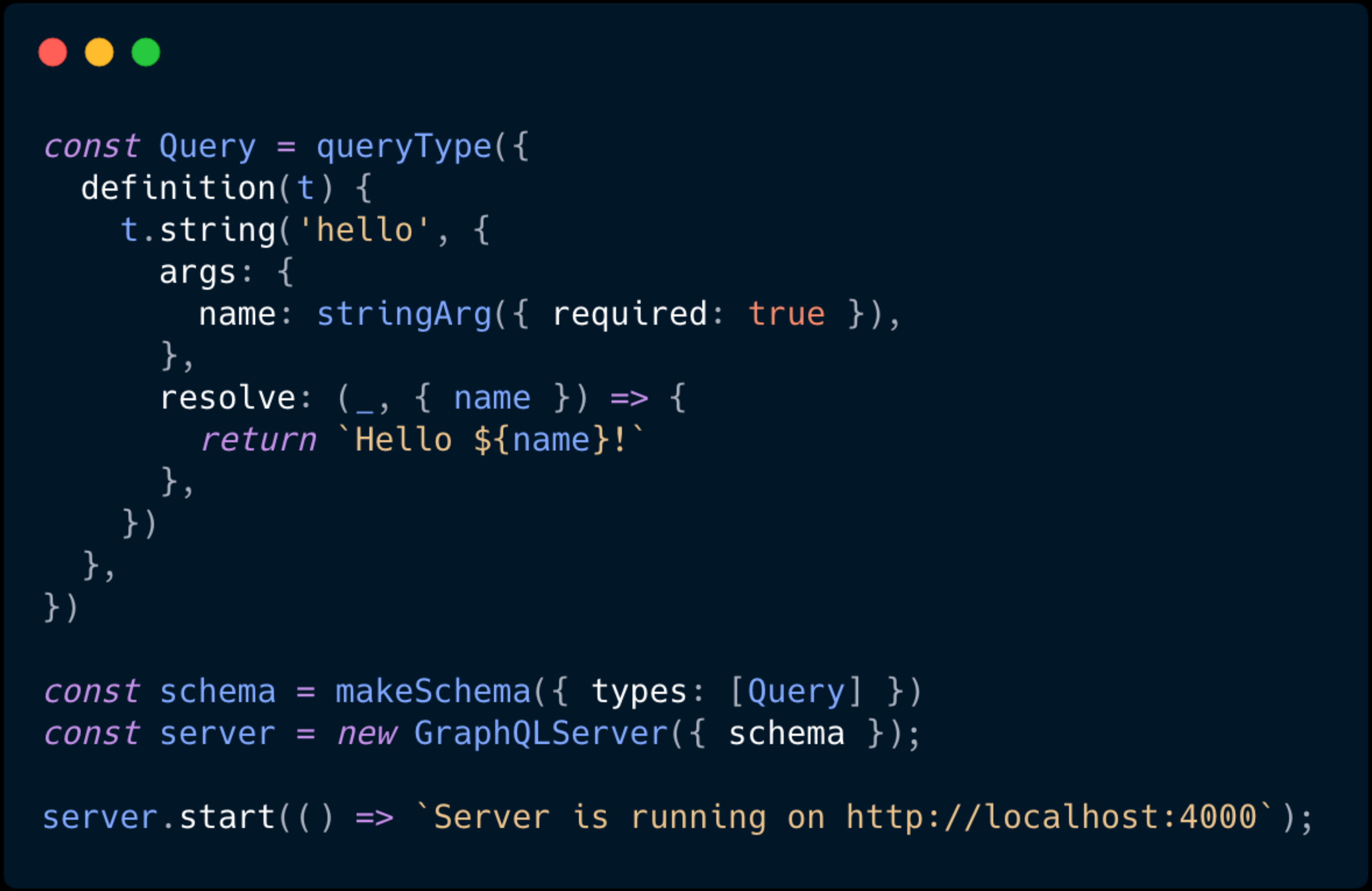
- HTTP implementation
- Middleware, logging
- Rate limiting
- Authorization, authentication and permissions

2. Code first approach with **GraphQL Nexus**

GraphQL Nexus

- **Expressive, declarative** API for building the schema
- Built on top of the **graphql-js**
- **Type safe** by default, even with JavaScript

Hello world!

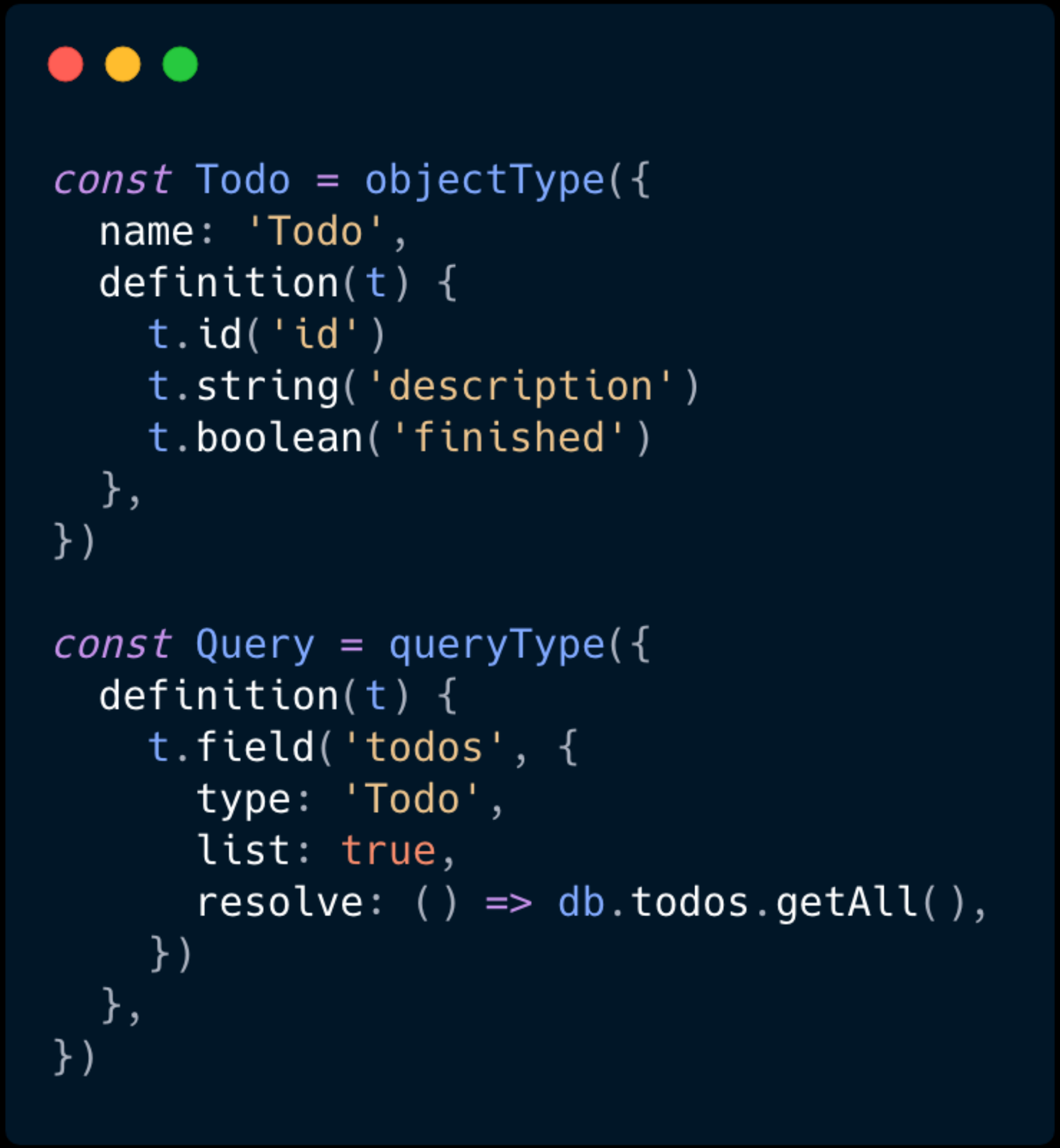


```
const Query = queryType({
  definition(t) {
    t.string('hello', {
      args: {
        name: stringArg({ required: true }),
      },
      resolve: (_, { name }) => {
        return `Hello ${name}!`
      },
    })
  },
})

const schema = makeSchema({ types: [Query] })
const server = new GraphQLServer({ schema });

server.start(() => `Server is running on http://localhost:4000`);
```

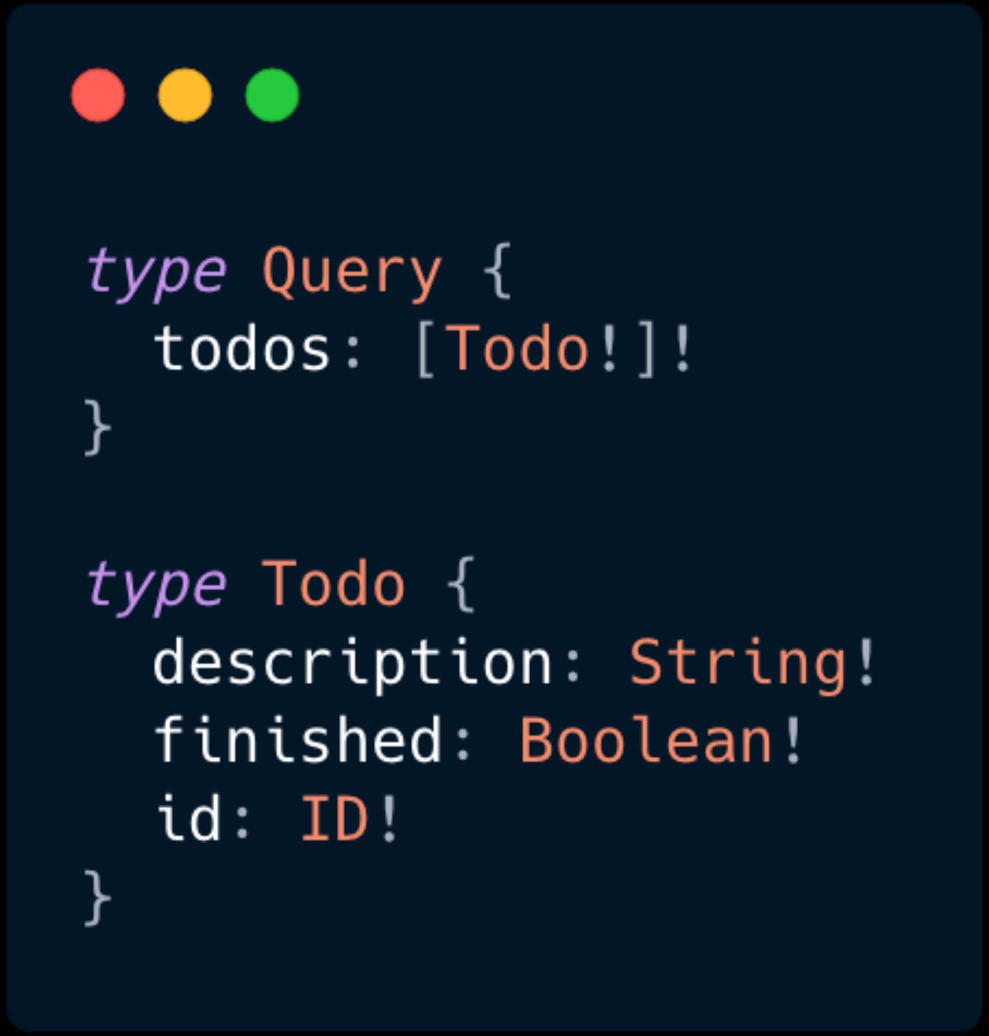
Query Todos



```
const Todo = objectType({
  name: 'Todo',
  definition(t) {
    t.id('id')
    t.string('description')
    t.boolean('finished')
  },
})

const Query = queryType({
  definition(t) {
    t.field('todos', {
      type: 'Todo',
      list: true,
      resolve: () => db.todos.getAll(),
    })
  },
})
```

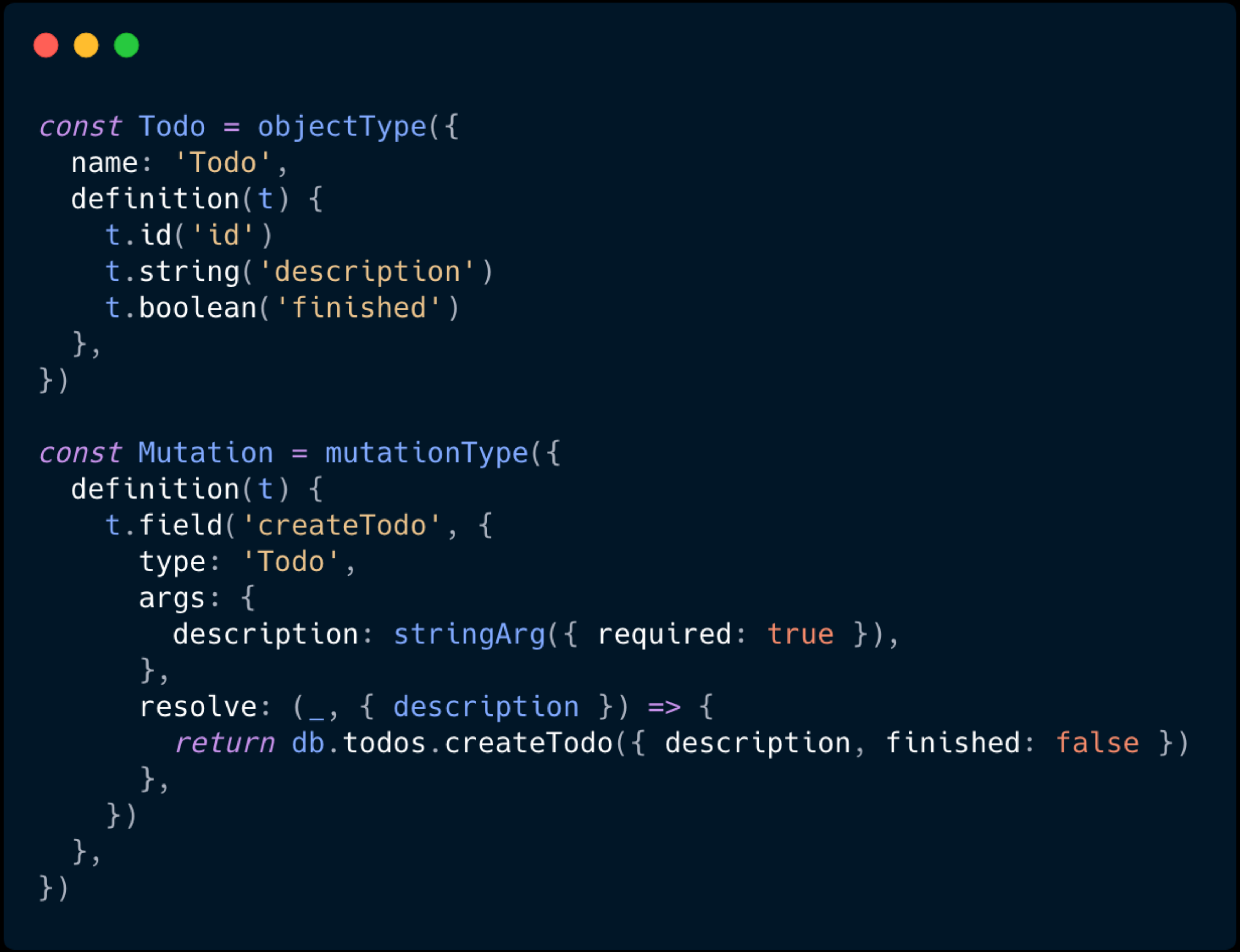
Query Todos



```graphql
type Query {
  todos: [Todo!]!
}

type Todo {
  description: String!
  finished: Boolean!
  id: ID!
}
```

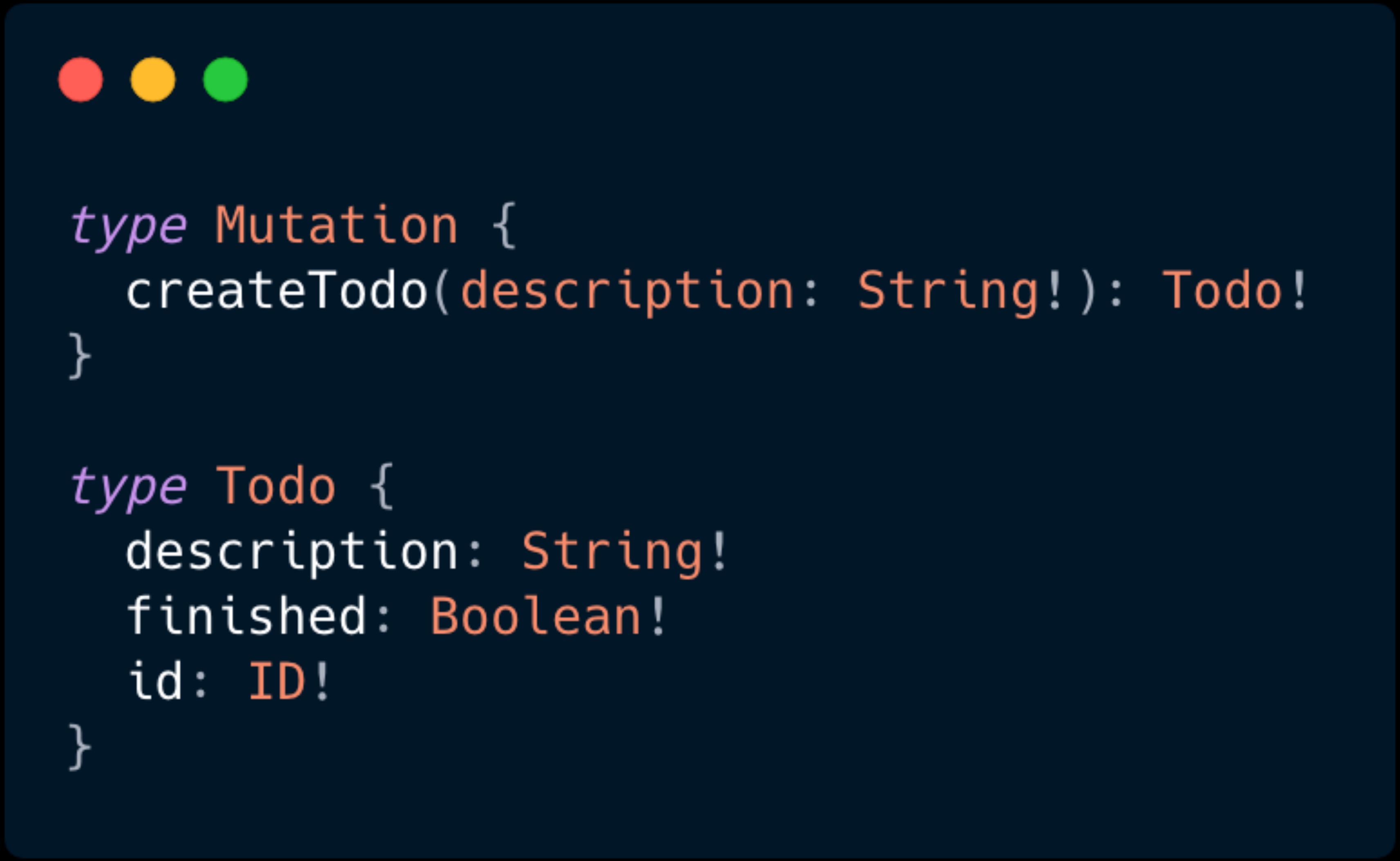
Mutation Todos



```
const Todo = objectType({
  name: 'Todo',
  definition(t) {
    t.id('id')
    t.string('description')
    t.boolean('finished')
  },
})

const Mutation = mutationType({
  definition(t) {
    t.field('createTodo', {
      type: 'Todo',
      args: {
        description: stringArg({ required: true }),
      },
      resolve: (_, { description }) => {
        return db.todos.createTodo({ description, finished: false })
      },
    })
  },
})
```

Mutation Todos



```graphql
type Mutation {
  createTodo(description: String!): Todo!
}

type Todo {
  description: String!
  finished: Boolean!
  id: ID!
}
```

Demo 💻

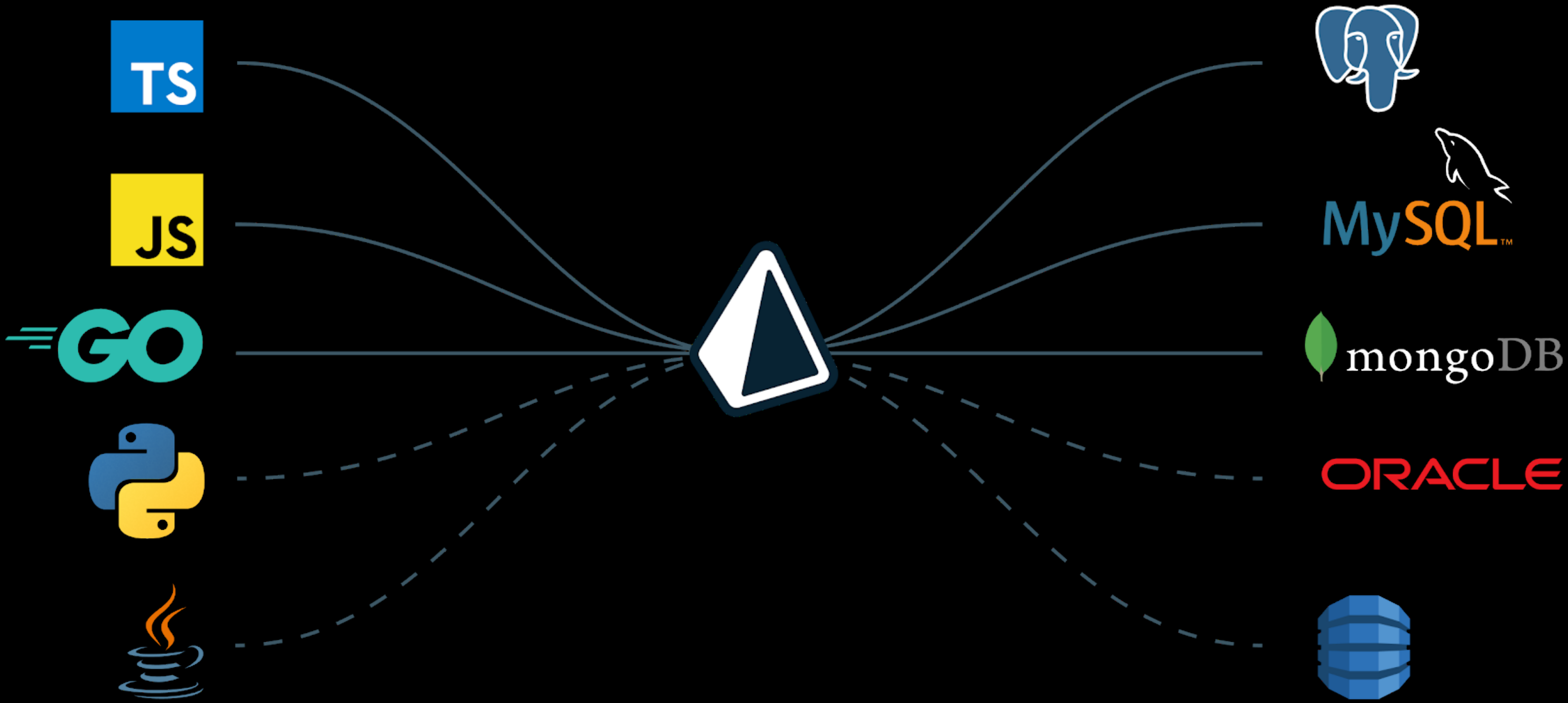
3. Building GraphQL server with **Prisma** and **Yoga2**

What is Prisma?

Prisma replaces traditional ORMs and simplifies database workflows

- **Access:** Type-safe database access with **Prisma client**
- **Migrate:** Declarative data modeling and migrations
- **Manage:** Visual data management with **Prisma Admin**

Prisma is the database-interface for application developers



GraphQL + Prisma = Yoga

- GraphQL framework built with **conventions** over **configurations**
- Modern alternative to Ruby on Rails, Spring, Django
- **Features:**
  - ✓ Fully **type-safe** (supports JavaScript & Typescript)
  - ✓ Deep **database integration** with Prisma
  - ✓ Powerful **CLI** (scaffolding, dev server, build, …)
  - ✓ Compatible with **GraphQL ecosystem** (GraphQL Shield)

Demo 💻

Thank you 😇

https://www.huvik.dev

@huv1k

@huv1k