Building a GraphQL API with **MongoDB, Prisma & TypeScript**

Matthias Oertel

Based in Berlin
Engineer at @prisma

@do4gr

@oertel_matthias

Agenda

1. GraphQL Introduction
2. Understanding GraphQL Servers
3. Building GraphQL Servers with MongoDB, Prisma & TypeScript

1 GraphQL Introduction

What is GraphQL?

- **A query language for APIs** (alternative to REST)
- **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 as **cross-team** communication tool (decouples teams)

```
GET /user/123

GET /user/123/posts
```

REST

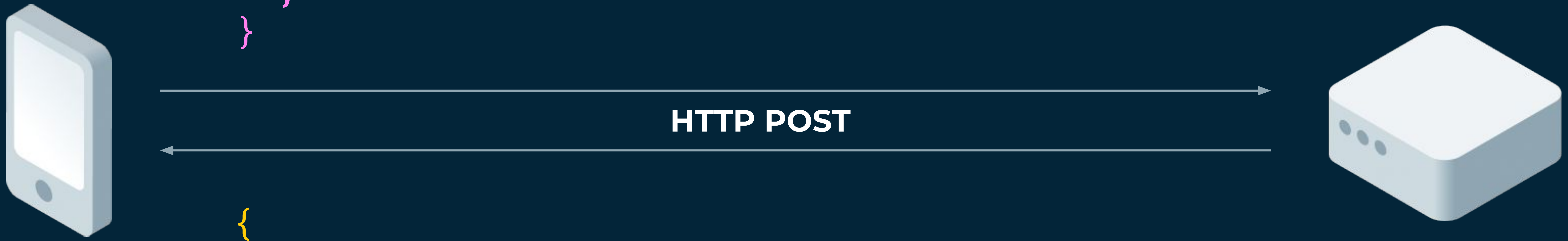
- **Multiple** endpoints
- **Server** decides what data is returned

```
POST /graphql { "query":
 "query {
    user(id: 123){
      name
      posts {
        title
      }
    }
  }"
}
```

GraphQL

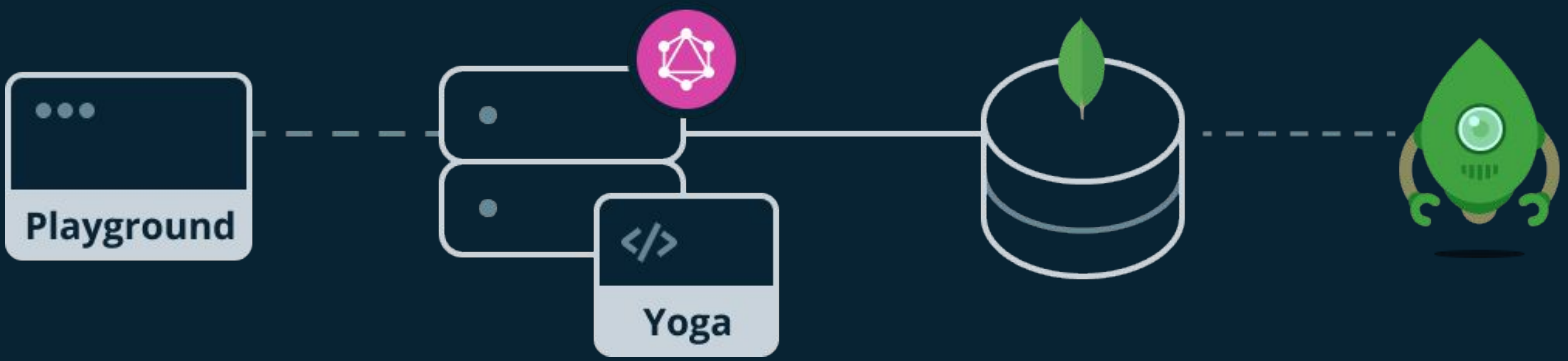
- **Single** endpoint
- **Client** decides what data is returned

```
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" },
      ]
    }
  }
}
```

Architecture of demo



🍿 Demo

2

Understanding GraphQL Servers

Three parts of a GraphQL server

- API definition: GraphQL schema
- Implementation: Resolver Functions
- Server: Network (HTTP), Middleware, Auth ...

SDL-first vs 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

```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)

```graphql
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('user', {
     type: 'User',
     args: {id: idArg()},
     resolve: () => userCollection.findOne(id)
   })
 },
})
```

`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('user', {
     type: 'User',
     args: {id: idArg()},
     resolve: () => userCollection.findOne(id)
   })
 },
})
```

`User` model: Query

index.ts

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

const Query = queryType({
 definition(t) {
   t.field('user', {
     type: 'User',
     args: {id: idArg()},
     resolve: () => userCollection.findOne(id)
   })
 },
})
```

schema.graphql (generated)

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

type Query {
  user(id: ID): 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() },
     resolve: (_, args) => userCollection.insertOne({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() },
     resolve: (_, args) => userCollection.insertOne({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

GraphQL resolvers are hard

- ✗ A lot of **CRUD boilerplate**
- ✗ **Deeply nested** queries
- ✗ **Performant database access** & N+1 problem
- ✗ 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 datamodel migrations

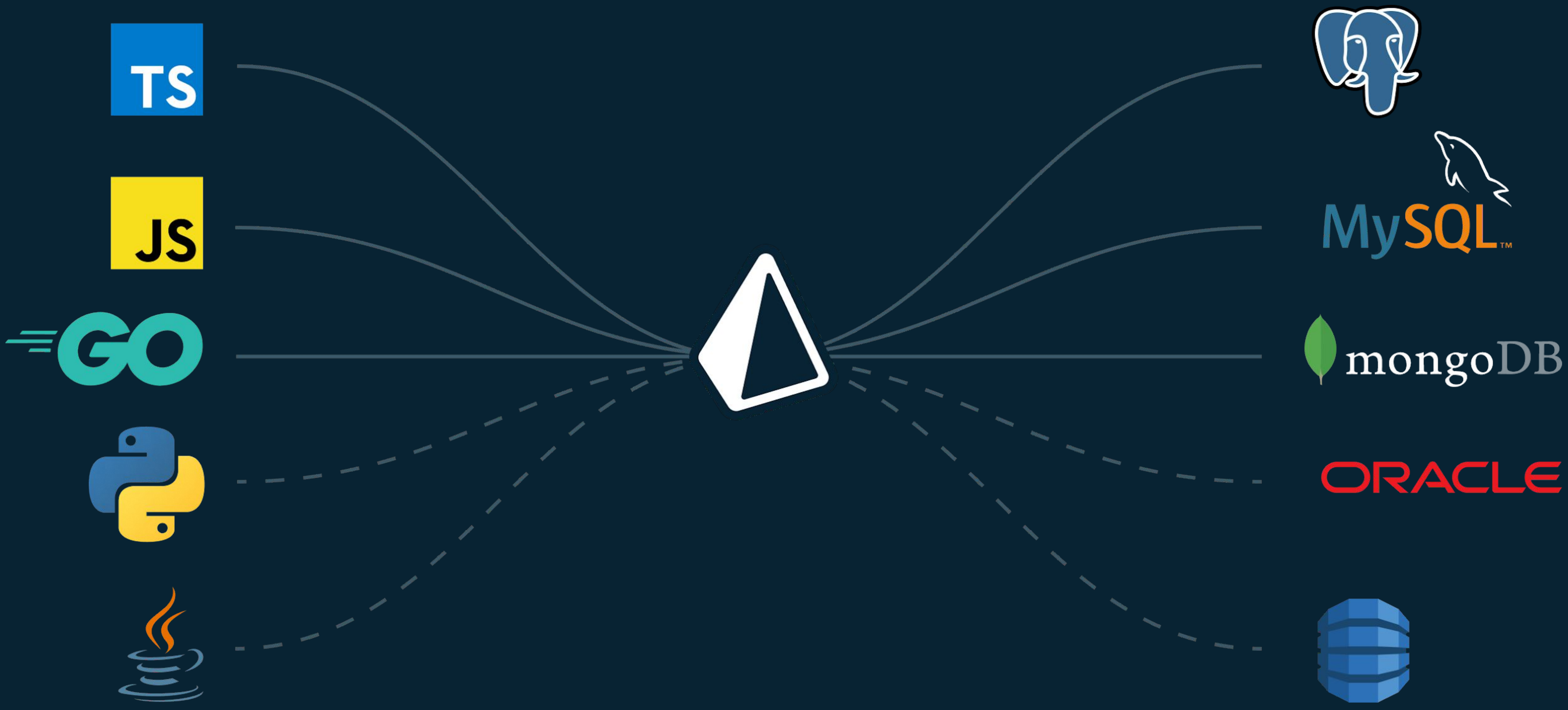
Admin UI

Visual data management with Prisma Studio

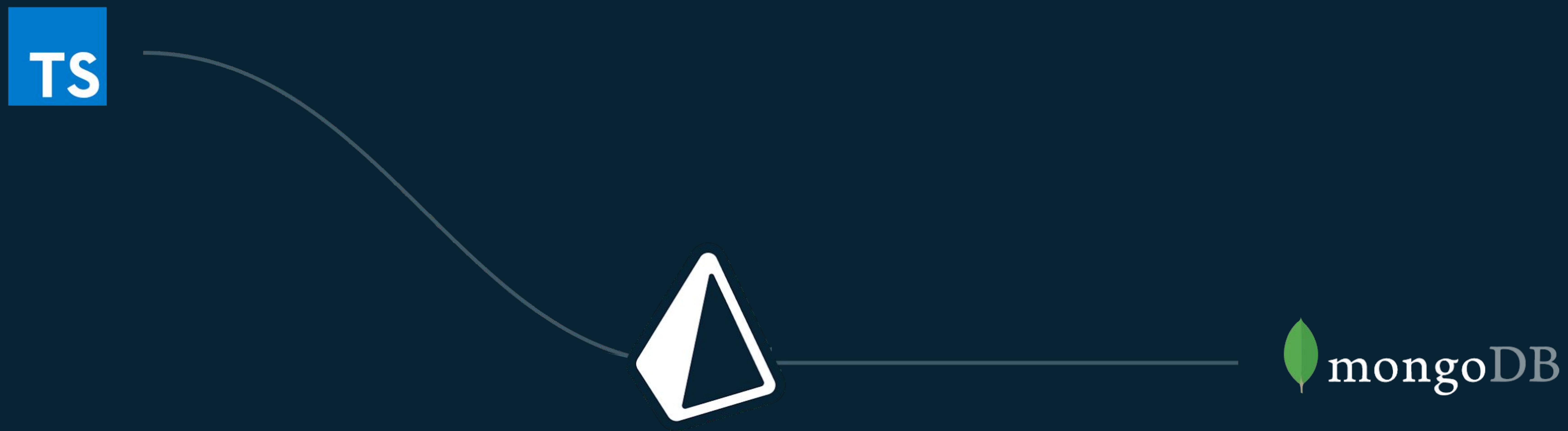
Query Analytics

Quickly identify slow data access patterns

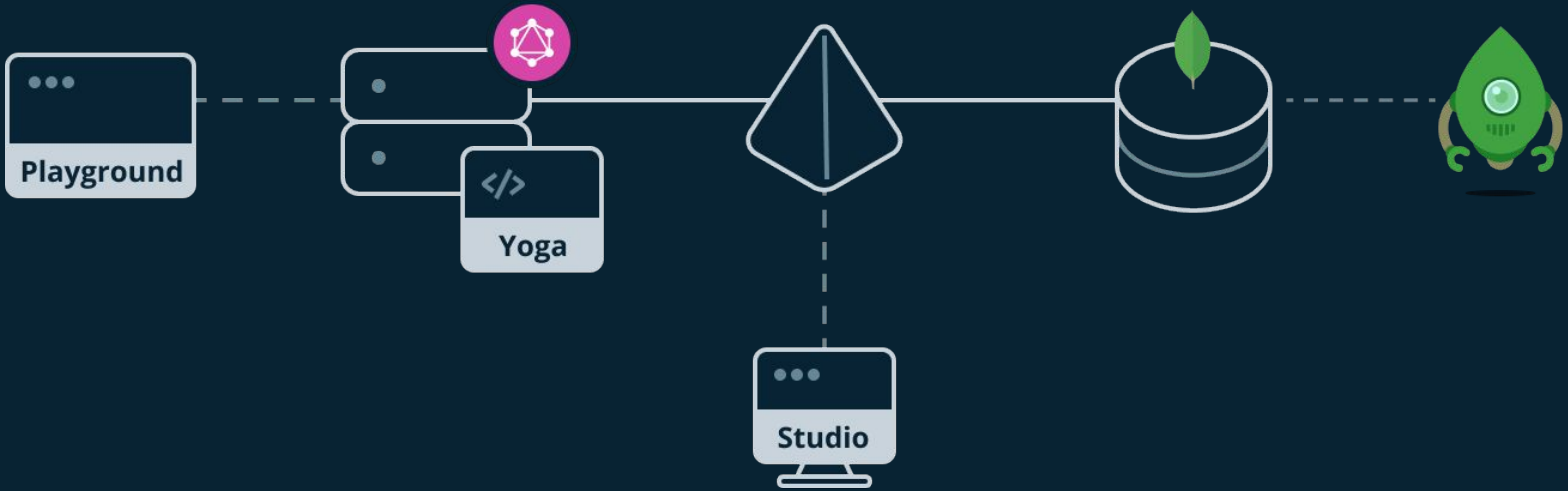
Prisma is the database-interface for application developers



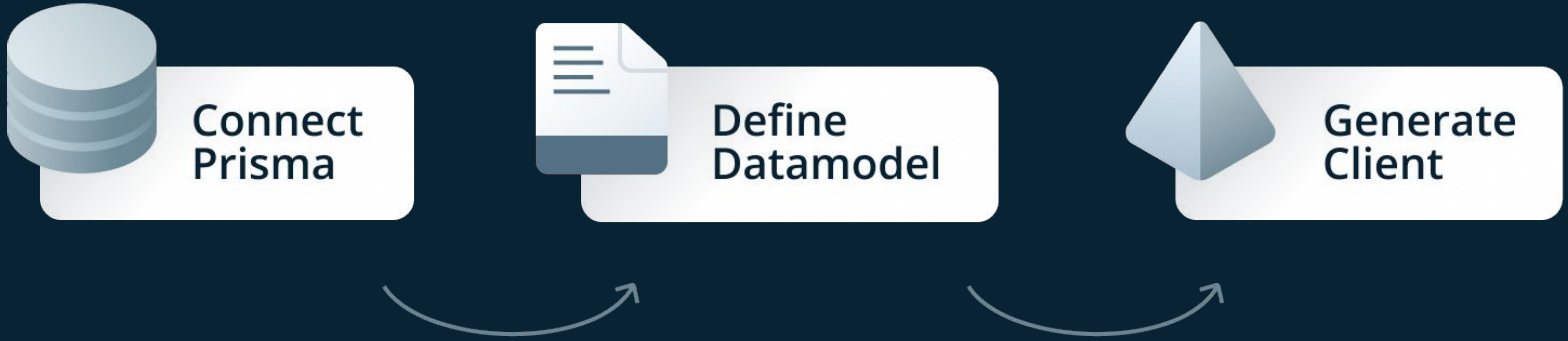
Prisma is the database-interface for application developers



Demo Architecture with Prisma



How Prisma works with MongoDB



🍿 Demo

Recap

GraphQL

- Flexible APIs
- Get what you need
- Schema decouples teams

Prisma

- Powerful primitives
- Auto-Completion
- Typesafety
- Multi-Language

MongoDB

- Flexibility
- High Scalability
- Embedded Documents

Thank you 🙏

Materials: https://github.com/do4gr/mongoworld

@do4gr

@oertel_matthias