4️⃣

4. GraphQL API

Goal

The goal of this lesson is to use your freshly gained knowlec
implement some resolvers of a GraphQL API using Apollo Se

Setup

You can continue working in the same `prisma-workshop` p
However, the starter for this lesson is located in the `graphq`
cloned.

Before you switch to that branch, you need to commit the cı
simplicity, you can use the `stash` command to do this:

```
git stash
```

After you ran this command, you can switch to the `graphql`
`migrations` directory and `dev.db` file:

```
git checkout graphql-api rm -rf prisma/migratior
```

Next, wipe your npm dependencies and re-install them to ac
`package.json`:

```
rm -rf node_modules npm install
```

The data model you will use here is similar to the one from t

```
model User { id Int @id @default(autoincrement()
String? posts Post[] } model Post { id Int @id @
createdAt DateTime @default(now()) updatedAt Dat
content String? published Boolean @default(false
author User? @relation(fields: [authorId], refer
```

Since you are starting over with your Prisma setup, you have
tables. Run the following command to do this:

```
npx prisma migrate dev --name init
```

Finally, you can seed the database with some sample data t
`prisma/seed.ts` file. You can execute this seed script with

```
npx prisma db seed --preview-feature
```

That's it, you're ready for your tasks now!

Tasks

You can find the tasks for this lesson inside the `src/index.`
is to insert the right Prisma Client queries for each GraphQL

You can test your implementations by starting the server an
at `http://localhost:4000`.

`Query.allUsers: [User!]!`

Fetches all users.

▼ Solution

```
allUsers: (_parent, _args, context: Context)
context.prisma.user.findMany() },
```

▼ Sample query

```
{ allUsers { id name email posts { id title
```

`Query.postById(id: Int!): Post`

Fetches a post by its ID.

▼ Solution

```
postById: (_parent, args: { id: number }, co
context.prisma.post.findUnique({ where: { id
```

▼ Sample query

```
{ postById(id: 1) { id title content publish
email } } }
```

`Query.feed(searchString: String, skip: Int`

Fetches all published posts and optionally paginates and/or
search string appears in either title or content.

▼ Solution

```
feed: ( _parent, args: { searchString: strin
undefined; take: number | undefined; }, cont
args.searchString ? { OR: [ { title: { conta
string } }, { content: { contains: args.sear
{}; return context.prisma.post.findMany({ wh
}, skip: Number(args.skip) || undefined, tak
undefined, }); },
```

▼ Sample query

```
{ feed { id title content published viewCoun
```

`Query.draftsByUser(id: Int!): [Post]`

Fetches the unpublished posts of a specific user.

▼ Solution

```
draftsByUser: (_parent, args: { id: number }
return context.prisma.user.findUnique({ wher
where: { published: false } }) },
```

▼ Sample query

```
{ draftsByUser(id: 3) { id title content pub
name email } } }
```

`Mutation.signupUser(name: String, email: S`

Creates a new user.

▼ Solution

```
signupUser: ( _parent, args: { name: string 
context: Context ) => { return context.prism
args.name, email: args.email } }) },
```

▼ Sample mutation

```
mutation { signupUser( name: "Nikolas" email
posts { id } } }
```

`Mutation.createDraft(title: String!, conte`
`String): Post`

Creates a new post.

▼ Solution

```
createDraft: ( _parent, args: { title: strin
undefined; authorEmail: string }, context: C
context.prisma.post.create({ data: { title: 
args.content, author: { connect: { email: ar
```

▼ Sample mutation

```
mutation { createDraft( title: "Hello World"
) { id published viewCount author { id email
```

`Mutation.incrementPostViewCount(id: Int!)`

Increments the views of a post by 1.

▼ Solution

```
incrementPostViewCount: ( _parent, args: { i
) => { return context.prisma.post.update({ w
viewCount: { increment: 1 } } }) },
```

▼ Sample mutation

```
mutation { incrementPostViewCount(id: 1) { i
```

`Mutation.deletePost(id: Int!): Post`

Deletes a post.

▼ Solution

```
deletePost: (_parent, args: { id: number },
context.prisma.post.delete({ where: { id: ar
```

▼ Sample mutation

```
mutation { deletePost(id: 1) { id } }
```

`User.posts: [Post!]!`

Returns the posts of a given user.

▼ Solution

```
User: { posts: (parent, _args, context: Cont
context.prisma.user.findUnique({ where: { id
```

`Post.author: User`

Returns the author of a given post.

▼ Solution

```
Post: { author: (parent, _args, context: Con
context.prisma.post.findUnique({ where: { id
},
```