ГЛАВА 4

Наш первый GraphQL API

Смею предположить, что вы человек, раз уж читаете эти строки. Будучи человеком, вы обладаете рядом интересов и пристрастий. У вас также есть члены семьи, друзья, знакомые, одноклассники и коллеги. У всех них, в свою очередь, тоже есть свои социальные связи, интересы и пристрастия. Некоторые из этих связей и интересов пересекаются, а некоторые — нет. В совокупности у каждого из нас есть связанный круг людей из нашей жизни.

GraphQL как раз и создавался для разрешения сложностей в таких запутанных типах взаимосвязей данных при разработке API. Написав GraphQL API, мы получаем возможность эффективно связывать данные, что снижает сложность и число запросов и в то же время позволяет нам передавать на клиент только нужные данные.

Не слишком ли это сложно для приложения, работающего с заметками? Может быть, так оно и звучит, но как вы увидите, инструменты и техники, предоставляемые экосистемой GraphQL JavaScript, не только делают возможными, но и упрощают любые виды разработки API.

В этой главе мы будем создавать GraphQL API, используя пакет `apollo-server-express`. Ради этого мы изучим фундаментальные темы, касающиеся GraphQL, напишем схему GraphQL, разработаем код для разрешения функций этой схемы и обратимся к нашему API через пользовательский интерфейс GraphQL Playground.

Превращение сервера в API (ну, вроде того)

Давайте начнем разработку API с преобразования нашего сервера Express в сервер GraphQL при помощи пакета `apollo-server-express`. Apollo Server (https://oreil.ly/1fNt3) — это открытая серверная библиотека GraphQL, работающая с большим числом серверных фреймворков Node.js, включая Express, Connect, Hapi и Koa. Он позволяет передавать данные из Node.js-приложения в виде GraphQL API и предоставляет полезные инструменты вроде GraphQL Playground — визуального помощника для взаимодействия с нашим API при разработке.

Чтобы написать API, мы изменим код веб-приложения из предыдущей главы. Давайте начнем с включения пакета `apollo-server-express`. Добавьте в начало файла src/index.js следующее:

```
const { ApolloServer, gql } = require('apollo-server-express');
```

Теперь, когда мы импортировали `apollo-server`, перейдем к настройке базового приложения GraphQL. Такие приложения состоят из двух основных компонентов: схемы определений типов и распознавателей, разрешающих запросы и мутации данных. Если вы ничего не поняли, это нормально. Мы реализуем ответ API «Hello World» и в процессе дальнейшей разработки будем подробнее изучать эти особенности GraphQL.

Для начала давайте построим базовую схему, которую будем хранить в переменной `typeDefs`. Эта схема будет описывать один `Query` («запрос») под названием `hello`, возвращающий строку:

```
// Построение схемы с использованием языка схем GraphQL
const typeDefs = gql`
  type Query {
    hello: String
  }
`;
```

Настроив схему, мы можем добавить распознаватель, который будет возвращать значение пользователю. Им будет простая функция, возвращающая строку «Hello World!»:

```
// Предоставляем функцию разрешения для полей схемы
const resolvers = {
  Query: {
    hello: () => 'Hello world!'
  }
};
```

Под конец мы интегрируем Apollo Server, который будет обслуживать наш GraphQL API. Для этого добавим некоторые специфичные для него настройки и промежуточное ПО, после чего обновим код `app.listen`:

```
// Настройка Apollo Server
const server = new ApolloServer({ typeDefs, resolvers });

// Применяем промежуточное ПО Apollo GraphQL и указываем путь к /api
server.applyMiddleware({ app, path: '/api' });

app.listen({ port }, () =>
  console.log(
    `GraphQL Server running at http://localhost:${port}${server.graphqlPath}`
  )
);
```

После всего этого файл src/index.js должен выглядеть так:

```
const express = require('express');
const { ApolloServer, gql } = require('apollo-server-express');

// Запускаем сервер на порте, указанном в файле .env, или на порте 4000
const port = process.env.PORT || 4000;

// Строим схему с помощью языка схем GraphQL
const typeDefs = gql`
  type Query {
    hello: String
  }
`;

// Предоставляем функции распознавания для полей схемы
const resolvers = {
  Query: {
    hello: () => 'Hello world!'
  }
};

const app = express();

// Настраиваем Apollo Server
const server = new ApolloServer({ typeDefs, resolvers });

// Применяем промежуточное ПО Apollo GraphQL и указываем путь к /api
server.applyMiddleware({ app, path: '/api' });

app.listen({ port }, () =>
  console.log(
    `GraphQL Server running at http://localhost:${port}${server.graphqlPath}`
  )
);
```

Если вы оставили процесс nodemon запущенным, то можете переходить прямо в браузер. В противном случае вам потребуется запустить сервер, набрав в терминале npm run dev. Далее перейдите по ссылке http://localhost:4000/api: там будет GraphQL Playground (рис. 4.1). Это веб-приложение, идущее в комплекте с Apollo Server, представляет одно из важных преимуществ работы с GraphQL. В нем вы можете выполнять запросы и вносить изменения и при этом сразу видеть результаты. Кроме того, можно найти автоматически создаваемую документацию для API во вкладке Schema.

У синтаксиса GraphQL Playground по умолчанию темная тема. В книге же я буду использовать светлую из-за более высокой контратстности. Изменить это можно в настройках самой GraphQL Playground, доступ к которым можно получить, кликнув на иконке шестеренки.

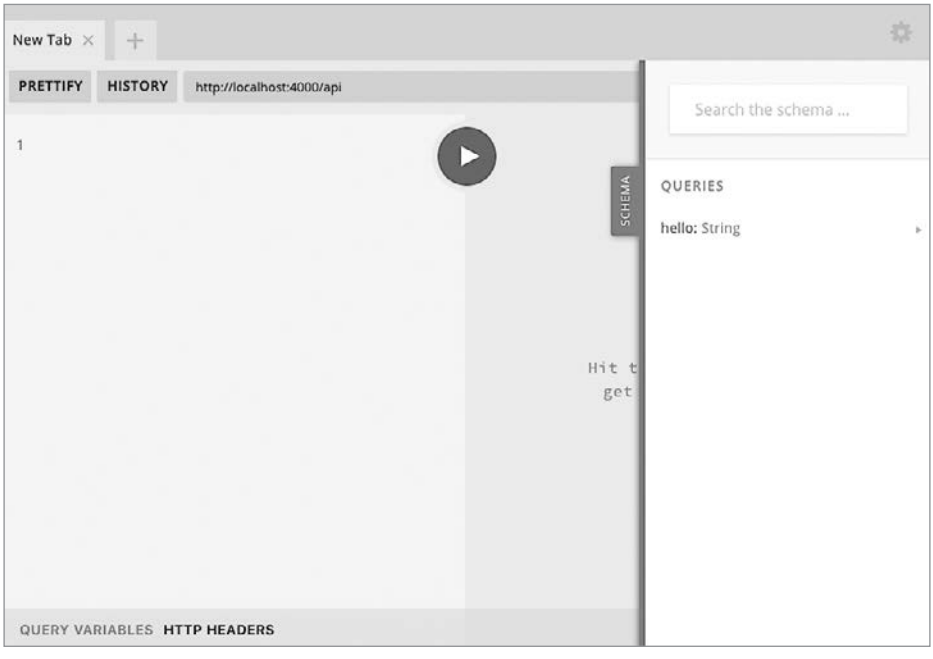

Рис. 4.1. GraphQL Playground

Рис. 4.2. Запрос hello

Теперь мы можем писать запрос к GraphQL API. Для этого наберите в GraphQL Playground следущее:

```
query {
  hello
}
```

Когда вы кликнете по кнопке Play, запрос должен вернуть следующее (рис. 4.2):

```
{
  "data": {
    "hello": "Hello world!"
  }
}
```

Вот и все! Теперь у нас есть рабочий GraphQL API, к которому мы обратились через GraphQL Playground. Наш API получает запрос `hello` и возвращает строку `Hello world!`. Но важнее то, что теперь у нас есть структура для построения полноценного API.

Основы GraphQL

В предыдущем разделе мы разработали наш первый API, но давайте ненадолго вернемся назад и взглянем на разные составляющие GraphQL API. Две из них — это схемы и распознаватели. Хорошенько разобравшись в этих компонентах, вы сможете более эффективно применять их в проектировании и разработке API.

Схемы

Схема — это письменное представление данных и взаимодействий. С ее помощью GraphQL обеспечивает соблюдение строгого плана нашего API, потому что API может возвращать данные и выполнять действия, которые определены в рамках этой схемы.

Основополагающим компонентом таких схем являются типы объектов. В предыдущем примере мы создали тип GraphQL-объекта `Query` с полем `hello`, который возвращал скалярный тип `String`. В GraphQL бывает пять встроенных скалярных типов:

`String`

Строка с кодировкой символов UTF-8.

`Boolean`

Значение true или false.