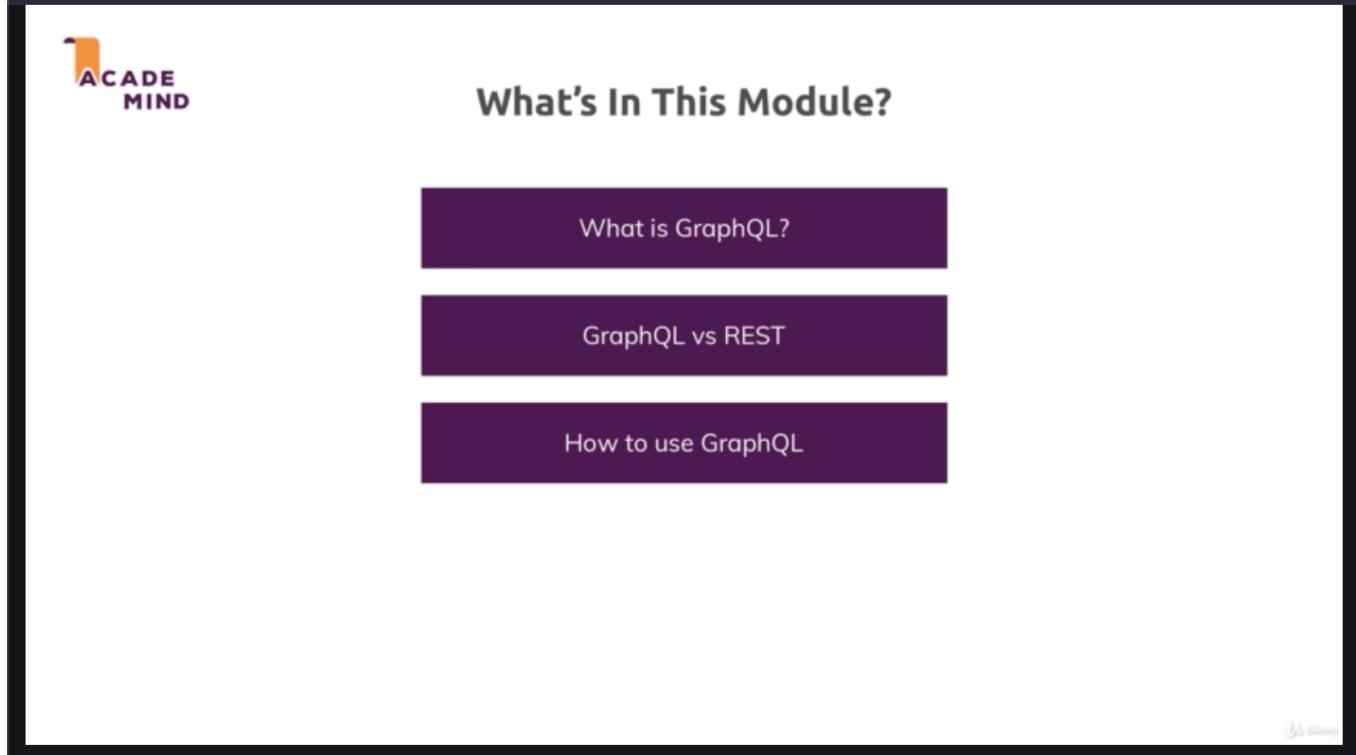


28. Working With GraphQL

* Chapter 415: Module Introduction



The slide features a dark purple header bar with the Academind logo on the left. Below the header, the title "What's In This Module?" is centered in a large, bold, dark gray font. Three dark purple rectangular buttons are stacked vertically, each containing white text: "What is GraphQL?", "GraphQL vs REST", and "How to use GraphQL". A small "View Details" button is located in the bottom right corner of the slide area.

What's In This Module?

What is GraphQL?

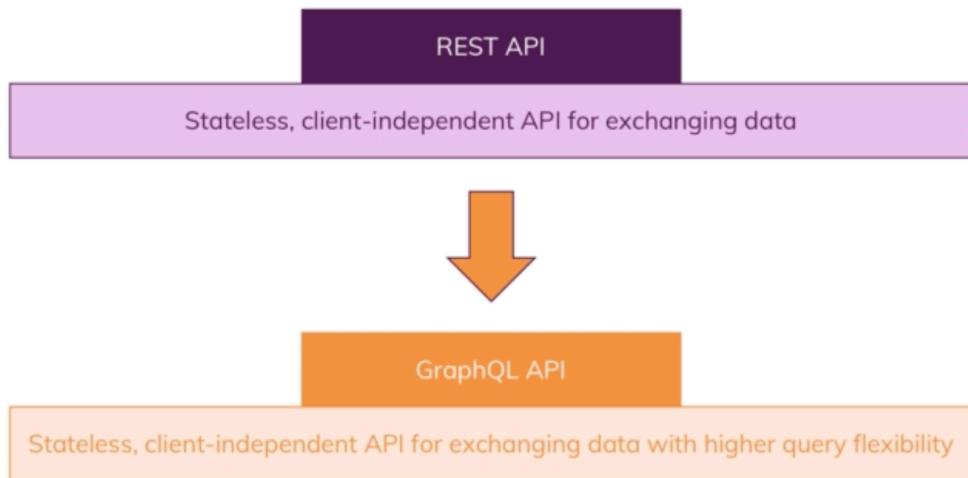
GraphQL vs REST

How to use GraphQL

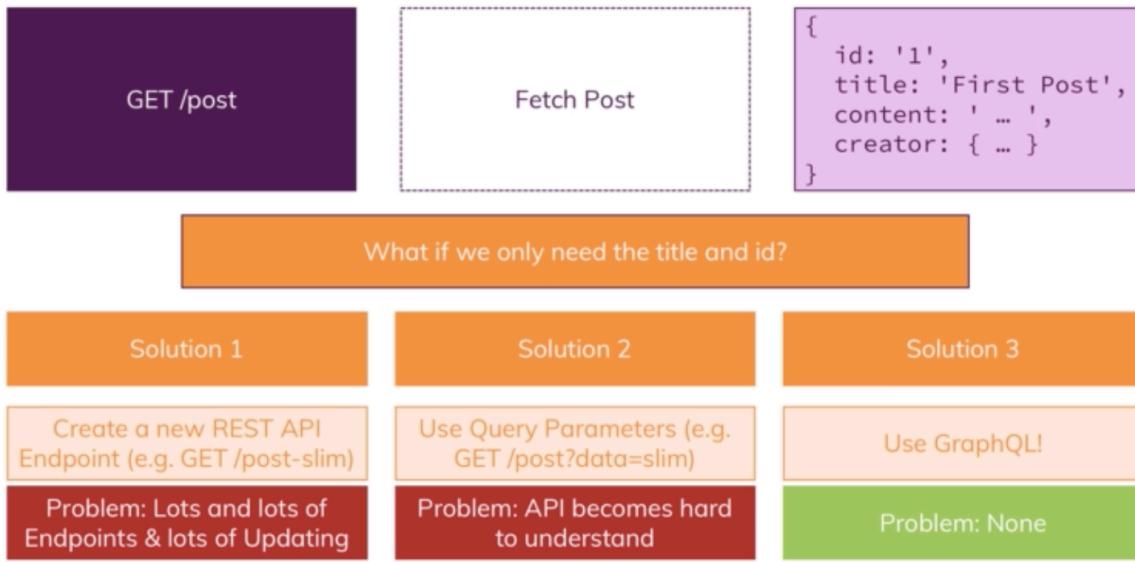
View Details

* Chapter 416: What Is GraphQL?

What is GraphQL?

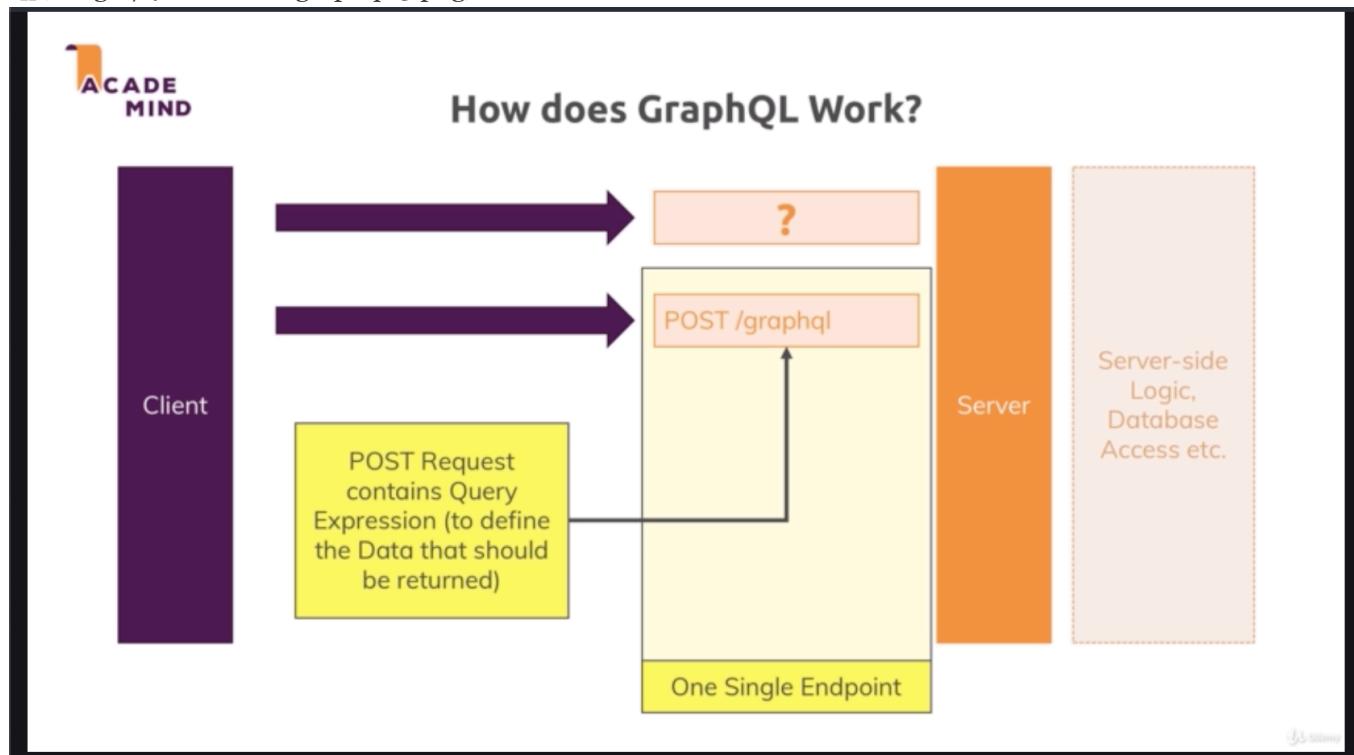


REST API Limitations

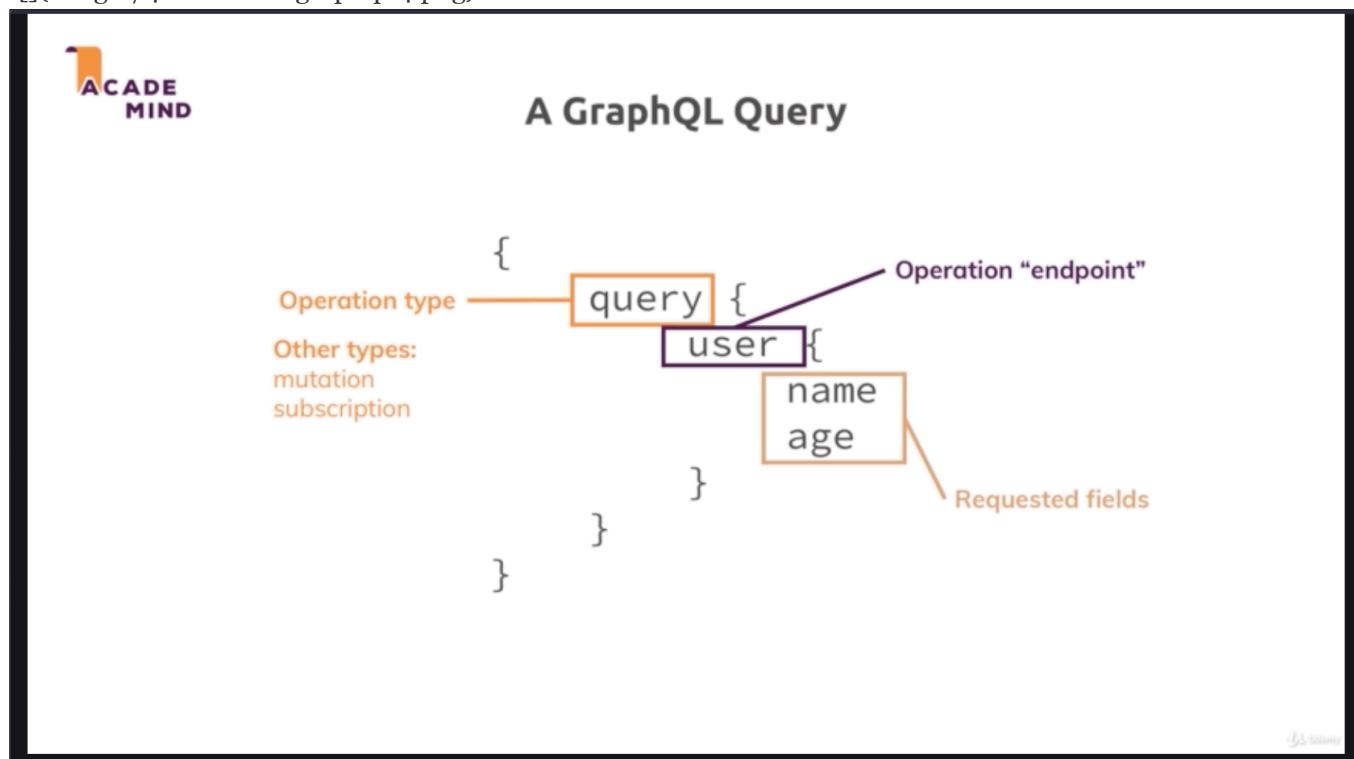


- what if we don't need the content or we don't need to creator data? we can have many scenarios where we use one at the same end point in our front end application. so in our single page application or our mobile application and in one place on one page we might need title and content, on other page, we might need content and creator. how we can solve this?
- solution 1 is that create more endpoints and returned to different types of data. we can create a new REST API endpoint for example sending a GET request to /post-slim. by the way you could use the same endpoint all the time and just parse or filter out the data you need on the frontend. but then you sending a lot of unnecessary data over the wire which is an issue especially when working with mobile devices.
- if you use graphQL, you don't have the problems i described before because as you will learn you have a rich query language that you use in your frontend to send it to the backend which is then parsed on the backend and dynamically retrieves just data you need. so it's almost like a database query language which you use on the backend like sequelize or mongodb query language almost like something like this for the frontend. so which you

put into the request you send to the backend.

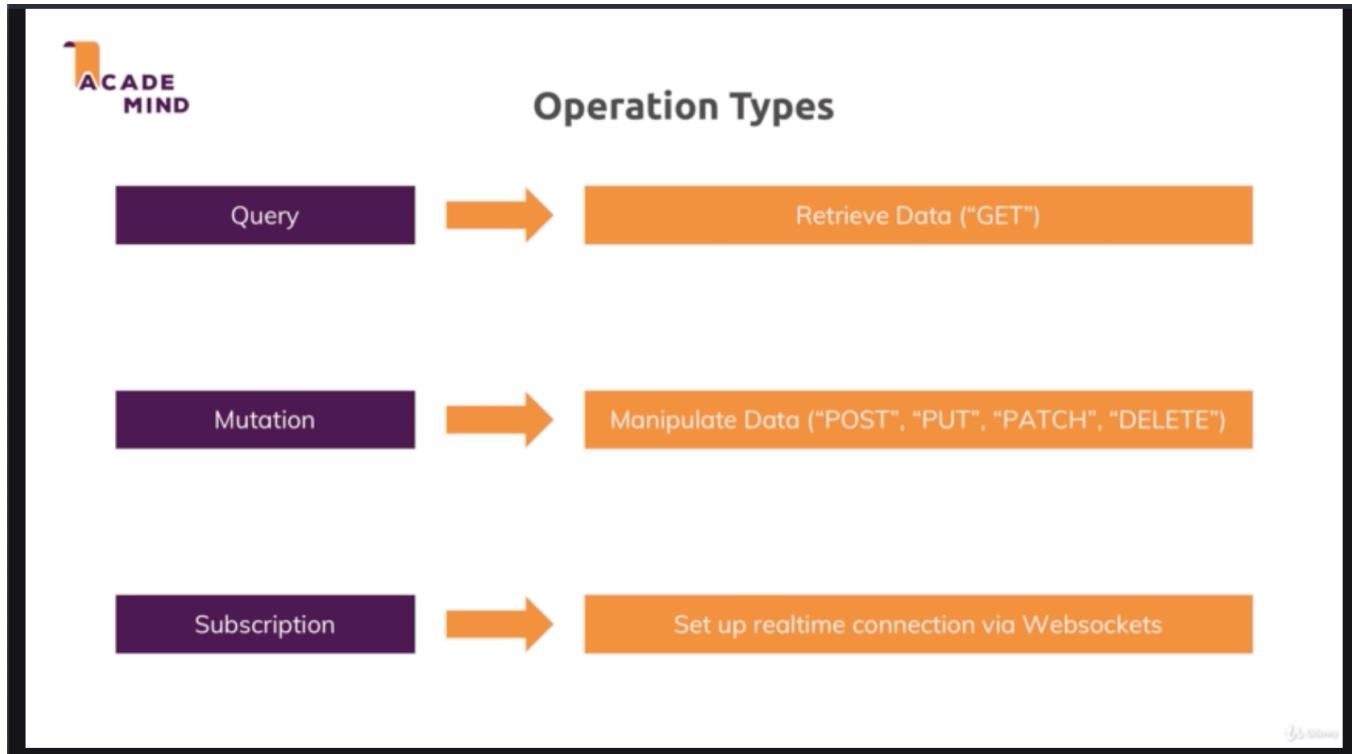


- what do you send from client to server in that GraphQL world? you only send one kind of request and that is POST/graphql. so you only have one single endpoint where you send your HTTP requests to even for getting data. because for POST request, you can add a request body and that request body will contain that query expression. you use query language you put it into a request body and you just can't send request body on GET request for example. so you put your query language expression into that request body and data will be parsed on a server to then do some magic on it and return you just the data you want.



- it's an JSON object-like structure where you have an operation type. query before getting data you also have other types like mutation for editing or deleting or inserting data or subscription for setting up real time data subscriptions using websockets.

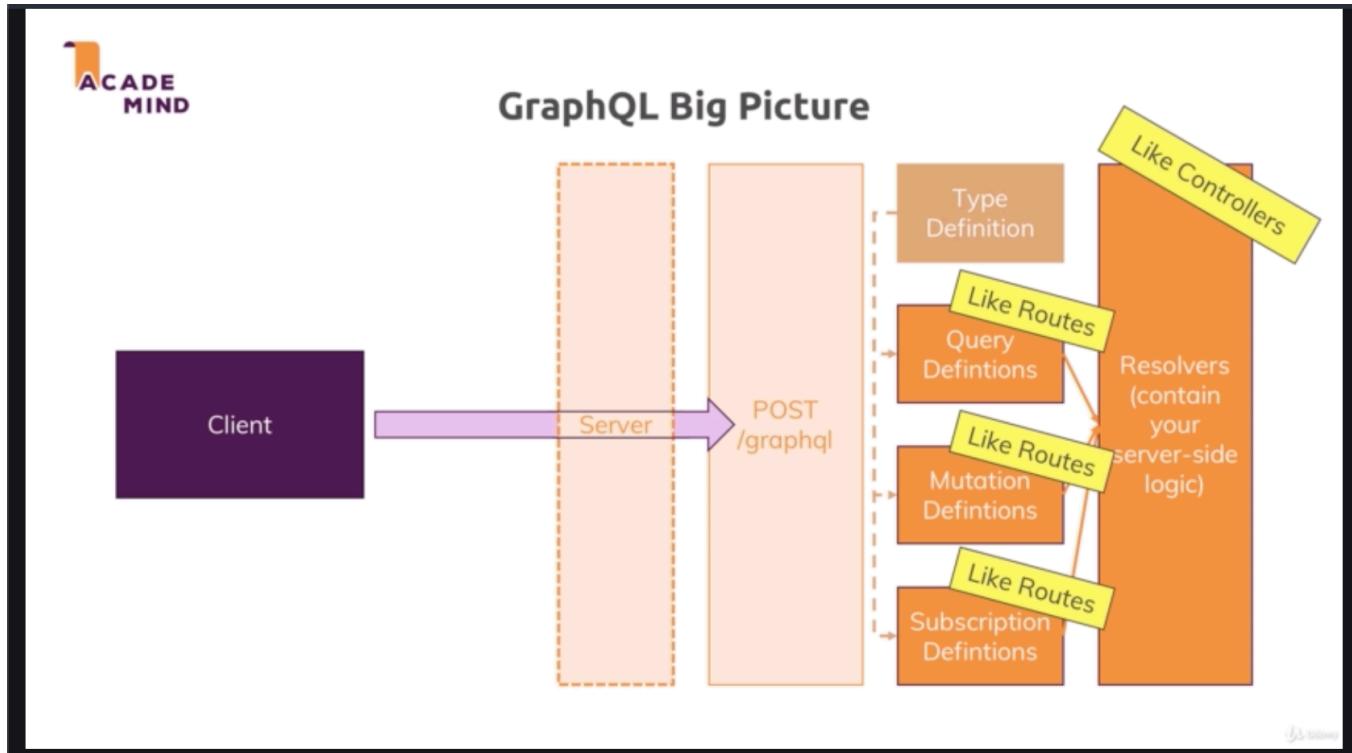
- you also have the endpoints. you could call them or the commands you can execute and you to find them as a developer on your backend. the available endpoints. and then the requested fields you wanna extract and that is the flexible part because you could in one place get the user with just a name and in another place you could get name age and email. so that is exactly what you put into your backend which is then parsed there.



- we retrieved data and we use a POST request for that. but if we wanna compare it to the REST API world, then this would be your equivalent to sending a GET request to some path there.

- we also got mutations which basically are used for everything that changes data.

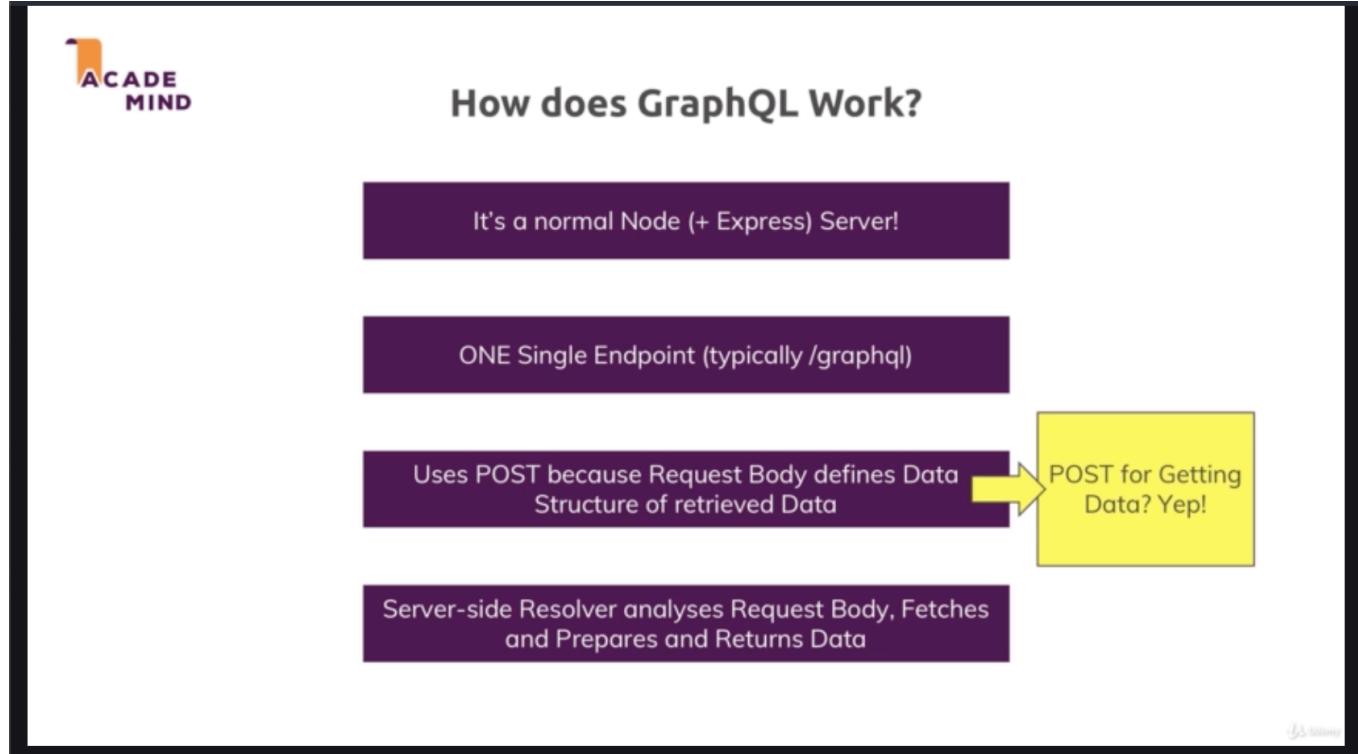
- subscription set up real time connections via websockets.



- to sum it up, in a big picture, we have our client we send a request to that single graphQL endpoint on our server. and then there and this is the part you will do with me in this module. and there you set up your

definitions for queries mutations and possibly also subscriptions definitions. in this definitions, you use type definition because graphQL you use is typed query language which means you define the types of data you work with, the types of data you return in a query and so on.

- this endpoint you define here, so these queries and mutations and subscriptions you define are connected to so-called ‘Resolvers’ which are functions that contain your server side logic and if you compare that to REST API. the definitions would be your routes and resolvers would be your controllers and that is how you can look at graphQL



* Chapter 417: Understanding The Setup & Writing Our First Query

1. update
- delete socketio.js(b)
- app.js(b)
- delete ./routes/auth.js(b)
- delete ./routes/feed.js(b)
- ./graphql/schema.js(b)
- ./graphql/resolvers.js(b)

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying a project structure for a Node.js application. The current file being edited is `app.js`. The code connects to a MongoDB database using Mongoose:

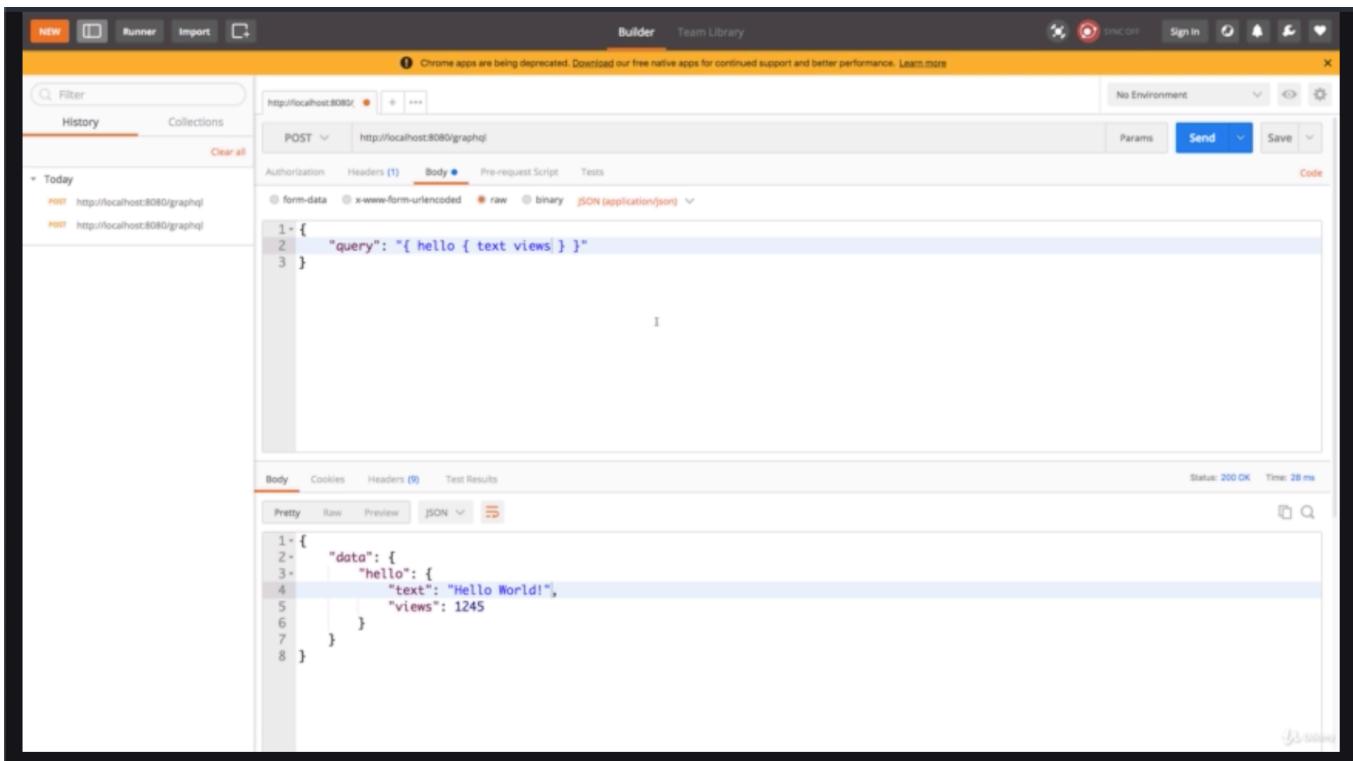
```
54  });
55
56  mongoose
57    .connect(
58      'mongodb+srv://maximilian:9u4biljMQc4jjqbe@cluster0-ntrwp.mongodb.net/messages?retryWrites=true'
59    )
60    .then(result => {
61      app.listen(8080);
62    })
63    .catch(err => console.log(err));
64
```

- we will install new packages. the first one is called 'graphql' which will be required for defining the schema of our GraphQL service. so the definition of the queries, mutations and so on we wanna allow
 - and 2nd is called 'express-graphql' which installs a simple server that will do the parsing of incoming requests.
-

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying a project structure for a Node.js application. The current file being edited is `schema.js`. It defines a GraphQL schema with a type `RootQuery` that has a field `hello` returning a `TestData` object:

```
1  const { buildSchema } = require('graphql');
2
3  module.exports = buildSchema(` 
4    type TestData {
5      text: String!
6      views: Int!
7    }
8
9    type RootQuery {
10      hello: TestData!
11    }
12
13    schema {
14      query: RootQuery
15    }
16 `);
```

The terminal shows the command `npm start` being run, and the output indicates that nodemon is monitoring changes and starting the node process.



- we will test with postman to test POST. you send a javascript object with a query key and this doesn't mean that we send a query and not mutation. it's something which the express-graphql package will look for inside of these curly braces.
- the value here between double quotation marks is your GraphQL query expression. and you add curly braces and in between the name of the query you wanna target. if we have a look at our schema, we see that we have 'hello' query so we can target.
- we need to find which data we wanna get back for that query and therefore offered the name of the query. and we add a number pair of curly braces and in between we list the properties the fields we wanna get back for that query. we separate them with blanks.
- so i could get my text for example not the views if and i would send

```

1 //app.js(b)
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9 /**
10 * will remove feedRoutes, authRoutes
11 * because we will have no more routes will not set up routes anymore
12 * we will use GraphQL endpoint instead.
13 */
14
15 const graphqlHttp = require('express-graphql');
16
17 const graphSchema = require('./graphql/schema');
18 const graphResolver = require('./graphql/resolvers');
19
20 const fileStorage = multer.diskStorage({
21   destination: (req, file, cb) => {
22     cb(null, 'images');
23   },

```

```

24   filename: (req, file, cb) => {
25     cb(null, new Date().toISOString() + '-' + file.originalname);
26   }
27 });
28
29 const fileFilter = (req, file, cb) => {
30   if(
31     file.mimetype === 'image/png' ||
32     file.mimetype === 'image/jpg' ||
33     file.mimetype === 'image/jpeg'
34   ) {
35     cb(null, true);
36   } else {
37     cb(null, false);
38   }
39 }
40
41 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
42 app.use(bodyParser.json()); // application/json
43 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
44 app.use('/images', express.static(path.join(__dirname, 'images')));
45
46 app.use((req, res, next) => {
47   res.setHeader('Access-Control-Allow-Origin', '*');
48   res.setHeader(
49     'Access-Control-Allow-Methods',
50     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
51   );
52   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
53   next();
54 });
55
56 app.use('/graphql', graphqlHttp({
57   schema: graphSchema,
58   rootValue: graphResolver
59 })
60 );
61
62 app.use((error, req, res, next) => {
63   console.log(error);
64   const status = error.statusCode || 500;
65   const message = error.message;
66   const data = error.data;
67   res.status(status).json({ message: message, data: data });
68 });
69
70 mongoose
71   .connect(
72     'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
73   )
74   .then(result => {
75     app.listen(8080);
76   })
77   .catch(err => console.log(err));
78
79 //./graphql/schema.js(b)
80
81

```

```

3  /**this is wherer i find the queries mutations and types
4   * i work with in my graphql service i will create
5   * */
6
7  /**'buildSchema' function allows me to build a schema
8   * which can be parsed by graphql by express-graphql
9   */
10
11 const { buildSchema } = require('graphql');
12
13 /**note that there is no colon after a schema
14  * and in that schema, we define a query field
15  *
16  * this will be an object with all the queries
17  * and queries are the parts where you get data
18  * so all the queries you wanna allow
19  *
20  * this is now a very basic schema
21  * where we can send a 'hello' queries to get back some text
22  *
23  * you don't need to resolve this 'query: RootQuery'
24  * because in your schema,
25  * you set up your RootQuery
26  * which has made up of this sub-queries 'hello: String'
27  * which need resolvers
28  *
29  * if you add '!',
30  * the result is that if you don't return a string,
31  * you will get an error
32  */
33 module.exports = buildSchema(`

34   type TestData {
35     text: String!
36     views: Int!
37   }

38

39   type RootQuery {
40     hello: TestData!
41   }

42

43   schema {
44     query: RootQuery
45   }
46 `);

```

```

1 //./graphql/resolvers.js(b)
2
3 /**this is where i will define the logic
4  * that then executed for incoming queries
5  * */
6
7 /**you need a method for every query or mutation
8  * you define your schema and the name has to match
9  * now we have resolver for 'hello' method
10 *
11 * we need to expose it to the public
12 * and that can be done with the 'express-graphql' package

```

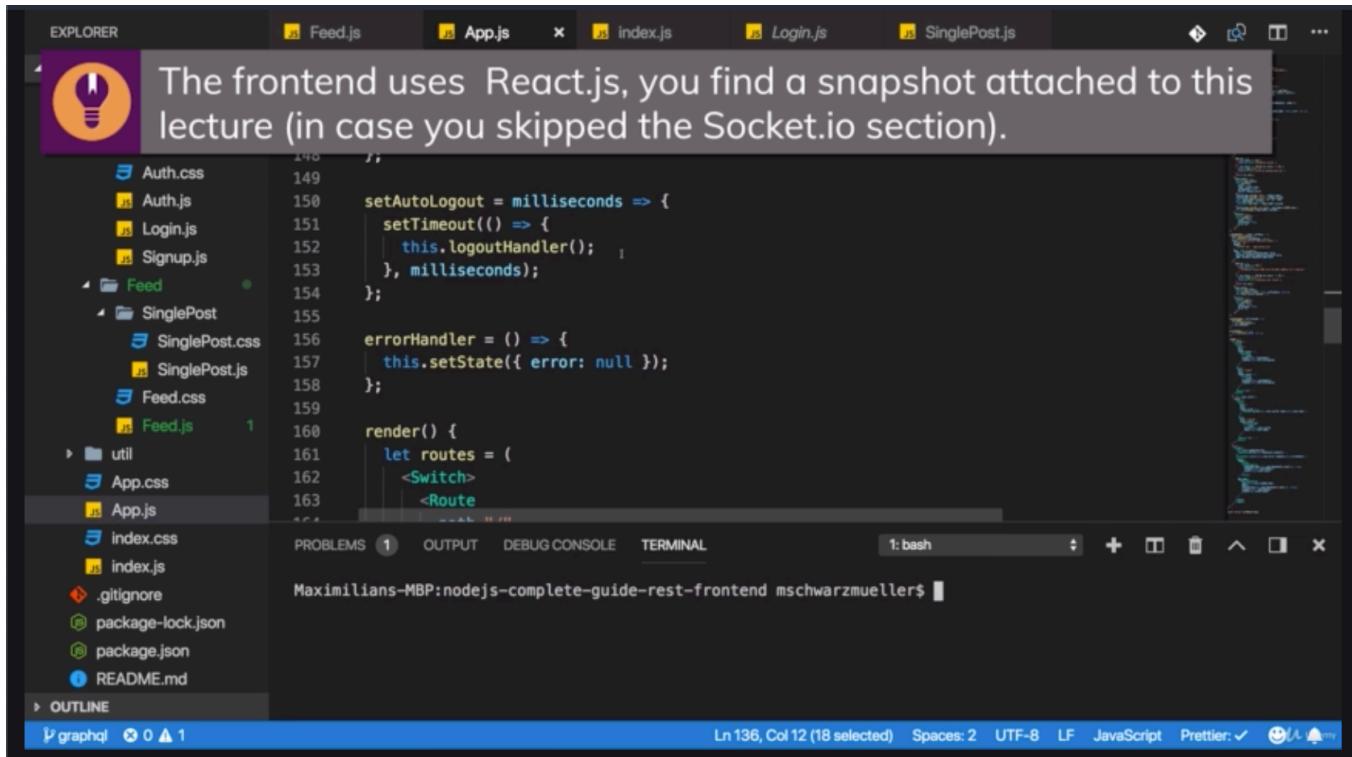
```

13 * so go to app.js(b)
14 */
15
16 module.exports = {
17   hello() {
18     return {
19       text: 'Hello World!',
20       views: 1245
21     }
22   }
23 }

```

* Chapter 418: Defining A Mutation Schema

1. update
 - ./src/pages/Feed/Feed.js(f)
 - ./graphql/resolvers.js(b)
 - ./graphql/schema.js(b)



```

1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {

```

```

15 state = {
16   isEditing: false,
17   posts: [],
18   totalPosts: 0,
19   editPost: null,
20   status: '',
21   postPage: 1,
22   postsLoading: true,
23   editLoading: false
24 };
25
26 componentDidMount() {
27   fetch('http://localhost:8080/auth/status', {
28     headers: {
29       Authorization: 'Bearer ' + this.props.token
30     }
31   })
32     .then(res => {
33       if (res.status !== 200) {
34         throw new Error('Failed to fetch user status.');
35       }
36       return res.json();
37     })
38     .then(resData => {
39       this.setState({ status: resData.status });
40     })
41     .catch(this.catchError);
42
43   this.loadPosts();
44
45 }
46
47 loadPosts = direction => {
48   if (direction) {
49     this.setState({ postsLoading: true, posts: [] });
50   }
51   let page = this.state.postPage;
52   if (direction === 'next') {
53     page++;
54     this.setState({ postPage: page });
55   }
56   if (direction === 'previous') {
57     page--;
58     this.setState({ postPage: page });
59   }
60   fetch('http://localhost:8080/feed/posts?page=' + page, {
61     headers: {
62       Authorization: 'Bearer ' + this.props.token
63     }
64   })
65     .then(res => {
66       if (res.status !== 200) {
67         throw new Error('Failed to fetch posts.');
68       }
69       return res.json();
70     })

```

```
71     .then(resData => {
72       this.setState({
73         posts: resData.posts.map(post => {
74           return {
75             ...post,
76             imagePath: post.imageUrl
77           };
78         }),
79         totalPosts: resData.totalItems,
80         postsLoading: false
81       });
82     })
83     .catch(this.catchError);
84   };
85
86 statusUpdateHandler = event => {
87   event.preventDefault();
88   fetch('http://localhost:8080/auth/status', {
89     method: 'PATCH',
90     headers: {
91       Authorization: 'Bearer ' + this.props.token,
92       'Content-Type': 'application/json'
93     },
94     body: JSON.stringify({
95       status: this.state.status
96     })
97   })
98   .then(res => {
99     if (res.status !== 200 && res.status !== 201) {
100       throw new Error("Can't update status!");
101     }
102     return res.json();
103   })
104   .then(resData => {
105     console.log(resData);
106   })
107   .catch(this.catchError);
108 };
109
110 newPostHandler = () => {
111   this.setState({ isEditing: true });
112 };
113
114 startEditPostHandler = postId => {
115   this.setState(prevState => {
116     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
117
118     return {
119       isEditing: true,
120       editPost: loadedPost
121     };
122   });
123 };
124
125 cancelEditHandler = () => {
126   this.setState({ isEditing: false, editPost: null });
```

```
127 };
128
129 finishEditHandler = postData => {
130   this.setState({
131     editLoading: true
132   });
133   const formData = new FormData();
134   formData.append('title', postData.title);
135   formData.append('content', postData.content);
136   formData.append('image', postData.image);
137   let url = 'http://localhost:8080/feed/post';
138   let method = 'POST';
139   if (this.state.editPost) {
140     url = 'http://localhost:8080/feed/post/' + this.state.editPost._id;
141     method = 'PUT';
142   }
143
144   fetch(url, {
145     method: method,
146     body: formData,
147     headers: {
148       Authorization: 'Bearer ' + this.props.token
149     }
150   })
151     .then(res => {
152       if (res.status !== 200 && res.status !== 201) {
153         throw new Error('Creating or editing a post failed!');
154       }
155       return res.json();
156     })
157     .then(resData => {
158       console.log(resData);
159       const post = {
160         _id: resData.post._id,
161         title: resData.post.title,
162         content: resData.post.content,
163         creator: resData.post.creator,
164         createdAt: resData.post.createdAt
165       };
166       this.setState(prevState => {
167         return {
168           isEditing: false,
169           editPost: null,
170           editLoading: false
171         };
172       });
173     })
174     .catch(err => {
175       console.log(err);
176       this.setState({
177         isEditing: false,
178         editPost: null,
179         editLoading: false,
180         error: err
181       });
182     });

```

```

183 };
184
185 statusInputChangeHandler = (input, value) => {
186   this.setState({ status: value });
187 };
188
189 deletePostHandler = postId => {
190   this.setState({ postsLoading: true });
191   fetch('http://localhost:8080/feed/post/' + postId, {
192     method: 'DELETE',
193     headers: {
194       Authorization: 'Bearer ' + this.props.token
195     }
196   })
197   .then(res => {
198     if (res.status !== 200 && res.status !== 201) {
199       throw new Error('Deleting a post failed!');
200     }
201     return res.json();
202   })
203   .then(resData => {
204     console.log(resData);
205     this.loadPosts();
206   /*
207     this.setState(prevState => {
208       const updatedPosts = prevState.posts.filter(p => p._id !== postId);
209       return { posts: updatedPosts, postsLoading: false };
210     });
211 */
212   })
213   .catch(err => {
214     console.log(err);
215     this.setState({ postsLoading: false });
216   });
217 };
218
219 errorHandler = () => {
220   this.setState({ error: null });
221 };
222
223 catchError = error => {
224   this.setState({ error: error });
225 };
226
227 render() {
228   return (
229     <Fragment>
230       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
231       <FeedEdit
232         editing={this.state.isEditing}
233         selectedPost={this.state.editPost}
234         loading={this.state.editLoading}
235         onCancelEdit={this.cancelEditHandler}
236         onFinishEdit={this.finishEditHandler}
237       />
238       <section className="feed_status">

```

```
239 <form onSubmit={this.statusUpdateHandler}>
240   <Input
241     type="text"
242     placeholder="Your status"
243     control="input"
244     onChange={this.statusInputChangeHandler}
245     value={this.state.status}
246   />
247   <Button mode="flat" type="submit">
248     Update
249   </Button>
250 </form>
251 </section>
252 <section className="feed__control">
253   <Button mode="raised" design="accent" onClick={this.newPostHandler}>
254     New Post
255   </Button>
256 </section>
257 <section className="feed">
258   {this.state.postsLoading && (
259     <div style={{ textAlign: 'center', marginTop: '2rem' }}>
260       <Loader />
261     </div>
262   )}
263   {this.state.posts.length <= 0 && !this.state.postsLoading ? (
264     <p style={{ textAlign: 'center' }}>No posts found.</p>
265   ) : null}
266   {!this.state.postsLoading && (
267     <Paginator
268       onPrevious={this.loadPosts.bind(this, 'previous')}
269       onNext={this.loadPosts.bind(this, 'next')}
270       lastPage={Math.ceil(this.state.totalPosts / 2)}
271       currentPage={this.state.postPage}
272     >
273       {this.state.posts.map(post => (
274         <Post
275           key={post._id}
276           id={post._id}
277           author={post.creator.name}
278           date={new Date(post.createdAt).toLocaleDateString('en-US')}
279           title={post.title}
280           image={post.imageUrl}
281           content={post.content}
282           onStartEdit={this.startEditPostHandler.bind(this, post._id)}
283           onDelete={this.deletePostHandler.bind(this, post._id)}
284         />
285       ))}
286     </Paginator>
287   )}
288 </section>
289 </Fragment>
290 );
291 }
292 }
293
294 export default Feed;
```

```
1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 /**that mutation here will require some input some arguments
6 * and that is a difference in text which you haven't seen before
7 * you can add parentheses after you require the name
8 * and this allows you to specify arguments
9 * that have to be given to that resolver that will run in the end
10 *
11 * we bundle it in one object
12 * which we expect and for data object,
13 * we can create a new type and we don't do with 'type' keyword
14 * but there's special keyword for data that is used as the input
15 * so for data as is used as an argument and that is the 'input' keyword
16 *
17 * 'ID' is a special type provided by graphql
18 * which signals that this is unique
19 * and well treated as an ID
20 *
21 * graphQL doesn't know the dates.
22 * so i will use string and the same for updatedAt
23 * remember we will get these 2 fields
24 * because in mongoose model for the post,
25 * we enable timestamps. '{ timestamps: true }'
26 */
27 module.exports = buildSchema(`

28   type Post {
29     _id: ID!
30     title: String!
31     Content: String!
32     imageUrl: String!
33     creator: User!
34     createdAt: String!
35     updatedAt: String!
36   }

37

38   type User {
39     _id: ID!
40     name: String!
41     email: String!
42     password: String
43     status: String!
44     posts: [Post!]!
45   }

46

47   input UserInput {
48     email: String!
49     name: String!
50     password: String!
51   }

52

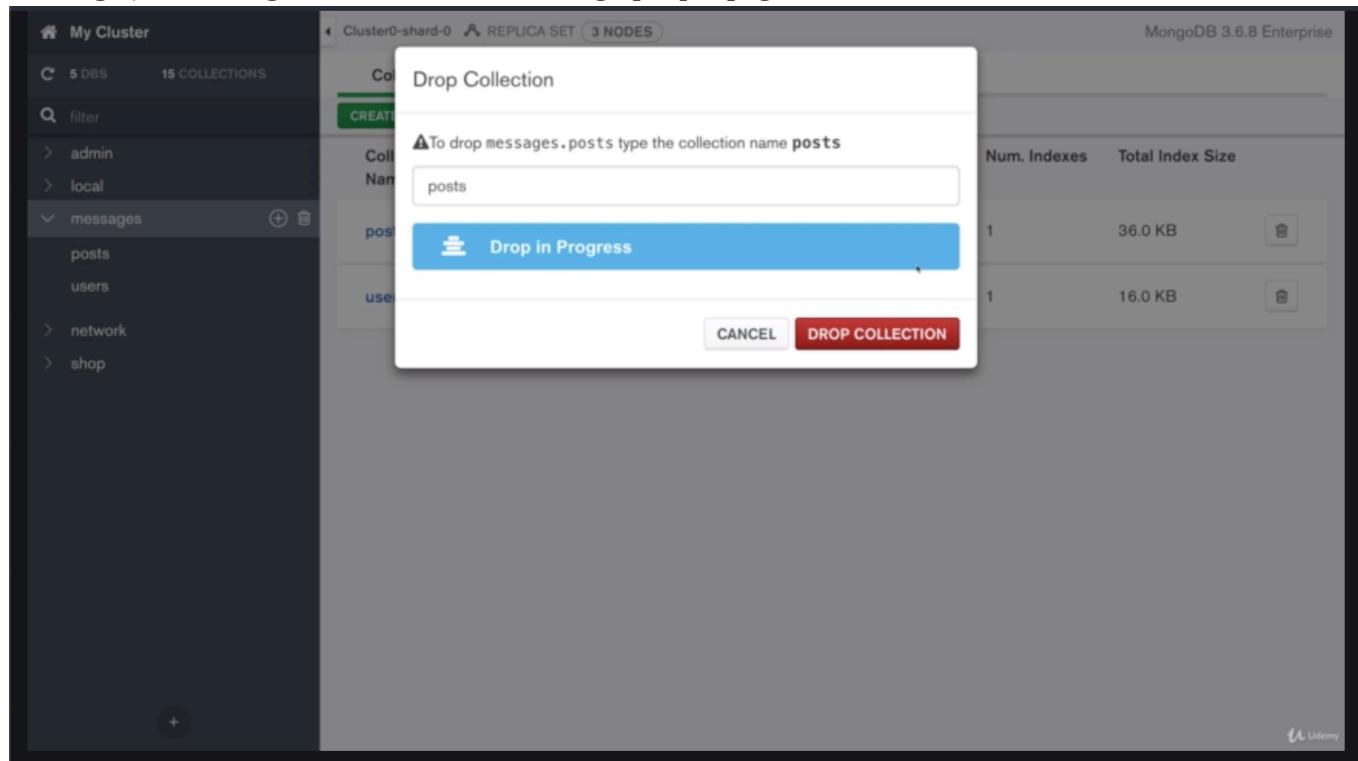
53   type RootMutation {
54     createUser(userInput: UserInputData): User!
55 }
```

```
56
57     schema {
58       mutation: RootMutation
59     }
60 `);
```

```
1 //./graphql/resolvers.js(b)
2
3 module.exports = {
4   ...
5 }
```

* Chapter 419: Adding A Mutation Resolver & GraphiQL

1. update
 - ./graphql/resolvers.js(b)
 - app.js(b)
 - ./graphql/schema.js(b)
 - graphiQL



The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with 'My Cluster' at the top, followed by '4 DBS' and '13 COLLECTIONS'. Below this is a 'filter' search bar and a list of databases: 'admin', 'local', 'network', and 'shop'. A green 'CREATE COLLECTION' button is at the bottom of this sidebar. The main area is titled 'Collections' and has a header with columns: 'Collection Name', 'Documents', 'Avg. Document Size', 'Total Document Size', 'Num. Indexes', and 'Total Index Size'. There is no data listed under this table.

- let's clean up 'message'

The screenshot shows a browser window with several tabs open. The address bar shows 'localhost:8080/graphql'. The tabs are:

- Code | GraphQL
- New Tab
- localhost:8080/graphql
- GraphiQL - localhost:8080/graphql
- GraphiQL - localhost:8080/graphql?query=%23 Welcome to GraphiQL%0A%23%0A%23 GraphiQL is an in-browser tool for
- localhost:8080/graphql - Google Search
- GraphiQL - localhost:8080/graphql?query=%23 Welcome to GraphiQL%0A%23%0A%23 GraphiQL is an in-browser tool for
- GraphiQL - localhost:8080/graphql?query=%23 Welcome to GraphiQL%0A%23%0A%23 GraphiQL is an in-browser tool for

```

1 # Welcome to GraphQL
2 #
3 # GraphQL is an in-browser tool for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will see intelligent
7 # typeahead aware of the current GraphQL type schema and live syntax and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character. Lines that starts
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Run Query: Ctrl-Enter (or press the play button above)
26 #
27 # Auto Complete: Ctrl-Space (or just start typing)
28 #
29 {
30   postData {
31     posts {
32       _id
33     }
34 }

```

QUERY VARIABLES

- localhost:8080/graphql will send a GET request and you will get this screen which allows you to play around with your GraphQL API


```

1 # Welcome to GraphQL
2 #
3 # GraphQL is an in-browser tool for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will see intelligent
7 # typeahead aware of the current GraphQL type schema and live syntax and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character. Lines that starts
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Run Query: Ctrl-Enter (or press the play button above)
26 #
27 # Auto Complete: Ctrl-Space (or just start typing)
28 #
29 {
30   postData {
31     posts {
32       _id
33     }
34 }

```

QUERY VARIABLES

- if you reload your graghQL inerface you have that documentation on the right where you see the operations you can do but you need to have a query defined for that even if it leads into white.

The screenshot shows the GraphiQL interface. On the left, a code editor displays a GraphQL query:

```

1 # Welcome to GraphiQL
2 #
3 # GraphiQL is an in-browser tool for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will see intelligent
7 # typeheads aware of the current GraphQL type schema and live syntax and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character. Lines that starts
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Run Query: Ctrl-Enter (or press the play button above)
26 #
27 # Auto Complete: Ctrl-Space (or just start typing)
28 #
29 {
30   postData {
31     posts {
32       _id
33     }
34 }

```

On the right, the results pane shows a single mutation result:

```

{
  "data": {
    "createUser": {
      "_id": "5bc5a62e6ad8b76087037aa1",
      "email": "test@test.com"
    }
  }
}

```

A cursor arrow points to the word "String" in the "email" field description.

The screenshot shows the GraphiQL interface. On the left, a code editor displays a GraphQL query:

```

1 # Welcome to GraphiQL
2 #
3 # GraphiQL is an in-browser tool for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will see intelligent
7 # typeheads aware of the current GraphQL type schema and live syntax and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character. Lines that starts
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Run Query: Ctrl-Enter (or press the play button above)
26 #
27 # Auto Complete: Ctrl-Space (or just start typing)
28 #
29 mutation {
30   createUser(userInput: {email: "test@test.com", name:"Max", password:"tes
31   _id
32   email
33 }
34 }

```

On the right, the results pane shows a single mutation result:

```

{
  "data": {
    "createUser": {
      "_id": "5bc5a62e6ad8b76087037aa1",
      "email": "test@test.com"
    }
  }
}

```

- you can click onto it to see which mutations you have which data you need to send and so on.
- GraphiQL is a great toll for playing around better than postman because you got this nice interactive support with the auto completion


```

1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 module.exports = buildSchema(`

6   type Post {
7     _id: ID!
8     title: String!
9     content: String!
10    imageUrl: String!
11    creator: User!
12    createdAt: String!
13    updatedAt: String!
14  }

15

16   type User {
17     _id: ID!
18     name: String!
19     email: String!
20     password: String
21     status: String!
22     posts: [Post!]!
23   }

24

25   input UserInputData {
26     email: String!
27     name: String!
28     password: String!
29   }

30

31   type RootQuery {
32     hello: String
33   }

34

35   type RootMutation {
```

```

36     createUser(userInput: UserInputData): User!
37   }
38
39   schema {
40     query: RootQuery
41     mutation: RootMutation
42   }
43 `);
44
45 //./graphql/resolvers.js(b)
46
47 const bcrypt = require('bcryptjs');
48
49 const User = require('../models/user');
50
51 /**
52  * instead our args will have a 'userInput' field
53  * because our args will be an object
54  * containing all the arguments passed to that function 'createUser'
55  * and in 'userInput' field,
56  * that will have email, name and password so i can retrieve them.
57 */
58
59 module.exports = {
60   createUser: async function({ userInput }, req) {
61     //const email = args.userInput.email;
62     /**
63      * if you are not using async, await
64      * you need to return your fineOne query
65      * which you are executing here
66      * where you would then add '.then()'
67      * because if you don't return your promises in the resolver,
68      * graphQL will not wait for it to resolve
69      *
70      *       return User.findOne().then()
71      *
72      * when using async await
73      * it's automatically returned for you
74      * we don't see the return statement but it is there.
75      */
76     const existingUser = await User.findOne({email: userInput.email});
77     if(existingUser) {
78       const error = new Error('User exists already!');
79       throw error;
80     }
81     const hashedPw = await bcrypt.hash(userInput.password, 12);
82     const user = new User({
83       email: userInput.email,
84       name: userInput.name,
85       password: hashedPw
86     });
87     const createdUser = await user.save();
88     /**
89      * will return my createdUser._doc field
90      * which contains just userData without all the meta data mongoose would add
91      * and i will override the _id field by adding it as a separate property
92      * and therefore it will override the one i'm pulling out of _doc
93      * because i need to convert it from an object id field to a string field
94      * otherwise it will fail
95      */
96   }
97 }

```

```
49
50     ...
51     return { ...createdUser._doc, _id: createdUser._id.toString() };
52 }
53
54 //app.js(b)
55
56 const path = require('path');
57
58 const express = require('express');
59 const bodyParser = require('body-parser');
60 const mongoose = require('mongoose');
61 const multer = require('multer');
62
63 /**
64  * **i will remove feedRoutes, authRoutes
65  * because we will have no more routes will not set up routes anymore
66  * we will use graphQL endpoint instead.
67 */
68
69 const graphqlHttp = require('express-graphql');
70
71
72 const graphSchema = require('./graphql/schema');
73 const graphResolver = require('./graphql/resolvers');
74
75
76 const app = express();
77
78
79 const fileStorage = multer.diskStorage({
80   destination: (req, file, cb) => {
81     cb(null, 'images');
82   },
83   filename: (req, file, cb) => {
84     cb(null, new Date().toISOString() + '-' + file.originalname);
85   }
86 });
87
88
89 const fileFilter = (req, file, cb) => {
90   if(
91     file.mimetype === 'image/png' ||
92     file.mimetype === 'image/jpg' ||
93     file.mimetype === 'image/jpeg'
94   ) {
95     cb(null, true);
96   } else {
97     cb(null, false);
98   }
99 }
100
101 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
102 app.use(bodyParser.json()); // application/json
103 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'));
104 app.use('/images', express.static(path.join(__dirname, 'images')));
105
106
107 app.use((req, res, next) => {
108   res.setHeader('Access-Control-Allow-Origin', '*');
109   res.setHeader(
110     'Access-Control-Allow-Methods',
111     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
112   );
113   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
114 }
```

```

53   next();
54 });
55
56 app.use('/graphql', graphqlHttp({
57   schema: graphSchema,
58   rootValue: graphResolver,
59   /**to test those mutation.
60    * 'graphiql' gives you a special tool
61    * and this is the reason why i'm not listening for a post request only here
62    * which is 'use' not 'post'
63    * because your running server,
64    * try visiting localhost:8080/graphql will send GET request
65    *
66   */
67   graphiql: true
68 })
69 );
70
71 app.use((error, req, res, next) => {
72   console.log(error);
73   const status = error.statusCode || 500;
74   const message = error.message;
75   const data = error.data;
76   res.status(status).json({ message: message, data: data });
77 });
78
79 mongoose
80   .connect(
81     'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
82     )
83   .then(result => {
84     app.listen(8080);
85   })
86   .catch(err => console.log(err));

```

```

1 //localhost:8080/graphql
2 //graphiQL
3
4 # Welcome to GraphiQL
5 #
6 # GraphiQL is an in-browser tool for writing, validating, and
7 # testing GraphQL queries.
8 #
9 # Type queries into this side of the screen, and you will see intelligent
10 # typeahead's aware of the current GraphQL type schema and live syntax and
11 # validation errors highlighted within the text.
12 #
13 # GraphQL queries typically start with a "{" character. Lines that starts
14 # with a # are ignored.
15 #
16 # An example GraphQL query might look like:
17 #
18 #   {
19 #     field(arg: "value") {
20 #       subField
21 #     }
22 #   }

```

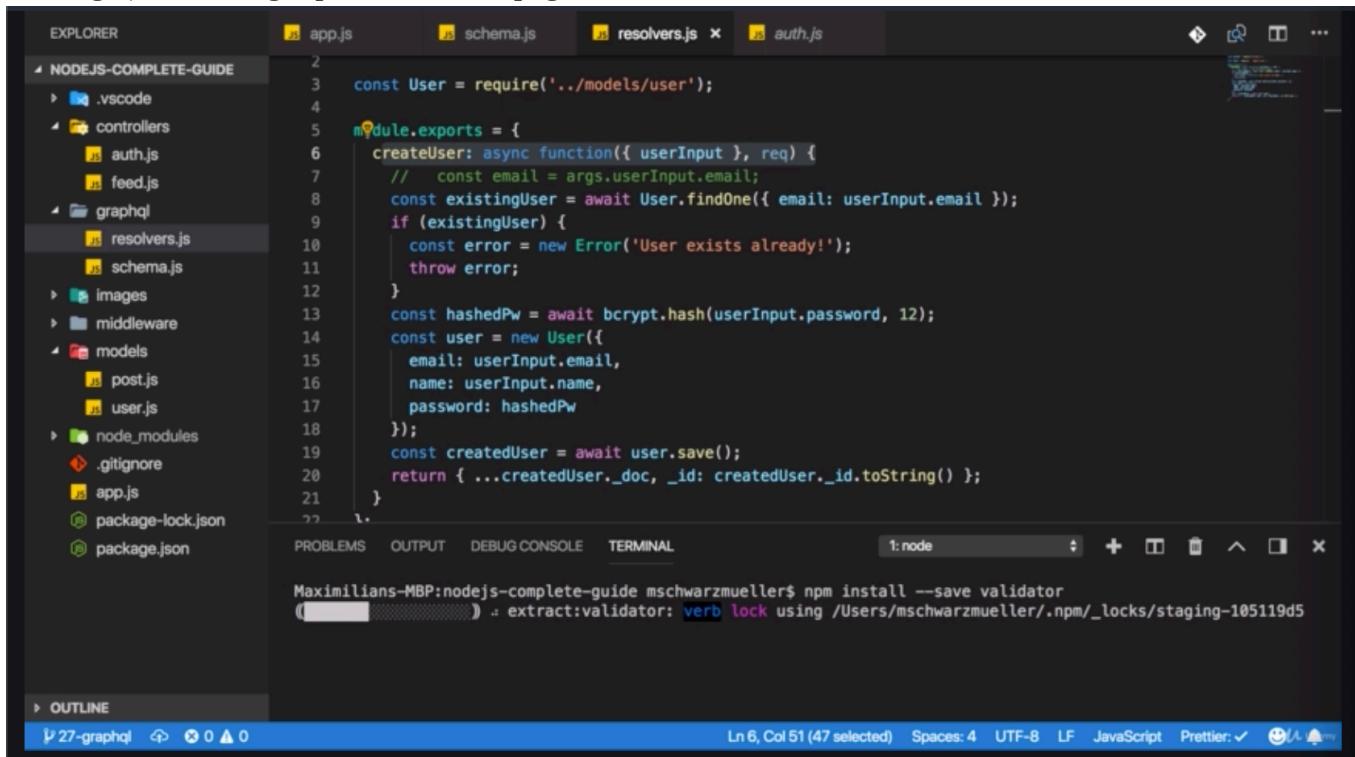
```

23 #
24 # Keyboard shortcuts:
25 #
26 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
27 #
28 # Run Query: Ctrl-Enter (or press the play button above)
29 #
30 # Auto Complete: Ctrl-Space (or just start typing)
31 #
32 mutation {
33   createUser(userInput: {email:"test@test.com", name: "Max", password: "tester"}){
34     _id
35     email
36   }
37 }

```

* Chapter 420: Adding Input Validation

1. update
- ./graphql/schema.js(b)
- app.js(b)
- ./graphql/resolvers.js(b)



```

const User = require('../models/user');

module.exports = {
  createUser: async function({ userInput }, req) {
    // const email = args.userInput.email;
    const existingUser = await User.findOne({ email: userInput.email });
    if (existingUser) {
      const error = new Error('User exists already!');
      throw error;
    }
    const hashedPw = await bcrypt.hash(userInput.password, 12);
    const user = new User({
      email: userInput.email,
      name: userInput.name,
      password: hashedPw
    });
    const createdUser = await user.save();
    return { ...createdUser._doc, _id: createdUser._id.toString() };
  }
};

```

- 'npm install --save validator' is the package which express validator use behind the scenes. we can use that directly in our code.

The screenshot shows the GraphiQL interface running in a browser. On the left, the code editor contains a sample GraphQL schema and a mutation. On the right, the Documentation Explorer panel displays the schema definition, highlighting an error in the mutation field. The error message indicates an invalid input at line 30, column 3, with the path to the 'createUser' field.

```

1 # Welcome to GraphiQL
2 #
3 # GraphQL is an in-browser tool for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will see intelligent
7 # typeahead aware of the current GraphQL type schema and live syntax and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character. Lines that starts
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 # Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 # Run Query: Ctrl-Enter (or press the play button above)
26 #
27 # Auto Complete: Ctrl-Space (or just start typing)
28 #
29 mutation {
30   createUser(userInput: {email: "test", name:"Max", password:"tester"}) {
31     _id
32     email
33   }
34 }
```

Documentation Explorer

Search Schema...

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: RootQuery
mutation: RootMutation

- if now i try to add an email address which is not an e-mail address and i hit play, i see my invalid input message and that is how we can add validation

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5
6 const User = require('../models/user');
7
8 module.exports = {
9   createUser: async function({ userInput }, req) {
10     //const email = args.userInput.email;
11     const errors = [];
12     if (!validator.isEmail(userInput.email)){
13       errors.push({ message: 'E-Mail is invalid.' });
14     }
15     if (validator.isEmpty(userInput.password) || !validator.minLength(userInput.password,
{ min: 5 })) {
16       errors.push({message: 'Password Too Short!'});
17     }
18     if(errors.length > 0){
19       const error = new Error('Invalid Input.');
20       throw error;
21     }
22     const existingUser = await User.findOne({email: userInput.email});
23     if(existingUser) {
24       const error = new Error('User exists already!');
25       throw error;
26     }
27     const hashedPw = await bcrypt.hash(userInput.password, 12);
28     const user = new User({
29       email: userInput.email,
30       name: userInput.name,
31       password: hashedPw
32     })
33     user.save();
34     return user;
35   }
36 }
```

```
32     });
33     const createdUser = await user.save();
34     return { ...createdUser._doc, _id: createdUser._id.toString() };
35   }
36 }

1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 /** we added our first mutation
6 * and mutation also means that we store data in the database
7 *
8 */
9 module.exports = buildSchema(`

10  type Post {
11    _id: ID!
12    title: String!
13    content: String!
14    imageUrl: String!
15    creator: User!
16    createdAt: String!
17    updatedAt: String!
18  }

19

20  type User {
21    _id: ID!
22    name: String!
23    email: String!
24    password: String
25    status: String!
26    posts: [Post!]!
27  }

28

29  input UserInputData {
30    email: String!
31    name: String!
32    password: String!
33  }

34

35  type RootQuery {
36    hello: String
37  }

38

39  type RootMutation {
40    createUser(userInput: UserInputData): User!
41  }

42

43  schema {
44    query: RootQuery
45    mutation: RootMutation
46  }
47 `);

1 //app.js(b)
2
3 const path = require('path');
```

```

4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9
10 const graphqlHttp = require('express-graphql');
11
12 const graphSchema = require('./graphql/schema');
13 const graphResolver = require('./graphql/resolvers');
14
15 const app = express();
16
17 const fileStorage = multer.diskStorage({
18   destination: (req, file, cb) => {
19     cb(null, 'images');
20   },
21   filename: (req, file, cb) => {
22     cb(null, new Date().toISOString() + '-' + file.originalname);
23   }
24 });
25
26 const fileFilter = (req, file, cb) => {
27   if(
28     file.mimetype === 'image/png' ||
29     file.mimetype === 'image/jpg' ||
30     file.mimetype === 'image/jpeg'
31   ) {
32     cb(null, true);
33   } else {
34     cb(null, false);
35   }
36 }
37
38 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
39 app.use(bodyParser.json()); // application/json
40 app.use(multer({storage: fileStorage, fileFilter}).single('image'))
41 app.use('/images', express.static(path.join(__dirname, 'images')));
42
43 app.use((req, res, next) => {
44   res.setHeader('Access-Control-Allow-Origin', '*');
45   res.setHeader(
46     'Access-Control-Allow-Methods',
47     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
48   );
49   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
50   next();
51 });
52
53 /**when something like this happens,
54 * we wanna ensure that the data we store is valid
55 * and previously in REST API
56 * and also in the normal node express application,
57 * where we rendered views
58 * we use express-validator for that
59 * and we added that as a middleware on our routes.

```

```

60  *
61  * with GraphQL is not an option
62  * because we only have one route
63  * this is the only end point
64  * and we don't wanna validate all requests in the same way
65  * so to change that and to adjust it to our needs
66  * we wanna move validation into our resolvers
67  */
68 app.use('/graphql', graphqlHttp({
69   schema: graphSchema,
70   rootValue: graphResolver,
71   graphiql: true
72 })
73 );
74
75 app.use((error, req, res, next) => {
76   console.log(error);
77   const status = error.statusCode || 500;
78   const message = error.message;
79   const data = error.data;
80   res.status(status).json({ message: message, data: data });
81 });
82
83 mongoose
84   .connect(
85     'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
86     )
87   .then(result => {
88     app.listen(8080);
89   })
90   .catch(err => console.log(err));

```

* Chapter 421: Handling Errors

1. update
 - app.js(b)
 - ./graphql/resolvers.js(b)

The screenshot shows the GraphiQL interface running in a browser. On the left, the code editor contains a GraphQL schema and a query. The query is:

```

query {
  createUser(userInput: {email: "test", name: "Max", password: "tester"}) {
    _id
    email
  }
}

```

An error is highlighted in the code editor:

```

{
  "errors": [
    {
      "message": "Invalid input.",
      "locations": [
        {
          "line": 30,
          "column": 3
        }
      ],
      "path": [
        "createUser"
      ]
    }
  ],
  "data": null
}

```

The Documentation Explorer panel on the right shows the schema definition for the `createUser` mutation.

The Network tab shows a single request named "graphql" with a status of 500, indicating an internal server error.

- but sometimes you wanna get more detailed information. you can set your own status code.

The screenshot shows the GraphiQL interface running in a browser. The code editor contains the same GraphQL query as before:

```

query {
  createUser(userInput: {email: "test", name: "Max", password: "tester"}) {
    _id
    email
  }
}

```

The response in the Documentation Explorer panel now includes a "status" field in the error object:

```

{
  "errors": [
    {
      "message": "Invalid input.",
      "status": 422,
      "data": [
        {
          "message": "E-Mail is invalid."
        }
      ]
    }
  ],
  "data": null
}

```

```

1 //app.js(b)
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9
10 const graphqlHttp = require('express-graphql');
11

```

```

12 const graphSchema = require('./graphql/schema');
13 const graphResolver = require('./graphql/resolvers');
14
15 const app = express();
16
17 const fileStorage = multer.diskStorage({
18   destination: (req, file, cb) => {
19     cb(null, 'images');
20   },
21   filename: (req, file, cb) => {
22     cb(null, new Date().toISOString() + '-' + file.originalname);
23   }
24 });
25
26 const fileFilter = (req, file, cb) => {
27   if(
28     file.mimetype === 'image/png' ||
29     file.mimetype === 'image/jpg' ||
30     file.mimetype === 'image/jpeg'
31   ) {
32     cb(null, true);
33   } else {
34     cb(null, false);
35   }
36 }
37
38 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
39 app.use(bodyParser.json()); // application/json
40 app.use(multer({storage: fileStorage, fileFilter}).single('image'))
41 app.use('/images', express.static(path.join(__dirname, 'images')));
42
43 app.use((req, res, next) => {
44   res.setHeader('Access-Control-Allow-Origin', '*');
45   res.setHeader(
46     'Access-Control-Allow-Methods',
47     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
48   );
49   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
50   next();
51 });
52
53 app.use('/graphql', graphqlHttp({
54   schema: graphSchema,
55   rootValue: graphResolver,
56   graphiql: true,
57   /**'formatError()' is a method
58    * which receives the error detected by graphQL
59    * and allows you to return your own format
60    * if you return error,
61    * you keep the default format
62    * which is the format we see in the graphiQL localhost:8080/graphql
63    *
64    * and you can check if in your error,
65    * you don't have the originalError field
66    * so if you don't have that
67    * hence exclamation mark at the begining

```

```

68 * or original error will be set by express-graphql
69 * when it detects an error thrown in your code either
70 * by you or some 3rd party package
71 * if you have a technical error,
72 * for example a missing character in your query or anything like that
73 * then it will not have that original error.
74 *
75 * and if we don't have that
76 * then i will return the error that was generated by graphQL
77 * but if i do have my originalError
78 * then i can extract useful information from it that i can add in other places
79 * and that is what we can do into resolver
80 */
81 formatError(err) {
82   if (!err.originalError) {
83     return err;
84   }
85   const data = err.originalError.data;
86   const message = err.message || 'An error occurred.';
87   const code = err.originalError.code || 500
88   return { message: message, status: code, data: data };
89 }
90 })
91 );
92
93 app.use((error, req, res, next) => {
94   console.log(error);
95   const status = error.statusCode || 500;
96   const message = error.message;
97   const data = error.data;
98   res.status(status).json({ message: message, data: data });
99 });
100
101 mongoose
102   .connect(
103     'mongodb+srv://maximilian:rldnj12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
104   )
105   .then(result => {
106     app.listen(8080);
107   })
108   .catch(err => console.log(err));

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5
6 const User = require('../models/user');
7
8 module.exports = {
9   createUser: async function({ userInput }, req) {
10     //const email = args.userInput.email;
11     const errors = [];
12     if (!validator.isEmail(userInput.email)){
13       errors.push({ message: 'E-Mail is invalid.' });
14     }
15     if (validator.isEmpty(userInput.password) || !validator.minLength(userInput.password,

```

```

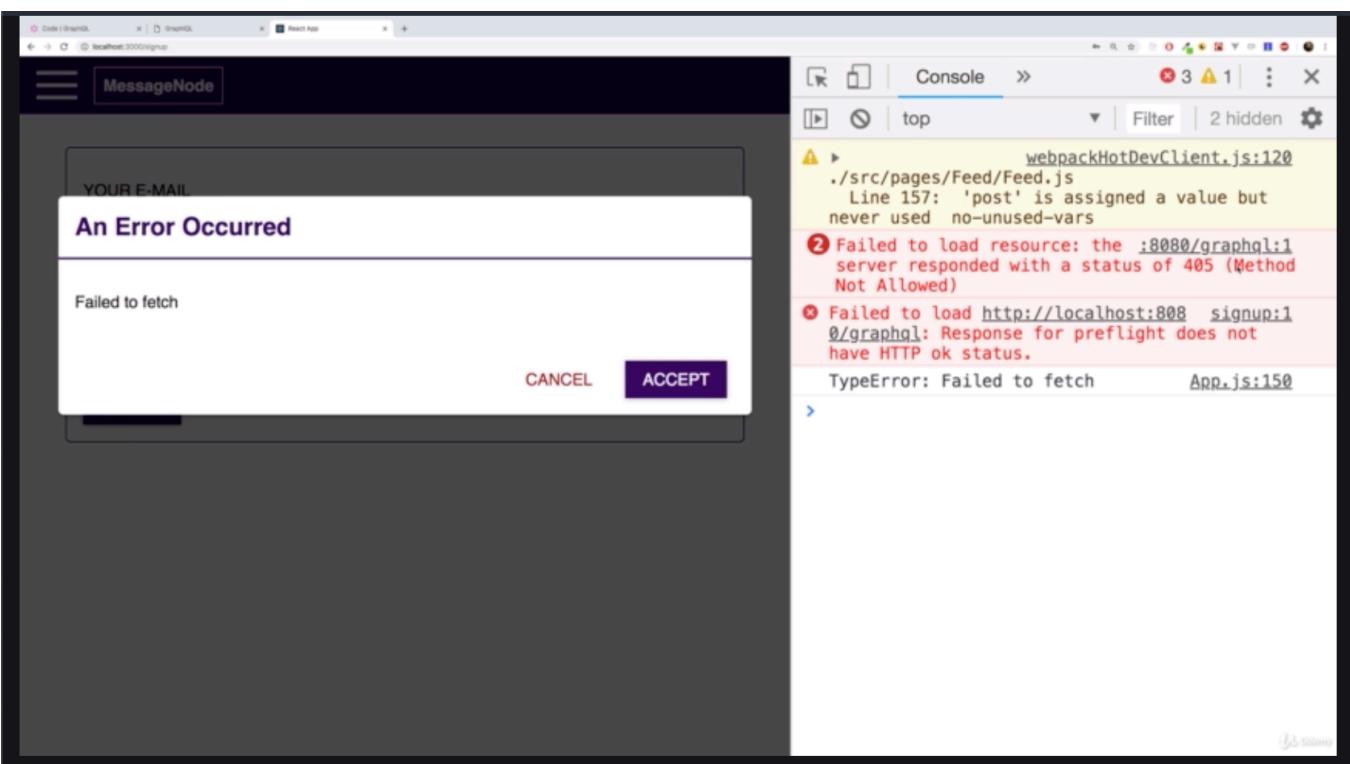
{ min: 5 }}){
16     errors.push({message: 'Password Too Short!'});
17 }
18 if(errors.length > 0){
19     const error = new Error('Invalid Input.');
20     error.data = errors;
21     error.code = 422;
22     throw error;
23 }
24 const existingUser = await User.findOne({email: userInput.email});
25 if(existingUser) {
26     const error = new Error('User exists already!');
27     throw error;
28 }
29 const hashedPw = await bcrypt.hash(userInput.password, 12);
30 const user = new User({
31     email: userInput.email,
32     name: userInput.name,
33     password: hashedPw
34 });
35 const createdUser = await user.save();
36 return { ...createdUser._doc, _id: createdUser._id.toString() };
37 }
38 }

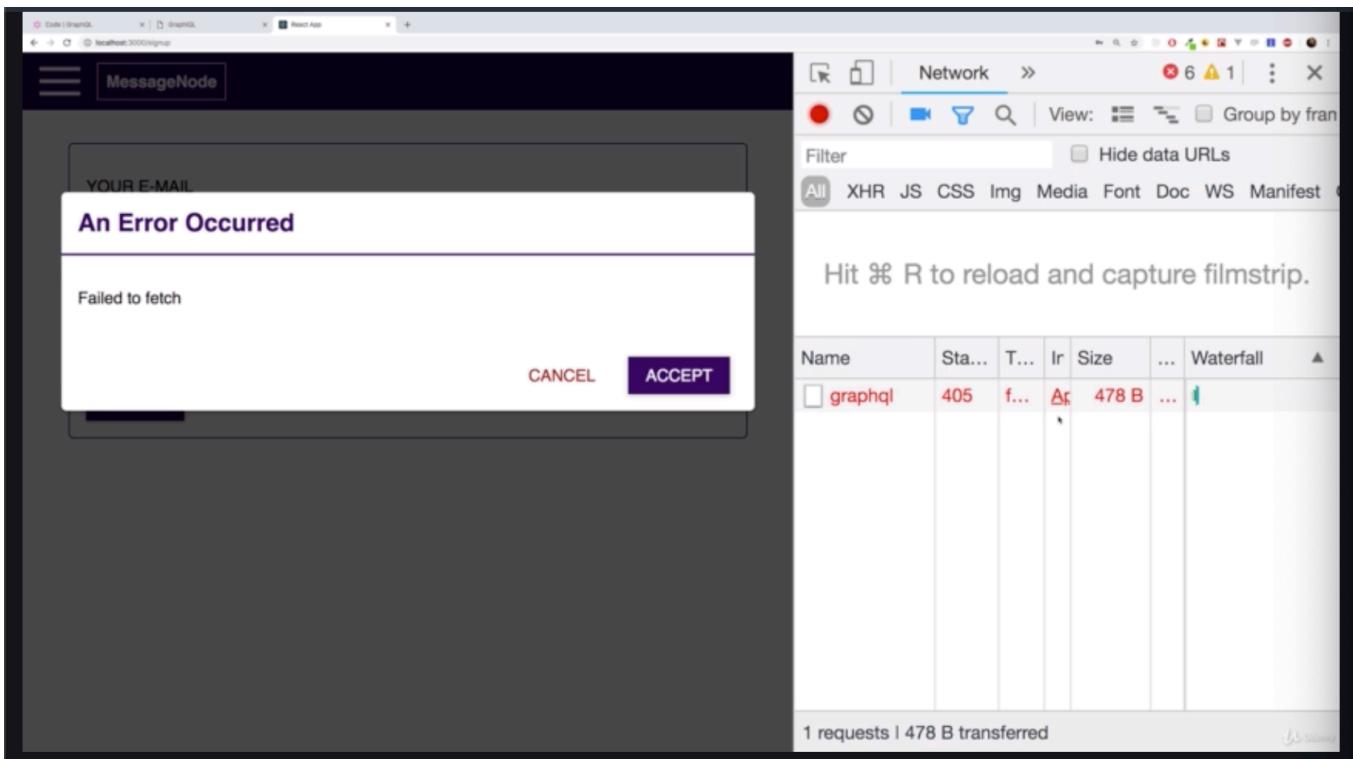
```

* Chapter 422: Connecting The Frontend To The GraphQL API

1. update
- App.js(f)
- app.js(b)

A screenshot of a web browser window. The title bar shows 'Code | GraphQl' and 'localhost:3000/signup'. The main content area has a purple header bar with the text 'MessageNode'. In the center is a sign-up form with three input fields: 'YOUR E-MAIL' containing 'test@test.com', 'YOUR NAME' containing 'Max', and 'PASSWORD' containing '*****'. Below the inputs is a purple 'SIGNUP' button.





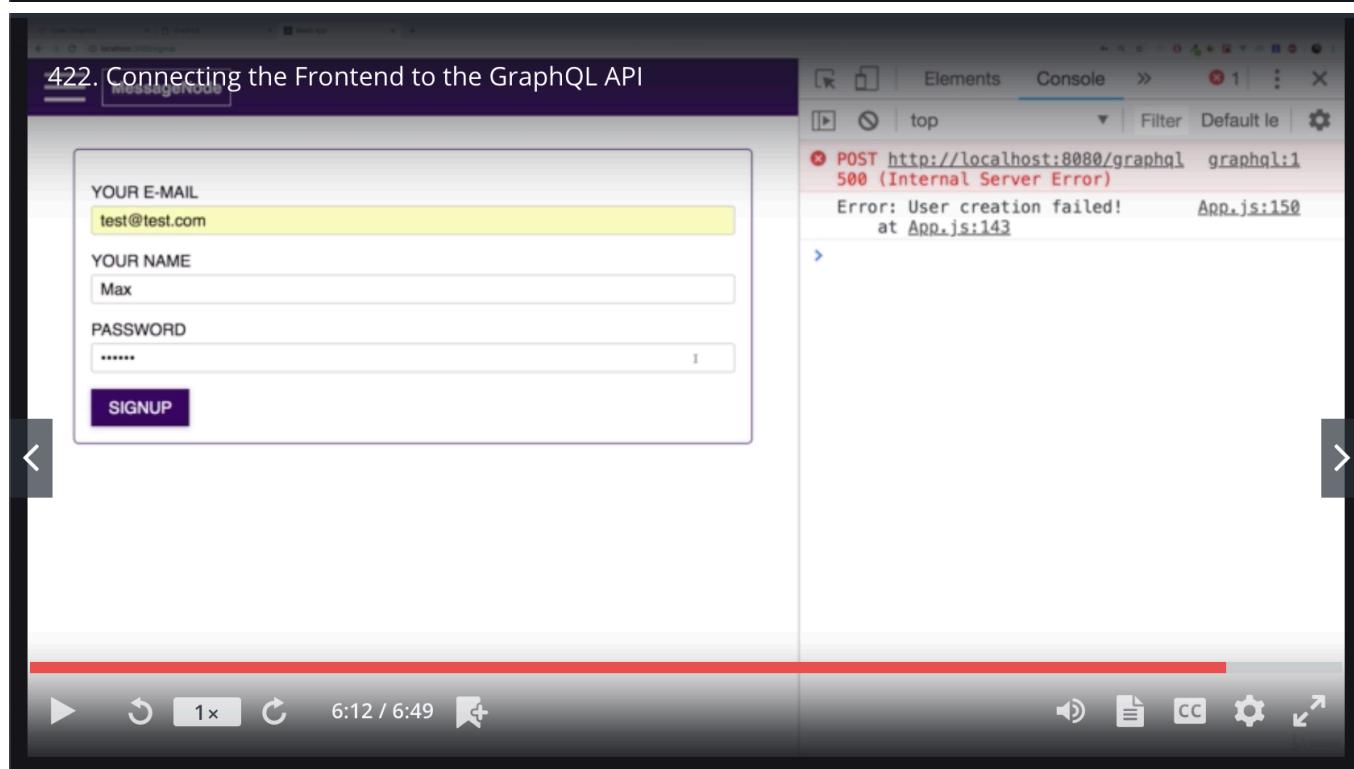
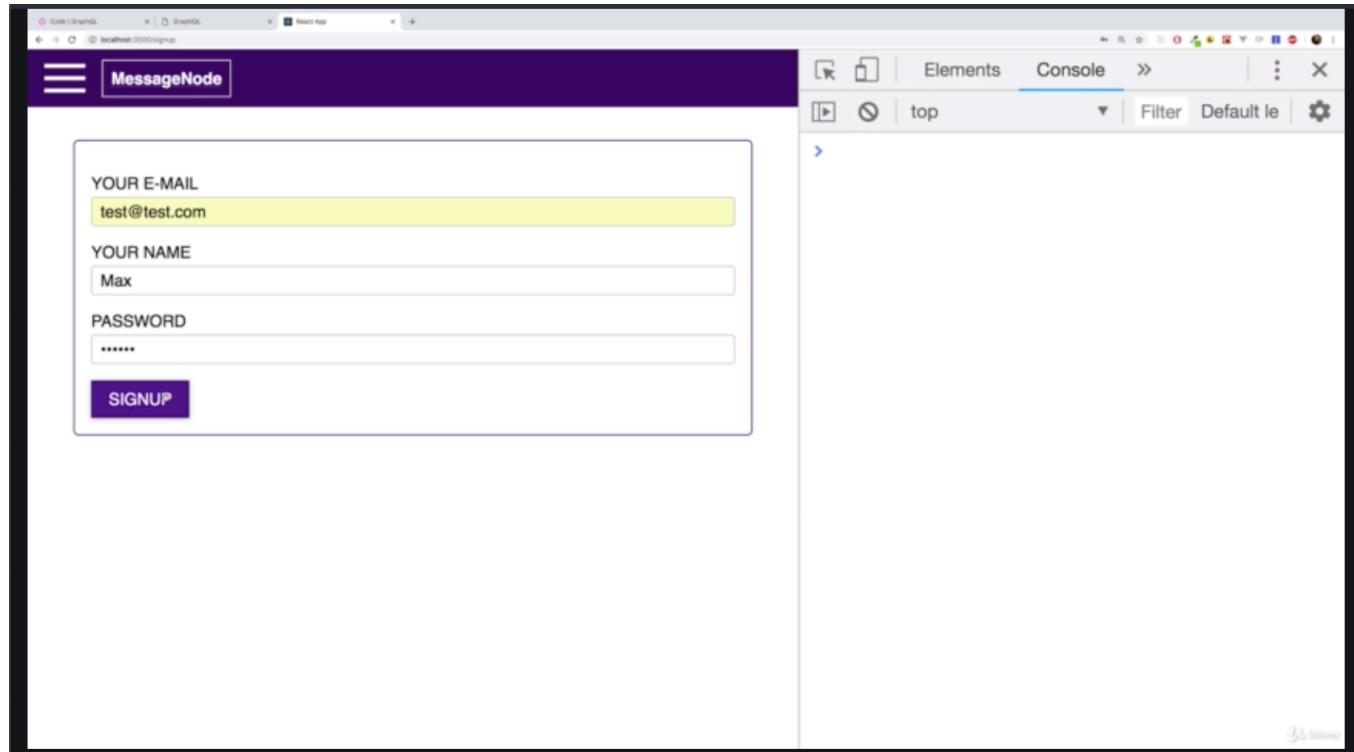
- this is a email address which is already taken. and therefore i should fail doing that. but this is a different error and you might expect an error where my method is not allowed and this can be a tricky thing to fix if you are brand new to craft.
- the reason for this error is we get this error as a response to our options request not to our post request. you might remember that i explained that the browser sends an options request before it sends the POST PATCH PUT DELETE or so on request.
- the problem is express-graphql automatically declines anything which is not a POST or GET request. so the options request is denied.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure with files like app.js, schema.js, resolvers.js, auth.js, controllers/auth.js, controllers/feed.js, graphql/resolvers.js, graphql/schema.js, images, middleware, models/post.js, models/user.js, node_modules, .gitignore, package-lock.json, and package.json.
- CODE EDITOR**: The main area displays the `app.js` file content. The code handles static file serving for images and sets CORS headers. It then handles OPTIONS requests by sending status 200. For other requests, it calls the next middleware function. Finally, it sets up a GraphQL endpoint at `/graphql` using the `graphqlHttp` middleware.
- STATUS BAR**: Shows the current file is `app.js`, line 52, column 10, 7 selected. It also shows the code is in JavaScript mode, using Prettier for styling, and includes icons for file operations like save and close.

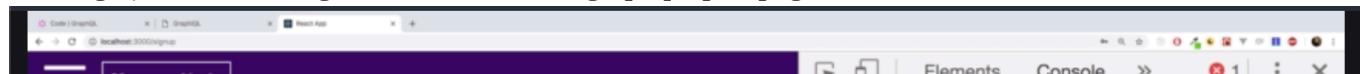
- so the solution is to go to that middleware where i set up my course headers and there i check if my req.method is equal to options and if it's all options request then, i will return res.sendStatus(200). so i will send an empty response with a status code of 200. i return so that 'next()' is not executed and therefore options requests never

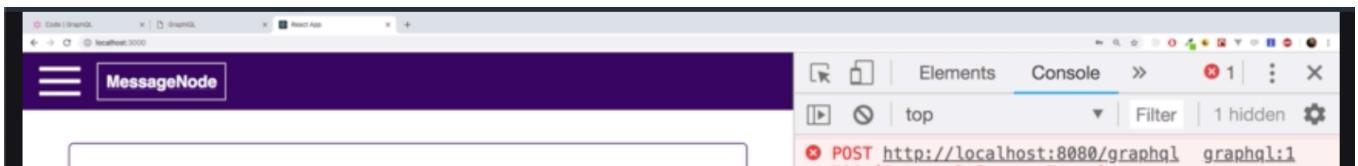
make it to the graphQL endpoint but still get a valid response 'return res.sendStatus(200)'





- so i will go back to my frontend, i clear my error and sign up again. now the user creation fails but now it simply fails because if you look into our data we see that the user exists already and that is what we expect.





- if we take a valid email address which is not taken yet, it succeeds.

```
1 //App.js(f)
2
3 import React, { Component, Fragment } from 'react';
4 import { Route, Switch, Redirect, withRouter } from 'react-router-dom';
5
6 import Layout from './components/Layout/Layout';
7 import Backdrop from './components/Backdrop/Backdrop';
8 import Toolbar from './components/Toolbar/Toolbar';
9 import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
10 import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
11 import ErrorHandler from './components/ErrorHandler/ErrorHandler';
12 import FeedPage from './pages/Feed/Feed';
13 import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
14 import LoginPage from './pages/Auth/Login';
15 import SignupPage from './pages/Auth/Signup';
16 import './App.css';
17
18 class App extends Component {
19   state = {
20     showBackdrop: false,
21     showMobileNav: false,
22     isAuth: false,
23     token: null,
24     userId: null,
25     authLoading: false,
26     error: null
27   };
28
29   componentDidMount() {
30     const token = localStorage.getItem('token');
31     const expiryDate = localStorage.getItem('expiryDate');
32     if (!token || !expiryDate) {
33       return;
```

```
34     }
35     if (new Date(expiryDate) <= new Date()) {
36         this.logoutHandler();
37         return;
38     }
39     const userId = localStorage.getItem('userId');
40     const remainingMilliseconds =
41         new Date(expiryDate).getTime() - new Date().getTime();
42     this.setState({ isAuth: true, token: token, userId: userId });
43     this.setAutoLogout(remainingMilliseconds);
44 }
45
46 mobileNavHandler = isOpen => {
47     this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
48 };
49
50 backdropClickHandler = () => {
51     this.setState({ showBackdrop: false, showMobileNav: false, error: null });
52 };
53
54 logoutHandler = () => {
55     this.setState({ isAuth: false, token: null });
56     localStorage.removeItem('token');
57     localStorage.removeItem('expiryDate');
58     localStorage.removeItem('userId');
59 };
60
61 loginHandler = (event, authData) => {
62     event.preventDefault();
63     this.setState({ authLoading: true });
64     fetch('http://localhost:8080/auth/login', {
65         method: 'POST',
66         headers: {
67             'Content-Type': 'application/json'
68         },
69         body: JSON.stringify({
70             email: authData.email,
71             password: authData.password
72         })
73     })
74     .then(res => {
75         if (res.status === 422) {
76             throw new Error('Validation failed.');
77         }
78         if (res.status !== 200 && res.status !== 201) {
79             console.log('Error!');
80             throw new Error('Could not authenticate you!');
81         }
82         return res.json();
83     })
84     .then(resData => {
85         console.log(resData);
86         this.setState({
87             isAuth: true,
88             token: resData.token,
89             authLoading: false,
```

```

90     userId: resData.userId
91   });
92   localStorage.setItem('token', resData.token);
93   localStorage.setItem('userId', resData.userId);
94   const remainingMilliseconds = 60 * 60 * 1000;
95   const expiryDate = new Date(
96     new Date().getTime() + remainingMilliseconds
97   );
98   localStorage.setItem('expiryDate', expiryDate.toISOString());
99   this.setAutoLogout(remainingMilliseconds);
100 }
101 .catch(err => {
102   console.log(err);
103   this.setState({
104     isAuthenticated: false,
105     authLoading: false,
106     error: err
107   });
108 });
109 };
110
111 signupHandler = (event, authData) => {
112   event.preventDefault();
113   this.setState({ authLoading: true });
114   const graphqlQuery = {
115     /**'query' key is required even for mutation
116      *
117      * and make sure that
118      * you keep the double quotation marks "" around
119      * and check that values
120      * otherwise this will fail
121      * because this query language needs to enclose strings in double quotation marks
122      */
123     query: `mutation {
124       createUser(userInput: {email: "${authData.signupForm.email.value}", name:
125         "${authData.signupForm.name.value}", password: "${authData.signupForm.password.value}"}) {
126         _id
127         email
128       }
129     }
130   }
131
132   fetch('http://localhost:8080/graphql', {
133     method: 'POST',
134     headers: {
135       'Content-Type': 'application/json'
136     },
137     body: JSON.stringify({graphqlQuery})
138   })
139   .then(res => {
140     return res.json();
141   })
142   .then(resData => {
143     if (resData.errors && resData.errors[0].status === 422) {
144       throw new Error(

```

```
145         "Validation failed. Make sure the email address isn't used yet!"  
146     );  
147 }  
148 if(resData.errors) {  
149     throw new Error('User creation failed!');  
150 }  
151 console.log(resData);  
152 this.setState({ isAuth: false, authLoading: false });  
153 this.props.history.replace('/');  
154 })  
155 .catch(err => {  
156     console.log(err);  
157     this.setState({  
158         isAuth: false,  
159         authLoading: false,  
160         error: err  
161     });  
162 });  
163 };  
164  
165 setAutoLogout = milliseconds => {  
166     setTimeout(() => {  
167         this.logoutHandler();  
168     }, milliseconds);  
169 };  
170  
171 errorHandler = () => {  
172     this.setState({ error: null });  
173 };  
174  
175 render() {  
176     let routes = (  
177         <Switch>  
178             <Route  
179                 path="/"  
180                 exact  
181                 render={props => (  
182                     <LoginPage  
183                         {...props}  
184                         onLogin={this.loginHandler}  
185                         loading={this.state.authLoading}  
186                     />  
187                 )}  
188             />  
189             <Route  
190                 path="/signup"  
191                 exact  
192                 render={props => (  
193                     <SignupPage  
194                         {...props}  
195                         onSignup={this.signupHandler}  
196                         loading={this.state.authLoading}  
197                     />  
198                 )}  
199             />  
200             <Redirect to="/" />
```

```

201     </Switch>
202   );
203   if (this.state.isAuthenticated) {
204     routes = (
205       <Switch>
206         <Route
207           path="/"
208           exact
209           render={props => (
210             <FeedPage userId={this.state.userId} token={this.state.token} />
211           )}
212         />
213         <Route
214           path="/:postId"
215           render={props => (
216             <SinglePostPage
217               {...props}
218               userId={this.state.userId}
219               token={this.state.token}
220             />
221           )}
222         />
223         <Redirect to="/" />
224       </Switch>
225     );
226   }
227   return (
228     <Fragment>
229       {this.state.showBackdrop && (
230         <Backdrop onClick={this.backdropClickHandler} />
231       )}
232       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
233       <Layout
234         header={
235           <Toolbar>
236             <MainNavigation
237               onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
238               onLogout={this.logoutHandler}
239               isAuthenticated={this.state.isAuthenticated}
240             />
241           </Toolbar>
242         }
243         mobileNav={
244           <MobileNavigation
245             open={this.state.showMobileNav}
246             mobile
247             onChooseItem={this.mobileNavHandler.bind(this, false)}
248             onLogout={this.logoutHandler}
249             isAuthenticated={this.state.isAuthenticated}
250           />
251         }
252       />
253       {routes}
254     </Fragment>
255   );
256 }

```

```
257 }
258
259 export default withRouter(App);
260
261
1 //app.js(b)
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9
10 const graphqlHttp = require('express-graphql');
11
12 const graphSchema = require('./graphql/schema');
13 const graphResolver = require('./graphql/resolvers');
14
15 const app = express();
16
17 const fileStorage = multer.diskStorage({
18   destination: (req, file, cb) => {
19     cb(null, 'images');
20   },
21   filename: (req, file, cb) => {
22     cb(null, new Date().toISOString() + '-' + file.originalname);
23   }
24 });
25
26 const fileFilter = (req, file, cb) => {
27   if(
28     file.mimetype === 'image/png' ||
29     file.mimetype === 'image/jpg' ||
30     file.mimetype === 'image/jpeg'
31   ) {
32     cb(null, true);
33   } else {
34     cb(null, false);
35   }
36 }
37
38 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
39 app.use(bodyParser.json()); // application/json
40 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
41 app.use('/images', express.static(path.join(__dirname, 'images')));
42
43 app.use((req, res, next) => {
44   res.setHeader('Access-Control-Allow-Origin', '*');
45   res.setHeader(
46     'Access-Control-Allow-Methods',
47     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
48   );
49   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
50   if(req.method === 'OPTIONS') {
51     return res.sendStatus(200);
52   }
53 }
```

```

53     next();
54   });
55
56 app.use('/graphql', graphqlHttp({
57   schema: graphSchema,
58   rootValue: graphResolver,
59   graphiql: true,
60   formatError(err) {
61     if (!err.originalError) {
62       return err;
63     }
64     const data = err.originalError.data;
65     const message = err.message || 'An error occurred.';
66     const code = err.originalError.code || 500
67     return { message: message, status: code, data: data };
68   }
69 })
70 );
71
72 app.use((error, req, res, next) => {
73   console.log(error);
74   const status = error.statusCode || 500;
75   const message = error.message;
76   const data = error.data;
77   res.status(status).json({ message: message, data: data });
78 });
79
80 mongoose
81   .connect(
82     'mongodb+srv://maximilian:rldnj12@cluster0-z3v1k.mongodb.net/message?retryWrites=true'
83   )
84   .then(result => {
85     app.listen(8080);
86   })
87   .catch(err => console.log(err));

```

* Chapter 423: Adding A Login Query & Resolver

1. update
 - ./graphql/schema.js(b)
 - ./graphql/resolvers.js(b)

```

1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 /**at 'login' in type RootQuery
6 * i will return not a user
7 * but some data that contains, for example, user_id, the token
8 * because these are the 2 pieces of information i happened to need here
9 * and that makes sense as a response
10 * of course you could return the overall user object
11 */
12 module.exports = buildSchema(`

13   type Post {
14     _id: ID!

```

```

15     title: String!
16     content: String!
17     imageUrl: String!
18     creator: User!
19     createdAt: String!
20     updatedAt: String!
21   }
22
23 type User {
24   _id: ID!
25   name: String!
26   email: String!
27   password: String
28   status: String!
29   posts: [Post!]!
30 }
31
32 type AuthData {
33   token: String!
34   userId: String!
35 }
36
37 input UserInputData {
38   email: String!
39   name: String!
40   password: String!
41 }
42
43 type RootQuery {
44   login(email: String!, password: String!): AuthData!
45 }
46
47 type RootMutation {
48   createUser(userInput: UserInputData): User!
49 }
50
51 schema {
52   query: RootQuery
53   mutation: RootMutation
54 }
55 `);

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8
9 module.exports = {
10   createUser: async function({ userInput }, req) {
11     //const email = args.userInput.email;
12     const errors = [];
13     if (!validator.isEmail(userInput.email)){
14       errors.push({ message: 'E-Mail is invalid.' });
15     }

```

```

16     if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password,
17   { min: 5 })) {
18       errors.push({message: 'Password Too Short!'});
19     }
20     if(errors.length > 0){
21       const error = new Error('Invalid Input.');
22       error.data = errors;
23       error.code = 422;
24       throw error;
25     }
26     const existingUser = await User.findOne({email: userInput.email});
27     if(existingUser) {
28       const error = new Error('User exists already!');
29       throw error;
30     }
31     const hashedPw = await bcrypt.hash(userInput.password, 12);
32     const user = new User({
33       email: userInput.email,
34       name: userInput.name,
35       password: hashedPw
36     });
37     const createdUser = await user.save();
38     return { ...createdUser._doc, _id: createdUser._id.toString() };
39   },
40   /**'{ email, password }' is from ./graphql/schema.js(b) 'type RootQuery'*/
41   login: async function({ email, password }) {
42     const user = await User.findOne({email: email});
43     if(!user) {
44       const error = new Error('User not found.');
45       error.code = 404;
46       throw error;
47     }
48     const isEqual = await bcrypt.compare(password, user.password);
49     if(!isEqual) {
50       const error = new Error('Password is incorrect.');
51       error.code = 401;
52       throw error;
53     }
54     const token = jwt.sign({
55       userId: user._id.toString(),
56       email: user.email
57     },
58     'somesupersecretsecret',
59     { expiresIn: '1h' })
60   /**'token' on the left is from ./graphql/schema.js(b) 'type AuthData' */
61   return { token: token, userId: user._id.toString() };
62 }

```

* Chapter 424: Adding Login Functionality

- this is my restData object in my App.js(f). so restData will be this entire object. so if you wanna get the token, i need to drill into it into data.login.token

- so now in places where i would just resData.token and resData.userId, change to resData.data.login.token and resData.data.login.userId

- log in. we get an error for fetching the posts which makes sense because we try to use our REST API which doesn't exist but logging in actually succeeded. so we got users sign up and log in in place.

```
1 //App.js(f)
2
3 import React, { Component, Fragment } from 'react';
4 import { Route, Switch, Redirect, withRouter } from 'react-router-dom';
5
6 import Layout from './components/Layout/Layout';
7 import Backdrop from './components/Backdrop/Backdrop';
8 import Toolbar from './components/Toolbar/Toolbar';
9 import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
10 import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
11 import ErrorHandler from './components/ErrorHandler/ErrorHandler';
12 import FeedPage from './pages/Feed/Feed';
13 import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
14 import LoginPage from './pages/Auth/Login';
15 import SignupPage from './pages/Auth/Signup';
16 import './App.css';
17
18 class App extends Component {
19   state = {
20     showBackdrop: false,
21     showMobileNav: false,
22     isAuth: false,
23     token: null,
24     userId: null,
25     authLoading: false,
26     error: null
27   };
28
29   componentDidMount() {
30     const token = localStorage.getItem('token');
31     const expiryDate = localStorage.getItem('expiryDate');
32     if (!token || !expiryDate) {
33       return;
34     }
35     if (new Date(expiryDate) <= new Date()) {
36       this.logoutHandler();
37       return;
38     }
39   }
40 }
```

```

39  const userId = localStorage.getItem('userId');
40  const remainingMilliseconds =
41    new Date(expiryDate).getTime() - new Date().getTime();
42  this.setState({ isAuthenticated: true, token, userId });
43  this.setAutoLogout(remainingMilliseconds);
44 }
45
46 mobileNavHandler = isOpen => {
47   this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
48 };
49
50 backdropClickHandler = () => {
51   this.setState({ showBackdrop: false, showMobileNav: false, error: null });
52 };
53
54 logoutHandler = () => {
55   this.setState({ isAuthenticated: false, token: null });
56   localStorage.removeItem('token');
57   localStorage.removeItem('expiryDate');
58   localStorage.removeItem('userId');
59 };
60
61 loginHandler = (event, authData) => {
62   event.preventDefault();
63   /**
64    * for a query,
65    * you don't repeat query
66    * you start right away with the curly braces
67    * as we did it in the graphiql interface
68    *
69    * 'login' is from ./graphql/schema.js(b) 'type RootQuery login'
70    */
71   const graphqlQuery = {
72     query: `

73       login(email: "${authData.email}", password: "${authData.password}") {
74         token
75         userId
76       }
77     `
78   };
79   this.setState({ authLoading: true });
80   fetch('http://localhost:8080/graphql', {
81     method: 'POST',
82     headers: {
83       'Content-Type': 'application/json'
84     },
85     body: JSON.stringify(graphqlQuery)
86   })
87     .then(res => {
88       return res.json();
89     })
90     .then(resData => {
91       if (resData.errors && resData.errors[0].status === 422) {
92         throw new Error(
93           "Validation failed. Make sure the email address isn't used yet!"
94         )

```

```

95     );
96   }
97   if (resData.errors) {
98     throw new Error('User login failed!');
99   }
100  console.log(resData);
101  this.setState({
102    isAuthenticated: true,
103    token: resData.data.login.token,
104    authLoading: false,
105    userId: resData.data.login.userId
106  });
107  localStorage.setItem('token', resData.data.login.token);
108  localStorage.setItem('userId', resData.data.login.userId);
109  const remainingMilliseconds = 60 * 60 * 1000;
110  const expiryDate = new Date(
111    new Date().getTime() + remainingMilliseconds
112  );
113  localStorage.setItem('expiryDate', expiryDate.toISOString());
114  this.setAutoLogout(remainingMilliseconds);
115})
116 .catch(err => {
117   console.log(err);
118   this.setState({
119     isAuthenticated: false,
120     authLoading: false,
121     error: err
122   });
123 });
124 };
125
126 signupHandler = (event, authData) => {
127   event.preventDefault();
128   this.setState({ authLoading: true });
129   const graphqlQuery = {
130     query: `
131       mutation {
132         createUser(userInput: {email: "${{
133           authData.signupForm.email.value
134         }", name:"${authData.signupForm.name.value}", password:"${{
135           authData.signupForm.password.value
136         }}") {
137           _id
138           email
139         }
140       }
141     `;
142   };
143   fetch('http://localhost:8080/graphql', {
144     method: 'POST',
145     headers: {
146       'Content-Type': 'application/json'
147     },
148     body: JSON.stringify(graphqlQuery)
149   })
150   .then(res => {

```

```
151     return res.json();
152   })
153   .then(resData => {
154     if (resData.errors && resData.errors[0].status === 422) {
155       throw new Error(
156         "Validation failed. Make sure the email address isn't used yet!"
157       );
158     }
159     if (resData.errors) {
160       throw new Error('User creation failed!');
161     }
162     console.log(resData);
163     this.setState({ isAuthenticated: false, authLoading: false });
164     this.props.history.replace('/');
165   })
166   .catch(err => {
167     console.log(err);
168     this.setState({
169       isAuthenticated: false,
170       authLoading: false,
171       error: err
172     });
173   });
174 };
175
176 setAutoLogout = milliseconds => {
177   setTimeout(() => {
178     this.logoutHandler();
179   }, milliseconds);
180 };
181
182 errorHandler = () => {
183   this.setState({ error: null });
184 };
185
186 render() {
187   let routes = (
188     <Switch>
189       <Route
190         path="/"
191         exact
192         render={props => (
193           <LoginPage
194             {...props}
195             onLogin={this.loginHandler}
196             loading={this.state.authLoading}
197           />
198         )}
199       />
200       <Route
201         path="/signup"
202         exact
203         render={props => (
204           <SignupPage
205             {...props}
206             onSignup={this.signupHandler}
```

```
207           loading={this.state.authLoading}
208         />
209       )}
210     />
211     <Redirect to="/" />
212   </Switch>
213 );
214 if (this.state.isAuthenticated) {
215   routes = (
216     <Switch>
217       <Route
218         path="/"
219         exact
220         render={props => (
221           <FeedPage userId={this.state.userId} token={this.state.token} />
222         )}
223       />
224       <Route
225         path="/:postId"
226         render={props => (
227           <SinglePostPage
228             {...props}
229             userId={this.state.userId}
230             token={this.state.token}
231           />
232         )}
233       />
234       <Redirect to="/" />
235     </Switch>
236 );
237 }
238 return (
239   <Fragment>
240     {this.state.showBackdrop && (
241       <Backdrop onClick={this.backdropClickHandler} />
242     )}
243     <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
244     <Layout
245       header={
246         <Toolbar>
247           <MainNavigation
248             onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
249             onLogout={this.logoutHandler}
250             isAuthenticated={this.state.isAuthenticated}
251           />
252         </Toolbar>
253       }
254       mobileNav={
255         <MobileNavigation
256           open={this.state.showMobileNav}
257           mobile
258           onChooseItem={this.mobileNavHandler.bind(this, false)}
259           onLogout={this.logoutHandler}
260           isAuthenticated={this.state.isAuthenticated}
261         />
262       }

```

```

263     />
264     {routes}
265   </Fragment>
266 );
267 }
268 }
269
270 export default withRouter(App);
271

```

* Chapter 425: Adding A Create Post Mutation

1. update
- ./graphql/schema.js(b)
- ./graphql/resolvers.js(b)

- i get an error from the database that the creator is missing which makes sense. but the general mutation is working.

```

1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 module.exports = buildSchema(`

6   type Post {
7     _id: ID!
8     title: String!
9     content: String!
10    imageUrl: String!
11    creator: User!
12    createdAt: String!
13    updatedAt: String!
14  }

15
16   type User {
17     _id: ID!
18     name: String!
19     email: String!
20     password: String
21     status: String!
22     posts: [Post!]!
23   }

24
25   type AuthData {
26     token: String!
27     userId: String!
28   }

29
30   input UserInputData {
31     email: String!
32     name: String!
33     password: String!
34   }

```

```

35
36 type RootQuery {
37     login(email: String!, password: String!): AuthData!
38 }
39
40 input PostInputData {
41     title: String!
42     content: String!
43     imageUrl: String!
44 }
45
46 type RootMutation {
47     createUser(userInput: UserInputData): User!
48     createPost(postInput: PostInputData): Post!
49 }
50
51 schema {
52     query: RootQuery
53     mutation: RootMutation
54 }
55 `);

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9
10 module.exports = {
11     createUser: async function({ userInput }, req) {
12         //const email = args.userInput.email;
13         const errors = [];
14         if (!validator.isEmail(userInput.email)){
15             errors.push({ message: 'E-Mail is invalid.' });
16         }
17         if (validator.isEmpty(userInput.password) || !validator.minLength(userInput.password, { min: 5 })) {
18             errors.push({ message: 'Password Too Short!' });
19         }
20         if(errors.length > 0){
21             const error = new Error('Invalid Input.');
22             error.data = errors;
23             error.code = 422;
24             throw error;
25         }
26         const existingUser = await User.findOne({email: userInput.email});
27         if(existingUser) {
28             const error = new Error('User exists already!');
29             throw error;
30         }
31         const hashedPw = await bcrypt.hash(userInput.password, 12);
32         const user = new User({
33             email: userInput.email,
34             name: userInput.name,

```

```
35         password: hashedPw
36     });
37     const createdUser = await user.save();
38     return { ...createdUser._doc, _id: createdUser._id.toString() };
39 },
40 login: async function({ email, password }) {
41     const user = await User.findOne({email: email});
42     if(!user) {
43         const error = new Error('User not found.');
44         error.code = 404;
45         throw error;
46     }
47     const isEqual = await bcrypt.compare(password, user.password);
48     if(!isEqual) {
49         const error = new Error('Password is incorrect.');
50         error.code = 401;
51         throw error;
52     }
53     const token = jwt.sign({
54         userId: user._id.toString(),
55         email: user.email
56     },
57     'somesupersecretsecret',
58     { expiresIn: '1h' })
59     return { token: token, userId: user._id.toString() };
60 },
61 /**'postInput' is from 'type RootMutation' */
62 createPost: async function({ postInput }, req){
63     const errors = [];
64     if(
65         validator.isEmpty(postInput.title) ||
66         !validator.isLength(postInput.title, { min: 5 })
67     ) {
68         errors.push({ message: 'Title is invalid.' });
69     }
70     if(
71         validator.isEmpty(postInput.content) ||
72         !validator.isLength(postInput.content, { min: 5 })
73     ) {
74         errors.push({ message: 'Content is invalid.' });
75     }
76     if(errors.length > 0){
77         const error = new Error('Invalid Input.');
78         error.data = errors;
79         error.code = 422;
80         throw error;
81     }
82     const post = new User({
83         title: postInput.title,
84         content: postInput.content,
85         imageUrl: postInput.imageUrl
86     });
87     const createdPost = await post.save();
88     //Add post to users' posts
89     return {
90         ...createdPost._doc,
```

```

91     .....
92     .....
93     .....
94     .....
95   }
96 }

```

* Chapter 426: Extracting User Data From The Auth Token

1. update
- ./middleware/auth.js(b)
- app.js(b)
- ./graphql/resolvers.js(b)

```

1 //./middleware/auth.js(b)
2
3 const jwt = require('jsonwebtoken');
4
5 module.exports = (req, res, next) => {
6   const authHeader = req.get('Authorization');
7   if (!authHeader) {
8     /**'req.isAuthenticated' = false'
9      * so that i can handle this inside of my resolvers.
10     */
11     req.isAuthenticated = false
12     return next();
13   }
14   const token = authHeader.split(' ')[1];
15   let decodedToken;
16   try {
17     decodedToken = jwt.verify(token, 'somesupersecretsecret');
18   } catch (err) {
19     req.isAuthenticated = false;
20     return next();
21   }
22   if (!decodedToken) {
23     req.isAuthenticated = false;
24     return next();
25   }
26   req.userId = decodedToken.userId;
27   req.isAuthenticated = true;
28   next();
29 };
30

```

```

1 //app.js(b)
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9
10 const graphqlHttp = require('express-graphql');

```

```
11
12 const graphSchema = require('./graphql/schema');
13 const graphResolver = require('./graphql/resolvers');
14 const auth = require('./middleware/auth');
15
16 const app = express();
17
18 const fileStorage = multer.diskStorage({
19   destination: (req, file, cb) => {
20     cb(null, 'images');
21   },
22   filename: (req, file, cb) => {
23     cb(null, new Date().toISOString() + '-' + file.originalname);
24   }
25 });
26
27 const fileFilter = (req, file, cb) => {
28   if(
29     file.mimetype === 'image/png' ||
30     file.mimetype === 'image/jpg' ||
31     file.mimetype === 'image/jpeg'
32   ) {
33     cb(null, true);
34   } else {
35     cb(null, false);
36   }
37 }
38
39 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
40 app.use(bodyParser.json()); // application/json
41 app.use(multer({storage: fileStorage, fileFilter}).single('image'))
42 app.use('/images', express.static(path.join(__dirname, 'images')));
43
44 app.use((req, res, next) => {
45   res.setHeader('Access-Control-Allow-Origin', '*');
46   res.setHeader(
47     'Access-Control-Allow-Methods',
48     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
49   );
50   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
51   if(req.method === 'OPTIONS') {
52     return res.sendStatus(200);
53   }
54   next();
55 });
56
57 /**this is my middleware
58 * and i will add it like that
59 * this middleware will run on every request
60 * that reaches my GraphQL end point
61 * but it will not deny the request if there's no token
62 * the only thing is what set off to false
63 * and then i can decide in my resolver whether i wanna continue or not
64 */
65 app.use(auth);
66
```

```

67 app.use('/graphql', graphqlHttp({
68   schema: graphSchema,
69   rootValue: graphResolver,
70   graphiql: true,
71   formatError(err) {
72     if (!err.originalError) {
73       return err;
74     }
75     const data = err.originalError.data;
76     const message = err.message || 'An error occurred.';
77     const code = err.originalError.code || 500
78     return { message: message, status: code, data: data };
79   }
80 })
81 );
82
83 app.use((error, req, res, next) => {
84   console.log(error);
85   const status = error.statusCode || 500;
86   const message = error.message;
87   const data = error.data;
88   res.status(status).json({ message: message, data: data });
89 });
90
91 mongoose
92   .connect(
93     'mongodb+srv://maximilian:rldnj12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
94   )
95   .then(result => {
96     app.listen(8080);
97   })
98   .catch(err => console.log(err));

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9
10 module.exports = {
11   createUser: async function({ userInput }, req) {
12     //const email = args.userInput.email;
13     const errors = [];
14     if (!validator.isEmail(userInput.email)){
15       errors.push({ message: 'E-Mail is invalid.' });
16     }
17     if (validator.isEmpty(userInput.password) || !validator.minLength(userInput.password, { min: 5 })){
18       errors.push({message: 'Password Too Short!'});
19     }
20     if(errors.length > 0){
21       const error = new Error('Invalid Input.');
22       error.data = errors;
23       error.code = 422;

```

```
24     throw error;
25 }
26 const existingUser = await User.findOne({email: userInput.email});
27 if(existingUser) {
28     const error = new Error('User exists already!');
29     throw error;
30 }
31 const hashedPw = await bcrypt.hash(userInput.password, 12);
32 const user = new User({
33     email: userInput.email,
34     name: userInput.name,
35     password: hashedPw
36 });
37 const createdUser = await user.save();
38 return { ...createdUser._doc, _id: createdUser._id.toString() };
39 },
40 login: async function({ email, password }) {
41     const user = await User.findOne({email: email});
42     if(!user) {
43         const error = new Error('User not found.');
44         error.code = 404;
45         throw error;
46     }
47     const isEqual = await bcrypt.compare(password, user.password);
48     if(!isEqual) {
49         const error = new Error('Password is incorrect.');
50         error.code = 401;
51         throw error;
52     }
53     const token = jwt.sign({
54         userId: user._id.toString(),
55         email: user.email
56     },
57     'somesupersecretsecret',
58     { expiresIn: '1h' })
59     return { token: token, userId: user._id.toString() };
60 },
61 /**'postInput' is from 'type RootMutation ' */
62 createPost: async function({ postInput }, req){
63     if(!req.isAuthenticated){
64         const error = new Error('Not authenticated!');
65         error.code = 401;
66         throw error;
67     }
68     const errors = [];
69     if(
70         validator.isEmpty(postInput.title) ||
71         !validator.isLength(postInput.title, { min: 5 })
72     ) {
73         errors.push({ message: 'Title is invalid.' });
74     }
75     if(
76         validator.isEmpty(postInput.content) ||
77         !validator.isLength(postInput.content, { min: 5 })
78     ) {
79         errors.push({ message: 'Content is invalid.' });
80     }
81     if(errors.length > 0) {
82         return { errors: errors };
83     }
84     const newPost = new Post({
85         title: postInput.title,
86         content: postInput.content,
87         author: req.user._id
88     });
89     await newPost.save();
90     return { post: newPost };
91 }
```

```

80     }
81     if(errors.length > 0){
82       const error = new Error('Invalid Input.');
83       error.data = errors;
84       error.code = 422;
85       throw error;
86     }
87     const user = await User.findById(req.userId);
88     if(!user){
89       const error = new Error('Invalid user.');
90       error.code = 401;
91       throw error;
92     }
93     const post = new User({
94       title: postInput.title,
95       content: postInput.content,
96       imageUrl: postInput.imageUrl,
97       creator: user
98     });
99     const createdPost = await post.save();
100    user.posts.push(createdPost);
101    return {
102      ...createdPost._doc,
103      _id: createdPost._id.toString(),
104      createdAt: createdPost.createdAt.toISOString(),
105      updatedAt: createdPost.updatedAt.toISOString()
106    };
107  };
108}

```

* Chapter 427: Sending The “Create Post” Query

1. update
 - ./src/pages/Feed/Feed.js(f)

- i get an error which was expected. but i also see a console log which doesn't look too bad. i have all the data i entered and we can now have a look at more.

- in posts collection i see a post with the data i entered with that creator ending with ‘aa1’

- if i have a look at my users, i see that it's that user and they are on the posts. i'm unfortunately missing my post so that is one thign i will need to tweak but the rest is already working well

1 //./src/pages/Feed/Feed.js(f)
 2

```
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('http://localhost:8080/auth/status', {
28       headers: {
29         Authorization: 'Bearer ' + this.props.token
30       }
31     })
32       .then(res => {
33         if (res.status !== 200) {
34           throw new Error('Failed to fetch user status.');
35         }
36         return res.json();
37       })
38       .then(resData => {
39         this.setState({ status: resData.status });
40       })
41       .catch(this.catchError);
42
43     this.loadPosts();
44   }
45
46   loadPosts = direction => {
47     if (direction) {
48       this.setState({ postsLoading: true, posts: [] });
49     }
50     let page = this.state.postPage;
51     if (direction === 'next') {
52       page++;
53       this.setState({ postPage: page });
54     }
55     if (direction === 'previous') {
56       page--;
57       this.setState({ postPage: page });
58     }

```

```

59   fetch('http://localhost:8080/feed/posts?page=' + page, {
60     headers: {
61       Authorization: 'Bearer ' + this.props.token
62     }
63   })
64   .then(res => {
65     if (res.status !== 200) {
66       throw new Error('Failed to fetch posts.');
67     }
68     return res.json();
69   })
70   .then(resData => {
71     this.setState({
72       posts: resData.posts.map(post => {
73         return {
74           ...post,
75           imagePath: post.imageUrl
76         };
77       }),
78       totalPosts: resData.totalItems,
79       postsLoading: false
80     });
81   })
82   .catch(this.catchError);
83 };
84
85 statusUpdateHandler = event => {
86   event.preventDefault();
87   fetch('http://localhost:8080/auth/status', {
88     method: 'PATCH',
89     headers: {
90       Authorization: 'Bearer ' + this.props.token,
91       'Content-Type': 'application/json'
92     },
93     body: JSON.stringify({
94       status: this.state.status
95     })
96   })
97   .then(res => {
98     if (res.status !== 200 && res.status !== 201) {
99       throw new Error("Can't update status!");
100    }
101    return res.json();
102  })
103  .then(resData => {
104    console.log(resData);
105  })
106  .catch(this.catchError);
107};
108
109 newPostHandler = () => {
110   this.setState({isEditing: true});
111 };
112
113 startEditPostHandler = postId => {
114   this.setState(prevState => {

```

```
115     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
116
117     return {
118       isEditing: true,
119       editPost: loadedPost
120     };
121   );
122 }
123
124 cancelEditHandler = () => {
125   this.setState({ isEditing: false, editPost: null });
126 };
127
128 finishEditHandler = postData => {
129   this.setState({
130     editLoading: true
131   });
132   const formData = new FormData();
133   formData.append('title', postData.title);
134   formData.append('content', postData.content);
135   formData.append('image', postData.image);
136
137   let graphqlQuery = {
138     query: `
139       mutation {
140         createPost(postInput: {title: "${postData.title}", content: "${{
141           postData.content
142         }", imageUrl: "some url"}) {
143           _id
144           title
145           content
146           imageUrl
147           creator {
148             name
149           }
150           createdAt
151         }
152       }
153     `
154   };
155
156   fetch('http://localhost:8080/graphql', {
157     method: 'POST',
158     body: JSON.stringify(graphqlQuery),
159     headers: {
160       Authorization: 'Bearer ' + this.props.token,
161       'Content-Type': 'application/json'
162     }
163   })
164     .then(res => {
165       return res.json();
166     })
167     .then(resData => {
168       if (resData.errors && resData.errors[0].status === 422) {
169         throw new Error(
170           "Validation failed. Make sure the email address isn't used yet!"
171         )
172     }
173   );
174 }
```

```

171     );
172   }
173   if (resData.errors) {
174     throw new Error('User login failed!');
175   }
176   console.log(resData);
177   const post = {
178     _id: resData.data.createPost._id,
179     title: resData.data.createPost.title,
180     content: resData.data.createPost.content,
181     creator: resData.data.createPost.creator,
182     createdAt: resData.data.createPost.createdAt
183   };
184   this.setState(prevState => {
185     return {
186       isEditing: false,
187       editPost: null,
188       editLoading: false
189     };
190   });
191 })
192 .catch(err => {
193   console.log(err);
194   this.setState({
195     isEditing: false,
196     editPost: null,
197     editLoading: false,
198     error: err
199   });
200 });
201 );
202
203 statusInputChangeHandler = (input, value) => {
204   this.setState({ status: value });
205 };
206
207 deletePostHandler = postId => {
208   this.setState({ postsLoading: true });
209   fetch('http://localhost:8080/feed/post/' + postId, {
210     method: 'DELETE',
211     headers: {
212       Authorization: 'Bearer ' + this.props.token
213     }
214   })
215   .then(res => {
216     if (res.status !== 200 && res.status !== 201) {
217       throw new Error('Deleting a post failed!');
218     }
219     return res.json();
220   })
221   .then(resData => {
222     console.log(resData);
223     this.loadPosts();
224     // this.setState(prevState => {
225     //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
226     //   return { posts: updatedPosts, postsLoading: false };

```

```
227     // });
228   })
229   .catch(err => {
230     console.log(err);
231     this.setState({ postsLoading: false });
232   });
233 };
234
235 errorHandler = () => {
236   this.setState({ error: null });
237 };
238
239 catchError = error => {
240   this.setState({ error: error });
241 };
242
243 render() {
244   return (
245     <Fragment>
246       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
247       <FeedEdit
248         editing={this.state.isEditing}
249         selectedPost={this.state.editPost}
250         loading={this.state.editLoading}
251         onCancelEdit={this.cancelEditHandler}
252         onFinishEdit={this.finishEditHandler}
253       />
254       <section className="feed__status">
255         <form onSubmit={this.statusUpdateHandler}>
256           <Input
257             type="text"
258             placeholder="Your status"
259             control="input"
260             onChange={this.statusInputChangeHandler}
261             value={this.state.status}
262           />
263           <Button mode="flat" type="submit">
264             Update
265           </Button>
266         </form>
267       </section>
268       <section className="feed__control">
269         <Button mode="raised" design="accent" onClick={this.newPostHandler}>
270           New Post
271         </Button>
272       </section>
273       <section className="feed">
274         {this.state.postsLoading && (
275           <div style={{ textAlign: 'center', marginTop: '2rem' }}>
276             <Loader />
277           </div>
278         )}
279         {this.state.posts.length <= 0 && !this.state.postsLoading ? (
280           <p style={{ textAlign: 'center' }}>No posts found.</p>
281         ) : null}
282         {!this.state.postsLoading && (
```

```

283     <Paginator
284       onPrevious={this.loadPosts.bind(this, 'previous')}
285       onNext={this.loadPosts.bind(this, 'next')}
286       lastPage={Math.ceil(this.state.totalPosts / 2)}
287       currentPage={this.state.postPage}
288     >
289     {this.state.posts.map(post => (
290       <Post
291         key={post._id}
292         id={post._id}
293         author={post.creator.name}
294         date={new Date(post.createdAt).toLocaleDateString('en-US')}
295         title={post.title}
296         image={post.imageUrl}
297         content={post.content}
298         onStartEdit={this.startEditPostHandler.bind(this, post._id)}
299         onDelete={this.deletePostHandler.bind(this, post._id)}
300       />
301     ))}
302   </Paginator>
303 }
304 </section>
305 </Fragment>
306 );
307 }
308 }
309
310 export default Feed;
311

```

* Chapter 428: Fixing A Bug & Adding New Posts Correctly

1. update
 - ./graphql/resolevers.js(b)
 - ./src/pages/Feed/Feed.js(f)


```
1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9
10 module.exports = {
11     createUser: async function({ userInput }, req) {
12         //const email = args.userInput.email;
13         const errors = [];
14         if (!validator.isEmail(userInput.email)){
15             errors.push({ message: 'E-Mail is invalid.' });
16         }
17         if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password,
18 { min: 5 })) {
19             errors.push({message: 'Password Too Short!'});
20         }
21         if(errors.length > 0){
22             const error = new Error('Invalid Input.');
23             error.data = errors;
24             error.code = 422;
25             throw error;
26         }
27         const existingUser = await User.findOne({email: userInput.email});
28         if(existingUser) {
29             const error = new Error('User exists already!');
30             throw error;
31         }
32         const hashedPw = await bcrypt.hash(userInput.password, 12);
33         const user = new User({
34             email: userInput.email,
35             name: userInput.name,
36             password: hashedPw
37         });
38         const createdUser = await user.save();
39         return { ...createdUser._doc, _id: createdUser._id.toString() };
40     },
41     login: async function({ email, password }) {
42         const user = await User.findOne({email: email});
43         if(!user) {
44             const error = new Error('User not found.');
45             error.code = 404;
46             throw error;
47         }
48         const isEqual = await bcrypt.compare(password, user.password);
49         if(!isEqual) {
50             const error = new Error('Password is incorrect.');
51             error.code = 401;
52             throw error;
53     }
```

```
53  const token = jwt.sign({
54    userId: user._id.toString(),
55    email: user.email
56  },
57  'somesupersecretsecret',
58  { expiresIn: '1h' })
59  return { token: token, userId: user._id.toString() };
60},
61 /**'postInput' is from 'type RootMutation' */
62 createPost: async function({ postInput }, req){
63  if(!req.isAuthenticated){
64    const error = new Error('Not authenticated!');
65    error.code = 401;
66    throw error;
67  }
68  const errors = [];
69  if(
70    validator.isEmpty(postInput.title) ||
71    !validator.isLength(postInput.title, { min: 5 })
72  ) {
73    errors.push({ message: 'Title is invalid.' });
74  }
75  if(
76    validator.isEmpty(postInput.content) ||
77    !validator.isLength(postInput.content, { min: 5 })
78  ) {
79    errors.push({ message: 'Content is invalid.' });
80  }
81  if(errors.length > 0){
82    const error = new Error('Invalid Input.');
83    error.data = errors;
84    error.code = 422;
85    throw error;
86  }
87  const user = await User.findById(req.userId);
88  if(!user){
89    const error = new Error('Invalid user.');
90    error.code = 401;
91    throw error;
92  }
93  const post = new User({
94    title: postInput.title,
95    content: postInput.content,
96    imageUrl: postInput.imageUrl,
97    creator: user
98  });
99  const createdPost = await post.save();
100 user.posts.push(createdPost);
101 await user.save()
102 return {
103   ...createdPost._doc,
104   _id: createdPost._id.toString(),
105   createdAt: createdPost.createdAt.toISOString(),
106   updatedAt: createdPost.updatedAt.toISOString()
107 };
108 }
```

109 }

```
1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('http://localhost:8080/auth/status', {
28       headers: {
29         Authorization: `Bearer ${this.props.token}`
30       }
31     })
32     .then(res => {
33       if (res.status !== 200) {
34         throw new Error('Failed to fetch user status.');
35       }
36       return res.json();
37     })
38     .then(resData => {
39       this.setState({ status: resData.status });
40     })
41     .catch(this.catchError);
42
43     this.loadPosts();
44   }
45
46   loadPosts = direction => {
47     if (direction) {
48       this.setState({ postsLoading: true, posts: [] });
49     }
50     let page = this.state.postPage;
51     if (direction === 'next') {
52       page++;
53       this.setState({ postPage: page });
54     }
55     if (direction === 'previous') {
```

```
56     page--;
57     this.setState({ postPage: page });
58   }
59   fetch('http://localhost:8080/feed/posts?page=' + page, {
60     headers: {
61       Authorization: 'Bearer ' + this.props.token
62     }
63   })
64   .then(res => {
65     if (res.status !== 200) {
66       throw new Error('Failed to fetch posts.');
67     }
68     return res.json();
69   })
70   .then(resData => {
71     this.setState({
72       posts: resData.posts.map(post => {
73         return {
74           ...post,
75           imagePath: post.imageUrl
76         };
77       }),
78       totalPosts: resData.totalItems,
79       postsLoading: false
80     });
81   })
82   .catch(this.catchError);
83 };
84
85 statusUpdateHandler = event => {
86   event.preventDefault();
87   fetch('http://localhost:8080/auth/status', {
88     method: 'PATCH',
89     headers: {
90       Authorization: 'Bearer ' + this.props.token,
91       'Content-Type': 'application/json'
92     },
93     body: JSON.stringify({
94       status: this.state.status
95     })
96   })
97   .then(res => {
98     if (res.status !== 200 && res.status !== 201) {
99       throw new Error("Can't update status!");
100     }
101     return res.json();
102   })
103   .then(resData => {
104     console.log(resData);
105   })
106   .catch(this.catchError);
107 };
108
109 newPostHandler = () => {
110   this.setState({ isEditing: true });
111};
```

```
112
113 startEditPostHandler = postId => {
114   this.setState(prevState => {
115     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
116
117     return {
118       isEditing: true,
119       editPost: loadedPost
120     };
121   });
122 };
123
124 cancelEditHandler = () => {
125   this.setState({ isEditing: false, editPost: null });
126 };
127
128 finishEditHandler = postData => {
129   this.setState({
130     editLoading: true
131   });
132   const formData = new FormData();
133   formData.append('title', postData.title);
134   formData.append('content', postData.content);
135   formData.append('image', postData.image);
136
137   let graphqlQuery = {
138     query: `
139       mutation {
140         createPost(postInput: {title: "${postData.title}", content: "${{
141           postData.content
142         }", imageUrl: "some url"}) {
143           _id
144           title
145           content
146           imageUrl
147           creator {
148             name
149           }
150           createdAt
151         }
152       }
153     `
154   };
155
156   fetch('http://localhost:8080/graphql', {
157     method: 'POST',
158     body: JSON.stringify(graphqlQuery),
159     headers: {
160       Authorization: 'Bearer ' + this.props.token,
161       'Content-Type': 'application/json'
162     }
163   })
164     .then(res => {
165       return res.json();
166     })
167     .then(resData => {
```

```
168     if (resData.errors && resData.errors[0].status === 422) {
169         throw new Error(
170             "Validation failed. Make sure the email address isn't used yet!"
171         );
172     }
173     if (resData.errors) {
174         throw new Error('User login failed!');
175     }
176     console.log(resData);
177     const post = {
178         _id: resData.data.createPost._id,
179         title: resData.data.createPost.title,
180         content: resData.data.createPost.content,
181         creator: resData.data.createPost.creator,
182         createdAt: resData.data.createPost.createdAt
183     };
184     this.setState(prevState => {
185         return {
186             isEditing: false,
187             editPost: null,
188             editLoading: false
189         };
190     });
191 })
192 .catch(err => {
193     console.log(err);
194     this.setState({
195         isEditing: false,
196         editPost: null,
197         editLoading: false,
198         error: err
199     });
200 });
201 );
202
203 statusInputChangeHandler = (input, value) => {
204     this.setState({ status: value });
205 };
206
207 deletePostHandler = postId => {
208     this.setState({ postsLoading: true });
209     fetch('http://localhost:8080/feed/post/' + postId, {
210         method: 'DELETE',
211         headers: {
212             Authorization: 'Bearer ' + this.props.token
213         }
214     })
215     .then(res => {
216         if (res.status !== 200 && res.status !== 201) {
217             throw new Error('Deleting a post failed!');
218         }
219         return res.json();
220     })
221     .then(resData => {
222         console.log(resData);
223         this.loadPosts();
```

```
224 // this.setState(prevState => {
225 //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
226 //   return { posts: updatedPosts, postsLoading: false };
227 // });
228 })
229 .catch(err => {
230   console.log(err);
231   this.setState({ postsLoading: false });
232 });
233 };
234
235 errorHandler = () => {
236   this.setState({ error: null });
237 };
238
239 catchError = error => {
240   this.setState({ error: error });
241 };
242
243 render() {
244   return (
245     <Fragment>
246       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
247       <FeedEdit
248         editing={this.state.isEditing}
249         selectedPost={this.state.editPost}
250         loading={this.state.editLoading}
251         onCancelEdit={this.cancelEditHandler}
252         onFinishEdit={this.finishEditHandler}
253       />
254       <section className="feed__status">
255         <form onSubmit={this.statusUpdateHandler}>
256           <Input
257             type="text"
258             placeholder="Your status"
259             control="input"
260             onChange={this.statusInputChangeHandler}
261             value={this.state.status}
262           />
263           <Button mode="flat" type="submit">
264             Update
265           </Button>
266         </form>
267       </section>
268       <section className="feed__control">
269         <Button mode="raised" design="accent" onClick={this.newPostHandler}>
270           New Post
271         </Button>
272       </section>
273       <section className="feed">
274         {this.state.postsLoading && (
275           <div style={{ textAlign: 'center', marginTop: '2rem' }}>
276             <Loader />
277           </div>
278         )}
279         {this.state.posts.length <= 0 && !this.state.postsLoading ? (
```

```

280         <p style={{ textAlign: 'center' }}>No posts found.</p>
281     ) : null
282     {!this.state.postsLoading && (
283       <Paginator
284         onPrevious={this.loadPosts.bind(this, 'previous')}
285         onNext={this.loadPosts.bind(this, 'next')}
286         lastPage={Math.ceil(this.state.totalPosts / 2)}
287         currentPage={this.state.postPage}
288       >
289         {this.state.posts.map(post => (
290           <Post
291             key={post._id}
292             id={post._id}
293             author={post.creator.name}
294             date={new Date(post.createdAt).toLocaleDateString('en-US')}
295             title={post.title}
296             image={post.imageUrl}
297             content={post.content}
298             onStartEdit={this.startEditPostHandler.bind(this, post._id)}
299             onDelete={this.deletePostHandler.bind(this, post._id)}
300           />
301         )))
302       </Paginator>
303     )}
304   </section>
305 </Fragment>
306 );
307 }
308 }
309
310 export default Feed;
311

```

* Chapter 429: Adding A “Get Post” Query & Resolver

1. update
 - ./graphql/schema.js(b)
 - ./graphql/resolvers.js(b)

- i get not authenticated which is correct because i'm not sending a token here.

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9
10 module.exports = {
11   createUser: async function({ userInput }, req) {

```

```
12 //const email = args.userInput.email;
13 const errors = [];
14 if (!validator.isEmail(userInput.email)){
15     errors.push({ message: 'E-Mail is invalid.' });
16 }
17 if (validator.isEmpty(userInput.password) || !validator.minLength(userInput.password,
{ min: 5 })) {
18     errors.push({ message: 'Password Too Short!' });
19 }
20 if(errors.length > 0){
21     const error = new Error('Invalid Input.');
22     error.data = errors;
23     error.code = 422;
24     throw error;
25 }
26 const existingUser = await User.findOne({email: userInput.email});
27 if(existingUser) {
28     const error = new Error('User exists already!');
29     throw error;
30 }
31 const hashedPw = await bcrypt.hash(userInput.password, 12);
32 const user = new User({
33     email: userInput.email,
34     name: userInput.name,
35     password: hashedPw
36 });
37 const createdUser = await user.save();
38 return { ...createdUser._doc, _id: createdUser._id.toString() };
39 },
40 login: async function({ email, password }) {
41     const user = await User.findOne({email: email});
42     if(!user) {
43         const error = new Error('User not found.');
44         error.code = 404;
45         throw error;
46     }
47     const isEqual = await bcrypt.compare(password, user.password);
48     if(!isEqual) {
49         const error = new Error('Password is incorrect.');
50         error.code = 401;
51         throw error;
52     }
53     const token = jwt.sign({
54         userId: user._id.toString(),
55         email: user.email
56     },
57     'somesupersecretsecret',
58     { expiresIn: '1h' })
59     return { token: token, userId: user._id.toString() };
60 },
61 createPost: async function({ postInput }, req){
62     if(!req.isAuthenticated()){
63         const error = new Error('Not authenticated!');
64         error.code = 401;
65         throw error;
66 }
```

```
67  const errors = [];
68  if(
69    validator.isEmpty(postInput.title) ||
70    !validator.isLength(postInput.title, { min: 5 })
71  ) {
72    errors.push({ message: 'Title is invalid.' });
73  }
74  if(
75    validator.isEmpty(postInput.content) ||
76    !validator.isLength(postInput.content, { min: 5 })
77  ) {
78    errors.push({ message: 'Content is invalid.' });
79  }
80  if(errors.length > 0){
81    const error = new Error('Invalid Input.');
82    error.data = errors;
83    error.code = 422;
84    throw error;
85  }
86  const user = await User.findById(req.userId);
87  if(!user){
88    const error = new Error('Invalid user.');
89    error.code = 401;
90    throw error;
91  }
92  const post = new User({
93    title: postInput.title,
94    content: postInput.content,
95    imageUrl: postInput.imageUrl,
96    creator: user
97  });
98  const createdPost = await post.save();
99  user.posts.push(createdPost);
100 await user.save()
101 return {
102   ...createdPost._doc,
103   _id: createdPost._id.toString(),
104   createdAt: createdPost.createdAt.toISOString(),
105   updatedAt: createdPost.updatedAt.toISOString()
106 };
107 },
108 posts: async function(args, req){
109  if(!req.isAuthenticated()){
110    const error = new Error('Not authenticated!');
111    error.code = 401;
112    throw error;
113  }
114  const totalPosts = await Post.find().countDocuments();
115  const posts = await Post.find()
116    .sort({createdAt: -1})
117    .populate('creator');
118  return {posts: posts.map(p => {
119    return {
120      ...p._doc,
121      _id: p._id.toString(),
122      createdAt: p.createdAt.toISOString(),
123      updatedAt: p.updatedAt.toISOString()
124    };
125  });
126}
127
```

```
123     updatedAt: p.updatedAt.toISOString()
124   };
125 }, totalPosts: totalPosts};
126 }
127 }

1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 module.exports = buildSchema(`

6   type Post {
7     _id: ID!
8     title: String!
9     content: String!
10    imageUrl: String!
11    creator: User!
12    createdAt: String!
13    updatedAt: String!
14  }

15

16   type User {
17     _id: ID!
18     name: String!
19     email: String!
20     password: String
21     status: String!
22     posts: [Post!]!
23   }

24

25   type PostData {
26     posts: [Post!]!
27     totalPosts: Int!
28   }

29

30   type AuthData {
31     token: String!
32     userId: String!
33   }

34

35   input UserInputData {
36     email: String!
37     name: String!
38     password: String!
39   }

40

41   type RootQuery {
42     login(email: String!, password: String!): AuthData!
43     posts: PostData!
44   }

45

46   input PostInputData {
47     title: String!
48     content: String!
49     imageUrl: String!
50   }

51 
```

```

52   type RootMutation {
53     createUser(userInput: UserInputData): User!
54     createPost(postInput: PostInputData): Post!
55   }
56
57   schema {
58     query: RootQuery
59     mutation: RootMutation
60   }
61 `);

```

* Chapter 430: Sending “Create Post” And “Get Post” Queries

1. update
- ./src/pages/Feed/Feed.js(f)


```

1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('http://localhost:8080/auth/status', {
28       headers: {

```

```

29         Authorization: 'Bearer ' + this.props.token
30     }
31   })
32   .then(res => {
33     if (res.status !== 200) {
34       throw new Error('Failed to fetch user status.');
35     }
36     return res.json();
37   })
38   .then(resData => {
39     this.setState({ status: resData.status });
40   })
41   .catch(this.catchError);
42
43   this.loadPosts();
44 }
45
46 loadPosts = direction => {
47   if (direction) {
48     this.setState({ postsLoading: true, posts: [] });
49   }
50   let page = this.state.postPage;
51   if (direction === 'next') {
52     page++;
53     this.setState({ postPage: page });
54   }
55   if (direction === 'previous') {
56     page--;
57     this.setState({ postPage: page });
58   }
59   /*note that we see the entire flexibility of GraphQL
60   * all doing its job here
61   * we are fetching exactly that posts that we need
62   * we don't fetch things like create, email, updated field, imageUrl
63   * which don't need
64   * we only fetch what we need
65   * so we only fetch what we need
66   * and we use graphQL to its fullest potential here.
67   */
68   const graphqlQuery = {
69     query: `

70     {
71       posts {
72         posts {
73           _id
74           title
75           content
76           creator {
77             name
78           }
79           createdAt
80         }
81         totalPosts
82       }
83     }
84   `

```

```

85 }
86 fetch('http://localhost:8080/graphql', {
87   method: 'POST',
88   headers: {
89     Authorization: 'Bearer ' + this.props.token,
90     'Content-Type': 'application/json'
91   },
92   body: JSON.stringify(graphqlQuery)
93 })
94 .then(res => {
95   return res.json();
96 })
97 .then(resData => {
98   if (resData.errors) {
99     throw new Error('Fetching posts failed!');
100 }
101 this.setState({
102   posts: resData.data.posts.posts.map(post => {
103     return {
104       ...post,
105       imagePath: post.imageUrl
106     };
107   }),
108   totalPosts: resData.data.posts.totalPosts,
109   postsLoading: false
110 });
111 })
112 .catch(this.catchError);
113 };
114
115 statusUpdateHandler = event => {
116   event.preventDefault();
117   fetch('http://localhost:8080/auth/status', {
118     method: 'PATCH',
119     headers: {
120       Authorization: 'Bearer ' + this.props.token,
121       'Content-Type': 'application/json'
122     },
123     body: JSON.stringify({
124       status: this.state.status
125     })
126   })
127   .then(res => {
128     if (res.status !== 200 && res.status !== 201) {
129       throw new Error("Can't update status!");
130     }
131     return res.json();
132   })
133   .then(resData => {
134     console.log(resData);
135   })
136   .catch(this.catchError);
137 };
138
139 newPostHandler = () => {
140   this.setState({ isEditing: true });

```

```
141 };
142
143 startEditPostHandler = postId => {
144     this.setState(prevState => {
145         const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
146
147         return {
148             isEditing: true,
149             editPost: loadedPost
150         };
151     });
152 };
153
154 cancelEditHandler = () => {
155     this.setState({ isEditing: false, editPost: null });
156 };
157
158 finishEditHandler = postData => {
159     this.setState({
160         editLoading: true
161     });
162     const formData = new FormData();
163     formData.append('title', postData.title);
164     formData.append('content', postData.content);
165     formData.append('image', postData.image);
166
167     let graphqlQuery = {
168         query: `
169             mutation {
170                 createPost(postInput: {title: "${postData.title}", content: "${{
171                     postData.content
172                 }", imageUrl: "some url"}) {
173                     _id
174                     title
175                     content
176                     imageUrl
177                     creator {
178                         name
179                     }
180                     createdAt
181                 }
182             }
183         `,
184     };
185
186     fetch('http://localhost:8080/graphql', {
187         method: 'POST',
188         body: JSON.stringify(graphqlQuery),
189         headers: {
190             Authorization: 'Bearer ' + this.props.token,
191             'Content-Type': 'application/json'
192         }
193     })
194     .then(res => {
195         return res.json();
196     })

```

```
197     .then(resData => {
198       if (resData.errors && resData.errors[0].status === 422) {
199         throw new Error(
200           "Validation failed. Make sure the email address isn't used yet!"
201         );
202       }
203       if (resData.errors) {
204         throw new Error('User login failed!');
205       }
206       console.log(resData);
207       const post = {
208         _id: resData.data.createPost._id,
209         title: resData.data.createPost.title,
210         content: resData.data.createPost.content,
211         creator: resData.data.createPost.creator,
212         createdAt: resData.data.createPost.createdAt
213       };
214       this.setState(prevState => {
215         let updatedPosts = [...prevState.posts];
216         if (prevState.editPost) {
217           const postIndex = prevState.posts.findIndex(
218             p => p._id === prevState.editPost._id
219           );
220           updatedPosts[postIndex] = post;
221         } else {
222           updatedPosts.unshift(post);
223         }
224         return {
225           posts: updatedPosts,
226           isEditing: false,
227           editPost: null,
228           editLoading: false
229         };
230       });
231     })
232     .catch(err => {
233       console.log(err);
234       this.setState({
235         isEditing: false,
236         editPost: null,
237         editLoading: false,
238         error: err
239       });
240     });
241   );
242
243   statusInputChangeHandler = (input, value) => {
244     this.setState({ status: value });
245   };
246
247   deletePostHandler = postId => {
248     this.setState({ postsLoading: true });
249     fetch('http://localhost:8080/feed/post/' + postId, {
250       method: 'DELETE',
251       headers: {
252         Authorization: 'Bearer ' + this.props.token
```

```
253     }
254   })
255   .then(res => {
256     if (res.status !== 200 && res.status !== 201) {
257       throw new Error('Deleting a post failed!');
258     }
259     return res.json();
260   })
261   .then(resData => {
262     console.log(resData);
263     this.loadPosts();
264     // this.setState(prevState => {
265     //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
266     //   return { posts: updatedPosts, postsLoading: false };
267     // });
268   })
269   .catch(err => {
270     console.log(err);
271     this.setState({ postsLoading: false });
272   });
273 }
274
275 errorHandler = () => {
276   this.setState({ error: null });
277 };
278
279 catchError = error => {
280   this.setState({ error: error });
281 };
282
283 render() {
284   return (
285     <Fragment>
286       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
287       <FeedEdit
288         editing={this.state.isEditing}
289         selectedPost={this.state.editPost}
290         loading={this.state.editLoading}
291         onCancelEdit={this.cancelEditHandler}
292         onFinishEdit={this.finishEditHandler}
293       />
294       <section className="feed__status">
295         <form onSubmit={this.statusUpdateHandler}>
296           <Input
297             type="text"
298             placeholder="Your status"
299             control="input"
300             onChange={this.statusInputChangeHandler}
301             value={this.state.status}
302           />
303           <Button mode="flat" type="submit">
304             Update
305           </Button>
306         </form>
307       </section>
308       <section className="feed__control">
```

```

309         <Button mode="raised" design="accent" onClick={this.newPostHandler}>
310             New Post
311         </Button>
312     </section>
313     <section className="feed">
314         {this.state.postsLoading &&
315             <div style={{ textAlign: 'center', marginTop: '2rem' }}>
316                 <Loader />
317             </div>
318         )}
319         {this.state.posts.length <= 0 && !this.state.postsLoading ? (
320             <p style={{ textAlign: 'center' }}>No posts found.</p>
321         ) : null}
322         {!this.state.postsLoading &&
323             <Paginator
324                 onPrevious={this.loadPosts.bind(this, 'previous')}
325                 onNext={this.loadPosts.bind(this, 'next')}
326                 lastPage={Math.ceil(this.state.totalPosts / 2)}
327                 currentPage={this.state.postPage}
328             >
329                 {this.state.posts.map(post => (
330                     <Post
331                         key={post._id}
332                         id={post._id}
333                         author={post.creator.name}
334                         date={new Date(post.createdAt).toLocaleDateString('en-US')}
335                         title={post.title}
336                         image={post.imageUrl}
337                         content={post.content}
338                         onStartEdit={this.startEditPostHandler.bind(this, post._id)}
339                         onDelete={this.deletePostHandler.bind(this, post._id)}
340                     />
341                 )))
342             </Paginator>
343         )}
344     </section>
345   </Fragment>
346 );
347 }
348 }
349
350 export default Feed;
351

```

* Chapter 431: Adding Pagination

1. update
 - ./graphql/schema.js(b)
 - ./graphql/resolvers.js(b)
 - ./src/pages/Feed/Feed.js(f)


```
1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 module.exports = buildSchema(`

6   type Post {
7     _id: ID!
8     title: String!
9     content: String!
10    imageUrl: String!
11    creator: User!
12    createdAt: String!
13    updatedAt: String!
14  }

15

16   type User {
17     _id: ID!
18     name: String!
19     email: String!
20     password: String
21     status: String!
22     posts: [Post!]!
23   }

24

25   type PostData {
26     posts: [Post!]!
27     totalPosts: Int!
28   }

29

30   type AuthData {
31     token: String!
32     userId: String!
33   }

34

35   input UserInputData {
36     email: String!
37     name: String!
38     password: String!
39   }

40

41   type RootQuery {
42     login(email: String!, password: String!): AuthData!
43     posts(page: Int!): PostData!
44 }
```

```

45
46   input PostInputData {
47     title: String!
48     content: String!
49     imageUrl: String!
50   }
51
52   type RootMutation {
53     createUser(userInput: UserInputData): User!
54     createPost(postInput: PostInputData): Post!
55   }
56
57   schema {
58     query: RootQuery
59     mutation: RootMutation
60   }
61 `);

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9
10 module.exports = {
11   createUser: async function({ userInput }, req) {
12     //const email = args.userInput.email;
13     const errors = [];
14     if (!validator.isEmail(userInput.email)){
15       errors.push({ message: 'E-Mail is invalid.' });
16     }
17     if (validator.isEmpty(userInput.password) || !validator.minLength(userInput.password,
18 { min: 5 })) {
19       errors.push({message: 'Password Too Short!'});
20     }
21     if(errors.length > 0){
22       const error = new Error('Invalid Input.');
23       error.data = errors;
24       error.code = 422;
25       throw error;
26     }
27     const existingUser = await User.findOne({email: userInput.email});
28     if(existingUser) {
29       const error = new Error('User exists already!');
30       throw error;
31     }
32     const hashedPw = await bcrypt.hash(userInput.password, 12);
33     const user = new User({
34       email: userInput.email,
35       name: userInput.name,
36       password: hashedPw
37     });
38     const createdUser = await user.save();
39     return { ...createdUser._doc, _id: createdUser._id.toString() };

```

```
39 },
40 login: async function({ email, password }) {
41     const user = await User.findOne({email: email});
42     if(!user) {
43         const error = new Error('User not found.');
44         error.code = 404;
45         throw error;
46     }
47     const isEqual = await bcrypt.compare(password, user.password);
48     if(!isEqual) {
49         const error = new Error('Password is incorrect.');
50         error.code = 401;
51         throw error;
52     }
53     const token = jwt.sign({
54         userId: user._id.toString(),
55         email: user.email
56     },
57     'somesupersecretsecret',
58     { expiresIn: '1h' })
59     return { token: token, userId: user._id.toString() };
60 },
61 createPost: async function({ postInput }, req){
62     if(!req.isAuthenticated){
63         const error = new Error('Not authenticated!');
64         error.code = 401;
65         throw error;
66     }
67     const errors = [];
68     if(
69         validator.isEmpty(postInput.title) ||
70         !validator.isLength(postInput.title, { min: 5 })
71     ) {
72         errors.push({ message: 'Title is invalid.' });
73     }
74     if(
75         validator.isEmpty(postInput.content) ||
76         !validator.isLength(postInput.content, { min: 5 })
77     ) {
78         errors.push({ message: 'Content is invalid.' });
79     }
80     if(errors.length > 0){
81         const error = new Error('Invalid Input.');
82         error.data = errors;
83         error.code = 422;
84         throw error;
85     }
86     const user = await User.findById(req.userId);
87     if(!user){
88         const error = new Error('Invalid user.');
89         error.code = 401;
90         throw error;
91     }
92     const post = new User({
93         title: postInput.title,
94         content: postInput.content,
```

```

95     imageUrl: postInput.imageUrl,
96     creator: user
97   });
98   const createdPost = await post.save();
99   user.posts.push(createdPost);
100  await user.save()
101  return {
102    ...createdPost._doc,
103    _id: createdPost._id.toString(),
104    createdAt: createdPost.createdAt.toISOString(),
105    updatedAt: createdPost.updatedAt.toISOString()
106  };
107 },
108 posts: async function({page}, req){
109   if(!req.isAuthenticated){
110     const error = new Error('Not authenticated!');
111     error.code = 401;
112     throw error;
113   }
114   if(!page) {
115     page = 1;
116   }
117   const perPage = 2;
118   const totalPosts = await Post.find().countDocuments();
119   const posts = await Post.find()
120     .sort({createdAt: -1})
121     /**if you are on page 2,
122      * you have 2 - 1 = 1
123      * if you skip 1 * 2,
124      * so the first 2 items were on page 1
125      */
126     .skip((page - 1) * perPage)
127     .limit(perPage)
128     .populate('creator');
129   return {posts: posts.map(p => {
130     return {
131       ...p._doc,
132       _id: p._id.toString(),
133       createdAt: p.createdAt.toISOString(),
134       updatedAt: p.updatedAt.toISOString()
135     };
136   }), totalPosts};
137 }
138 }

```

```

1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';

```

```

13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26 componentDidMount() {
27   fetch('http://localhost:8080/auth/status', {
28     headers: {
29       Authorization: 'Bearer ' + this.props.token
30     }
31   })
32     .then(res => {
33       if (res.status !== 200) {
34         throw new Error('Failed to fetch user status.');
35       }
36       return res.json();
37     })
38     .then(resData => {
39       this.setState({ status: resData.status });
40     })
41     .catch(this.catchError);
42
43   this.loadPosts();
44 }
45
46 loadPosts = direction => {
47   if (direction) {
48     this.setState({ postsLoading: true, posts: [] });
49   }
50   let page = this.state.postPage;
51   if (direction === 'next') {
52     page++;
53     this.setState({ postPage: page });
54   }
55   if (direction === 'previous') {
56     page--;
57     this.setState({ postPage: page });
58   }
59   /**note that we see the entire flexibility of GraphQL
60   * all doing its job here
61   * we are fetching exactly that posts that we need
62   * we don't fetch things like create, email, updated field, imageUrl
63   * which don't need
64   * we only fetch what we need
65   * so we only fetch what we need
66   * and we use graphQL to its fullest potential here.
67   */
68   const graphqlQuery = {

```

```
69 query: ` 
70   {
71     posts(page: $(page)) {
72       posts {
73         _id
74         title
75         content
76         creator {
77           name
78         }
79         createdAt
80       }
81       totalPosts
82     }
83   }
84 `
85 }
86 fetch('http://localhost:8080/graphql', {
87   method: 'POST',
88   headers: {
89     Authorization: 'Bearer ' + this.props.token,
90     'Content-Type': 'application/json'
91   },
92   body: JSON.stringify(graphqlQuery)
93 })
94 .then(res => {
95   return res.json();
96 })
97 .then(resData => {
98   if (resData.errors) {
99     throw new Error('Fetching posts failed!');
100 }
101 this.setState({
102   posts: resData.data.posts.posts.map(post => {
103     return {
104       ...post,
105       imagePath: post.imageUrl
106     };
107   }),
108   totalPosts: resData.data.posts.totalPosts,
109   postsLoading: false
110 });
111 })
112 .catch(this.catchError);
113 };
114
115 statusUpdateHandler = event => {
116   event.preventDefault();
117   fetch('http://localhost:8080/auth/status', {
118     method: 'PATCH',
119     headers: {
120       Authorization: 'Bearer ' + this.props.token,
121       'Content-Type': 'application/json'
122     },
123     body: JSON.stringify({
124       status: this.state.status

```

```
125     })
126   })
127   .then(res => {
128     if (res.status !== 200 && res.status !== 201) {
129       throw new Error("Can't update status!");
130     }
131     return res.json();
132   })
133   .then(resData => {
134     console.log(resData);
135   })
136   .catch(this.catchError);
137 };
138
139 newPostHandler = () => {
140   this.setState({ isEditing: true });
141 };
142
143 startEditPostHandler = postId => {
144   this.setState(prevState => {
145     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
146
147     return {
148       isEditing: true,
149       editPost: loadedPost
150     };
151   });
152 };
153
154 cancelEditHandler = () => {
155   this.setState({ isEditing: false, editPost: null });
156 };
157
158 finishEditHandler = postData => {
159   this.setState({
160     editLoading: true
161   });
162   const formData = new FormData();
163   formData.append('title', postData.title);
164   formData.append('content', postData.content);
165   formData.append('image', postData.image);
166
167   let graphqlQuery = {
168     query: `
169       mutation {
170         createPost(postInput: {title: "${postData.title}", content: "${{
171           postData.content
172         }", imageUrl: "some url"}) {
173           _id
174             title
175             content
176             imageUrl
177             creator {
178               name
179             }
180             createdAt

```

```

181
182     }
183   }
184 };
185
186 fetch('http://localhost:8080/graphql', {
187   method: 'POST',
188   body: JSON.stringify(graphqlQuery),
189   headers: {
190     Authorization: 'Bearer ' + this.props.token,
191     'Content-Type': 'application/json'
192   }
193 })
194 .then(res => {
195   return res.json();
196 })
197 .then(resData => {
198   if (resData.errors && resData.errors[0].status === 422) {
199     throw new Error(
200       "Validation failed. Make sure the email address isn't used yet!"
201     );
202   }
203   if (resData.errors) {
204     throw new Error('User login failed!');
205   }
206   console.log(resData);
207   const post = {
208     _id: resData.data.createPost._id,
209     title: resData.data.createPost.title,
210     content: resData.data.createPost.content,
211     creator: resData.data.createPost.creator,
212     createdAt: resData.data.createPost.createdAt
213   };
214   this.setState(prevState => {
215     let updatedPosts = [...prevState.posts];
216     if (prevState.editPost) {
217       const postIndex = prevState.posts.findIndex(
218         p => p._id === prevState.editPost._id
219       );
220       updatedPosts[postIndex] = post;
221     } else {
222       /**this will remove one element
223        * and add a new one at the beginning.
224        */
225       updatedPosts.pop();
226       updatedPosts.unshift(post);
227     }
228     return {
229       posts: updatedPosts,
230       isEditing: false,
231       editPost: null,
232       editLoading: false
233     };
234   });
235 }
236 .catch(err => {

```

```
237     console.log(err);
238     this.setState({
239       |   isEditing: false,
240       |   editPost: null,
241       |   editLoading: false,
242       |   error: err
243     });
244   });
245 }
246
247 statusInputChangeHandler = (input, value) => {
248   this.setState({ status: value });
249 };
250
251 deletePostHandler = postId => {
252   this.setState({ postsLoading: true });
253   fetch('http://localhost:8080/feed/post/' + postId, {
254     method: 'DELETE',
255     headers: {
256       Authorization: 'Bearer ' + this.props.token
257     }
258   })
259   .then(res => {
260     if (res.status !== 200 && res.status !== 201) {
261       |   throw new Error('Deleting a post failed!');
262     }
263     return res.json();
264   })
265   .then(resData => {
266     console.log(resData);
267     this.loadPosts();
268     // this.setState(prevState => {
269     //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
270     //   return { posts: updatedPosts, postsLoading: false };
271     // });
272   })
273   .catch(err => {
274     console.log(err);
275     this.setState({ postsLoading: false });
276   });
277 }
278
279 errorHandler = () => {
280   this.setState({ error: null });
281 };
282
283 catchError = error => {
284   this.setState({ error: error });
285 };
286
287 render() {
288   return (
289     <Fragment>
290       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
291       <FeedEdit
292         |   editing={this.state.isEditing}
```

```
293     selectedPost={this.state.editPost}
294     loading={this.state.editLoading}
295     onCancelEdit={this.cancelEditHandler}
296     onFinishEdit={this.finishEditHandler}
297   />
298   <section className="feed__status">
299     <form onSubmit={this.statusUpdateHandler}>
300       <Input
301         type="text"
302         placeholder="Your status"
303         control="input"
304         onChange={this.statusInputChangeHandler}
305         value={this.state.status}
306       />
307       <Button mode="flat" type="submit">
308         Update
309       </Button>
310     </form>
311   </section>
312   <section className="feed__control">
313     <Button mode="raised" design="accent" onClick={this.newPostHandler}>
314       New Post
315     </Button>
316   </section>
317   <section className="feed">
318     {this.state.postsLoading && (
319       <div style={{ textAlign: 'center', marginTop: '2rem' }}>
320         <Loader />
321       </div>
322     )}
323     {this.state.posts.length <= 0 && !this.state.postsLoading ? (
324       <p style={{ textAlign: 'center' }}>No posts found.</p>
325     ) : null}
326     {!this.state.postsLoading && (
327       <Paginator
328         onPrevious={this.loadPosts.bind(this, 'previous')}
329         onNext={this.loadPosts.bind(this, 'next')}
330         lastPage={Math.ceil(this.state.totalPosts / 2)}
331         currentPage={this.state.postPage}
332       >
333         {this.state.posts.map(post => (
334           <Post
335             key={post._id}
336             id={post._id}
337             author={post.creator.name}
338             date={new Date(post.createdAt).toLocaleDateString('en-US')}
339             title={post.title}
340             image={post.imageUrl}
341             content={post.content}
342             onStartEdit={this.startEditPostHandler.bind(this, post._id)}
343             onDelete={this.deletePostHandler.bind(this, post._id)}
344           />
345         )))
346       </Paginator>
347     )}
348   </section>
```

```

349     .... </Fragment>
350   );
351 }
352 }
353
354 export default Feed;
355

```

* Chapter 432: Uploading Images

1. update
 - app.js(b)
 - ./src/pages/Feed/Feed.js(f)


```

1 //app.js(b)
2
3 const path = require('path');
4 const fs = require('fs');
5
6 const express = require('express');
7 const bodyParser = require('body-parser');
8 const mongoose = require('mongoose');
9 const multer = require('multer');
10
11 const graphqlHttp = require('express-graphql');
12
13 const graphSchema = require('./graphql/schema');
14 const graphResolver = require('./graphql/resolvers');
15 const auth = require('./middleware/auth');
16
17 const app = express();
18
19 const fileStorage = multer.diskStorage({
20   destination: (req, file, cb) => {
21     cb(null, 'images');
22   },
23   filename: (req, file, cb) => {
24     cb(null, new Date().toISOString() + '-' + file.originalname);
25   }
26 });
27
28 const fileFilter = (req, file, cb) => {
29   if(
30     file.mimetype === 'image/png' ||
31     file.mimetype === 'image/jpg' ||
32     file.mimetype === 'image/jpeg'

```

```

33     ) {
34       cb(null, true);
35     } else {
36       cb(null, false);
37     }
38   }
39
40 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
41 app.use(bodyParser.json()); // application/json
42 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
43 app.use('/images', express.static(path.join(__dirname, 'images')));
44
45 app.use((req, res, next) => {
46   res.setHeader('Access-Control-Allow-Origin', '*');
47   res.setHeader(
48     'Access-Control-Allow-Methods',
49     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
50   );
51   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
52   if(req.method === 'OPTIONS') {
53     return res.sendStatus(200);
54   }
55   next();
56 });
57
58 app.use(auth);
59
60 app.put('/post-image', (req, res, next) => {
61   if(!req.isAuthenticated){
62     throw new Error('Not authenticated!');
63   }
64   if(!req.file){
65     return res.status(200).json({message: 'No file provided!'});
66   }
67   if(req.body.oldPath) {
68     /**i will check for the existence of a body field
69      * which is named 'oldPath'
70      * which means that an oldPath was passed
71      * with the incoming request in which case i wanna clear my old image
72      * and i will pass in the oldPath
73      * because we added a new image
74      */
75     clearImage(req.body.oldPath);
76   }
77   /**'req.file.path' is the path where multer stored the image
78   * this is the path we can use the front-end
79 */
80   return res.status(201).json({message: 'File stored.', filePath: req.file.path});
81 });
82
83 app.use('/graphql', graphqlHttp({
84   schema: graphSchema,
85   rootValue: graphResolver,
86   graphiql: true,
87   customFormatErrorFn(err) {
88     if (!err.originalError) {

```

```

89         return err;
90     }
91     const data = err.originalError.data;
92     const message = err.message || 'An error occurred.';
93     const code = err.originalError.code || 500
94     return { message: message, status: code, data: data };
95   }
96 }
97 );
98
99 app.use((error, req, res, next) => {
100   console.log(error);
101   const status = error.statusCode || 500;
102   const message = error.message;
103   const data = error.data;
104   res.status(status).json({ message: message, data: data });
105 });
106
107 mongoose
108   .connect(
109     'mongodb+srv://maximilian:rldnj12@cluster0-z3v1k.mongodb.net/message?retryWrites=true'
110   )
111   .then(result => {
112     app.listen(8080);
113   })
114   .catch(err => console.log(err));
115
116   const clearImage = filePath => {
117     filePath = path.join(__dirname, '...', filePath);
118     fs.unlink(filePath, err => console.log(err));
119   };

```

```

1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25

```

```
26 componentDidMount() {
27   fetch('http://localhost:8080/auth/status', {
28     headers: {
29       Authorization: 'Bearer ' + this.props.token
30     }
31   })
32   .then(res => {
33     if (res.status !== 200) {
34       throw new Error('Failed to fetch user status.');
35     }
36     return res.json();
37   })
38   .then(resData => {
39     this.setState({ status: resData.status });
40   })
41   .catch(this.catchError);
42
43   this.loadPosts();
44 }
45
46 loadPosts = direction => {
47   if (direction) {
48     this.setState({ postsLoading: true, posts: [] });
49   }
50   let page = this.state.postPage;
51   if (direction === 'next') {
52     page++;
53     this.setState({ postPage: page });
54   }
55   if (direction === 'previous') {
56     page--;
57     this.setState({ postPage: page });
58   }
59   const graphqlQuery = {
60     query: `
61       {
62         posts(page: ${page}) {
63           posts {
64             _id
65             title
66             content
67             imageUrl
68             creator {
69               name
70             }
71             createdAt
72           }
73           totalPosts
74         }
75       }
76     `;
77   };
78   fetch('http://localhost:8080/graphQL', {
79     method: 'POST',
80     headers: {
81       Authorization: 'Bearer ' + this.props.token,
```

```
82     'Content-Type': 'application/json'
83   },
84   body: JSON.stringify(graphqlQuery)
85 )
86 .then(res => {
87   return res.json();
88 })
89 .then(resData => {
90   if (resData.errors) {
91     throw new Error('Fetching posts failed!');
92   }
93   this.setState({
94     posts: resData.data.posts.posts.map(post => {
95       return {
96         ...post,
97         imagePath: post.imageUrl
98       };
99     }),
100    totalPosts: resData.data.posts.totalPosts,
101    postsLoading: false
102  });
103 })
104 .catch(this.catchError);
105 };
106
107 statusUpdateHandler = event => {
108   event.preventDefault();
109   fetch('http://localhost:8080/auth/status', {
110     method: 'PATCH',
111     headers: {
112       Authorization: 'Bearer ' + this.props.token,
113       'Content-Type': 'application/json'
114     },
115     body: JSON.stringify({
116       status: this.state.status
117     })
118   })
119   .then(res => {
120     if (res.status !== 200 && res.status !== 201) {
121       throw new Error("Can't update status!");
122     }
123     return res.json();
124   })
125   .then(resData => {
126     console.log(resData);
127   })
128   .catch(this.catchError);
129 };
130
131 newPostHandler = () => {
132   this.setState({ isEditing: true });
133 };
134
135 startEditPostHandler = postId => {
136   this.setState(prevState => {
137     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
```

```
138
139     return {
140         isEditing: true,
141         editPost: loadedPost
142     };
143 );
144 };
145
146 cancelEditHandler = () => {
147     this.setState({ isEditing: false, editPost: null });
148 };
149
150 finishEditHandler = postData => {
151     this.setState({
152         editLoading: true
153     });
154     const formData = new FormData();
155     formData.append('image', postData.image);
156     if(this.state.editPost) {
157         formData.append('oldPath', this.state.editPost.imagePath);
158     }
159     fetch('http://localhost:8080/post-image', {
160         /**method is 'PUT'
161          * because on my app.js(b)
162          * and i defined this to handle 'PUT' request
163          */
164         method: 'PUT',
165         headers: {
166             /**i should not set the content-type to application/json
167              * otherwise it will be parsed as data
168              * and that doesn't work for my binary data here.
169              */
170             Authorization: 'Bearer ' + this.props.token
171         },
172         body: formData
173     })
174     .then(res => res.json())
175     .then(fileResData => {
176         /**'filePath' is from app.js(b) */
177         const imageUrl = fileResData.filePath;
178         let graphqlQuery = {
179             query: `
180                 mutation {
181                     createPost(postInput: {title: "${postData.title}", content: "${postData.content
182                         }", imageUrl: "${imageUrl}"}) {
183                         _id
184                         title
185                         content
186                         imageUrl
187                         creator {
188                             name
189                         }
190                         createdAt
191                     }
192                 }
193             
```

```
194
195    );
196    return fetch('http://localhost:8080/graphql', {
197      method: 'POST',
198      body: JSON.stringify(graphqlQuery),
199      headers: {
200        Authorization: 'Bearer ' + this.props.token,
201        'Content-Type': 'application/json'
202      }
203    })
204  ).then(res => {
205    return res.json();
206  })
207  .then(resData => {
208    if (resData.errors && resData.errors[0].status === 422) {
209      throw new Error(
210        "Validation failed. Make sure the email address isn't used yet!"
211      );
212    }
213    if (resData.errors) {
214      throw new Error('User login failed!');
215    }
216    console.log(resData);
217    const post = {
218      _id: resData.data.createPost._id,
219      title: resData.data.createPost.title,
220      content: resData.data.createPost.content,
221      creator: resData.data.createPost.creator,
222      createdAt: resData.data.createPost.createdAt,
223      imagePath: resData.data.createPost.imageUrl
224    };
225    this.setState(prevState => {
226      let updatedPosts = [...prevState.posts];
227      if (prevState.editPost) {
228        const postIndex = prevState.posts.findIndex(
229          p => p._id === prevState.editPost._id
230        );
231        updatedPosts[postIndex] = post;
232      } else {
233        updatedPosts.pop();
234        updatedPosts.unshift(post);
235      }
236      return {
237        posts: updatedPosts,
238        isEditing: false,
239        editPost: null,
240        editLoading: false
241      };
242    });
243  })
244  .catch(err => {
245    console.log(err);
246    this.setState({
247      isEditing: false,
248      editPost: null,
249      editLoading: false,
```

```
250     error: err
251   });
252 });
253 };
254
255 statusInputChangeHandler = (input, value) => {
256   this.setState({ status: value });
257 };
258
259 deletePostHandler = postId => {
260   this.setState({ postsLoading: true });
261   fetch('http://localhost:8080/feed/post/' + postId, {
262     method: 'DELETE',
263     headers: {
264       Authorization: 'Bearer ' + this.props.token
265     }
266   })
267   .then(res => {
268     if (res.status !== 200 && res.status !== 201) {
269       throw new Error('Deleting a post failed!');
270     }
271     return res.json();
272   })
273   .then(resData => {
274     console.log(resData);
275     this.loadPosts();
276     // this.setState(prevState => {
277     //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
278     //   return { posts: updatedPosts, postsLoading: false };
279     // });
280   })
281   .catch(err => {
282     console.log(err);
283     this.setState({ postsLoading: false });
284   });
285 };
286
287 errorHandler = () => {
288   this.setState({ error: null });
289 };
290
291 catchError = error => {
292   this.setState({ error: error });
293 };
294
295 render() {
296   return (
297     <Fragment>
298       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
299       <FeedEdit
300         editing={this.state.isEditing}
301         selectedPost={this.state.editPost}
302         loading={this.state.editLoading}
303         onCancelEdit={this.cancelEditHandler}
304         onFinishEdit={this.finishEditHandler}
305       />

```

```
306     <section className="feed__status">
307         <form onSubmit={this.statusUpdateHandler}>
308             <Input
309                 type="text"
310                 placeholder="Your status"
311                 control="input"
312                 onChange={this.statusInputChangeHandler}
313                 value={this.state.status}
314             />
315             <Button mode="flat" type="submit">
316                 Update
317             </Button>
318         </form>
319     </section>
320     <section className="feed__control">
321         <Button mode="raised" design="accent" onClick={this.newPostHandler}>
322             New Post
323         </Button>
324     </section>
325     <section className="feed">
326         {this.state.postsLoading && (
327             <div style={{ textAlign: 'center', marginTop: '2rem' }}>
328                 <Loader />
329             </div>
330         )}
331         {this.state.posts.length <= 0 && !this.state.postsLoading ? (
332             <p style={{ textAlign: 'center' }}>No posts found.</p>
333         ) : null}
334         {!this.state.postsLoading && (
335             <Paginator
336                 onPrevious={this.loadPosts.bind(this, 'previous')}
337                 onNext={this.loadPosts.bind(this, 'next')}
338                 lastPage={Math.ceil(this.state.totalPosts / 2)}
339                 currentPage={this.state.postPage}
340             >
341                 {this.state.posts.map(post => (
342                     <Post
343                         key={post._id}
344                         id={post._id}
345                         author={post.creator.name}
346                         date={new Date(post.createdAt).toLocaleDateString('en-US')}
347                         title={post.title}
348                         image={post.imageUrl}
349                         content={post.content}
350                         onStartEdit={this.startEditPostHandler.bind(this, post._id)}
351                         onDelete={this.deletePostHandler.bind(this, post._id)}
352                     />
353                 )))
354             </Paginator>
355         )}
356     </section>
357 </Fragment>
358 );
359 }
360 }
```

```
362 export default Feed;  
363
```

* Chapter 434: Viewing A Single Post

- ./graphql/schema.js(b)
- ./graphql/resolvers.js(b)
- ./src/pages/Feed/SinglePost/SinglePost.js(f)


```
1 //./graphql/schema.js(b)  
2  
3 const { buildSchema } = require('graphql');  
4  
5 module.exports = buildSchema(`  
6   type Post {  
7     _id: ID!  
8     title: String!  
9     content: String!  
10    imageUrl: String!  
11    creator: User!  
12    createdAt: String!  
13    updatedAt: String!  
14  }  
15  
16  type User {  
17    _id: ID!  
18    name: String!  
19    email: String!  
20    password: String  
21    status: String!  
22    posts: [Post!]!  
23  }  
24  
25  type AuthData {  
26    token: String!  
27    userId: String!  
28  }  
29  
30  type PostData {  
31    posts: [Post!]!  
32    totalPosts: Int!  
33  }  
34  
35  input UserInputData {  
36    email: String!  
37    name: String!  
38    password: String!  
39  }  
40  
41  input PostInputData {
```

```

42     title: String!
43     content: String!
44     imageUrl: String!
45   }
46
47   type RootQuery {
48     login(email: String!, password: String!): AuthData!
49     posts(page: Int): PostData!
50     post(id: ID!): Post!
51   }
52
53   type RootMutation {
54     createUser(userInput: UserInputData): User!
55     createPost(postInput: PostInputData): Post!
56   }
57
58   schema {
59     query: RootQuery
60     mutation: RootMutation
61   }
62 `);
63

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9
10 module.exports = {
11   createUser: async function({ userInput }, req) {
12     // const email = args.userInput.email;
13     const errors = [];
14     if (!validator.isEmail(userInput.email)) {
15       errors.push({ message: 'E-Mail is invalid.' });
16     }
17     if (
18       validator.isEmpty(userInput.password) ||
19       !validator.minLength(userInput.password, { min: 5 })
20     ) {
21       errors.push({ message: 'Password too short!' });
22     }
23     if (errors.length > 0) {
24       const error = new Error('Invalid input.');
25       error.data = errors;
26       error.code = 422;
27       throw error;
28     }
29     const existingUser = await User.findOne({ email: userInput.email });
30     if (existingUser) {
31       const error = new Error('User exists already!');
32       throw error;
33     }
34     const hashedPw = await bcrypt.hash(userInput.password, 12);

```

```
35  const user = new User({
36    email: userInput.email,
37    name: userInput.name,
38    password: hashedPw
39  });
40  const createdUser = await user.save();
41  return { ...createdUser._doc, _id: createdUser._id.toString() };
42},
43 login: async function({ email, password }) {
44  const user = await User.findOne({ email: email });
45  if (!user) {
46    const error = new Error('User not found.');
47    error.code = 401;
48    throw error;
49  }
50  const isEqual = await bcrypt.compare(password, user.password);
51  if (!isEqual) {
52    const error = new Error('Password is incorrect.');
53    error.code = 401;
54    throw error;
55  }
56  const token = jwt.sign(
57    {
58      userId: user._id.toString(),
59      email: user.email
60    },
61    'somesupersecretsecret',
62    { expiresIn: '1h' }
63  );
64  return { token: token, userId: user._id.toString() };
65},
66 createPost: async function({ postInput }, req) {
67  if (!req.isAuthenticated()) {
68    const error = new Error('Not authenticated!');
69    error.code = 401;
70    throw error;
71  }
72  const errors = [];
73  if (
74    validator.isEmpty(postInput.title) ||
75    !validator.isLength(postInput.title, { min: 5 })
76  ) {
77    errors.push({ message: 'Title is invalid.' });
78  }
79  if (
80    validator.isEmpty(postInput.content) ||
81    !validator.isLength(postInput.content, { min: 5 })
82  ) {
83    errors.push({ message: 'Content is invalid.' });
84  }
85  if (errors.length > 0) {
86    const error = new Error('Invalid input.');
87    error.data = errors;
88    error.code = 422;
89    throw error;
90  }
```

```
91  const user = await User.findById(req.userId);
92  if (!user) {
93    const error = new Error('Invalid user.');
94    error.code = 401;
95    throw error;
96  }
97  const post = new Post({
98    title: postInput.title,
99    content: postInput.content,
100   imageUrl: postInput.imageUrl,
101   creator: user
102 });
103 const createdPost = await post.save();
104 user.posts.push(createdPost);
105 await user.save();
106 return {
107   ...createdPost._doc,
108   _id: createdPost._id.toString(),
109   createdAt: createdPost.createdAt.toISOString(),
110   updatedAt: createdPost.updatedAt.toISOString()
111 };
112 },
113 posts: async function({ page }, req) {
114  if (!req.isAuthenticated) {
115    const error = new Error('Not authenticated!');
116    error.code = 401;
117    throw error;
118  }
119  if (!page) {
120    page = 1;
121  }
122  const perPage = 2;
123  const totalPosts = await Post.find().countDocuments();
124  const posts = await Post.find()
125    .sort({ createdAt: -1 })
126    .skip((page - 1) * perPage)
127    .limit(perPage)
128    .populate('creator');
129  return {
130    posts: posts.map(p => {
131      return {
132        ...p._doc,
133        _id: p._id.toString(),
134        createdAt: p.createdAt.toISOString(),
135        updatedAt: p.updatedAt.toISOString()
136      };
137    }),
138    totalPosts: totalPosts
139  };
140 },
141 post: async function({ id }, req) {
142  if (!req.isAuthenticated) {
143    const error = new Error('Not authenticated!');
144    error.code = 401;
145    throw error;
146  }
```

```

147 const post = await Post.findById(id).populate('creator');
148 if (!post) {
149   const error = new Error('No post found!');
150   error.code = 404;
151   throw error;
152 }
153 return {
154   ...post._doc,
155   _id: post._id.toString(),
156   createdAt: post.createdAt.toISOString(),
157   updatedAt: post.updatedAt.toISOString()
158 };
159 }
160 };
161

```

```

1 //./src/pages/Feed/SinglePost/SinglePost.js(f)
2
3 import React, { Component } from 'react';
4
5 import Image from '../../../../../components/Image/Image';
6 import './SinglePost.css';
7
8 class SinglePost extends Component {
9   state = {
10     title: '',
11     author: '',
12     date: '',
13     image: '',
14     content: ''
15   };
16
17   componentDidMount() {
18     const postId = this.props.match.params.postId;
19     const graphqlQuery = {
20       query: `{
21         post(id: "${postId}") {
22           title
23           content
24           imageUrl
25           creator {
26             name
27           }
28           createdAt
29         }
30       }
31     `;
32   };
33   fetch('http://localhost:8080/graphql', {
34     method: 'POST',
35     headers: {
36       Authorization: 'Bearer ' + this.props.token,
37       'Content-Type': 'application/json'
38     },
39     body: JSON.stringify(graphqlQuery)
40   })
41     .then(res => {

```

```

42     return res.json();
43   })
44   .then(resData => {
45     if (resData.errors) {
46       throw new Error('Fetching post failed!');
47     }
48     this.setState({
49       title: resData.data.post.title,
50       author: resData.data.post.creator.name,
51       image: 'http://localhost:8080/' + resData.data.post.imageUrl,
52       date: new Date(resData.data.post.createdAt).toLocaleDateString('en-US'),
53       content: resData.data.post.content
54     });
55   })
56   .catch(err => {
57     console.log(err);
58   });
59 }
60
61 render() {
62   return (
63     <section className="single-post">
64       <h1>{this.state.title}</h1>
65       <h2>
66         Created by {this.state.author} on {this.state.date}
67       </h2>
68       <div className="single-post__image">
69         <Image contain imageUrl={this.state.image} />
70       </div>
71       <p>{this.state.content}</p>
72     </section>
73   );
74 }
75 }
76
77 export default SinglePost;
78

```

* Chapter 435: Updating Posts

1. update
 - ./graphql/schema.js(b)
 - ./graphql/resolvers.js(b)
 - ./src/pages/Feed/Feed.js(f)


```

1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4

```

```
5 module.exports = buildSchema(`\n6   type Post {`  
7     _id: ID!  
8     title: String!  
9     content: String!  
10    imageUrl: String!  
11    creator: User!  
12    createdAt: String!  
13    updatedAt: String!  
14  }`  
15  
16  type User {`  
17    _id: ID!  
18    name: String!  
19    email: String!  
20    password: String  
21    status: String!  
22    posts: [Post!]!  
23  }`  
24  
25  type AuthData {`  
26    token: String!  
27    userId: String!  
28  }`  
29  
30  type PostData {`  
31    posts: [Post!]!  
32    totalPosts: Int!  
33  }`  
34  
35  input UserInputData {`  
36    email: String!  
37    name: String!  
38    password: String!  
39  }`  
40  
41  input PostInputData {`  
42    title: String!  
43    content: String!  
44    imageUrl: String!  
45  }`  
46  
47  type RootQuery {`  
48    login(email: String!, password: String!): AuthData!  
49    posts(page: Int): PostData!  
50    post(id: ID!): Post!  
51  }`  
52  
53  type RootMutation {`  
54    createUser(userInput: UserInputData): User!  
55    createPost(postInput: PostInputData): Post!  
56    updatePost(id: ID!, postInput: PostInputData): Post!  
57  }`  
58  
59  schema {`  
60    query: RootQuery
```

```
61     mutation: RootMutation
62   }
63 `);
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
633
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
```

```
53     error.code = 401;
54     throw error;
55   }
56   const token = jwt.sign(
57     {
58       userId: user._id.toString(),
59       email: user.email
60     },
61     'somesupersecretsecret',
62     { expiresIn: '1h' }
63   );
64   return { token: token, userId: user._id.toString() };
65 },
66 createPost: async function({ postInput }, req) {
67   if (!req.isAuthenticated) {
68     const error = new Error('Not authenticated!');
69     error.code = 401;
70     throw error;
71   }
72   const errors = [];
73   if (
74     validator.isEmpty(postInput.title) ||
75     !validator.isLength(postInput.title, { min: 5 })
76   ) {
77     errors.push({ message: 'Title is invalid.' });
78   }
79   if (
80     validator.isEmpty(postInput.content) ||
81     !validator.isLength(postInput.content, { min: 5 })
82   ) {
83     errors.push({ message: 'Content is invalid.' });
84   }
85   if (errors.length > 0) {
86     const error = new Error('Invalid input.');
87     error.data = errors;
88     error.code = 422;
89     throw error;
90   }
91   const user = await User.findById(req.userId);
92   if (!user) {
93     const error = new Error('Invalid user.');
94     error.code = 401;
95     throw error;
96   }
97   const post = new Post({
98     title: postInput.title,
99     content: postInput.content,
100    imageUrl: postInput.imageUrl,
101    creator: user
102  });
103  const createdPost = await post.save();
104  user.posts.push(createdPost);
105  await user.save();
106  return {
107    ...createdPost._doc,
108    _id: createdPost._id.toString(),
```

```
109     createdAt: createdPost.createdAt.toISOString(),
110     updatedAt: createdPost.updatedAt.toISOString()
111   };
112 },
113 posts: async function({ page }, req) {
114   if (!req.isAuthenticated) {
115     const error = new Error('Not authenticated!');
116     error.code = 401;
117     throw error;
118   }
119   if (!page) {
120     page = 1;
121   }
122   const perPage = 2;
123   const totalPosts = await Post.find().countDocuments();
124   const posts = await Post.find()
125     .sort({ createdAt: -1 })
126     .skip((page - 1) * perPage)
127     .limit(perPage)
128     .populate('creator');
129   return {
130     posts: posts.map(p => {
131       return {
132         ...p._doc,
133         _id: p._id.toString(),
134         createdAt: p.createdAt.toISOString(),
135         updatedAt: p.updatedAt.toISOString()
136       };
137     }),
138     totalPosts: totalPosts
139   };
140 },
141 post: async function({ id }, req) {
142   if (!req.isAuthenticated) {
143     const error = new Error('Not authenticated!');
144     error.code = 401;
145     throw error;
146   }
147   const post = await Post.findById(id).populate('creator');
148   if (!post) {
149     const error = new Error('No post found!');
150     error.code = 404;
151     throw error;
152   }
153   return {
154     ...post._doc,
155     _id: post._id.toString(),
156     createdAt: post.createdAt.toISOString(),
157     updatedAt: post.updatedAt.toISOString()
158   };
159 },
160 updatePost: async function({id, postInput}, req) {
161   if (!req.isAuthenticated) {
162     const error = new Error('Not authenticated!');
163     error.code = 401;
164     throw error;
```

```

165 }
166     const post = await Post.findById(id).populate('creator');
167     if(!post) {
168         const error = new Error('No post found!');
169         error.code = 404;
170         throw error;
171     }
172     if(post.creator._id.toString() !== req.userId.toString()) {
173         const error = new Error('Not authorized!');
174         error.code = 403;
175         throw error;
176     }
177     const errors = [];
178     if (
179         validator.isEmpty(postInput.title) ||
180         !validator.isLength(postInput.title, { min: 5 })
181     ) {
182         errors.push({ message: 'Title is invalid.' });
183     }
184     if (
185         validator.isEmpty(postInput.content) ||
186         !validator.isLength(postInput.content, { min: 5 })
187     ) {
188         errors.push({ message: 'Content is invalid.' });
189     }
190     if (errors.length > 0) {
191         const error = new Error('Invalid input.');
192         error.data = errors;
193         error.code = 422;
194         throw error;
195     }
196     post.title = postInput.title;
197     post.content = postInput.content;
198     /**it has to be checked against a text
199      * because it will be converted to text
200      * by the way i'm sending it on the front-end
201      */
202     if(postInput.imageUrl !== 'undefined') {
203         post.imageUrl = postInput.imageUrl;
204     }
205     const updatedPost = await post.save();
206     return {
207         ...updatedPost._doc,
208         _id: updatedPost._id.toString(),
209         createdAt: updatedPost.createdAt.toISOString(),
210         updatedAt: updatedPost.updatedAt.toISOString()
211     };
212 }
213 };
214

```

```

1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';

```

```

7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26 componentDidMount() {
27   fetch('http://localhost:8080/auth/status', {
28     headers: {
29       Authorization: `Bearer ${this.props.token}`
30     }
31   })
32     .then(res => {
33       if (res.status !== 200) {
34         throw new Error('Failed to fetch user status.');
35       }
36       return res.json();
37     })
38     .then(resData => {
39       this.setState({ status: resData.status });
40     })
41     .catch(this.catchError);
42
43   this.loadPosts();
44 }
45
46 loadPosts = direction => {
47   if (direction) {
48     this.setState({ postsLoading: true, posts: [] });
49   }
50   let page = this.state.postPage;
51   if (direction === 'next') {
52     page++;
53     this.setState({ postPage: page });
54   }
55   if (direction === 'previous') {
56     page--;
57     this.setState({ postPage: page });
58   }
59   const graphqlQuery = {
60     query: `
61       {
62         posts(page: ${page}) {

```

```
63
64     posts {
65         _id
66         title
67         content
68         imageUrl
69         creator {
70             name
71         }
72         createdAt
73     }
74     totalPosts
75 }
76
77 };
78 fetch('http://localhost:8080/graphql', {
79     method: 'POST',
80     headers: {
81         Authorization: 'Bearer ' + this.props.token,
82         'Content-Type': 'application/json'
83     },
84     body: JSON.stringify(graphqlQuery)
85 })
86     .then(res => {
87         return res.json();
88     })
89     .then(resData => {
90         if (resData.errors) {
91             throw new Error('Fetching posts failed!');
92         }
93         this.setState({
94             posts: resData.data.posts.posts.map(post => {
95                 return {
96                     ...post,
97                     imagePath: post.imageUrl
98                 };
99             }),
100            totalPosts: resData.data.posts.totalPosts,
101            postsLoading: false
102        });
103    })
104    .catch(this.catchError);
105};

106 statusUpdateHandler = event => {
107     event.preventDefault();
108     fetch('http://localhost:8080/auth/status', {
109         method: 'PATCH',
110         headers: {
111             Authorization: 'Bearer ' + this.props.token,
112             'Content-Type': 'application/json'
113         },
114         body: JSON.stringify({
115             status: this.state.status
116         })
117     })
118 }
```

```
119     .then(res => {
120       if (res.status !== 200 && res.status !== 201) {
121         throw new Error("Can't update status!");
122       }
123       return res.json();
124     })
125     .then(resData => {
126       console.log(resData);
127     })
128     .catch(this.catchError);
129   };
130
131 newPostHandler = () => {
132   this.setState({ isEditing: true });
133 };
134
135 startEditPostHandler = postId => {
136   this.setState(prevState => {
137     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
138
139     return {
140       isEditing: true,
141       editPost: loadedPost
142     };
143   });
144 };
145
146 cancelEditHandler = () => {
147   this.setState({ isEditing: false, editPost: null });
148 };
149
150 finishEditHandler = postData => {
151   this.setState({
152     editLoading: true
153   });
154   const formData = new FormData();
155   formData.append('image', postData.image);
156   if(this.state.editPost) {
157     formData.append('oldPath', this.state.editPost.imagePath);
158   }
159   fetch('http://localhost:8080/post-image', {
160     method: 'PUT',
161     headers: {
162       Authorization: 'Bearer ' + this.props.token
163     },
164     body: formData
165   })
166   .then(res => res.json())
167   .then(fileResData => {
168     const imageUrl = fileResData.filePath;
169     let graphqlQuery = {
170       query: `
171         mutation {
172           createPost(postInput: {title: "${postData.title}", content: "${{
173             postData.content
174           }", imageUrl: "${imageUrl}"}) {
```

```
175
176
177
178
179
180
181
182
183
184
185
186
187
188 if(this.state.editPost){
189     graphqlQuery = {
190         query: `

191         mutation {
192             updatePost(
193                 id: "${this.state.editPost._id}",
194                 postInput: {
195                     title: "${postData.title}",
196                     content: "${postData.content}",
197                     imageUrl: "${imageUrl}") {
198                         _id
199                         title
200                         content
201                         imageUrl
202                         creator {
203                             name
204                         }
205                         createdAt
206                     }
207                 }
208             }
209         };
210     }
211
212     return fetch('http://localhost:8080/graphql', {
213         method: 'POST',
214         body: JSON.stringify(graphqlQuery),
215         headers: {
216             Authorization: 'Bearer ' + this.props.token,
217             'Content-Type': 'application/json'
218         }
219     })
220 }).then(res => {
221     return res.json();
222 })
223 .then(resData => {
224     if (resData.errors && resData.errors[0].status === 422) {
225         throw new Error(
226             "Validation failed. Make sure the email address isn't used yet!"
227         );
228     }
229     if (resData.errors) {
230         throw new Error('User login failed!');
```

```
231     }
232     /**we can add a new variable 'resDataField'
233      * and by default, that should be 'createPost'
234      * because creating a post also is our default query up there.
235      */
236     let resDataField = 'createPost'
237     if (this.state.editPost) {
238       resDataField = 'updatePost';
239     }
240     const post = {
241       _id: resData.data[resDataField]._id,
242       title: resData.data[resDataField].title,
243       content: resData.data[resDataField].content,
244       creator: resData.data[resDataField].creator,
245       createdAt: resData.data[resDataField].createdAt,
246       imagePath: resData.data[resDataField].imageUrl
247     };
248     this.setState(prevState => {
249       let updatedPosts = [...prevState.posts];
250       if (prevState.editPost) {
251         const postIndex = prevState.posts.findIndex(
252           p => p._id === prevState.editPost._id
253         );
254         updatedPosts[postIndex] = post;
255       } else {
256         updatedPosts.pop();
257         updatedPosts.unshift(post);
258       }
259       return {
260         posts: updatedPosts,
261         isEditing: false,
262         editPost: null,
263         editLoading: false
264       };
265     });
266   })
267   .catch(err => {
268     console.log(err);
269     this.setState({
270       isEditing: false,
271       editPost: null,
272       editLoading: false,
273       error: err
274     });
275   });
276 };
277
278 statusInputChangeHandler = (input, value) => {
279   this.setState({ status: value });
280 };
281
282 deletePostHandler = postId => {
283   this.setState({ postsLoading: true });
284   fetch('http://localhost:8080/feed/post/' + postId, {
285     method: 'DELETE',
286     headers: {
```

```
287         Authorization: 'Bearer ' + this.props.token
288     }
289   })
290   .then(res => {
291     if (res.status !== 200 && res.status !== 201) {
292       throw new Error('Deleting a post failed!');
293     }
294     return res.json();
295   })
296   .then(resData => {
297     console.log(resData);
298     this.loadPosts();
299     // this.setState(prevState => {
300     //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
301     //   return { posts: updatedPosts, postsLoading: false };
302     // });
303   })
304   .catch(err => {
305     console.log(err);
306     this.setState({ postsLoading: false });
307   });
308 }
309
310 errorHandler = () => {
311   this.setState({ error: null });
312 };
313
314 catchError = error => {
315   this.setState({ error: error });
316 };
317
318 render() {
319   return (
320     <Fragment>
321       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
322       <FeedEdit
323         editing={this.state.isEditing}
324         selectedPost={this.state.editPost}
325         loading={this.state.editLoading}
326         onCancelEdit={this.cancelEditHandler}
327         onFinishEdit={this.finishEditHandler}
328       />
329       <section className="feed__status">
330         <form onSubmit={this.statusUpdateHandler}>
331           <Input
332             type="text"
333             placeholder="Your status"
334             control="input"
335             onChange={this.statusInputChangeHandler}
336             value={this.state.status}
337           />
338           <Button mode="flat" type="submit">
339             <Update
340             />
341           </form>
342         </section>
```

```

343     <section className="feed__control">
344         <Button mode="raised" design="accent" onClick={this.newPostHandler}>
345             New Post
346         </Button>
347     </section>
348     <section className="feed">
349         {this.state.postsLoading && (
350             <div style={{ textAlign: 'center', marginTop: '2rem' }}>
351                 <Loader />
352             </div>
353         )}
354         {this.state.posts.length <= 0 && !this.state.postsLoading ? (
355             <p style={{ textAlign: 'center' }}>No posts found.</p>
356         ) : null}
357         {!this.state.postsLoading && (
358             <Paginator
359                 onPrevious={this.loadPosts.bind(this, 'previous')}
360                 onNext={this.loadPosts.bind(this, 'next')}
361                 lastPage={Math.ceil(this.state.totalPosts / 2)}
362                 currentPage={this.state.postPage}
363             >
364                 {this.state.posts.map(post => (
365                     <Post
366                         key={post._id}
367                         id={post._id}
368                         author={post.creator.name}
369                         date={new Date(post.createdAt).toLocaleDateString('en-US')}
370                         title={post.title}
371                         image={post.imageUrl}
372                         content={post.content}
373                         onStartEdit={this.startEditPostHandler.bind(this, post._id)}
374                         onDelete={this.deletePostHandler.bind(this, post._id)}
375                     />
376                 )))
377             </Paginator>
378         )}
379     </section>
380   </Fragment>
381 );
382 }
383 }
384
385 export default Feed;
386

```

* Chapter 436: Deleting Posts

1. update
 - ./graphql/schema.js(b)
 - ./graphql/resolvers.js(b)
 - ./util/file.js(b)
 - app.js(b)
 - ./graphql/resolvers.js(b)
 - ./src/pages/Feed/Feed.js(f)


```















```

```
1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 module.exports = buildSchema(`

6   type Post {
7     _id: ID!
8     title: String!
9     content: String!
10    imageUrl: String!
11    creator: User!
12    createdAt: String!
13    updatedAt: String!
14  }

15

16   type User {
17     _id: ID!
18     name: String!
19     email: String!
20     password: String
```

```

21     status: String!
22     posts: [Post!]!
23   }
24
25   type AuthData {
26     token: String!
27     userId: String!
28   }
29
30   type PostData {
31     posts: [Post!]!
32     totalPosts: Int!
33   }
34
35   input UserInputData {
36     email: String!
37     name: String!
38     password: String!
39   }
40
41   input PostInputData {
42     title: String!
43     content: String!
44     imageUrl: String!
45   }
46
47   type RootQuery {
48     login(email: String!, password: String!): AuthData!
49     posts(page: Int): PostData!
50     post(id: ID!): Post!
51   }
52
53   type RootMutation {
54     createUser(userInput: UserInputData): User!
55     createPost(postInput: PostInputData): Post!
56     updatePost(id: ID!, postInput: PostInputData): Post!
57     deletePost(id: ID!): Boolean
58   }
59
60   schema {
61     query: RootQuery
62     mutation: RootMutation
63   }
64 `);
65

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9 const { clearImage } = require('../util/file');
10
11 module.exports = {

```

```
12 createUser: async function({ userInput }, req) {
13   // const email = args.userInput.email;
14   const errors = [];
15   if (!validator.isEmail(userInput.email)) {
16     errors.push({ message: 'E-Mail is invalid.' });
17   }
18   if (
19     validator.isEmpty(userInput.password) ||
20     !validator.minLength(userInput.password, { min: 5 })
21   ) {
22     errors.push({ message: 'Password too short!' });
23   }
24   if (errors.length > 0) {
25     const error = new Error('Invalid input.');
26     error.data = errors;
27     error.code = 422;
28     throw error;
29   }
30   const existingUser = await User.findOne({ email: userInput.email });
31   if (existingUser) {
32     const error = new Error('User exists already!');
33     throw error;
34   }
35   const hashedPw = await bcrypt.hash(userInput.password, 12);
36   const user = new User({
37     email: userInput.email,
38     name: userInput.name,
39     password: hashedPw
40   });
41   const createdUser = await user.save();
42   return { ...createdUser._doc, _id: createdUser._id.toString() };
43 },
44 login: async function({ email, password }) {
45   const user = await User.findOne({ email: email });
46   if (!user) {
47     const error = new Error('User not found.');
48     error.code = 401;
49     throw error;
50   }
51   const isEqual = await bcrypt.compare(password, user.password);
52   if (!isEqual) {
53     const error = new Error('Password is incorrect.');
54     error.code = 401;
55     throw error;
56   }
57   const token = jwt.sign(
58     {
59       userId: user._id.toString(),
60       email: user.email
61     },
62     'somesupersecretsecret',
63     { expiresIn: '1h' }
64   );
65   return { token: token, userId: user._id.toString() };
66 },
67 createPost: async function({ postInput }, req) {
```

```
68  if (!req.isAuthenticated) {
69    const error = new Error('Not authenticated!');
70    error.code = 401;
71    throw error;
72  }
73  const errors = [];
74  if (
75    validator.isEmpty(postInput.title) ||
76    !validator.isLength(postInput.title, { min: 5 })
77  ) {
78    errors.push({ message: 'Title is invalid.' });
79  }
80  if (
81    validator.isEmpty(postInput.content) ||
82    !validator.isLength(postInput.content, { min: 5 })
83  ) {
84    errors.push({ message: 'Content is invalid.' });
85  }
86  if (errors.length > 0) {
87    const error = new Error('Invalid input.');
88    error.data = errors;
89    error.code = 422;
90    throw error;
91  }
92  const user = await User.findById(req.userId);
93  if (!user) {
94    const error = new Error('Invalid user.');
95    error.code = 401;
96    throw error;
97  }
98  const post = new Post({
99    title: postInput.title,
100   content: postInput.content,
101   imageUrl: postInput.imageUrl,
102   creator: user
103 });
104 const createdPost = await post.save();
105 user.posts.push(createdPost);
106 await user.save();
107 return {
108   ...createdPost._doc,
109   _id: createdPost._id.toString(),
110   createdAt: createdPost.createdAt.toISOString(),
111   updatedAt: createdPost.updatedAt.toISOString()
112 };
113 },
114 posts: async function({ page }, req) {
115  if (!req.isAuthenticated) {
116    const error = new Error('Not authenticated!');
117    error.code = 401;
118    throw error;
119  }
120  if (!page) {
121    page = 1;
122  }
123  const perPage = 2;
```

```
124 const totalPosts = await Post.find().countDocuments();
125 const posts = await Post.find()
126     .sort({ createdAt: -1 })
127     .skip((page - 1) * perPage)
128     .limit(perPage)
129     .populate('creator');
130 
131 return {
132     posts: posts.map(p => {
133         return {
134             ...p._doc,
135             _id: p._id.toString(),
136             createdAt: p.createdAt.toISOString(),
137             updatedAt: p.updatedAt.toISOString()
138         };
139     }),
140     totalPosts: totalPosts
141 };
142 },
143 post: async function({ id }, req) {
144     if (!req.isAuthenticated) {
145         const error = new Error('Not authenticated!');
146         error.code = 401;
147         throw error;
148     }
149     const post = await Post.findById(id).populate('creator');
150     if (!post) {
151         const error = new Error('No post found!');
152         error.code = 404;
153         throw error;
154     }
155     return {
156         ...post._doc,
157         _id: post._id.toString(),
158         createdAt: post.createdAt.toISOString(),
159         updatedAt: post.updatedAt.toISOString()
160     };
161 },
162 updatePost: async function({ id, postInput }, req) {
163     if (!req.isAuthenticated) {
164         const error = new Error('Not authenticated!');
165         error.code = 401;
166         throw error;
167     }
168     const post = await Post.findById(id).populate('creator');
169     if (!post) {
170         const error = new Error('No post found!');
171         error.code = 404;
172         throw error;
173     }
174     if (post.creator._id.toString() !== req.userId.toString()) {
175         const error = new Error('Not authorized!');
176         error.code = 403;
177         throw error;
178     }
179     const errors = [];
180     if (
181         postInput.title.length < 1 ||
182         postInput.content.length < 1
183     ) {
184         errors.push('Post must have a title and content');
185     }
186     if (errors.length) {
187         const error = new Error('Validation failed');
188         error.code = 400;
189         error.validationErrors = errors;
190         throw error;
191     }
192     post.title = postInput.title;
193     post.content = postInput.content;
194     post.updatedAt = Date.now();
195     await post.save();
196     return post;
197 }
```

```
180     validator.isEmpty(postInput.title) ||
181     !validator.isLength(postInput.title, { min: 5 })
182   ) {
183     errors.push({ message: 'Title is invalid.' });
184   }
185   if (
186     validator.isEmpty(postInput.content) ||
187     !validator.isLength(postInput.content, { min: 5 })
188   ) {
189     errors.push({ message: 'Content is invalid.' });
190   }
191   if (errors.length > 0) {
192     const error = new Error('Invalid input.');
193     error.data = errors;
194     error.code = 422;
195     throw error;
196   }
197   post.title = postInput.title;
198   post.content = postInput.content;
199   if(postInput.imageUrl !== 'undefined') {
200     post.imageUrl = postInput.imageUrl;
201   }
202   const updatedPost = await post.save();
203   return {
204     ...updatedPost._doc,
205     _id: updatedPost._id.toString(),
206     createdAt: updatedPost.createdAt.toISOString(),
207     updatedAt: updatedPost.updatedAt.toISOString()
208   };
209 },
210 deletePost: async function({ id }, req) {
211   if(!req.isAuthenticated) {
212     const error = new Error('Not authenticated!');
213     error.code = 401;
214     throw error;
215   }
216   const post = await Post.findById(id);
217   if(!post) {
218     const error = new Error('No post found!');
219     error.code = 404;
220     throw error;
221   }
222   /**since i don't populate my creator field here,
223    * the creator here is a not an object with _id
224    * but creator itself is already the ID
225    * because that is how it's stored in a post.
226    * and we need to call populate to change that and the result
227    * we are not doing that here
228    * so we only access creator to directly get the ID of the user who created the post
229    */
230   if(post.creator.toString() !== req.userId.toString()) {
231     const error = new Error('Not authorized!');
232     error.code = 403;
233     throw error;
234   }
235   clearImage(post.imageUrl);
```

```
236     await Post.findByIdAndRemove(id);
237     const user = await User.findById(req.userId);
238     user.posts.pull(id);
239     await user.save();
240     /**in ./graphql/schema.js(b),
241      * i find that i wanna return boolean
242      */
243     return true;
244   }
245 };
246
```

```
1 //./util/file.js(b)
2
3 const path = require('path');
4 const fs = require('fs');
5
6 const clearImage = filePath => {
7   filePath = path.join(__dirname, '...', filePath);
8   fs.unlink(filePath, err => console.log(err));
9 };
10
11 exports.clearImage = clearImage;
```

```
1 //app.js(b)
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9
10 const graphqlHttp = require('express-graphql');
11
12 const graphSchema = require('./graphql/schema');
13 const graphResolver = require('./graphql/resolvers');
14 const auth = require('./middleware/auth');
15 const { clearImage } = require('./util/file');
16
17 const app = express();
18
19 const fileStorage = multer.diskStorage({
20   destination: (req, file, cb) => {
21     cb(null, 'images');
22   },
23   filename: (req, file, cb) => {
24     cb(null, new Date().toISOString() + '-' + file.originalname);
25   }
26 });
27
28 const fileFilter = (req, file, cb) => {
29   if(
30     file.mimetype === 'image/png' ||
31     file.mimetype === 'image/jpg' ||
32     file.mimetype === 'image/jpeg'
33   ) {
```

```
34     cb(null, true);
35   } else {
36     cb(null, false);
37   }
38 }
39
40 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
41 app.use(bodyParser.json()); // application/json
42 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
43 app.use('/images', express.static(path.join(__dirname, 'images')));
44
45 app.use((req, res, next) => {
46   res.setHeader('Access-Control-Allow-Origin', '*');
47   res.setHeader(
48     'Access-Control-Allow-Methods',
49     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
50   );
51   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
52   if(req.method === 'OPTIONS') {
53     return res.sendStatus(200);
54   }
55   next();
56 });
57
58 app.use(auth);
59
60 app.put('/post-image', (req, res, next) => {
61   if(!req.isAuthenticated){
62     throw new Error('Not authenticated!');
63   }
64   if(!req.file){
65     return res.status(200).json({message: 'No file provided!'});
66   }
67   if(req.body.oldPath) {
68     clearImage(req.body.oldPath);
69   }
70   return res.status(201).json({message: 'File stored.', filePath: req.file.path});
71 });
72
73 app.use('/graphql', graphqlHttp({
74   schema: graphSchema,
75   rootValue: graphResolver,
76   graphiql: true,
77   customFormatErrorFn(err) {
78     if (!err.originalError) {
79       return err;
80     }
81     const data = err.originalError.data;
82     const message = err.message || 'An error occurred.';
83     const code = err.originalError.code || 500
84     return { message: message, status: code, data: data };
85   }
86 })
87 );
88
89 app.use((error, req, res, next) => {
```

```

90  console.log(error);
91  const status = error.statusCode || 500;
92  const message = error.message;
93  const data = error.data;
94  res.status(status).json({ message: message, data: data });
95 });
96
97 mongoose
98   .connect(
99     'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/message?retryWrites=true'
100    )
101  .then(result => {
102    app.listen(8080);
103  })
104  .catch(err => console.log(err));

```

```

1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('http://localhost:8080/auth/status', {
28       headers: {
29         Authorization: `Bearer ${this.props.token}`
30       }
31     })
32     .then(res => {
33       if (res.status !== 200) {
34         throw new Error('Failed to fetch user status.');
35       }
36       return res.json();
37     })
38     .then(resData => {
39       this.setState({ status: resData.status });
40     })
41     .catch(this.catchError);

```

```
42
43     this.loadPosts();
44 }
45
46 loadPosts = direction => {
47     if (direction) {
48         this.setState({ postsLoading: true, posts: [] });
49     }
50     let page = this.state.postPage;
51     if (direction === 'next') {
52         page++;
53         this.setState({ postPage: page });
54     }
55     if (direction === 'previous') {
56         page--;
57         this.setState({ postPage: page });
58     }
59     const graphqlQuery = {
60         query: `
61             {
62                 posts(page: ${page}) {
63                     posts {
64                         _id
65                         title
66                         content
67                         imageUrl
68                         creator {
69                             name
70                         }
71                         createdAt
72                     }
73                     totalPosts
74                 }
75             }
76         `
77     };
78     fetch('http://localhost:8080/graphql', {
79         method: 'POST',
80         headers: {
81             Authorization: 'Bearer ' + this.props.token,
82             'Content-Type': 'application/json'
83         },
84         body: JSON.stringify(graphqlQuery)
85     })
86     .then(res => {
87         return res.json();
88     })
89     .then(resData => {
90         if (resData.errors) {
91             throw new Error('Fetching posts failed!');
92         }
93         this.setState({
94             posts: resData.data.posts.posts.map(post => {
95                 return {
96                     ...post,
97                     imagePath: post.imageUrl
98                 }
99             })
100        });
101    }
102 }
103
104 export default Posts;
```

```

98      );
99    },
100   totalPosts: resData.data.posts.totalPosts,
101   postsLoading: false
102 );
103 )
104 .catch(this.catchError);
105 };
106
107 statusUpdateHandler = event => {
108   event.preventDefault();
109   fetch('http://localhost:8080/auth/status', {
110     method: 'PATCH',
111     headers: {
112       Authorization: 'Bearer ' + this.props.token,
113       'Content-Type': 'application/json'
114     },
115     body: JSON.stringify({
116       status: this.state.status
117     })
118   })
119   .then(res => {
120     if (res.status !== 200 && res.status !== 201) {
121       throw new Error("Can't update status!");
122     }
123     return res.json();
124   })
125   .then(resData => {
126     console.log(resData);
127   })
128   .catch(this.catchError);
129 };
130
131 newPostHandler = () => {
132   this.setState({ isEditing: true });
133 };
134
135 startEditPostHandler = postId => {
136   this.setState(prevState => {
137     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
138
139     return {
140       isEditing: true,
141       editPost: loadedPost
142     };
143   });
144 };
145
146 cancelEditHandler = () => {
147   this.setState({ isEditing: false, editPost: null });
148 };
149
150 finishEditHandler = postData => {
151   this.setState({
152     editLoading: true
153   });

```

```
154 const formData = new FormData();
155 formData.append('image', postData.image);
156 if(this.state.editPost) {
157     formData.append('oldPath', this.state.editPost.imagePath);
158 }
159 fetch('http://localhost:8080/post-image', {
160     method: 'PUT',
161     headers: {
162         Authorization: 'Bearer ' + this.props.token
163     },
164     body: formData
165 })
166 .then(res => res.json())
167 .then(fileResData => {
168     const imageUrl = fileResData.filePath;
169     let graphqlQuery = {
170         query: `
171             mutation {
172                 createPost(postInput: {title: "${postData.title}", content: "${postData.content
173                 imageUrl: "${imageUrl}"}) {
174                     _id
175                     title
176                     content
177                     imageUrl
178                     creator {
179                         name
180                     }
181                     createdAt
182                 }
183             }
184         `
185     };
186     if(this.state.editPost){
187         graphqlQuery = {
188             query: `
189                 mutation {
190                     updatePost(
191                         id: "${this.state.editPost._id}",
192                         postInput: {
193                             title: "${postData.title}",
194                             content: "${postData.content}",
195                             imageUrl: "${imageUrl}") {
196                                 _id
197                                 title
198                                 content
199                                 imageUrl
200                                 creator {
201                                     name
202                                 }
203                                 createdAt
204                             }
205                         }
206                     `
207                 `;
208             }
209         };
210     }
211     return graphqlQuery;
212 }
213 .then(res => res.json())
214 .then(data => {
215     console.log(data);
216     this.setState({post: data});
217 }
218 );
```

```
210 }
211
212 return fetch('http://localhost:8080/graphql', {
213   method: 'POST',
214   body: JSON.stringify(graphqlQuery),
215   headers: {
216     Authorization: 'Bearer ' + this.props.token,
217     'Content-Type': 'application/json'
218   }
219 })
220 }).then(res => {
221   return res.json();
222 })
223 .then(resData => {
224   if (resData.errors && resData.errors[0].status === 422) {
225     throw new Error(
226       "Validation failed. Make sure the email address isn't used yet!"
227     );
228   }
229   if (resData.errors) {
230     throw new Error('User login failed!');
231   }
232   let resDataField = 'createPost'
233   if (this.state.editPost) {
234     resDataField = 'updatePost';
235   }
236   const post = {
237     _id: resData.data[resDataField]._id,
238     title: resData.data[resDataField].title,
239     content: resData.data[resDataField].content,
240     creator: resData.data[resDataField].creator,
241     createdAt: resData.data[resDataField].createdAt,
242     imagePath: resData.data[resDataField].imageUrl
243   };
244   this.setState(prevState => {
245     let updatedPosts = [...prevState.posts];
246     if (prevState.editPost) {
247       const postIndex = prevState.posts.findIndex(
248         p => p._id === prevState.editPost._id
249       );
250       updatedPosts[postIndex] = post;
251     } else {
252       updatedPosts.pop();
253       updatedPosts.unshift(post);
254     }
255     return {
256       posts: updatedPosts,
257       isEditing: false,
258       editPost: null,
259       editLoading: false
260     };
261   });
262 })
263 .catch(err => {
264   console.log(err);
265   this.setState({
```

```
266     isEditing: false,
267     editPost: null,
268     editLoading: false,
269     error: err
270   });
271 });
272 };
273
274 statusInputChangeHandler = (input, value) => {
275   this.setState({ status: value });
276 };
277
278 deletePostHandler = postId => {
279   this.setState({ postsLoading: true });
280   const graphqlQuery = {
281     /**we have no nested object here
282      * so we couldn't even get any nested or detailed data
283      * we only get true or false
284      */
285     query: `mutation {
286       deletePost(id: "${postId}")
287     }
288   `;
289 };
290
291 fetch('http://localhost:8080/graphql', {
292   method: 'POST',
293   headers: {
294     Authorization: 'Bearer ' + this.props.token,
295     'Content-Type': 'application/json'
296   },
297   body: JSON.stringify(graphqlQuery)
298 })
299   .then(res => {
300     return res.json();
301   })
302   .then(resData => {
303     if(resData.errors) {
304       throw new Error('Deleting the post failed!');
305     }
306     console.log(resData);
307     this.loadPosts();
308     // this.setState(prevState => {
309     //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
310     //   return { posts: updatedPosts, postsLoading: false };
311     // });
312   })
313   .catch(err => {
314     console.log(err);
315     this.setState({ postsLoading: false });
316   });
317 };
318
319 errorHandler = () => {
320   this.setState({ error: null });
321 };
```

```
322
323     catchError = error => {
324       this.setState({ error: error });
325     };
326
327   render() {
328     return (
329       <Fragment>
330         <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
331         <FeedEdit
332           editing={this.state.isEditing}
333           selectedPost={this.state.editPost}
334           loading={this.state.editLoading}
335           onCancelEdit={this.cancelEditHandler}
336           onFinishEdit={this.finishEditHandler}
337         />
338         <section className="feed__status">
339           <form onSubmit={this.statusUpdateHandler}>
340             <Input
341               type="text"
342               placeholder="Your status"
343               control="input"
344               onChange={this.statusInputChangeHandler}
345               value={this.state.status}
346             />
347             <Button mode="flat" type="submit">
348               Update
349             </Button>
350           </form>
351         </section>
352         <section className="feed__control">
353           <Button mode="raised" design="accent" onClick={this.newPostHandler}>
354             New Post
355           </Button>
356         </section>
357         <section className="feed">
358           {this.state.postsLoading && (
359             <div style={{ textAlign: 'center', marginTop: '2rem' }}>
360               <Loader />
361             </div>
362           )}
363           {this.state.posts.length <= 0 && !this.state.postsLoading ? (
364             <p style={{ textAlign: 'center' }}>No posts found.</p>
365           ) : null}
366           {!this.state.postsLoading && (
367             <Paginator
368               onPrevious={this.loadPosts.bind(this, 'previous')}
369               onNext={this.loadPosts.bind(this, 'next')}
370               lastPage={Math.ceil(this.state.totalPosts / 2)}
371               currentPage={this.state.postPage}
372             >
373               {this.state.posts.map(post => (
374                 <Post
375                   key={post._id}
376                   id={post._id}
377                   author={post.creator.name}
378                 </Post>
379               )}
380             </Paginator>
381           )}
382         </section>
383       </Fragment>
384     );
385   }
386 }
```

```

378     date={new Date(post.createdAt).toLocaleDateString('en-US')}
379     title={post.title}
380     image={post.imageUrl}
381     content={post.content}
382     onStartEdit={this.startEditPostHandler.bind(this, post._id)}
383     onDelete={this.deletePostHandler.bind(this, post._id)}
384   />
385 })
386 </Paginator>
387 )
388 </section>
389 </Fragment>
390 );
391 }
392 }
393
394 export default Feed;
395

```

* Chapter 438: Managing The User Status

1. update
 - ./graphql/schema.js(b)
 - ./graphql/resolvers.js(b)
 - ./src/pages/Feed/Feed.js(f)

- this is a change that is required and now that we see the status and this error message is gone


```

1 //./graphql/schema.js(b)
2
3 const { buildSchema } = require('graphql');
4
5 module.exports = buildSchema(`

6   type Post {
7     _id: ID!
8     title: String!
9     content: String!
10    imageUrl: String!
11    creator: User!
12    createdAt: String!
13    updatedAt: String!
14 }

```

```

15
16 type User {
17     _id: ID!
18     name: String!
19     email: String!
20     password: String
21     status: String!
22     posts: [Post!]!
23 }
24
25 type AuthData {
26     token: String!
27     userId: String!
28 }
29
30 type PostData {
31     posts: [Post!]!
32     totalPosts: Int!
33 }
34
35 input UserInputData {
36     email: String!
37     name: String!
38     password: String!
39 }
40
41 input PostInputData {
42     title: String!
43     content: String!
44     imageUrl: String!
45 }
46
47 type RootQuery {
48     login(email: String!, password: String!): AuthData!
49     posts(page: Int): PostData!
50     post(id: ID!): Post!
51     user: User!
52 }
53
54 type RootMutation {
55     createUser(userInput: UserInputData): User!
56     createPost(postInput: PostInputData): Post!
57     updatePost(id: ID!, postInput: PostInputData): Post!
58     deletePost(id: ID!): Boolean
59     updateStatus(status: String!): User!
60 }
61
62 schema {
63     query: RootQuery
64     mutation: RootMutation
65 }
66 `);
67

```

```

1 //./graphql/resolvers.js(b)
2
3 const bcrypt = require('bcryptjs');

```

```
4 const validator = require('validator');
5 const jwt = require('jsonwebtoken');
6
7 const User = require('../models/user');
8 const Post = require('../models/post');
9 const { clearImage } = require('../util/file');
10
11 module.exports = {
12   createUser: async function({ userInput }, req) {
13     // const email = args.userInput.email;
14     const errors = [];
15     if (!validator.isEmail(userInput.email)) {
16       errors.push({ message: 'E-Mail is invalid.' });
17     }
18     if (
19       validator.isEmpty(userInput.password) ||
20       !validator.minLength(userInput.password, { min: 5 })
21     ) {
22       errors.push({ message: 'Password too short!' });
23     }
24     if (errors.length > 0) {
25       const error = new Error('Invalid input.');
26       error.data = errors;
27       error.code = 422;
28       throw error;
29     }
30     const existingUser = await User.findOne({ email: userInput.email });
31     if (existingUser) {
32       const error = new Error('User exists already!');
33       throw error;
34     }
35     const hashedPw = await bcrypt.hash(userInput.password, 12);
36     const user = new User({
37       email: userInput.email,
38       name: userInput.name,
39       password: hashedPw
40     });
41     const createdUser = await user.save();
42     return { ...createdUser._doc, _id: createdUser._id.toString() };
43   },
44   login: async function({ email, password }) {
45     const user = await User.findOne({ email: email });
46     if (!user) {
47       const error = new Error('User not found.');
48       error.code = 401;
49       throw error;
50     }
51     const isEqual = await bcrypt.compare(password, user.password);
52     if (!isEqual) {
53       const error = new Error('Password is incorrect.');
54       error.code = 401;
55       throw error;
56     }
57     const token = jwt.sign(
58       {
59         userId: user._id.toString(),

```

```
60     email: user.email
61   },
62   'somesupersecretsecret',
63   { expiresIn: '1h' }
64 );
65   return { token: token, userId: user._id.toString() };
66 },
67 createPost: async function({ postInput }, req) {
68   if (!req.isAuthenticated()) {
69     const error = new Error('Not authenticated!');
70     error.code = 401;
71     throw error;
72   }
73   const errors = [];
74   if (
75     validator.isEmpty(postInput.title) ||
76     !validator.isLength(postInput.title, { min: 5 })
77   ) {
78     errors.push({ message: 'Title is invalid.' });
79   }
80   if (
81     validator.isEmpty(postInput.content) ||
82     !validator.isLength(postInput.content, { min: 5 })
83   ) {
84     errors.push({ message: 'Content is invalid.' });
85   }
86   if (errors.length > 0) {
87     const error = new Error('Invalid input.');
88     error.data = errors;
89     error.code = 422;
90     throw error;
91   }
92   const user = await User.findById(req.userId);
93   if (!user) {
94     const error = new Error('Invalid user.');
95     error.code = 401;
96     throw error;
97   }
98   const post = new Post({
99     title: postInput.title,
100    content: postInput.content,
101    imageUrl: postInput.imageUrl,
102    creator: user
103  });
104  const createdPost = await post.save();
105  user.posts.push(createdPost);
106  await user.save();
107  return {
108    ...createdPost._doc,
109    _id: createdPost._id.toString(),
110    createdAt: createdPost.createdAt.toISOString(),
111    updatedAt: createdPost.updatedAt.toISOString()
112  };
113},
114 posts: async function({ page }, req) {
115   if (!req.isAuthenticated()) {
```

```
116     const error = new Error('Not authenticated!');
117     error.code = 401;
118     throw error;
119   }
120   if (!page) {
121     page = 1;
122   }
123   const perPage = 2;
124   const totalPosts = await Post.find().countDocuments();
125   const posts = await Post.find()
126     .sort({ createdAt: -1 })
127     .skip((page - 1) * perPage)
128     .limit(perPage)
129     .populate('creator');
130   return {
131     posts: posts.map(p => {
132       return {
133         ...p._doc,
134         _id: p._id.toString(),
135         createdAt: p.createdAt.toISOString(),
136         updatedAt: p.updatedAt.toISOString()
137       };
138     }),
139     totalPosts
140   };
141 },
142 post: async function({ id }, req) {
143   if (!req.isAuthenticated) {
144     const error = new Error('Not authenticated!');
145     error.code = 401;
146     throw error;
147   }
148   const post = await Post.findById(id).populate('creator');
149   if (!post) {
150     const error = new Error('No post found!');
151     error.code = 404;
152     throw error;
153   }
154   return {
155     ...post._doc,
156     _id: post._id.toString(),
157     createdAt: post.createdAt.toISOString(),
158     updatedAt: post.updatedAt.toISOString()
159   };
160 },
161 updatePost: async function({ id, postInput }, req) {
162   if (!req.isAuthenticated) {
163     const error = new Error('Not authenticated!');
164     error.code = 401;
165     throw error;
166   }
167   const post = await Post.findById(id).populate('creator');
168   if (!post) {
169     const error = new Error('No post found!');
170     error.code = 404;
171     throw error;
```

```
172 }
173     if(post.creator._id.toString() !== req.userId.toString()) {
174         const error = new Error('Not authorized!');
175         error.code = 403;
176         throw error;
177     }
178     const errors = [];
179     if (
180         validator.isEmpty(postInput.title) ||
181         !validator.isLength(postInput.title, { min: 5 })
182     ) {
183         errors.push({ message: 'Title is invalid.' });
184     }
185     if (
186         validator.isEmpty(postInput.content) ||
187         !validator.isLength(postInput.content, { min: 5 })
188     ) {
189         errors.push({ message: 'Content is invalid.' });
190     }
191     if (errors.length > 0) {
192         const error = new Error('Invalid input.');
193         error.data = errors;
194         error.code = 422;
195         throw error;
196     }
197     post.title = postInput.title;
198     post.content = postInput.content;
199     if(postInput.imageUrl !== 'undefined') {
200         post.imageUrl = postInput.imageUrl;
201     }
202     const updatedPost = await post.save();
203     return {
204         ...updatedPost._doc,
205         _id: updatedPost._id.toString(),
206         createdAt: updatedPost.createdAt.toISOString(),
207         updatedAt: updatedPost.updatedAt.toISOString()
208     };
209 },
210 deletePost: async function({ id }, req) {
211     if(!req.isAuthenticated) {
212         const error = new Error('Not authenticated!');
213         error.code = 401;
214         throw error;
215     }
216     const post = await Post.findById(id);
217     if(!post) {
218         const error = new Error('No post found!');
219         error.code = 404;
220         throw error;
221     }
222     if(post.creator.toString() !== req.userId.toString()) {
223         const error = new Error('Not authorized!');
224         error.code = 403;
225         throw error;
226     }
227     clearImage(post.imageUrl);
```

```

228     await Post.findByIdAndRemove(id);
229     const user = await User.findById(req.userId);
230     user.posts.pull(id);
231     await user.save();
232     return true;
233 },
234 user: async function(args, req) {
235   if(!req.isAuthenticated) {
236     const error = new Error('Not authenticated!');
237     error.code = 401;
238     throw error;
239   }
240   const user = await User.findById(req.userId);
241   if(!user) {
242     const error = new Error('Not authenticated!');
243     error.code = 401;
244     throw error;
245   }
246   return {...user._doc, _id: user._id.toString()}
247 },
248 updateStatus: async function({status}, req) {
249   if(!req.isAuthenticated) {
250     const error = new Error('Not authenticated!');
251     error.code = 401;
252     throw error;
253   }
254   const user = await User.findById(req.userId);
255   if(!user) {
256     const error = new Error('Not authenticated!');
257     error.code = 401;
258     throw error;
259   }
260   user.status = status;
261   await user.save();
262   return {
263     ...user._doc,
264     _id: user._id.toString()
265   }
266 }
267 };
268 
```

```

1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = { 
```

```
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     const graphqlQuery = {
28       query: `
29         {
30           user {
31             status
32           }
33         }
34       `
35     }
36     fetch('http://localhost:8080/graphql', {
37       method: 'POST',
38       headers: {
39         Authorization: 'Bearer ' + this.props.token,
40         'Content-Type': 'application/json'
41       },
42       body: JSON.stringify(graphqlQuery)
43     })
44       .then(res => {
45         return res.json();
46       })
47       .then(resData => {
48         if (resData.errors) {
49           throw new Error('Fetching status failed!');
50         }
51         this.setState({ status: resData.data.user.status });
52       })
53       .catch(this.catchError);
54
55     this.loadPosts();
56   }
57
58   loadPosts = direction => {
59     if (direction) {
60       this.setState({ postsLoading: true, posts: [] });
61     }
62     let page = this.state.postPage;
63     if (direction === 'next') {
64       page++;
65       this.setState({ postPage: page });
66     }
67     if (direction === 'previous') {
68       page--;
69       this.setState({ postPage: page });
70     }
71     const graphqlQuery = {
```

```
72 query: ` 
73   {
74     posts(page: ${page}) {
75       posts {
76         _id
77         title
78         content
79         imageUrl
80         creator {
81           name
82         }
83         createdAt
84       }
85       totalPosts
86     }
87   }
88 `
89 };
90 fetch('http://localhost:8080/graphql', {
91   method: 'POST',
92   headers: {
93     Authorization: 'Bearer ' + this.props.token,
94     'Content-Type': 'application/json'
95   },
96   body: JSON.stringify(graphqlQuery)
97 })
98 .then(res => {
99   return res.json();
100 })
101 .then(resData => {
102   if (resData.errors) {
103     throw new Error('Fetching posts failed!');
104   }
105   this.setState({
106     posts: resData.data.posts.posts.map(post => {
107       return {
108         ...post,
109         imagePath: post.imageUrl
110       };
111     }),
112     totalPosts: resData.data.posts.totalPosts,
113     postsLoading: false
114   });
115 })
116 .catch(this.catchError);
117 };
118
119 statusUpdateHandler = event => {
120   event.preventDefault();
121   const graphqlQuery = {
122     query: ` 
123       mutation {
124         updateStatus(status: "${this.state.status}") {
125           status
126         }
127       }
128     `;
129   }
130   fetch('http://localhost:8080/graphql', {
131     method: 'POST',
132     headers: {
133       Authorization: 'Bearer ' + this.props.token,
134       'Content-Type': 'application/json'
135     },
136     body: JSON.stringify(graphqlQuery)
137   })
138   .then(res => {
139     return res.json();
140   })
141   .then(resData => {
142     if (resData.errors) {
143       throw new Error('Updating status failed!');
144     }
145     this.setState({
146       status: resData.data.updateStatus.status
147     });
148   })
149   .catch(this.catchError);
150 };
151
152 
```

```
128
129     }
130   fetch('http://localhost:8080/graphql', {
131     method: 'POST',
132     headers: {
133       Authorization: 'Bearer ' + this.props.token,
134       'Content-Type': 'application/json'
135     },
136     body: JSON.stringify(graphqlQuery)
137   })
138   .then(res => {
139     return res.json();
140   })
141   .then(resData => {
142     if (resData.errors) {
143       throw new Error('Fetching posts failed!');
144     }
145     console.log(resData);
146   })
147   .catch(this.catchError);
148 };
149
150 newPostHandler = () => {
151   this.setState({ isEditing: true });
152 };
153
154 startEditPostHandler = postId => {
155   this.setState(prevState => {
156     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
157
158     return {
159       isEditing: true,
160       editPost: loadedPost
161     };
162   });
163 };
164
165 cancelEditHandler = () => {
166   this.setState({ isEditing: false, editPost: null });
167 };
168
169 finishEditHandler = postData => {
170   this.setState({
171     editLoading: true
172   });
173   const formData = new FormData();
174   formData.append('image', postData.image);
175   if(this.state.editPost) {
176     formData.append('oldPath', this.state.editPost.imagePath);
177   }
178   fetch('http://localhost:8080/post-image', {
179     method: 'PUT',
180     headers: {
181       Authorization: 'Bearer ' + this.props.token
182     },
183     body: formData
```

```

184 })
185 .then(res => res.json())
186 .then(fileResData => {
187   const imageUrl = fileResData.filePath;
188   let graphqlQuery = {
189     query: `
190       mutation {
191         createPost(postInput: {title: "${postData.title}", content: "${postData.content
192           ", imageUrl: "${imageUrl}"}) {
193           _id
194           title
195           content
196           imageUrl
197           creator {
198             name
199           }
200           createdAt
201         }
202       }
203     `
204   };
205 
206   if(this.state.editPost){
207     graphqlQuery = {
208       query: `
209         mutation {
210           updatePost(
211             id: "${this.state.editPost._id}",
212             postInput: {
213               title: "${postData.title}",
214               content: "${postData.content}",
215               imageUrl: "${imageUrl}"}) {
216                 _id
217                 title
218                 content
219                 imageUrl
220                 creator {
221                   name
222                 }
223                 createdAt
224               }
225             }
226           }
227         `;
228   };
229 }
230 
231 return fetch('http://localhost:8080/graphql', {
232   method: 'POST',
233   body: JSON.stringify(graphqlQuery),
234   headers: {
235     Authorization: 'Bearer ' + this.props.token,
236     'Content-Type': 'application/json'
237   }
238 })
239 }).then(res => {

```

```
240     return res.json();
241   })
242   .then(resData => {
243     if (resData.errors && resData.errors[0].status === 422) {
244       throw new Error(
245         "Validation failed. Make sure the email address isn't used yet!"
246       );
247     }
248     if (resData.errors) {
249       throw new Error('User login failed!');
250     }
251     let resDataField = 'createPost'
252     if (this.state.editPost) {
253       resDataField = 'updatePost';
254     }
255     const post = {
256       _id: resData.data[resDataField]._id,
257       title: resData.data[resDataField].title,
258       content: resData.data[resDataField].content,
259       creator: resData.data[resDataField].creator,
260       createdAt: resData.data[resDataField].createdAt,
261       imagePath: resData.data[resDataField].imageUrl
262     };
263     this.setState(prevState => {
264       let updatedPosts = [...prevState.posts];
265       if (prevState.editPost) {
266         const postIndex = prevState.posts.findIndex(
267           p => p._id === prevState.editPost._id
268         );
269         updatedPosts[postIndex] = post;
270       } else {
271         if (prevState.posts.length >= 2) {
272           updatedPosts.pop();
273         }
274         updatedPosts.unshift(post);
275       }
276       return {
277         posts: updatedPosts,
278         isEditing: false,
279         editPost: null,
280         editLoading: false
281       };
282     });
283   })
284   .catch(err => {
285     console.log(err);
286     this.setState({
287       isEditing: false,
288       editPost: null,
289       editLoading: false,
290       error: err
291     });
292   });
293 }
294
295 statusInputChangeHandler = (input, value) => {
```

```
296     this.setState({ status: value });
297   };
298
299   deletePostHandler = postId => {
300     this.setState({ postsLoading: true });
301     const graphqlQuery = {
302       /**we have no nested object here
303        * so we couldn't even get any nested or detailed data
304        * we only get true or false
305        */
306       query: `

307         mutation {
308           deletePost(id: "${postId}")
309         }
310       `
311     };
312     fetch('http://localhost:8080/graphql', {
313       method: 'POST',
314       headers: {
315         Authorization: 'Bearer ' + this.props.token,
316         'Content-Type': 'application/json'
317       },
318       body: JSON.stringify(graphqlQuery)
319     })
320     .then(res => {
321       return res.json();
322     })
323     .then(resData => {
324       if(resData.errors) {
325         throw new Error('Deleting the post failed!');
326       }
327       console.log(resData);
328       this.loadPosts();
329       // this.setState(prevState => {
330       //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
331       //   return { posts: updatedPosts, postsLoading: false };
332       // });
333     })
334     .catch(err => {
335       console.log(err);
336       this.setState({ postsLoading: false });
337     });
338   };
339
340   errorHandler = () => {
341     this.setState({ error: null });
342   };
343
344   catchError = error => {
345     this.setState({ error: error });
346   };
347
348   render() {
349     return (
350       <Fragment>
351         <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
```

```
352 <FeedEdit
353   editing={this.state.isEditing}
354   selectedPost={this.state.editPost}
355   loading={this.state.editLoading}
356   onCancelEdit={this.cancelEditHandler}
357   onFinishEdit={this.finishEditHandler}
358 />
359 <section className="feed__status">
360   <form onSubmit={this.statusUpdateHandler}>
361     <Input
362       type="text"
363       placeholder="Your status"
364       control="input"
365       onChange={this.statusInputChangeHandler}
366       value={this.state.status}
367     />
368     <Button mode="flat" type="submit">
369       Update
370     </Button>
371   </form>
372 </section>
373 <section className="feed__control">
374   <Button mode="raised" design="accent" onClick={this.newPostHandler}>
375     New Post
376   </Button>
377 </section>
378 <section className="feed">
379   {this.state.postsLoading && (
380     <div style={{ textAlign: 'center', marginTop: '2rem' }}>
381       <Loader />
382     </div>
383   )}
384   {this.state.posts.length <= 0 && !this.state.postsLoading ? (
385     <p style={{ textAlign: 'center' }}>No posts found.</p>
386   ) : null}
387   {!this.state.postsLoading && (
388     <Paginator
389       onPrevious={this.loadPosts.bind(this, 'previous')}
390       onNext={this.loadPosts.bind(this, 'next')}
391       lastPage={Math.ceil(this.state.totalPosts / 2)}
392       currentPage={this.state.postPage}
393     >
394       {this.state.posts.map(post => (
395         <Post
396           key={post._id}
397           id={post._id}
398           author={post.creator.name}
399           date={new Date(post.createdAt).toLocaleDateString('en-US')}
400           title={post.title}
401           image={post.imageUrl}
402           content={post.content}
403           onStartEdit={this.startEditPostHandler.bind(this, post._id)}
404           onDelete={this.deletePostHandler.bind(this, post._id)}
405         />
406       ))}
407     </Paginator>
```

```
408         })
409     </section>
410   </Fragment>
411 );
412 }
413 }
414
415 export default Feed;
416
```

* Chapter 439: Using Variables

1. update

- ./src/pages/Feed/Feed.js(f)
- ./src/pages/Feed/SinglePost/SinglePost.js(f)
- App.js(f)

* Chapter 440: Fixing A Pagination Bug

1. update
- ./src/pages/Feed/Feed.js(f)
- ./src/pages/Feed/SinglePost/SinglePost.js(f)
- App.js(f)


```
1 //./src/pages/Feed/Feed.js(f)
2
3 import React, { Component, Fragment } from 'react';
4
5 import Post from '../../../../../components/Feed/Post/Post';
6 import Button from '../../../../../components/Button/Button';
7 import FeedEdit from '../../../../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../../../../components/Form/Input/Input';
9 import Paginator from '../../../../../components/Paginator/Paginator';
10 import Loader from '../../../../../components/Loader/Loader';
11 import ErrorHandler from '../../../../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     const graphqlQuery = {
28       query: `

29         {
30           user {
31             status
32           }
33         }
34       `
35     };
36     fetch('http://localhost:8080/graphql', {
```

```

37     method: 'POST',
38     headers: {
39       Authorization: 'Bearer ' + this.props.token,
40       'Content-Type': 'application/json'
41     },
42     body: JSON.stringify(graphqlQuery)
43   })
44   .then(res => {
45     return res.json();
46   })
47   .then(resData => {
48     if (resData.errors) {
49       throw new Error('Fetching status failed!');
50     }
51     this.setState({ status: resData.data.user.status });
52   })
53   .catch(this.catchError);
54
55   this.loadPosts();
56 }
57
58 loadPosts = direction => {
59   if (direction) {
60     this.setState({ postsLoading: true, posts: [] });
61   }
62   let page = this.state.postPage;
63   if (direction === 'next') {
64     page++;
65     this.setState({ postPage: page });
66   }
67   if (direction === 'previous') {
68     page--;
69     this.setState({ postPage: page });
70   }
71   /**you also remember that
72    * for mutations we had to add mutation here
73    * but for queries, we didn't have to add 'query' here
74    * and if you would have added a query like this,
75    *
76    *   query: `query {posts{}}()`
77    *
78    * we would have got an error
79    * now i will add it
80    * because i will add something else
81    * i will give this query a name
82    * that doesn't make a difference.
83    * it doesn't make it behave differently
84    * it will help in error messages
85    * but that name also allows us to do something else
86    * so here we are getting all posts
87    * here i will named as 'FetchPosts'
88    * with this name assigned,
89    * i can add parentheses after this name to define
90    * which variables this query will use
91    * and we create such a variable with a dollar sign
92    * but then no curly braces

```

```
93 * and then the name of the variable you wanna use
94 * and that name is up to you
95 * i will name it 'page'
96 *
97 * 'query FetchPosts($page: Int)'
98 * this graphql syntax will be parsed on the server
99 * this is not javascript code that runs on a client
100 * this will only tell our graphql server
101 * that we have a query which will use an internal variable
102 * now i'm saying internal to a question is where do we use that '$page'
103 * we use it in the place where we have dynamic variable value
104 * and that would be where we currently inject our value.
105 */
106 const graphqlQuery = {
107   query: `
108     query FetchPosts($page: Int) {
109       posts(page: $page) {
110         posts {
111           _id
112           title
113           content
114           imageUrl
115           creator {
116             name
117           }
118           createdAt
119         }
120         totalPosts
121       }
122     }
123   `,
124   /**
125    * the only question is
126    * how do we get our page variable in javascript
127    * into page variable in graphql '$page'
128    * we add a 2nd property to that query object we are creating
129    * 2nd property is 'variables' which has to be named 'variables'
130    * and 1st property name has to be 'query'
131    *
132    * 'variables' is the object
133    * where we can assign values to the variables we pass into our 'query'
134    * it should be just 'page' without the $
135    * and then my javascript variable or value
136    * 'page' on the left is page in '$page'
137    * 'page' on the right is page the javascript variable in 'page++' or 'page--'
138    *
139   variables: {
140     page: page
141   }
142 };
143 fetch('http://localhost:8080/graphql', {
144   method: 'POST',
145   headers: {
146     Authorization: 'Bearer ' + this.props.token,
147     'Content-Type': 'application/json'
148   },

```

```
149     body: JSON.stringify(graphqlQuery)
150   })
151   .then(res => {
152     return res.json();
153   })
154   .then(resData => {
155     if (resData.errors) {
156       throw new Error('Fetching posts failed!');
157     }
158     this.setState({
159       posts: resData.data.posts.posts.map(post => {
160         return {
161           ...post,
162           imagePath: post.imageUrl
163         };
164       }),
165       totalPosts: resData.data.posts.totalPosts,
166       postsLoading: false
167     });
168   })
169   .catch(this.catchError);
170 };
171
172 statusUpdateHandler = event => {
173   event.preventDefault();
174   const graphqlQuery = {
175     query: `
176       mutation UpdateUserStatus($userStatus: String!) {
177         updateStatus(status: $userStatus) {
178           status
179         }
180       }
181     `,
182     variables: {
183       userStatus: this.state.status
184     }
185   };
186   fetch('http://localhost:8080/graphql', {
187     method: 'POST',
188     headers: {
189       Authorization: 'Bearer ' + this.props.token,
190       'Content-Type': 'application/json'
191     },
192     body: JSON.stringify(graphqlQuery)
193   })
194   .then(res => {
195     return res.json();
196   })
197   .then(resData => {
198     if (resData.errors) {
199       throw new Error('Fetching posts failed!');
200     }
201     console.log(resData);
202   })
203   .catch(this.catchError);
204};
```

```

205
206 newPostHandler = () => {
207     this.setState({ isEditing: true });
208 };
209
210 startEditPostHandler = postId => {
211     this.setState(prevState => {
212         const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
213
214         return {
215             isEditing: true,
216             editPost: loadedPost
217         };
218     });
219 };
220
221 cancelEditHandler = () => {
222     this.setState({ isEditing: false, editPost: null });
223 };
224
225 finishEditHandler = postData => {
226     this.setState({
227         editLoading: true
228     });
229     const formData = new FormData();
230     formData.append('image', postData.image);
231     if (this.state.editPost) {
232         formData.append('oldPath', this.state.editPost.imagePath);
233     }
234     fetch('http://localhost:8080/post-image', {
235         method: 'PUT',
236         headers: {
237             Authorization: 'Bearer ' + this.props.token
238         },
239         body: formData
240     })
241         .then(res => res.json())
242         .then(fileResData => {
243             /**we at least pass that text to go through that check */
244             const imageUrl = fileResData.filePath || 'undefined';
245             /**the type you assign on your front-end for your variables
246             * have to match the type in your backend ./graphql/schema.js(b)
247             * whether exclamation mark are there or not.
248             */
249             let graphqlQuery = {
250                 query: `
251                     mutation CreateNewPost($title: String!, $content: String!, $imageUrl: String!) {
252                         createPost(postInput: {title: $title, content: $content, imageUrl: $imageUrl}) {
253                             _id
254                             title
255                             content
256                             imageUrl
257                             creator {
258                                 name
259                             }
260                             createdAt

```

```

261         }
262     }
263     `,
264     variables: {
265       title: postData.title,
266       content: postData.content,
267       imageUrl: imageUrl
268     }
269   };
270
271   if (this.state.editPost) {
272     graphqlQuery = {
273       query: `
274         mutation UpdateExistingPost($postId: ID!, $title: String!, $content: String!,
275           $imageUrl: String!) {
276           updatePost(id: $postId, postInput: {title: $title, content: $content,
277             imageUrl: $imageUrl}) {
278             _id
279             title
280             content
281             imageUrl
282             creator {
283               name
284             }
285           }
286         `,
287         variables: {
288           postId: this.state.editPost._id,
289           title: postData.title,
290           content: postData.content,
291           imageUrl: imageUrl
292         }
293       };
294     }
295
296     return fetch('http://localhost:8080/graphql', {
297       method: 'POST',
298       body: JSON.stringify(graphqlQuery),
299       headers: {
300         Authorization: 'Bearer ' + this.props.token,
301         'Content-Type': 'application/json'
302       }
303     });
304   })
305   .then(res => {
306     return res.json();
307   })
308   .then(resData => {
309     if (resData.errors && resData.errors[0].status === 422) {
310       throw new Error(
311         "Validation failed. Make sure the email address isn't used yet!"
312       );
313     }
314     if (resData.errors) {

```

```
315         throw new Error('User login failed!');
316     }
317     let resDataField = 'createPost';
318     if (this.state.editPost) {
319         resDataField = 'updatePost';
320     }
321     const post = {
322         _id: resData.data[resDataField]._id,
323         title: resData.data[resDataField].title,
324         content: resData.data[resDataField].content,
325         creator: resData.data[resDataField].creator,
326         createdAt: resData.data[resDataField].createdAt,
327         imagePath: resData.data[resDataField].imageUrl
328     };
329 /**
330 * need to make sure
331 * that i increase the amount of total posts in our post
332 * in our state in react here where we manage how many posts we have on that page,
333 *
334 *     totalPosts: resData.data.posts.totalPosts
335 *
336 * i set total posts to a different amount of posts
337 * when i fetch all posts
338 * where i get that total posts information.
339 * when adding a new element,
340 * i know that total posts will be the old total posts + 1
341 */
342     this.setState(prevState => {
343         let updatedPosts = [...prevState.posts];
344         let updatedTotalPosts = prevState.totalPosts;
345         if (prevState.editPost) {
346             const postIndex = prevState.posts.findIndex(
347                 p => p._id === prevState.editPost._id
348             );
349             updatedPosts[postIndex] = post;
350         } else {
351             updatedTotalPosts++;
352             if (prevState.posts.length >= 2) {
353                 updatedPosts.pop();
354             }
355             updatedPosts.unshift(post);
356         }
357         return {
358             posts: updatedPosts,
359             isEditing: false,
360             editPost: null,
361             editLoading: false,
362             totalPosts: updatedTotalPosts
363         };
364     });
365 })
366 .catch(err => {
367     console.log(err);
368     this.setState({
369         isEditing: false,
370         editPost: null,
```

```
371     editLoading: false,
372     error: err
373   });
374 );
375 );
376
377 statusInputChangeHandler = (input, value) => {
378   this.setState({ status: value });
379 };
380
381 deletePostHandler = postId => {
382   this.setState({ postsLoading: true });
383   const graphqlQuery = {
384     query: `
385       mutation {
386         deletePost(id: "${postId}")
387       }
388     `
389   };
390   fetch('http://localhost:8080/graphql', {
391     method: 'POST',
392     headers: {
393       Authorization: 'Bearer ' + this.props.token,
394       'Content-Type': 'application/json'
395     },
396     body: JSON.stringify(graphqlQuery)
397   })
398     .then(res => {
399       return res.json();
400     })
401     .then(resData => {
402       if (resData.errors) {
403         throw new Error('Deleting the post failed!');
404       }
405       console.log(resData);
406       this.loadPosts();
407       // this.setState(prevState => {
408       //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
409       //   return { posts: updatedPosts, postsLoading: false };
410       // });
411     })
412     .catch(err => {
413       console.log(err);
414       this.setState({ postsLoading: false });
415     });
416 };
417
418 errorHandler = () => {
419   this.setState({ error: null });
420 };
421
422 catchError = error => {
423   this.setState({ error: error });
424 };
425
426 render() {
```

```
427 return (
428   <Fragment>
429     <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
430     <FeedEdit
431       editing={this.state.isEditing}
432       selectedPost={this.state.editPost}
433       loading={this.state.editLoading}
434       onCancelEdit={this.cancelEditHandler}
435       onFinishEdit={this.finishEditHandler}
436     />
437     <section className="feed__status">
438       <form onSubmit={this.statusUpdateHandler}>
439         <Input
440           type="text"
441           placeholder="Your status"
442           control="input"
443           onChange={this.statusInputChangeHandler}
444           value={this.state.status}
445         />
446         <Button mode="flat" type="submit">
447           Update
448         </Button>
449       </form>
450     </section>
451     <section className="feed__control">
452       <Button mode="raised" design="accent" onClick={this.newPostHandler}>
453         New Post
454       </Button>
455     </section>
456     <section className="feed">
457       {this.state.postsLoading && (
458         <div style={{ textAlign: 'center', marginTop: '2rem' }}>
459           <Loader />
460         </div>
461       )}
462       {this.state.posts.length <= 0 && !this.state.postsLoading ? (
463         <p style={{ textAlign: 'center' }}>No posts found.</p>
464       ) : null}
465       {!this.state.postsLoading && (
466         <Paginator
467           onPrevious={this.loadPosts.bind(this, 'previous')}
468           onNext={this.loadPosts.bind(this, 'next')}
469           lastPage={Math.ceil(this.state.totalPosts / 2)}
470           currentPage={this.state.postPage}
471         >
472           {this.state.posts.map(post => (
473             <Post
474               key={post._id}
475               id={post._id}
476               author={post.creator.name}
477               date={new Date(post.createdAt).toLocaleDateString('en-US')}
478               title={post.title}
479               image={post.imageUrl}
480               content={post.content}
481               onStartEdit={this.startEditPostHandler.bind(this, post._id)}
482               onDelete={this.deletePostHandler.bind(this, post._id)}>
```

```

483           />
484       })
485     </Paginator>
486   )}
487   </section>
488 </Fragment>
489 );
490 }
491 }
492
493 export default Feed;

```

```

1 //./src/pages/Feed/SinglePost/SinglePost.js(f)
2
3 import React, { Component } from 'react';
4
5 import Image from ' ../../components/Image/Image';
6 import './SinglePost.css';
7
8 class SinglePost extends Component {
9   state = {
10   title: '',
11   author: '',
12   date: '',
13   image: '',
14   content: ''
15 };
16
17 componentDidMount() {
18   const postId = this.props.match.params.postId;
19   const graphqlQuery = {
20     /**i also have a query
21      * where i can check the value
22      * when i get a single post
23      * so i will give it as a name by adding 'query'
24      * and the name 'FetchSinglePost'
25      */
26     query: `query FetchSinglePost($postId: ID!) {
27       post(id: $postId) {
28         title
29         content
30         imageUrl
31         creator {
32           name
33         }
34         createdAt
35       }
36     }
37   `,
38   variables: {
39     postId: postId
40   }
41 };
42 fetch('http://localhost:8080/graphql', {
43   method: 'POST',
44   headers: {
45     Authorization: 'Bearer ' + this.props.token,

```

```

46     'Content-Type': 'application/json'
47   },
48   body: JSON.stringify(graphqlQuery)
49 )
50 .then(res => {
51   return res.json();
52 })
53 .then(resData => {
54   if (resData.errors) {
55     throw new Error('Fetching post failed!');
56   }
57   this.setState({
58     title: resData.data.post.title,
59     author: resData.data.post.creator.name,
60     image: 'http://localhost:8080/' + resData.data.post.imageUrl,
61     date: new Date(resData.data.post.createdAt).toLocaleDateString('en-US'),
62     content: resData.data.post.content
63   });
64 })
65 .catch(err => {
66   console.log(err);
67 });
68 }
69
70 render() {
71   return (
72     <section className="single-post">
73       <h1>{this.state.title}</h1>
74       <h2>
75         Created by {this.state.author} on {this.state.date}
76       </h2>
77       <div className="single-post__image">
78         <Image contain imageUrl={this.state.image} />
79       </div>
80       <p>{this.state.content}</p>
81     </section>
82   );
83 }
84 }
85
86 export default SinglePost;
87

```

```

1 //App.js(f)
2
3 import React, { Component, Fragment } from 'react';
4 import { Route, Switch, Redirect, withRouter } from 'react-router-dom';
5
6 import Layout from './components/Layout/Layout';
7 import Backdrop from './components/Backdrop/Backdrop';
8 import Toolbar from './components/Toolbar/Toolbar';
9 import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
10 import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
11 import ErrorHandler from './components/ErrorHandler/ErrorHandler';
12 import FeedPage from './pages/Feed/Feed';
13 import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
14 import LoginPage from './pages/Auth/Login';

```

```
15 import SignupPage from './pages/Auth/Signup';
16 import './App.css';
17
18 class App extends Component {
19   state = {
20     showBackdrop: false,
21     showMobileNav: false,
22     isAuth: false,
23     token: null,
24     userId: null,
25     authLoading: false,
26     error: null
27   };
28
29   componentDidMount() {
30     const token = localStorage.getItem('token');
31     const expiryDate = localStorage.getItem('expiryDate');
32     if (!token || !expiryDate) {
33       return;
34     }
35     if (new Date(expiryDate) <= new Date()) {
36       this.logoutHandler();
37       return;
38     }
39     const userId = localStorage.getItem('userId');
40     const remainingMilliseconds =
41       new Date(expiryDate).getTime() - new Date().getTime();
42     this.setState({ isAuth: true, token: token, userId: userId });
43     this.setAutoLogout(remainingMilliseconds);
44   }
45
46   mobileNavHandler = isOpen => {
47     this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
48   };
49
50   backdropClickHandler = () => {
51     this.setState({ showBackdrop: false, showMobileNav: false, error: null });
52   };
53
54   logoutHandler = () => {
55     this.setState({ isAuth: false, token: null });
56     localStorage.removeItem('token');
57     localStorage.removeItem('expiryDate');
58     localStorage.removeItem('userId');
59   };
60
61   loginHandler = (event, authData) => {
62     event.preventDefault();
63     const graphqlQuery = {
64       query: `query UserLogin($email: String!, $password: String!) {
65       login(email: $email, password: $password) {
66         token
67         userId
68       }
69     }
70   
```

```
71     `,
72     variables: {
73       email: authData.email,
74       password: authData.password
75     }
76   };
77   this.setState({ authLoading: true });
78   fetch('http://localhost:8080/graphql', {
79     method: 'POST',
80     headers: {
81       'Content-Type': 'application/json'
82     },
83     body: JSON.stringify(graphqlQuery)
84   })
85   .then(res => {
86     return res.json();
87   })
88   .then(resData => {
89     if (resData.errors && resData.errors[0].status === 422) {
90       throw new Error(
91         "Validation failed. Make sure the email address isn't used yet!"
92       );
93     }
94     if (resData.errors) {
95       throw new Error('User login failed!');
96     }
97     console.log(resData);
98     this.setState({
99       isAuthenticated: true,
100      token: resData.data.login.token,
101      authLoading: false,
102      userId: resData.data.login.userId
103    });
104    localStorage.setItem('token', resData.data.login.token);
105    localStorage.setItem('userId', resData.data.login.userId);
106    const remainingMilliseconds = 60 * 60 * 1000;
107    const expiryDate = new Date(
108      new Date().getTime() + remainingMilliseconds
109    );
110    localStorage.setItem('expiryDate', expiryDate.toISOString());
111    this.setAutoLogout(remainingMilliseconds);
112  })
113  .catch(err => {
114    console.log(err);
115    this.setState({
116      isAuthenticated: false,
117      authLoading: false,
118      error: err
119    });
120  });
121};

122 signupHandler = (event, authData) => {
123   event.preventDefault();
124   this.setState({ authLoading: true });
125   const graphqlQuery = {
```

```
127 query: ` 
128   mutation CreateNewUser($email: String!, $name: String!, $password: String!) { 
129     createUser(userInput: {email: $email, name: $name, password: $password}) { 
130       _id 
131       email 
132     } 
133   } 
134   `, 
135   variables: { 
136     email: authData.signupForm.email.value, 
137     name: authData.signupForm.name.value, 
138     password: authData.signupForm.password.value 
139   } 
140 }; 
141 fetch('http://localhost:8080/graphql', { 
142   method: 'POST', 
143   headers: { 
144     'Content-Type': 'application/json' 
145   }, 
146   body: JSON.stringify(graphqlQuery) 
147 }) 
148   .then(res => { 
149     return res.json(); 
150   }) 
151   .then(resData => { 
152     if (resData.errors && resData.errors[0].status === 422) { 
153       throw new Error( 
154         "Validation failed. Make sure the email address isn't used yet!" 
155       ); 
156     } 
157     if (resData.errors) { 
158       throw new Error('User creation failed!'); 
159     } 
160     console.log(resData); 
161     this.setState({ isAuth: false, authLoading: false }); 
162     this.props.history.replace('/'); 
163   }) 
164   .catch(err => { 
165     console.log(err); 
166     this.setState({ 
167       isAuth: false, 
168       authLoading: false, 
169       error: err 
170     }); 
171   }); 
172 }; 
173 
174 setAutoLogout = milliseconds => { 
175   setTimeout(() => { 
176     this.logoutHandler(); 
177   }, milliseconds); 
178 }; 
179 
180 errorHandler = () => { 
181   this.setState({ error: null }); 
182 };
```

```
183
184 render() {
185   let routes = (
186     <Switch>
187       <Route
188         path="/"
189         exact
190         render={props => (
191           <LoginPage
192             {...props}
193             onLogin={this.loginHandler}
194             loading={this.state.authLoading}
195           />
196         )}
197       />
198       <Route
199         path="/signup"
200         exact
201         render={props => (
202           <SignupPage
203             {...props}
204             onSignup={this.signupHandler}
205             loading={this.state.authLoading}
206           />
207         )}
208       />
209       <Redirect to="/" />
210     </Switch>
211   );
212   if (this.state.isAuthenticated) {
213     routes = (
214       <Switch>
215         <Route
216           path="/"
217           exact
218           render={props => (
219             <FeedPage userId={this.state.userId} token={this.state.token} />
220           )}
221         />
222         <Route
223           path="/:postId"
224           render={props => (
225             <SinglePostPage
226               {...props}
227               userId={this.state.userId}
228               token={this.state.token}
229             />
230           )}
231         />
232         <Redirect to="/" />
233       </Switch>
234   );
235 }
236 return (
237   <Fragment>
238     {this.state.showBackdrop && (
```

```

239         <Backdrop onClick={this.backdropClickHandler} />
240     )}
241     <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
242     <Layout
243       header={
244         <Toolbar>
245           <MainNavigation
246             onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
247             onLogout={this.logoutHandler}
248             isAuthenticated={this.state.isAuthenticated}
249           />
250         </Toolbar>
251     }
252     mobileNav={
253       <MobileNavigation
254         open={this.state.showMobileNav}
255         mobile
256         onChooseItem={this.mobileNavHandler.bind(this, false)}
257         onLogout={this.logoutHandler}
258         isAuthenticated={this.state.isAuthenticated}
259       />
260     }
261   />
262   {routes}
263 </Fragment>
264 );
265 }
266 }
267
268 export default withRouter(App);
269

```

* Chapter 441: Wrap Up

