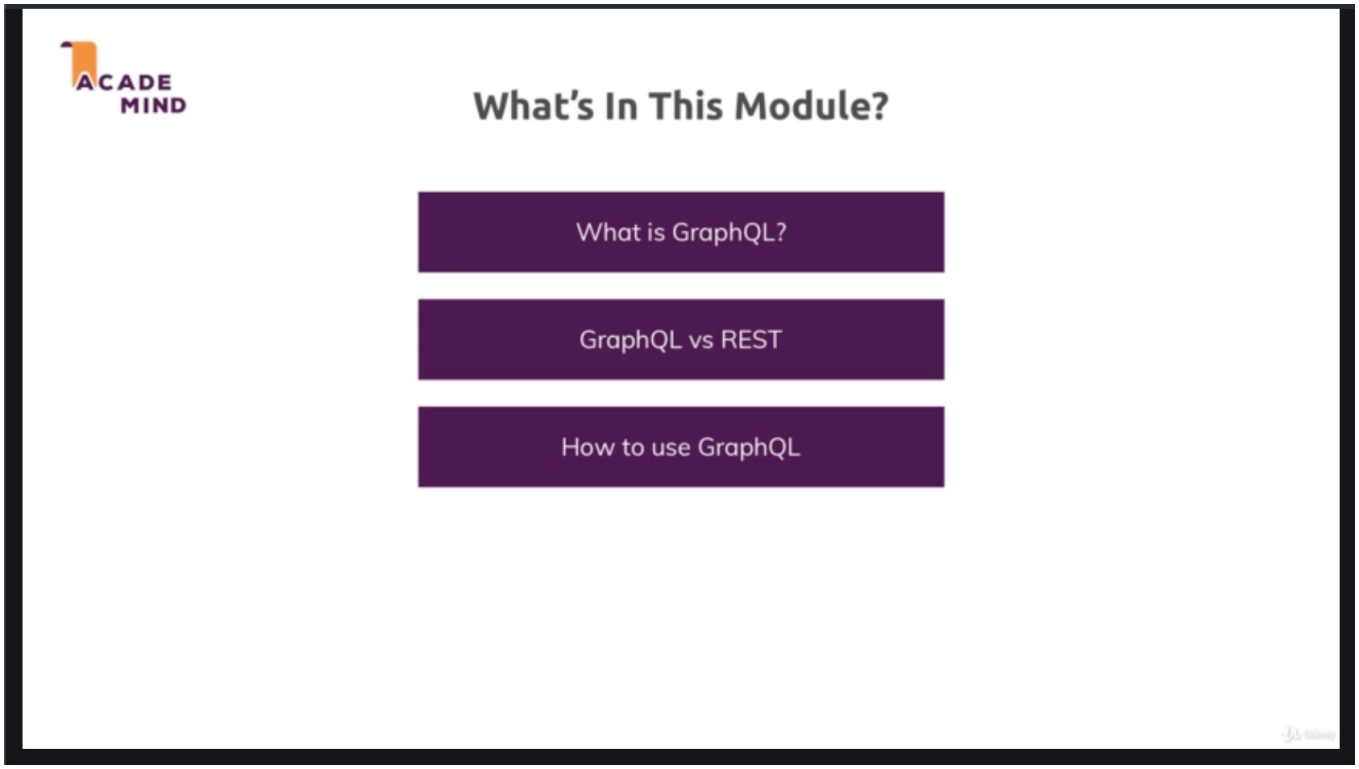
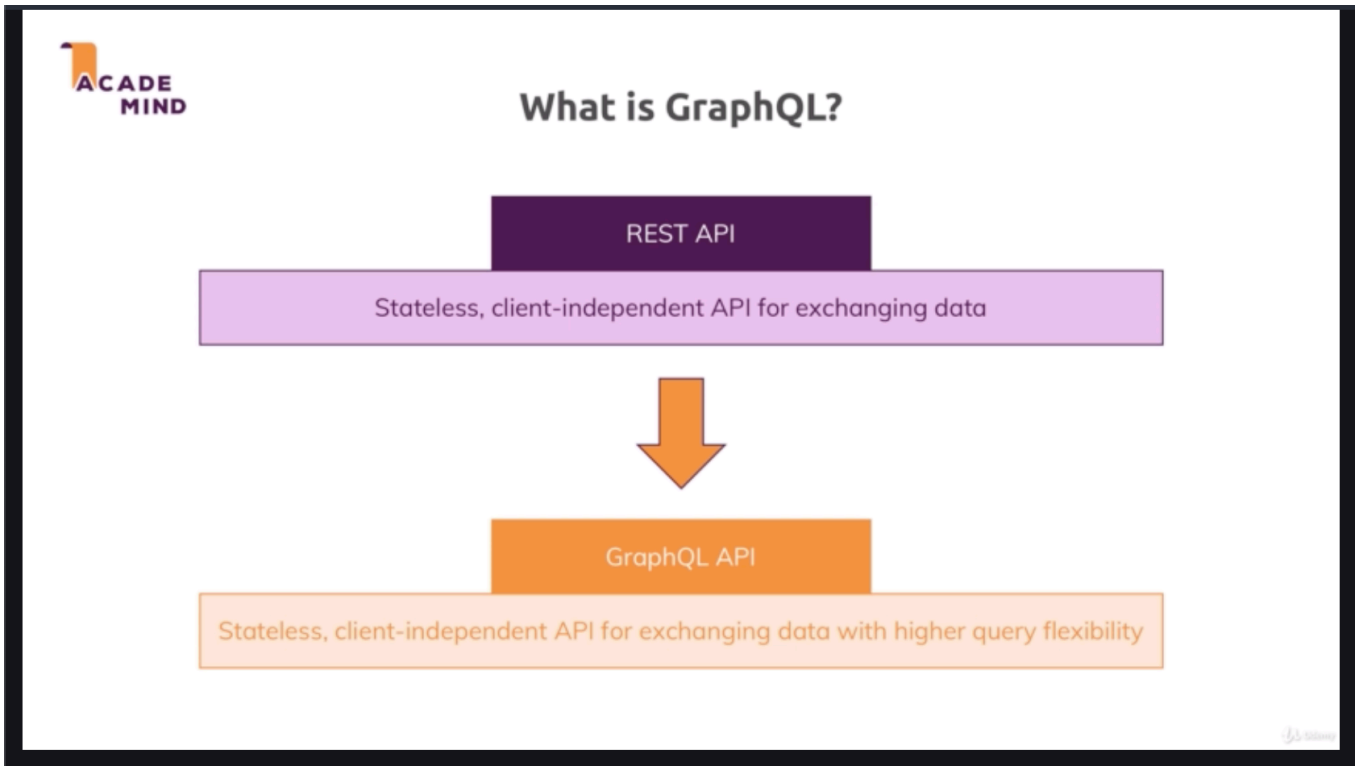
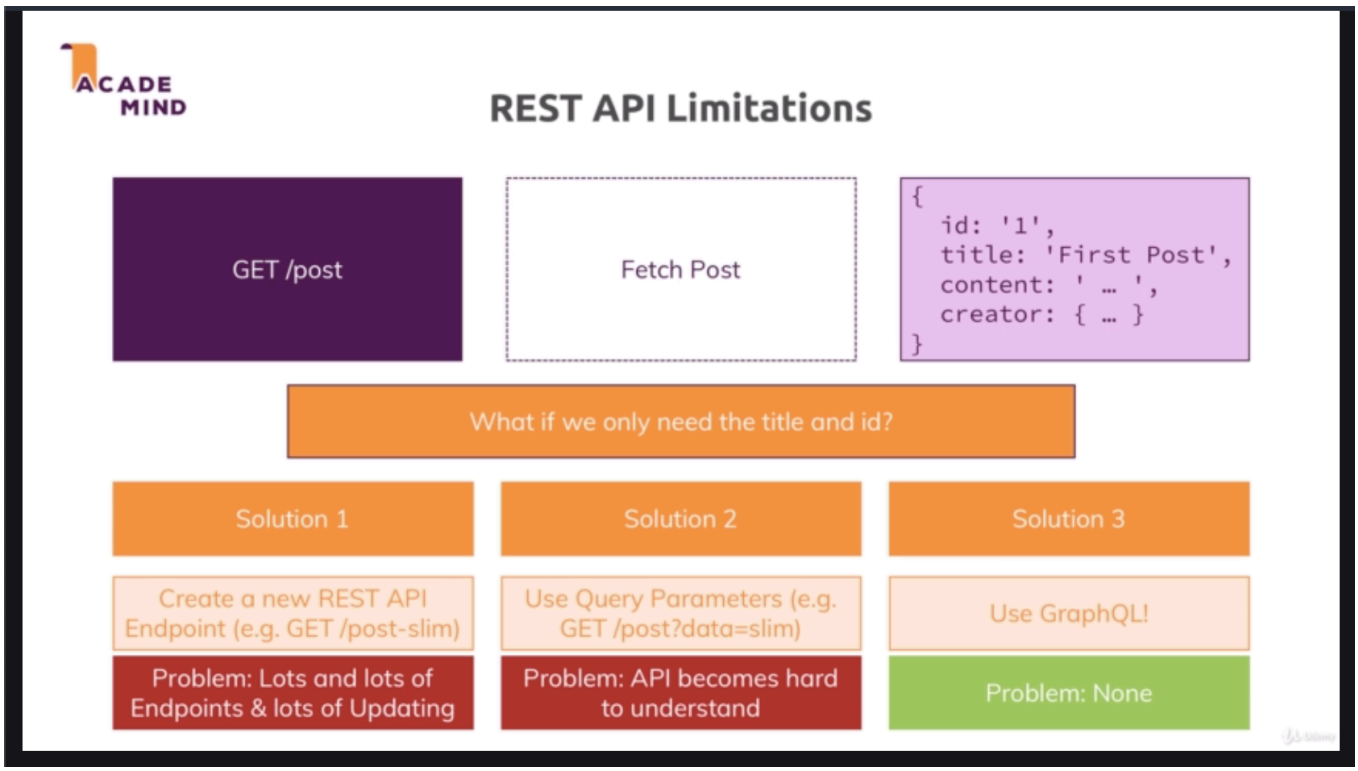
28. Working With GraphQL

* Chapter 415: Module Introduction



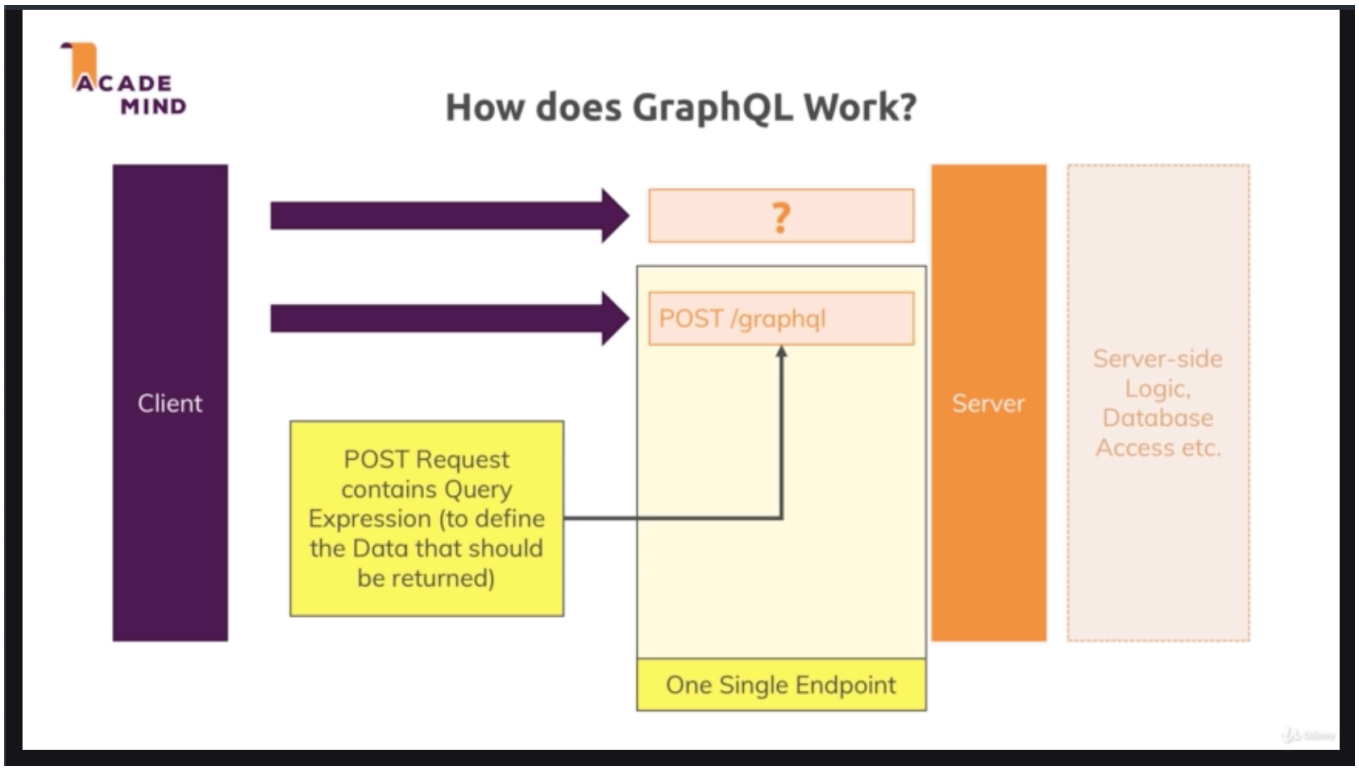
* Chapter 416: What Is GraphQL?



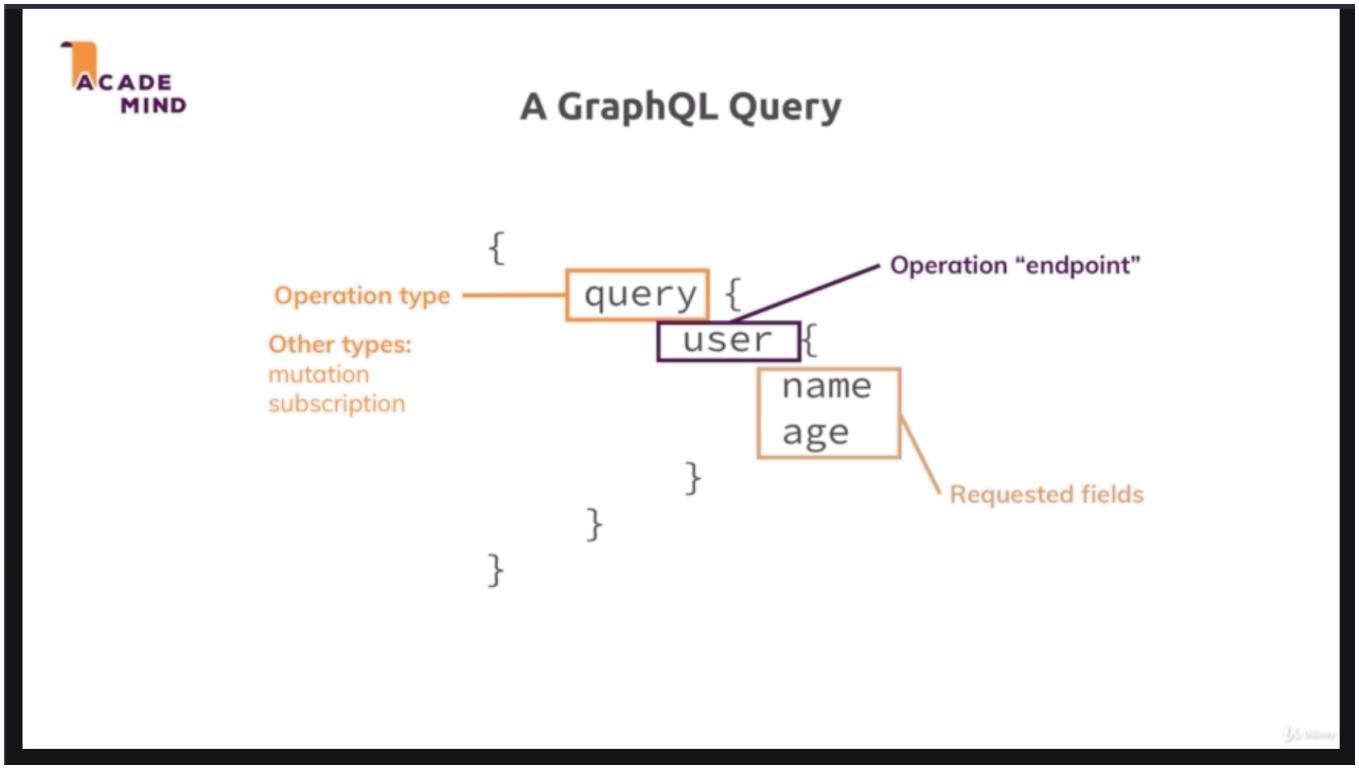


- what if we don't need the content or we don't need to creator data? we can have many scenarios where we use one at the same end point in our front end application. so in our single page application or our mobile application and in one place on one page we might need title and content, on other page, 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.

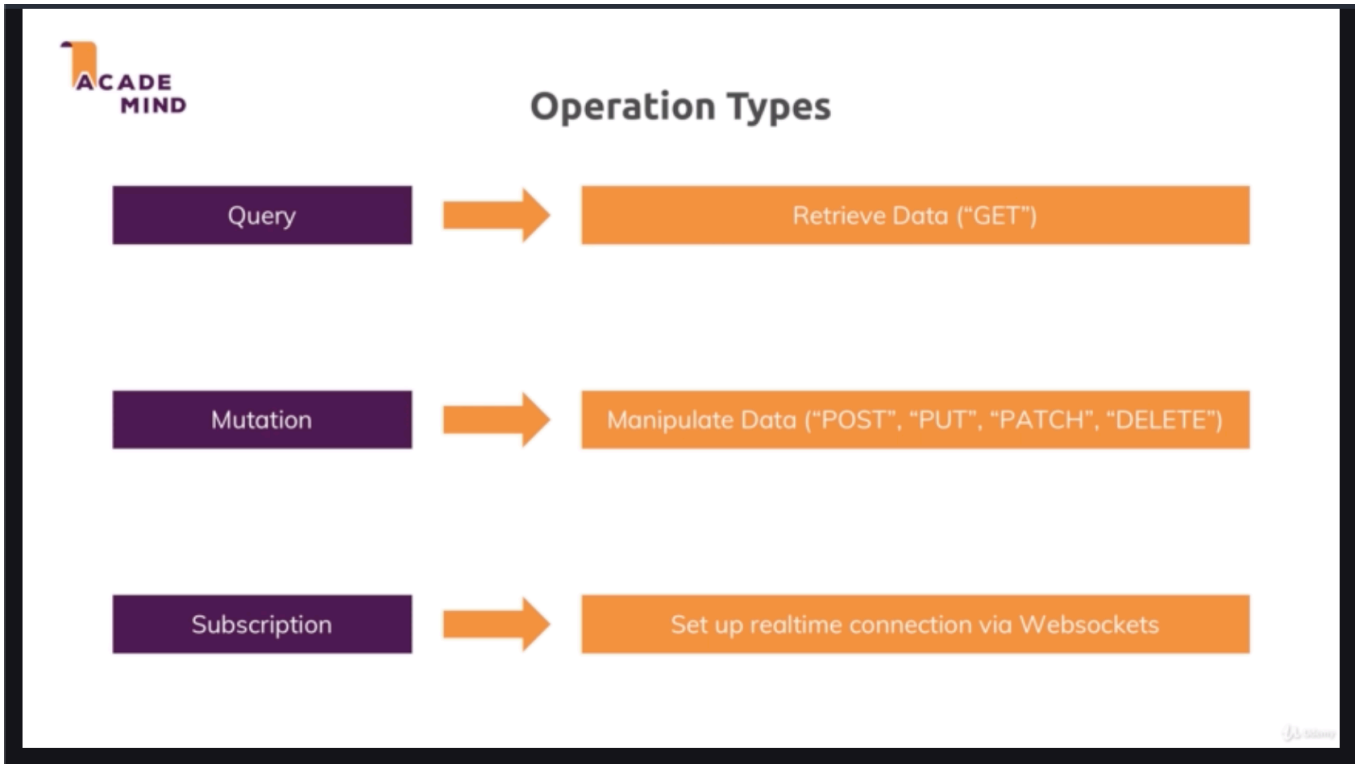


- what do you send from client to server in that graphQL world? you only send one kind of request and that is POST/graphql. so you only have one single end point where you send your HTTP requests to even for getting data. because for POST request, you can add a request body and that request body will 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.

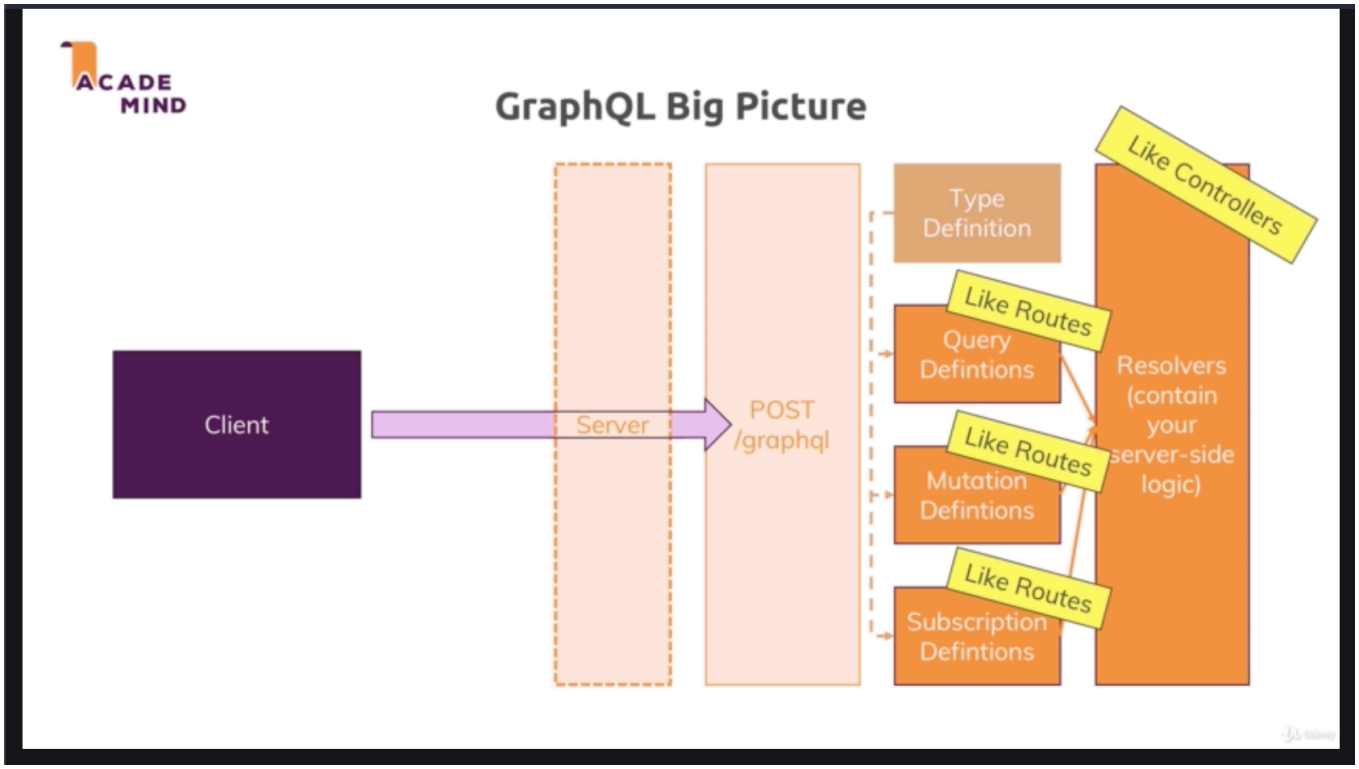


- it's an JSON object-like structure where you have an operation type. query before getting data you also have other types like mutation for editing or deleting or inserting data or subscription for setting up real time data subscriptions using websockets.

- 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.

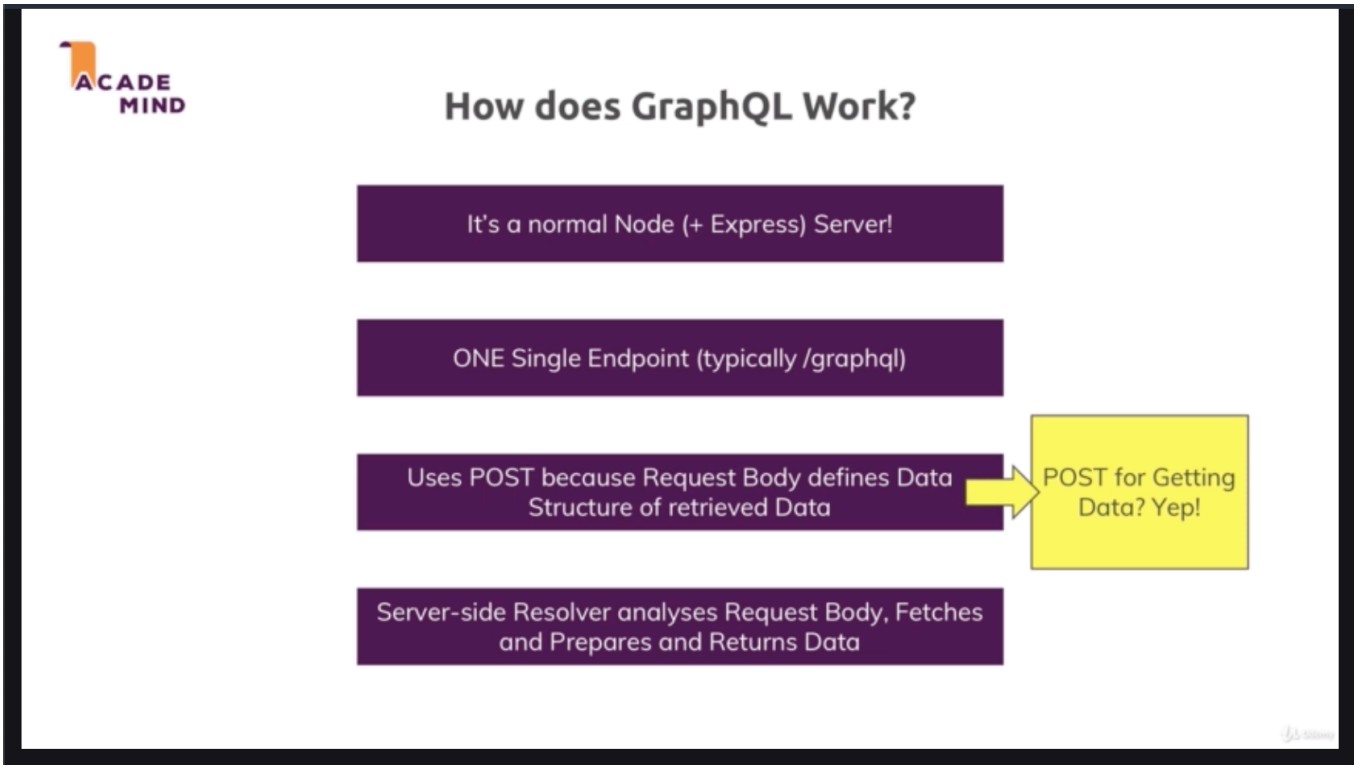


- we retrieved data and we use a POST request for that. but if we wanna compare it to the REST API world, then this would be your equivalent to sending a GET request to some path there.
- we also got mutations which basically are used for everything that changes data.
- subscription set up real time connections via websockets.



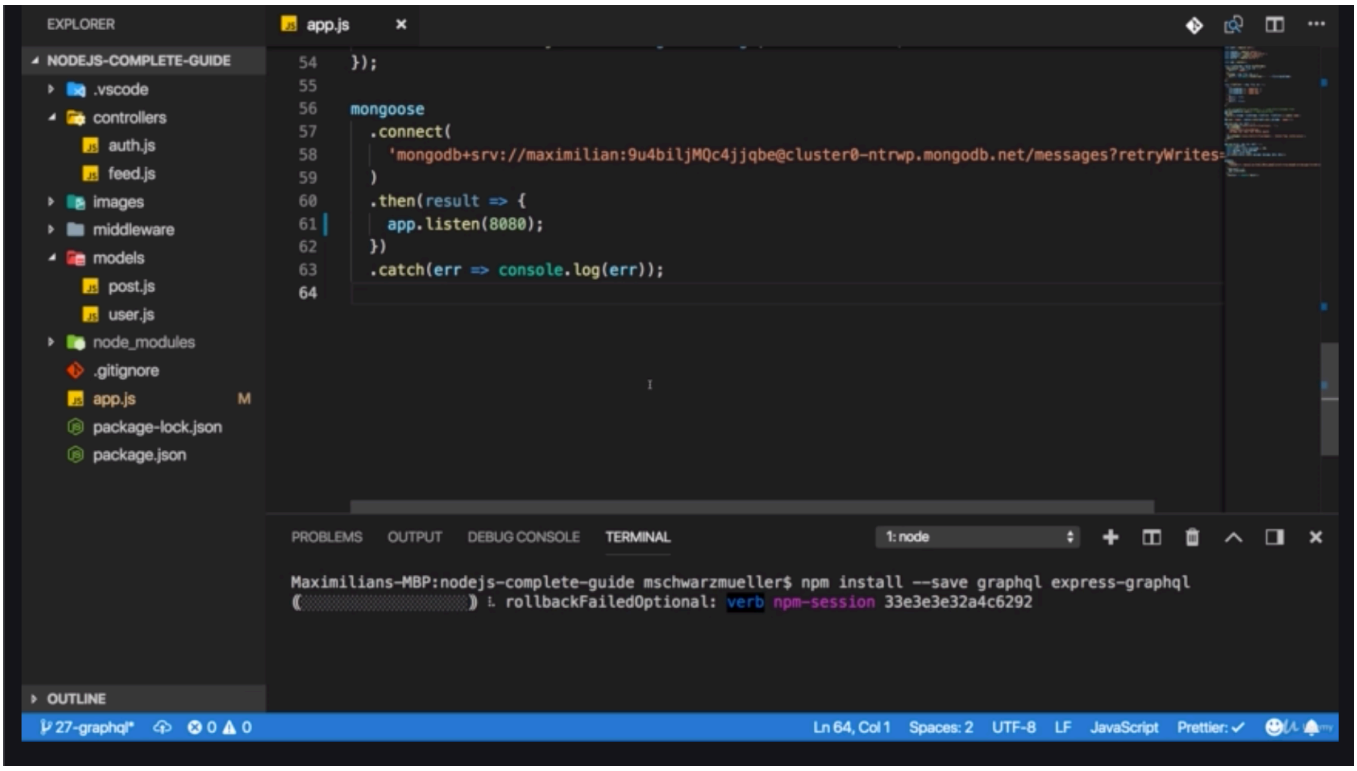
- to sum it up, in a big picture, we have our client we send a request to that single graphQL endpoint on our server. and then there and this is the part you will do with me in this module. and there you set up your

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

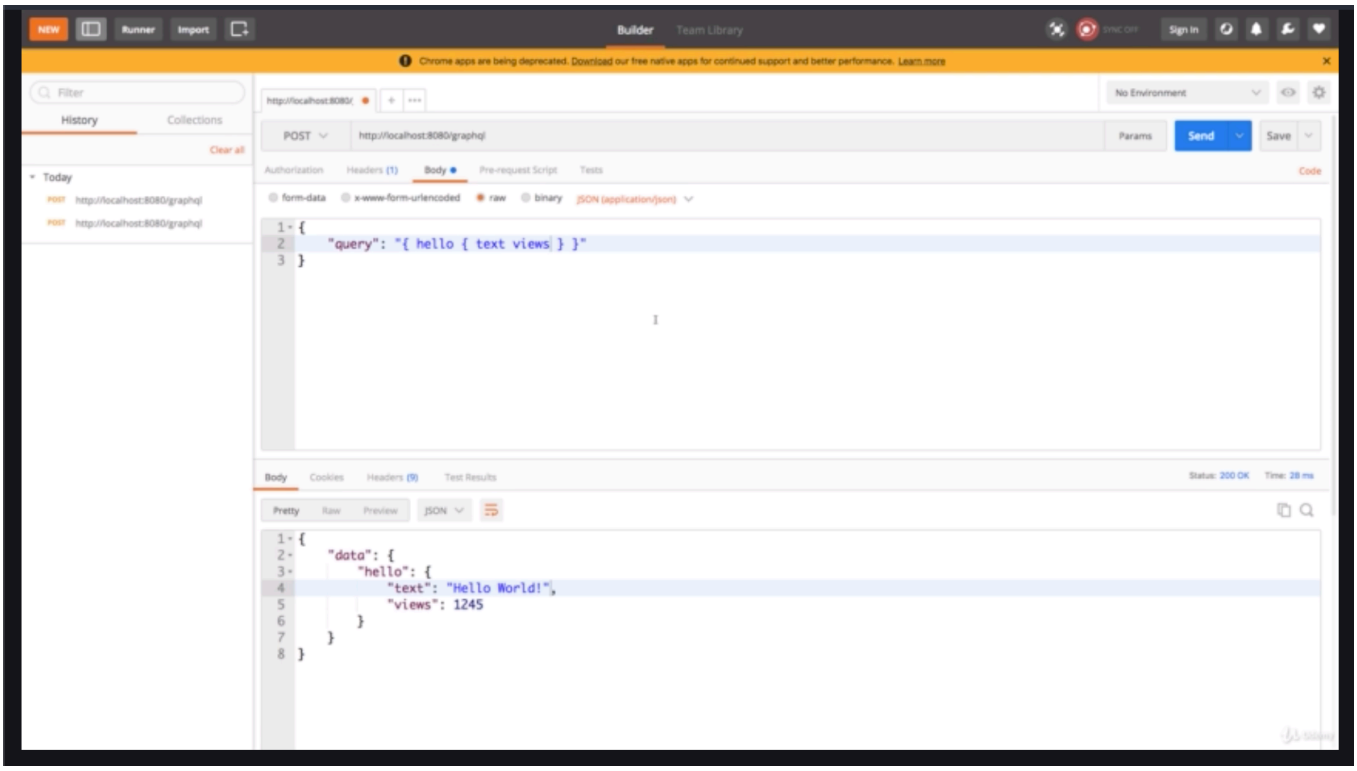


* Chapter 417: Understanding The Setup & Writing Our First Query

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)



- 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, amputations and so on we wanna allow
- and 2nd is called 'express-graphql' which install a simple server that will do the parsing of incoming requests.



- we will test with postman to test POST. you send a javascript object with a queriy 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 you 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

```
//app.js(b)

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const multer = require('multer');
/**i will remove feedRoutes, authRoutes
 * because we will have no more routes will not set up routes anymore
 * we will use graphQL endpoint instead.
 */
const graphqlHttp = require('express-graphql');

const graphSchema = require('./graphql/schema');
const graphResolver = require('./graphql/resolvers');

const app = express();

const fileStorage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'images');
  },
```

```
  filename: (req, file, cb) => {
    cb(null, new Date().toISOString() + '-' + file.originalname);
  }
});

const fileFilter = (req, file, cb) => {
  if(
    file.mimetype === 'image/png' ||
    file.mimetype === 'image/jpg' ||
    file.mimetype === 'image/jpeg'
    ) {
      cb(null, true);
    } else {
      cb(null, false);
    }
}

// app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
app.use(bodyParser.json()); // application/json
app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
app.use('/images', express.static(path.join(__dirname, 'images')));

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader(
    'Access-Control-Allow-Methods',
    'OPTIONS, GET, POST, PUT, PATCH, DELETE'
  );
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
  next();
});

app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver
  })
);

app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));
```

```
//./graphql/schema.js(b)

```

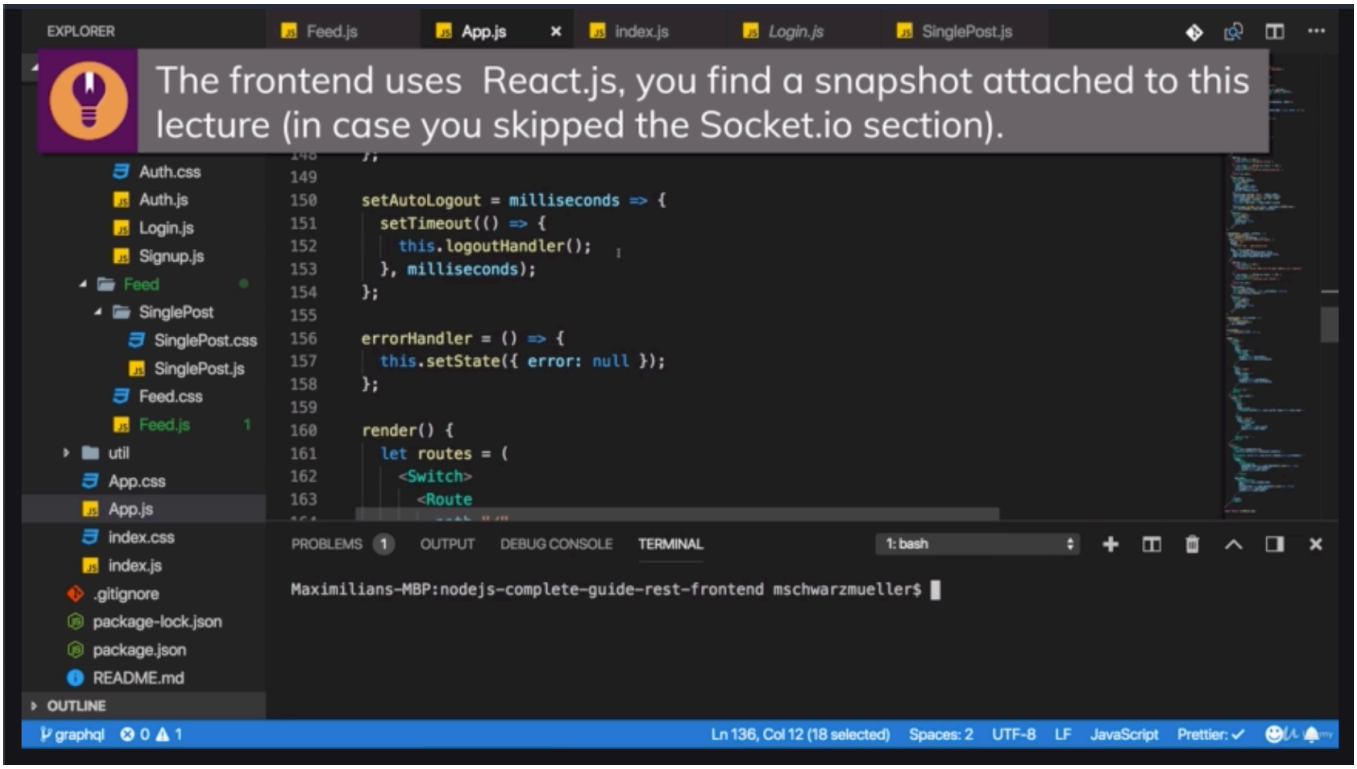
```
/**this is wherer i find the queries mutations and types
 * i work with in my graphql service i will create
 * */

 /**'buildSchema' function allows me to build a schema
  * which cna be parsed by graphql by express-graphql
  */

const { buildSchema } = require('graphql');

/**note that there is no colon after a schema
 * and in that schema, we define a query field
 *
 * this will be an object with all the queries
 * and queries are the parts where you get data
 * so all the queries you wanna allow
 *
 * this is now a very basic schema
 * where we can send a 'hello' queries to get back some text
 *
 * you don't need to resolve this 'query: RootQuery'
 * because in your schema,
 * you set up your RootQuery
 * which has made up of this sub-queries 'hello: String'
 * which need resolvers
 *
 * if you add '!',
 * the result is that if you don't return a string,
 * you will get an error
 */
module.exports = buildSchema(`
    type TestData {
        text: String!
        views: Int!
    }

    type RootQuery {
        hello: TestData!
    }

    schema {
        query: RootQuery
    }
`);
```

```
//./graphql/resolvers.js(b)

/**this is where i will define the logic
 * that then executed for incoming queries
 * */

 /**you need a method for every query or mutation
  * you define your schema and the name has to match
  * now we have resolver for 'hello' method
  *
  * we need to expose it to the public
  * and that can be done with the 'express-graphql' package
```

```
 * so go to app.js(b)
 */

module.exports = {
    hello() {
        return {
            text: 'Hello World!',
            views: 1245
        }
    }
}
```

* Chapter 418: Defining A Mutation Schema

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




```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
```

```
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    fetch('http://localhost:8080/auth/status', {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch user status.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({ status: resData.status });
      })
      .catch(this.catchError);

    this.loadPosts();

  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
      page--;
      this.setState({ postPage: page });
    }
    fetch('http://localhost:8080/feed/posts?page=' + page, {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch posts.');
        }
        return res.json();
      })
```

```
    .then(resData => {
      this.setState({
        posts: resData.posts.map(post => {
          return {
            ...post,
            imagePath: post.imageUrl
          };
        }),
        totalPosts: resData.totalItems,
        postsLoading: false
      });
    })
    .catch(this.catchError);
};

statusUpdateHandler = event => {
  event.preventDefault();
  fetch('http://localhost:8080/auth/status', {
    method: 'PATCH',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      status: this.state.status
    })
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error("Can't update status!");
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
    })
    .catch(this.catchError);
};

newPostHandler = () => {
  this.setState({ isEditing: true });
};

startEditPostHandler = postId => {
  this.setState(prevState => {
    const loadedPost = { ...prevState.posts.find(p => p._id === postId) };

    return {
      isEditing: true,
      editPost: loadedPost
    };
  });
};

cancelEditHandler = () => {
  this.setState({ isEditing: false, editPost: null });
```

```
};

finishEditHandler = postData => {
  this.setState({
    editLoading: true
  });
  const formData = new FormData();
  formData.append('title', postData.title);
  formData.append('content', postData.content);
  formData.append('image', postData.image);
  let url = 'http://localhost:8080/feed/post';
  let method = 'POST';
  if (this.state.editPost) {
    url = 'http://localhost:8080/feed/post/' + this.state.editPost._id;
    method = 'PUT';
  }

  fetch(url, {
    method: method,
    body: formData,
    headers: {
      Authorization: 'Bearer ' + this.props.token
    }
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error('Creating or editing a post failed!');
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
      const post = {
        _id: resData.post._id,
        title: resData.post.title,
        content: resData.post.content,
        creator: resData.post.creator,
        createdAt: resData.post.createdAt
      };
      this.setState(prevState => {
        return {
          isEditing: false,
          editPost: null,
          editLoading: false
        };
      });
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isEditing: false,
        editPost: null,
        editLoading: false,
        error: err
      });
    });
```

```
  };

  statusInputChangeHandler = (input, value) => {
    this.setState({ status: value });
  };

  deletePostHandler = postId => {
    this.setState({ postsLoading: true });
    fetch('http://localhost:8080/feed/post/' + postId, {
      method: 'DELETE',
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200 && res.status !== 201) {
          throw new Error('Deleting a post failed!');
        }
        return res.json();
      })
      .then(resData => {
        console.log(resData);
        this.loadPosts();
/*
        this.setState(prevState => {
          const updatedPosts = prevState.posts.filter(p => p._id !== postId);
          return { posts: updatedPosts, postsLoading: false };
        });
*/
      })
      .catch(err => {
        console.log(err);
        this.setState({ postsLoading: false });
      });
  };

  errorHandler = () => {
    this.setState({ error: null });
  };

  catchError = error => {
    this.setState({ error: error });
  };

  render() {
    return (
      <Fragment>
        <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
        <FeedEdit
          editing={this.state.isEditing}
          selectedPost={this.state.editPost}
          loading={this.state.editLoading}
          onCancelEdit={this.cancelEditHandler}
          onFinishEdit={this.finishEditHandler}
        />
        <section className="feed__status">
```

```
          <form onSubmit={this.statusUpdateHandler}>
            <Input
              type="text"
              placeholder="Your status"
              control="input"
              onChange={this.statusInputChangeHandler}
              value={this.state.status}
            />
            <Button mode="flat" type="submit">
              Update
            </Button>
          </form>
        </section>
        <section className="feed__control">
          <Button mode="raised" design="accent" onClick={this.newPostHandler}>
            New Post
          </Button>
        </section>
        <section className="feed">
          {this.state.postsLoading && (
            <div style={{ textAlign: 'center', marginTop: '2rem' }}>
              <Loader />
            </div>
          )}
          {this.state.posts.length <= 0 && !this.state.postsLoading ? (
            <p style={{ textAlign: 'center' }}>No posts found.</p>
          ) : null}
          {!this.state.postsLoading && (
            <Paginator
              onPrevious={this.loadPosts.bind(this, 'previous')}
              onNext={this.loadPosts.bind(this, 'next')}
              lastPage={Math.ceil(this.state.totalPosts / 2)}
              currentPage={this.state.postPage}
            >
              {this.state.posts.map(post => (
                <Post
                  key={post._id}
                  id={post._id}
                  author={post.creator.name}
                  date={new Date(post.createdAt).toLocaleDateString('en-US')}
                  title={post.title}
                  image={post.imageUrl}
                  content={post.content}
                  onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                  onDelete={this.deletePostHandler.bind(this, post._id)}
                />
              ))}
            </Paginator>
          )}
        </section>
      </Fragment>
    );
  }
}

export default Feed;
```

295

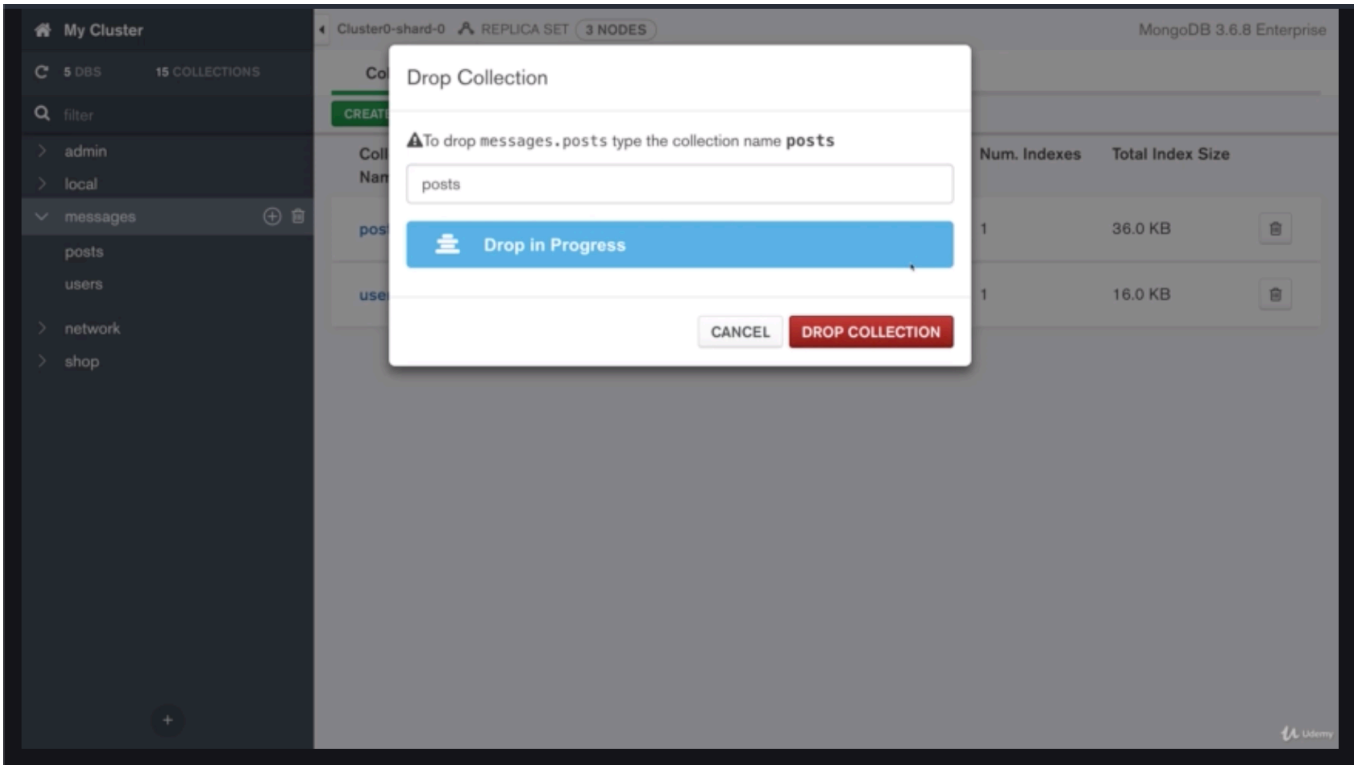
```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

/**that mutation here will require some input some arguments
 * and that is a difference in text which you haven't seen before
 * you can add parentheses after you require the name
 * and this allows you to specify arguments
 * that have to be given to that resolver that will run in the end
 *
 * we bundle it in one object
 * which we expect and for data object,
 * we can create a new type and we don't do with 'type' keyword
 * but there's special keyword for data that is used as the input
 * so for data as is used as an argument and that is the 'input' keyword
 *
 * 'ID' is a special type provided by graphql
 * which signals that this is unique
 * and well treated as an ID
 *
 * graphQL doesn't know the dates.
 * so i will use string and the same for updatedAt
 * remember we will get these 2 fields
 * because in mongoose model for the post,
 * we enable timestamps. '{ timestamps: true }'
 */
module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        Content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    input UserInput {
        email: String!
        name: String!
        password: String!
    }

    type RootMutation {
        creatUser(userInput: UserInputData): User!
    }
```

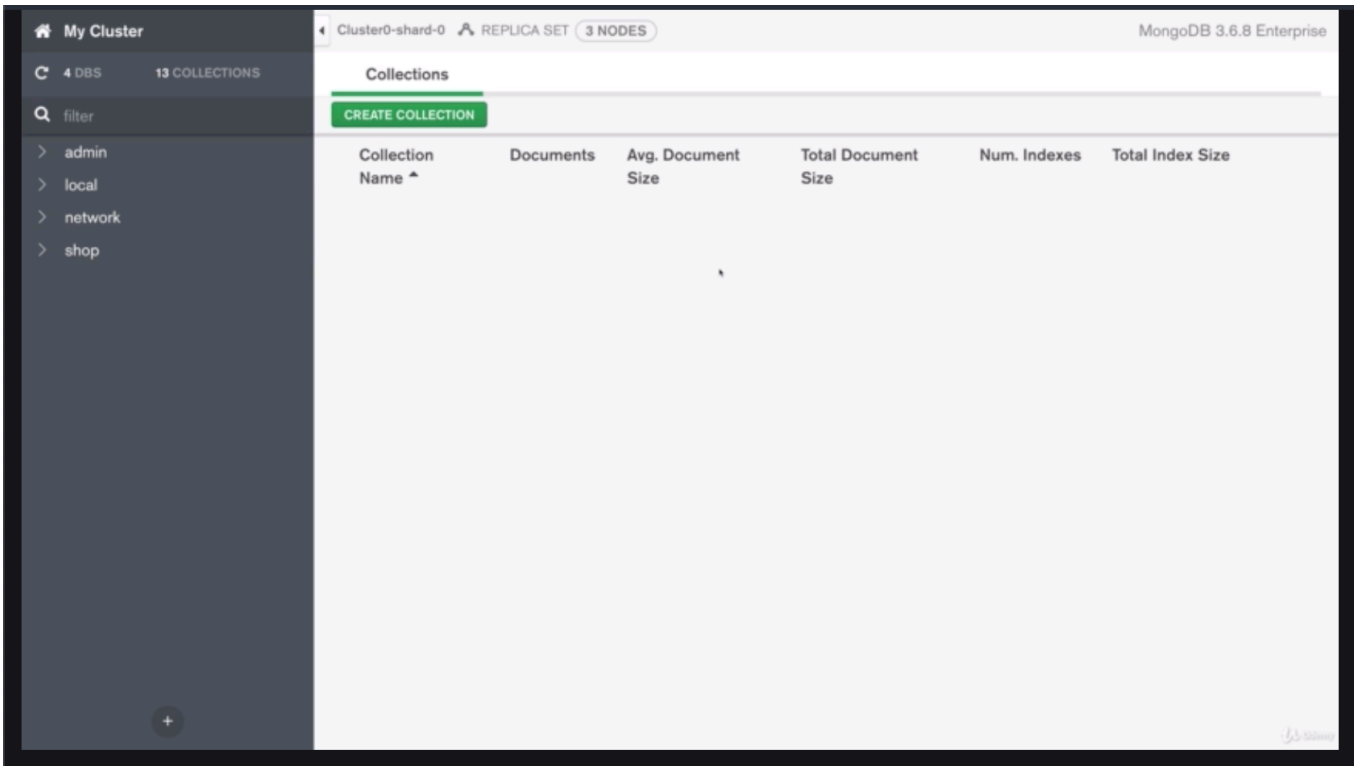
```
56
57      schema {
58          mutation: RootMutation
59      }
60  `);
```

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

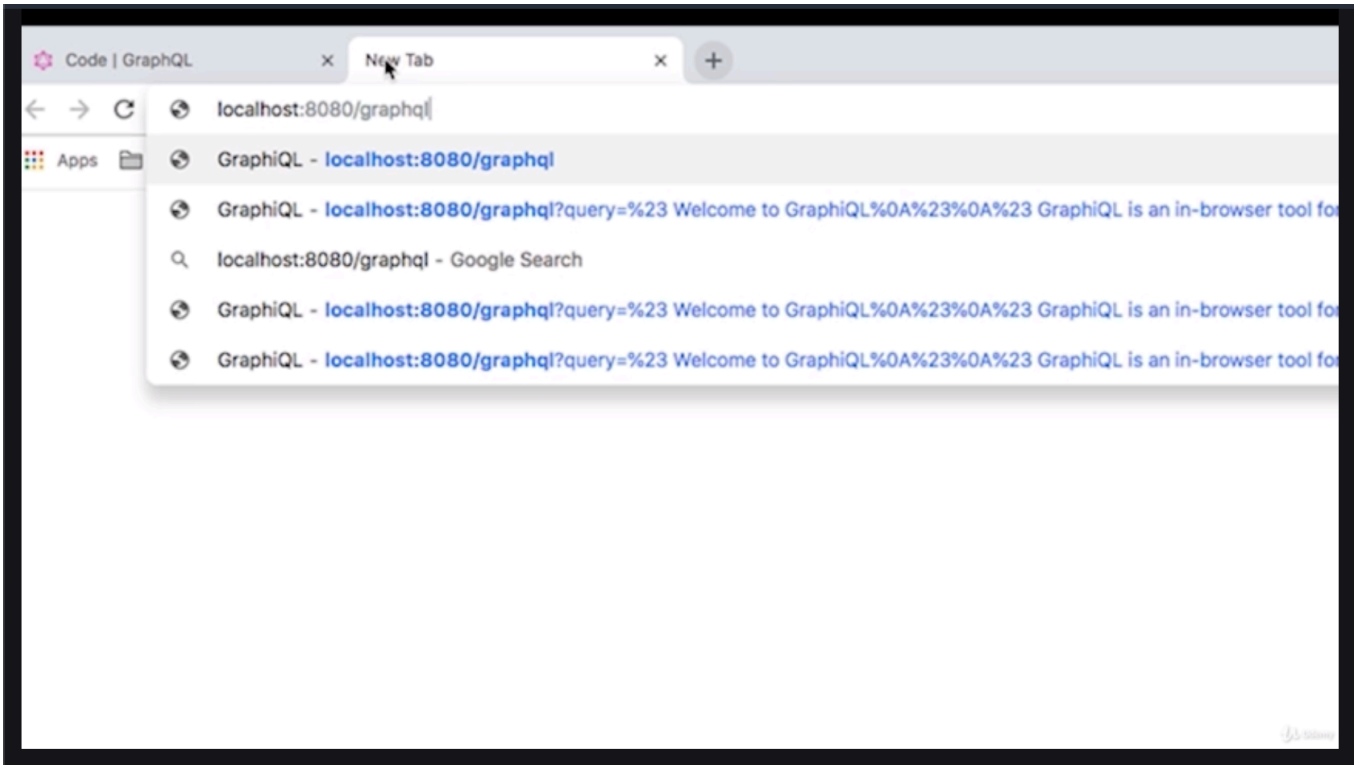
* Chapter 419: Adding A Mutation Resolver & GraphiQL

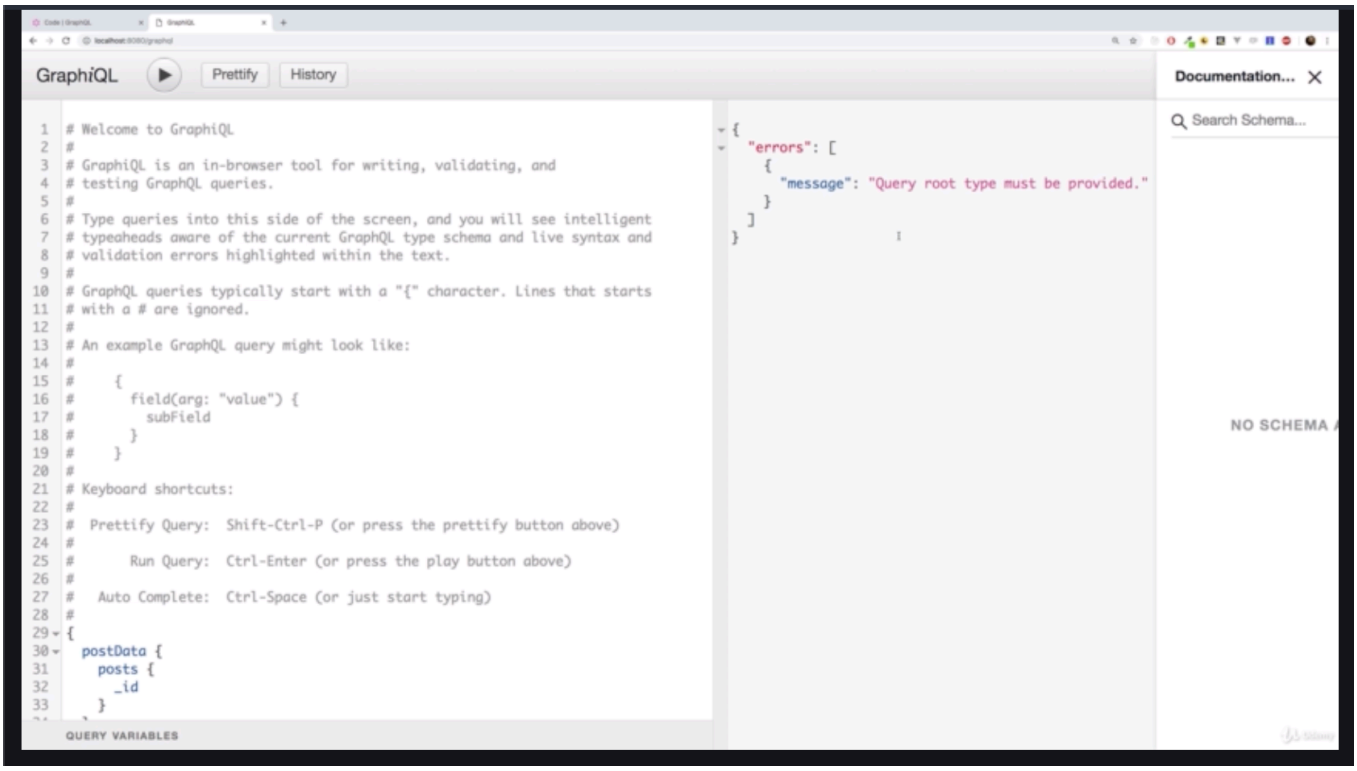
1. update
- ./graphql/resolvers.js(b)
- app.js(b)
- ./graphql/schema.js(b)
- graphiQL





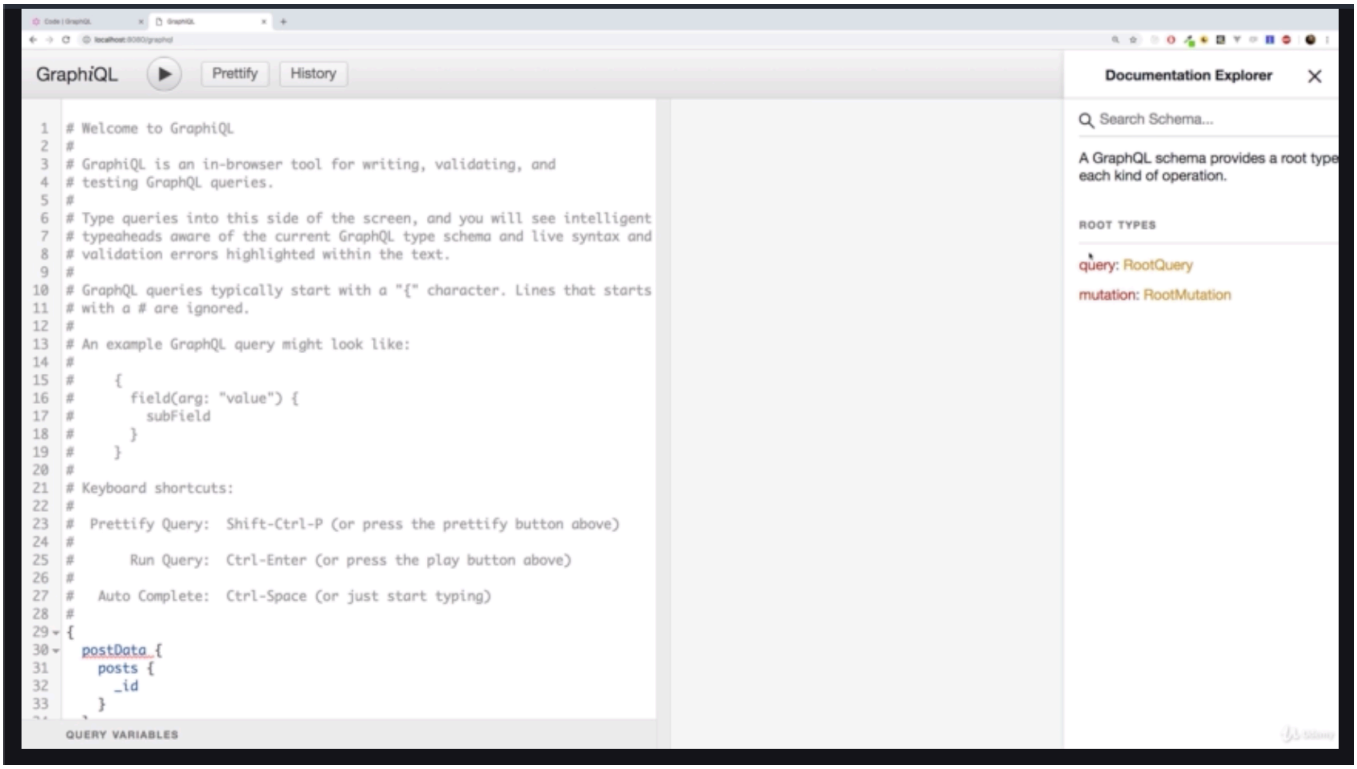
- let's clean up 'message'

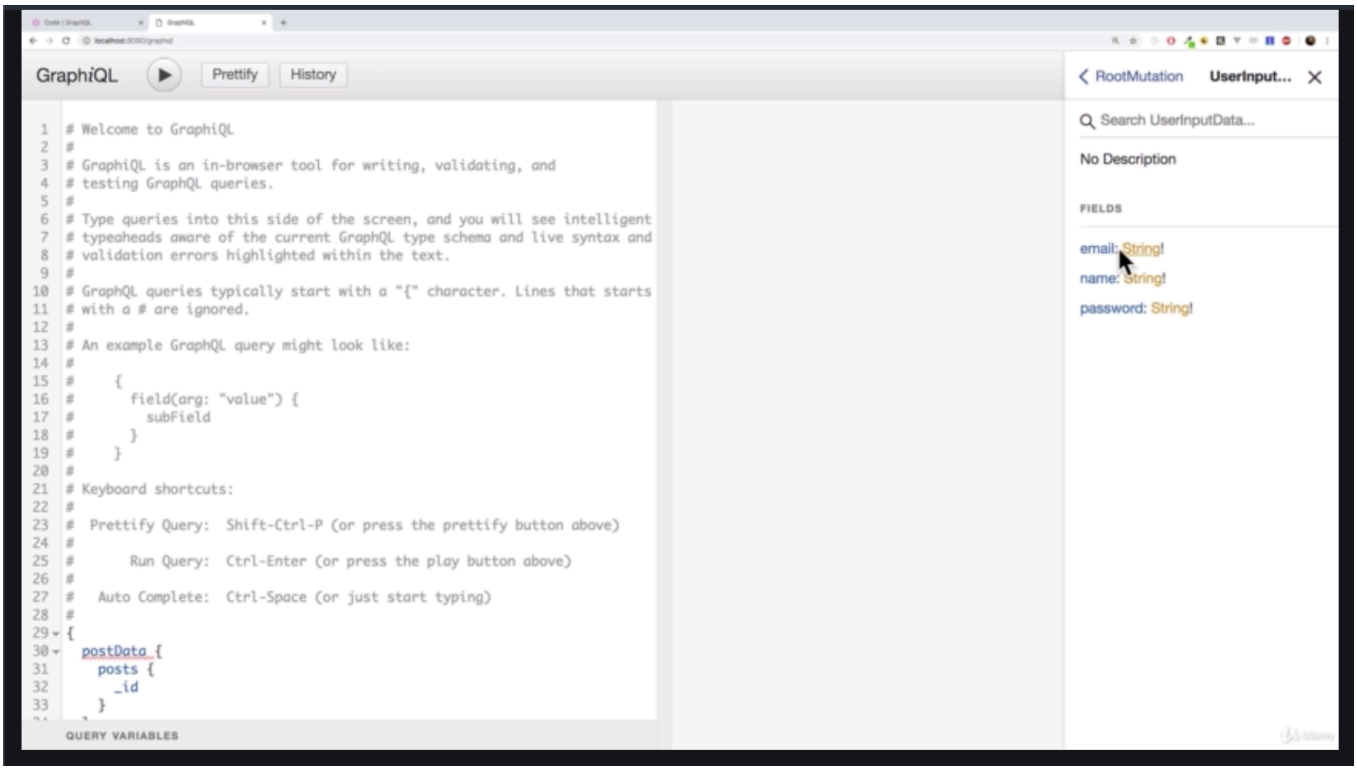


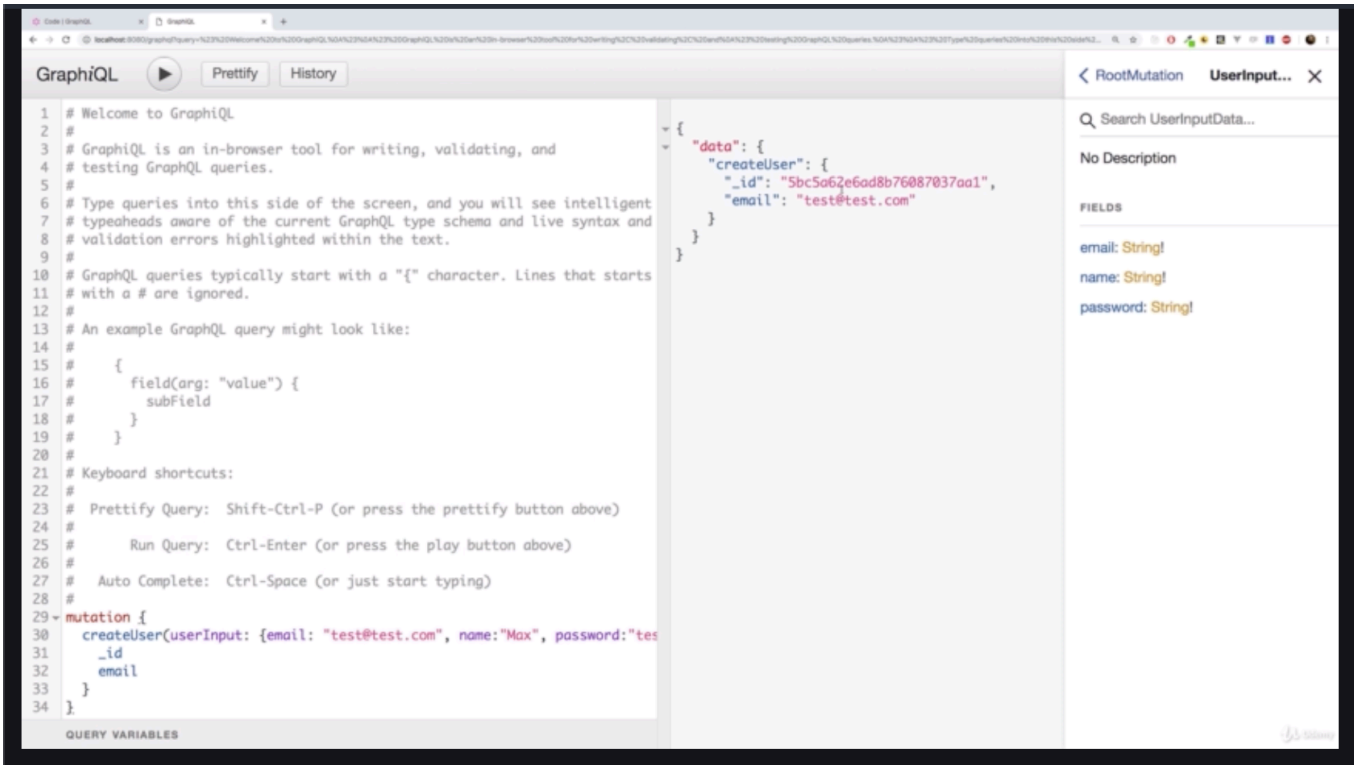


- localhost:8080/graphql will send a GET request and you will get this screen which allows you to play around with your graphQL API

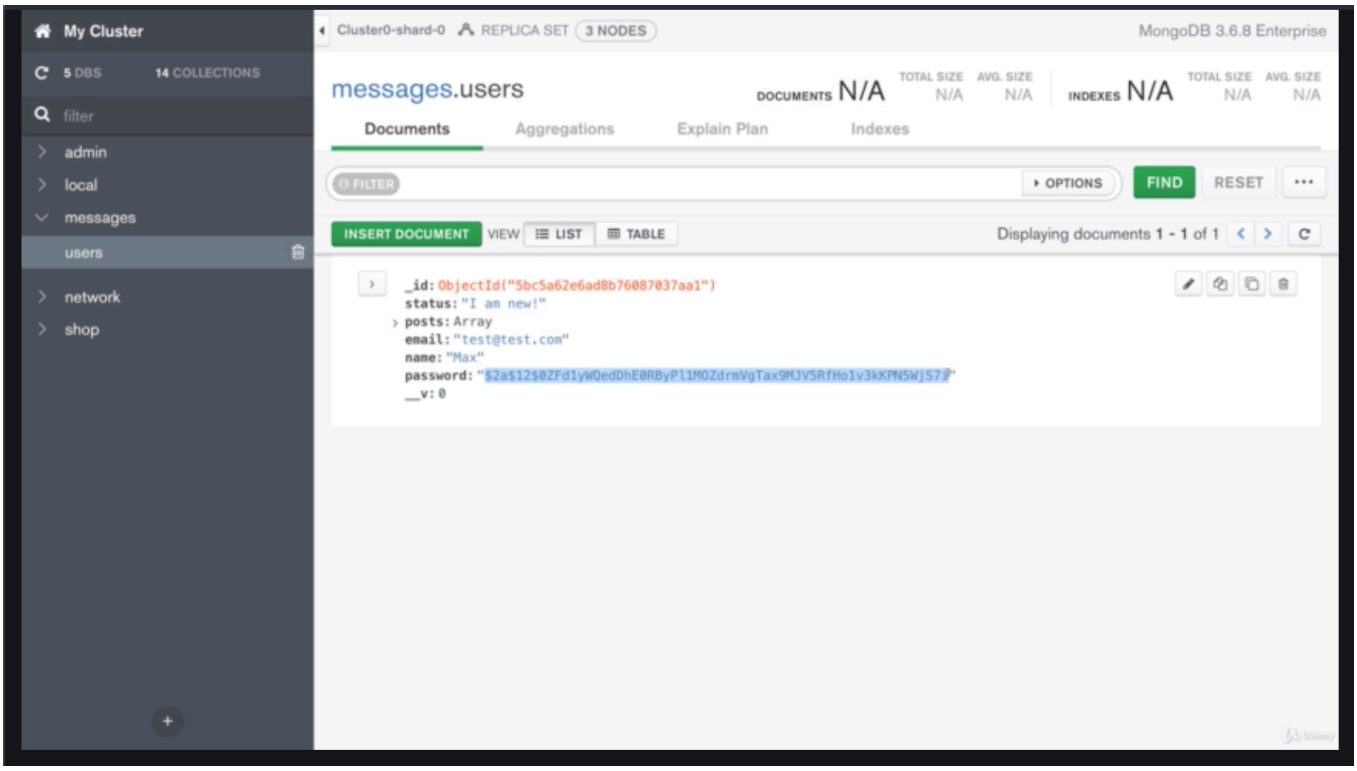
- if you reload your graghiQL inerface you have that documentation on the right where you see the operations you can do but you need to have a query defined for that even if it leads into white.







- 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




```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    type RootQuery {
        hello: String
    }

    type RootMutation {
```

```
        createUser(userInput: UserInputData): User!
    }

    schema {
        query: RootQuery
        mutation: RootMutation
    }
`);
```

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');

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

/**not directly on the args,
 * instead our args will have a 'userInput' field
 * because our args will be an object
 * containing all the arguments passed to that function 'createUser'
 * and in 'userInput' field,
 * that will have email, name and password so i can retrieve them.
 */
module.exports = {
    createUser: async function({ userInput }, req) {
        //const email = args.userInput.email;
        /**if you are not using async, await
         * you need to return your fineOne query
         * which you are executing here
         * where you would then add '.then()'
         * because if you don't return your promises in the resolver,
         * graphQL will not wait for it to resolve
         *
         *      return User.findOne().then()
         *
         * when using async await
         * it's automatically returned for you
         * we don't see the return statement but it is there.
         */
        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();
        /**i will return my createdUser._doc field
         * which contains just userData without all the meta data mongoose would add
         * and i will override the _id field by adding it as a separate property
         * and therefore it will override the one i'm pulling out of _doc
         * because i need to convert it from an object id field to a string field
         * otherwise it will fail
         *
```

```
        */
        return { ...createdUser._doc, _id: createdUser._id.toString() };
    }
}
```

```
//app.js(b)

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const multer = require('multer');
/**i will remove feedRoutes, authRoutes
 * because we will have no more routes will not set up routes anymore
 * we will use graphQL endpoint instead.
 */
const graphqlHttp = require('express-graphql');

const graphSchema = require('./graphql/schema');
const graphResolver = require('./graphql/resolvers');

const app = express();

const fileStorage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'images');
  },
  filename: (req, file, cb) => {
    cb(null, new Date().toISOString() + '-' + file.originalname);
  }
});

const fileFilter = (req, file, cb) => {
  if(
    file.mimetype === 'image/png' ||
    file.mimetype === 'image/jpg' ||
    file.mimetype === 'image/jpeg'
    ) {
      cb(null, true);
    } else {
      cb(null, false);
    }
}

// app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
app.use(bodyParser.json()); // application/json
app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
app.use('/images', express.static(path.join(__dirname, 'images')));

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader(
    'Access-Control-Allow-Methods',
    'OPTIONS, GET, POST, PUT, PATCH, DELETE'
  );
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
```

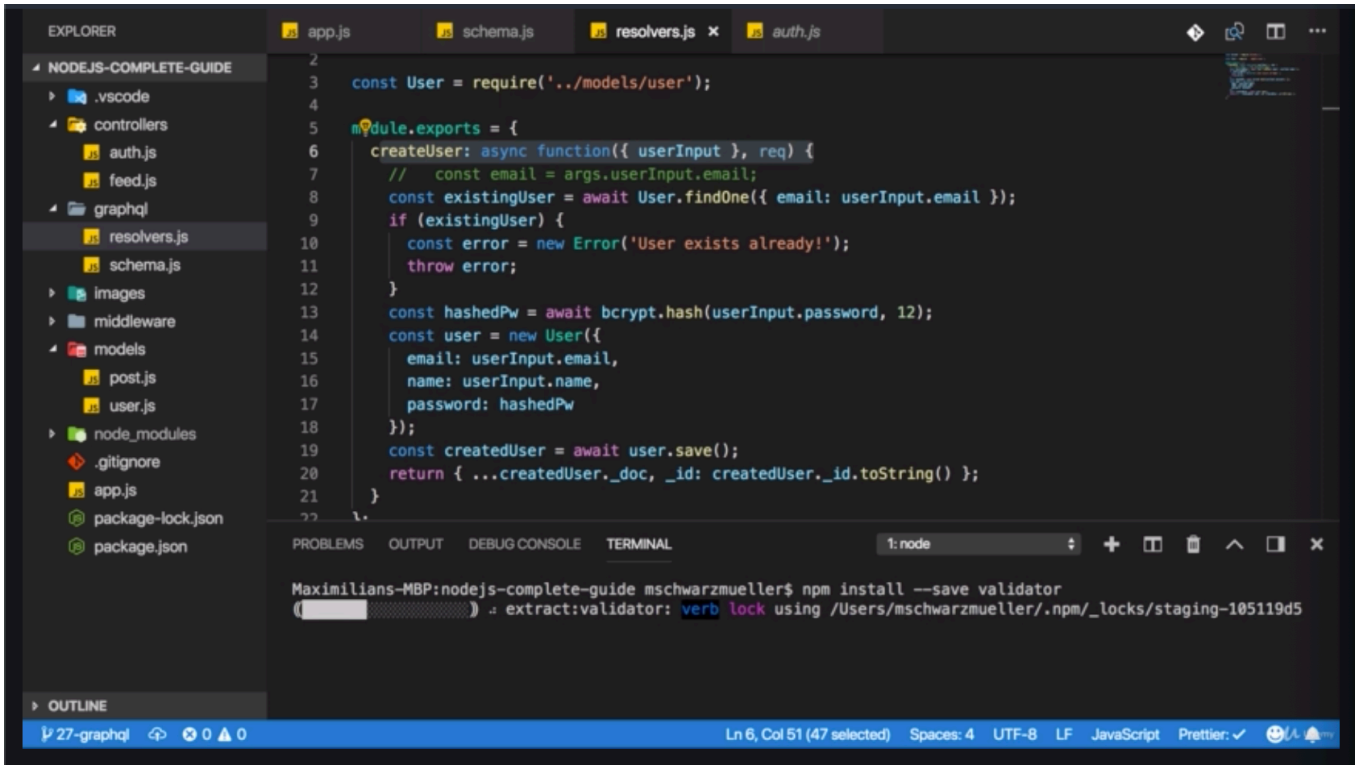
```
  next();
});

app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver,
    /**to test those mutation.
     * 'graphiql' gives you a special tool
     * and this is the reason why i'm not listening for a  post request only here
     * which is 'use' not 'post'
     * because your running server,
     * try visiting localhost:8080/graphql will send GET request
     *
    */
    graphiql: true
  })
);

app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));
```

```
//localhost:8080/graphql
//graphiQL

# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
```

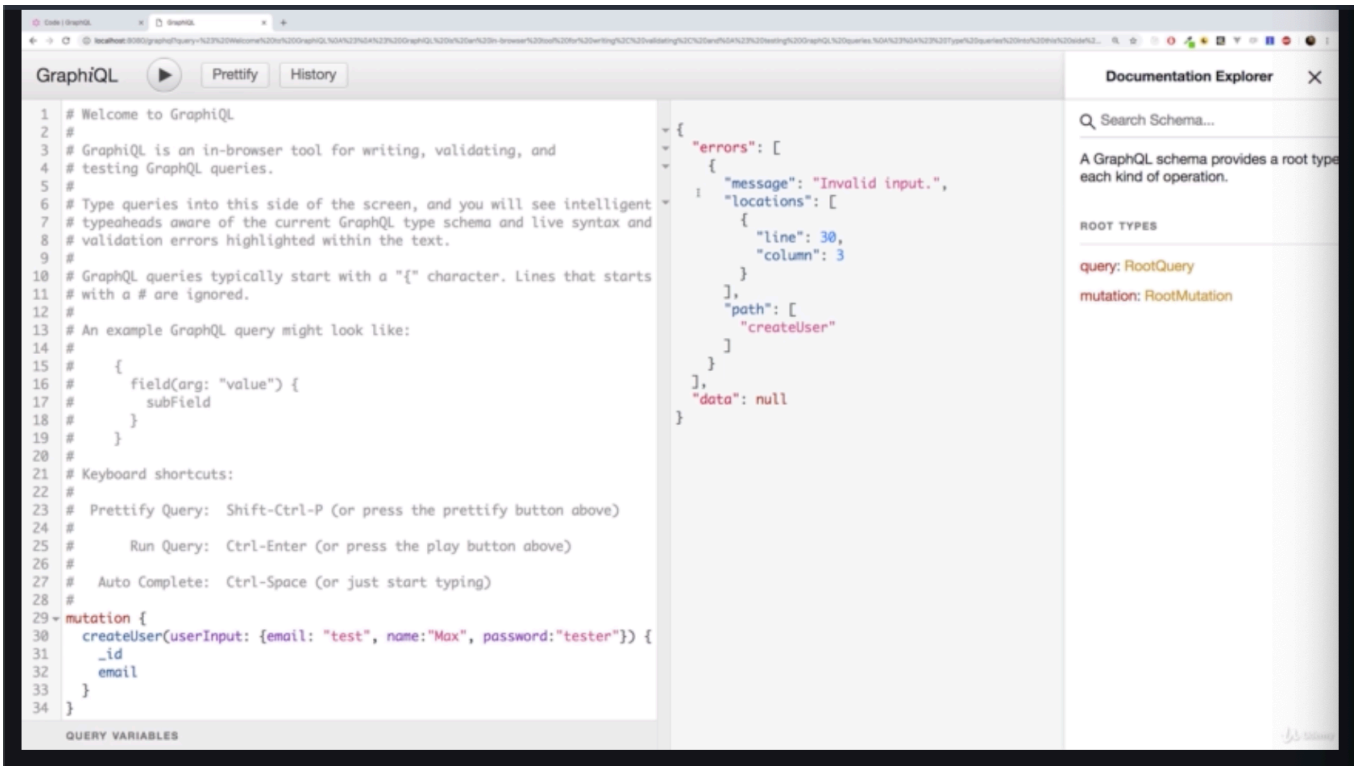
```
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#
mutation {
  createUser(userInput: {email:"test@test.com", name: "Max", password: "tester"}){
    _id
    email
  }
}
```

* Chapter 420: Adding Input Validation

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



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



- 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

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');

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

module.exports = {
    createUser: async function({ userInput }, req) {
        //const email = args.userInput.email;
        const errors = [];
        if (!validator.isEmail(userInput.email)){
            errors.push({ message: 'E-Mail is invalid.' });
        }
        if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password,
{ min: 5 })){
            errors.push({message: 'Password Too Short!'});
        }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            throw error;
        }
        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() };
    }
}
```

```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

/**we added our first mutation
 * and mutation also means that we store data in the database
 *
 */
module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    type RootQuery {
        hello: String
    }

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

    schema {
        query: RootQuery
        mutation: RootMutation
    }
`);
```

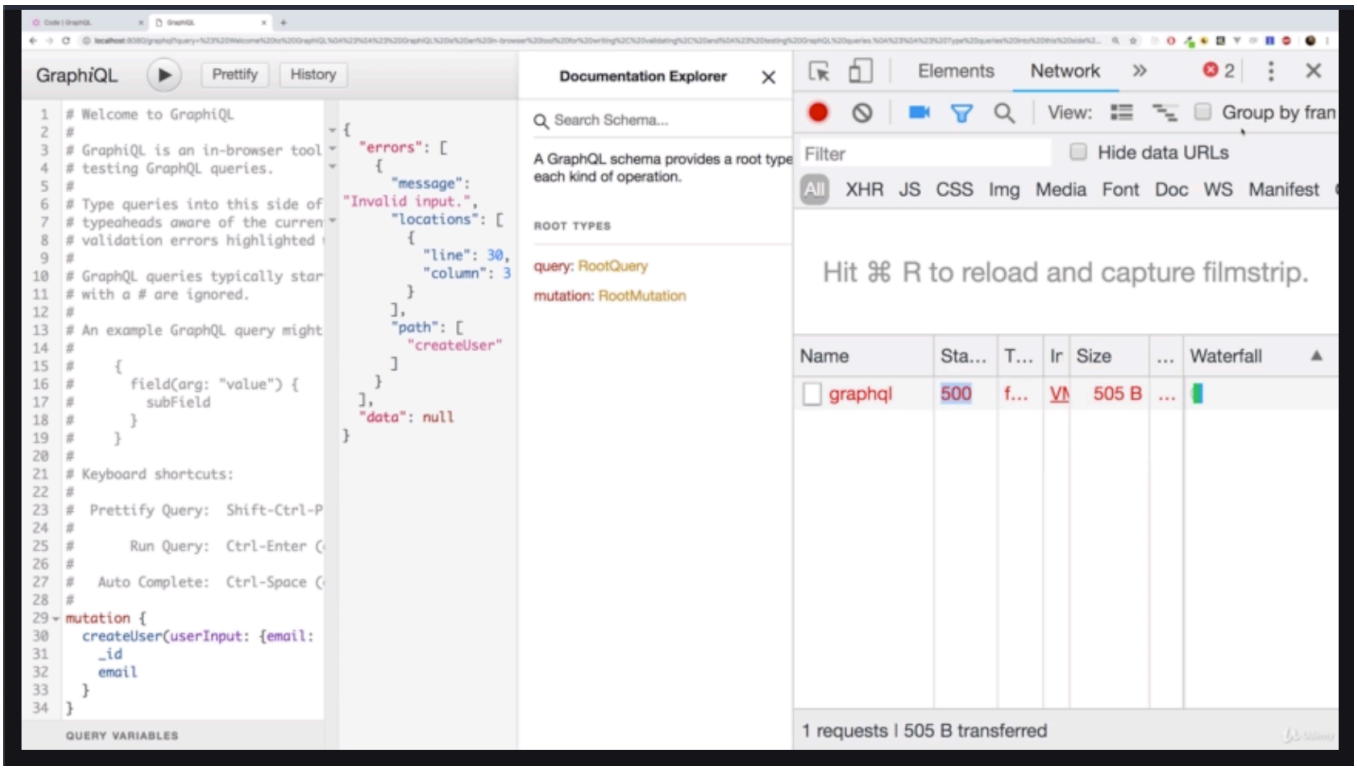
```
//app.js(b)

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   next();
51 });
52
53 /**when something like this happnes,
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.
```

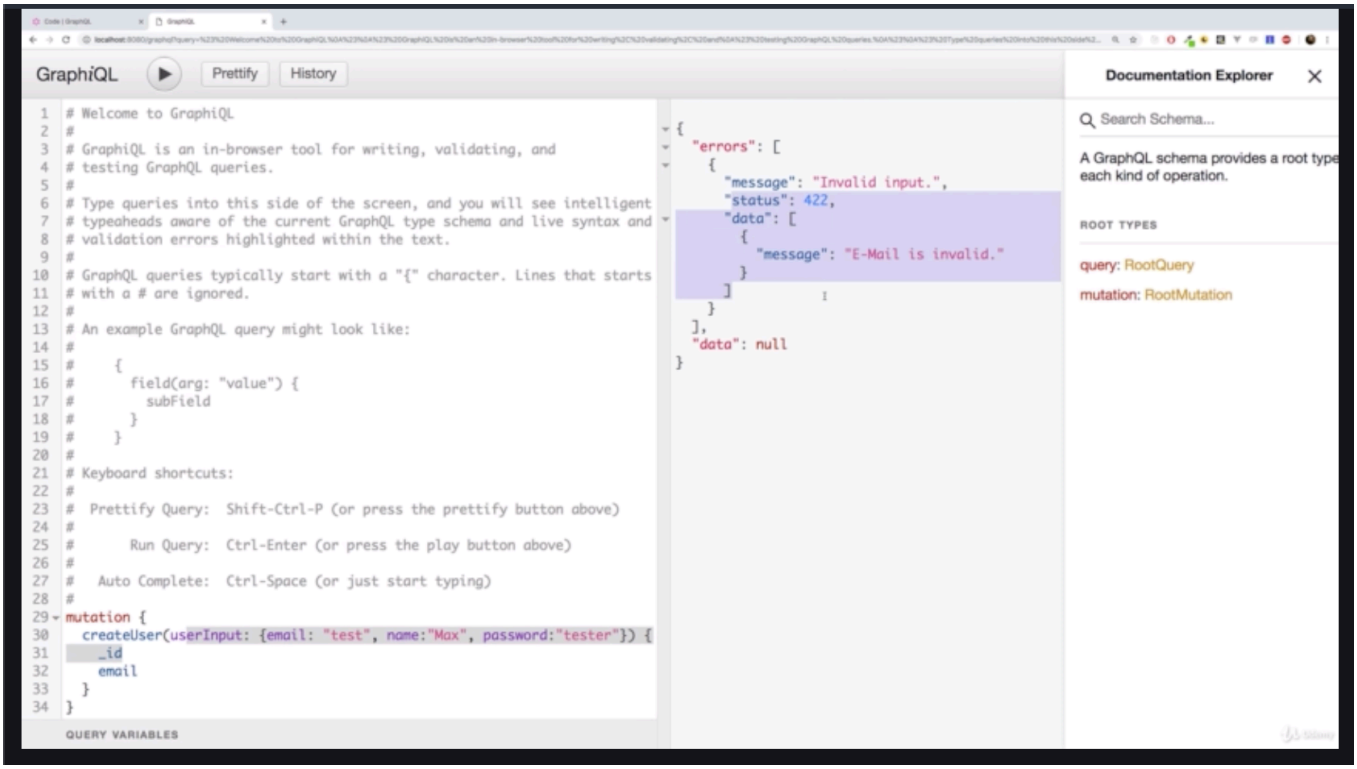
```
 *
 * with GraphQL is not an option
 * because we only have one route
 * this is the only end point
 * and we don't wanna validate all requests in the same way
 * so to change that and to adjust it to our needs
 * we wanna move validation into our resolvers
 */
app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver,
    graphiql: true
  })
);

app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));
```

* Chapter 421: Handling Errors

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



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




```
//app.js(b)

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const multer = require('multer');

const graphqlHttp = require('express-graphql');

```

```
const graphSchema = require('./graphql/schema');
const graphResolver = require('./graphql/resolvers');

const app = express();

const fileStorage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'images');
  },
  filename: (req, file, cb) => {
    cb(null, new Date().toISOString() + '-' + file.originalname);
  }
});

const fileFilter = (req, file, cb) => {
  if(
    file.mimetype === 'image/png' ||
    file.mimetype === 'image/jpg' ||
    file.mimetype === 'image/jpeg'
    ) {
      cb(null, true);
    } else {
      cb(null, false);
    }
}

// app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
app.use(bodyParser.json()); // application/json
app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
app.use('/images', express.static(path.join(__dirname, 'images')));

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader(
    'Access-Control-Allow-Methods',
    'OPTIONS, GET, POST, PUT, PATCH, DELETE'
  );
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
  next();
});

app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver,
    graphiql: true,
    /**'formatError()' is a method
     * which receives the error detected by graphQL
     * and allows you to return your own format
     * if you return error,
     * you keep the default format
     * which is the format we see in the graphiQL localhost:8080/graphql
     *
     * and you can check if in your error,
     * you don't have the originalError field
     * so if you don't have that
     * hence exclamation mark at the begining
```

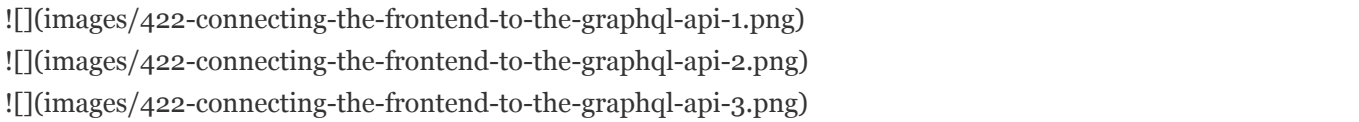
```
    * or original error will be set by express-graphql
    * when it detects an error thrown in your code either
    * by you or some 3rd party package
    * if you have a technical error,
    * for example a missing character in your query or anything like that
    * then it will not have that original error.
    *
    * and if we don't have that
    * then i will return the error that was generated by graphQL
    * but if i do have my originalError
    * then i can extract useful information from it that i can add in other plcaes
    * and that is what we can do into resolver
    */
    formatError(err) {
      if (!err.originalError) {
        return err;
      }
      const data = err.originalError.data;
      const message = err.message || 'An error occurred.';
      const code = err.originalError.code || 500
      return { message: message, status: code, data: data };
    }
  })
);

app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));
```

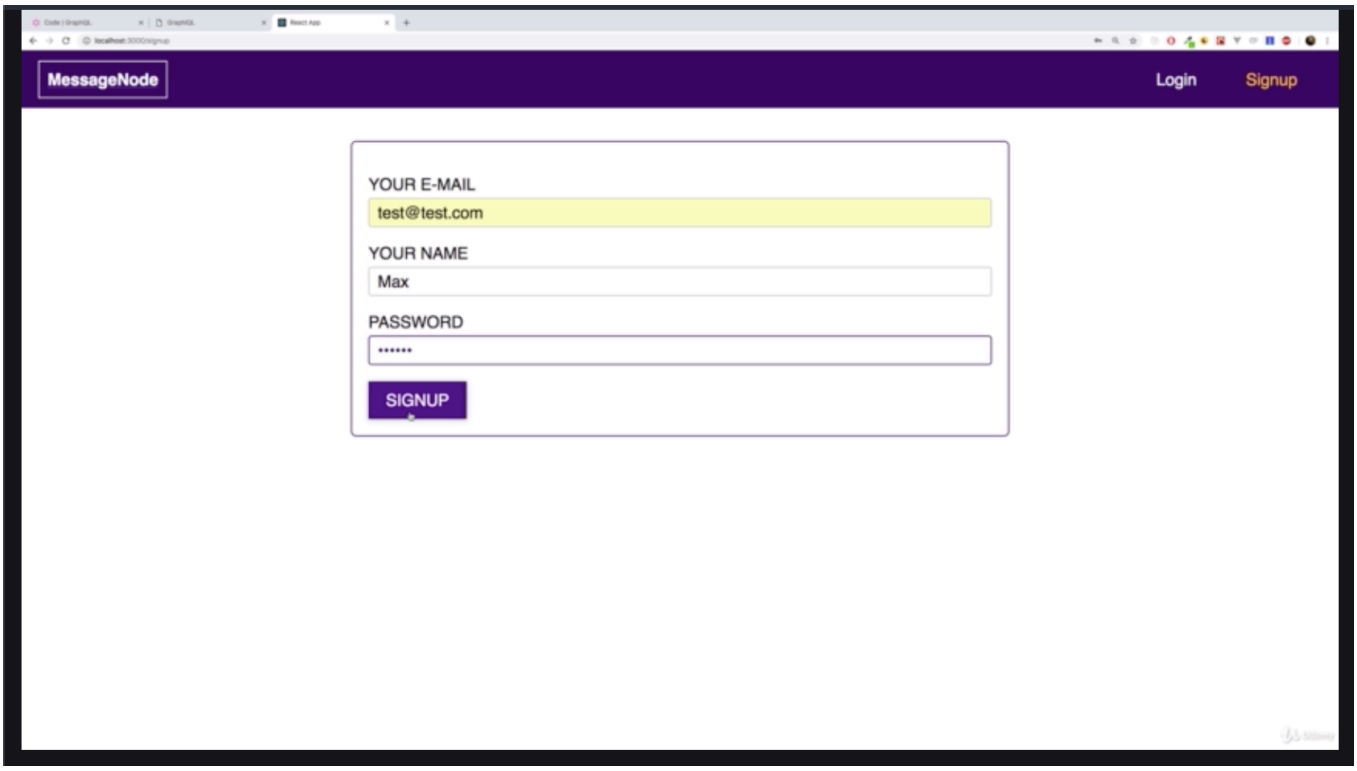
```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');

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

module.exports = {
    createUser: async function({ userInput }, req) {
        //const email = args.userInput.email;
        const errors = [];
        if (!validator.isEmail(userInput.email)){
            errors.push({ message: 'E-Mail is invalid.' });
        }
        if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password,
```

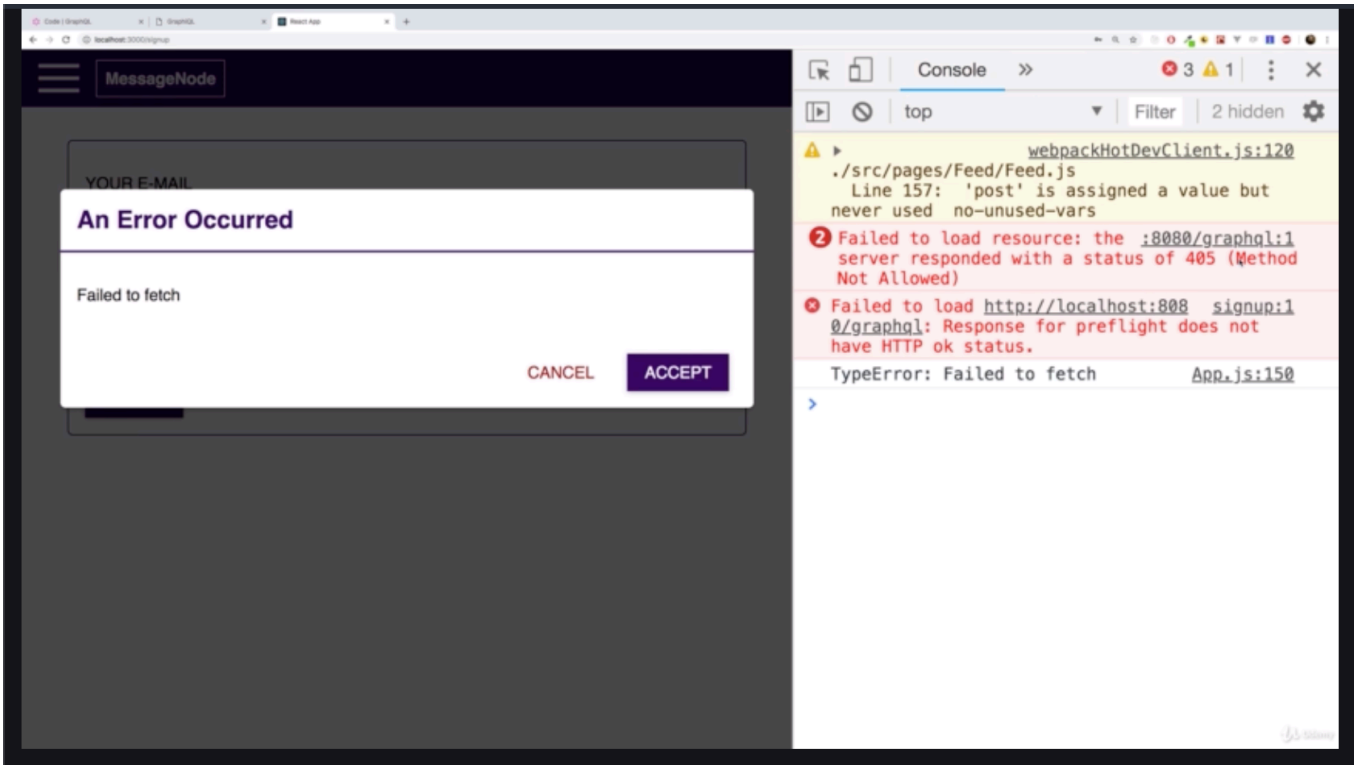
```
{ min: 5 })){
            errors.push({message: 'Password Too Short!'});
        }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        const existingUser = await 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() };
    }
}
```

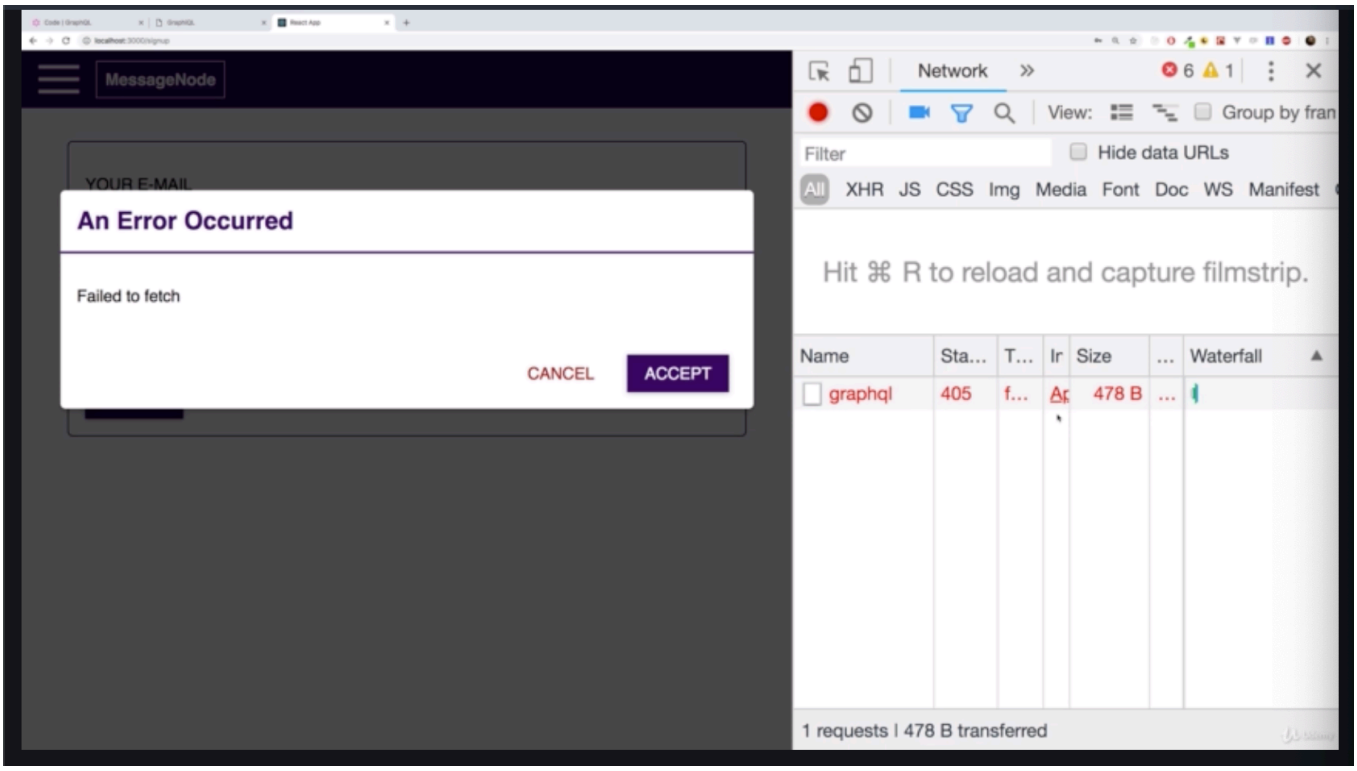
* Chapter 422: Connecting The Frontend To The GraphQL API

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

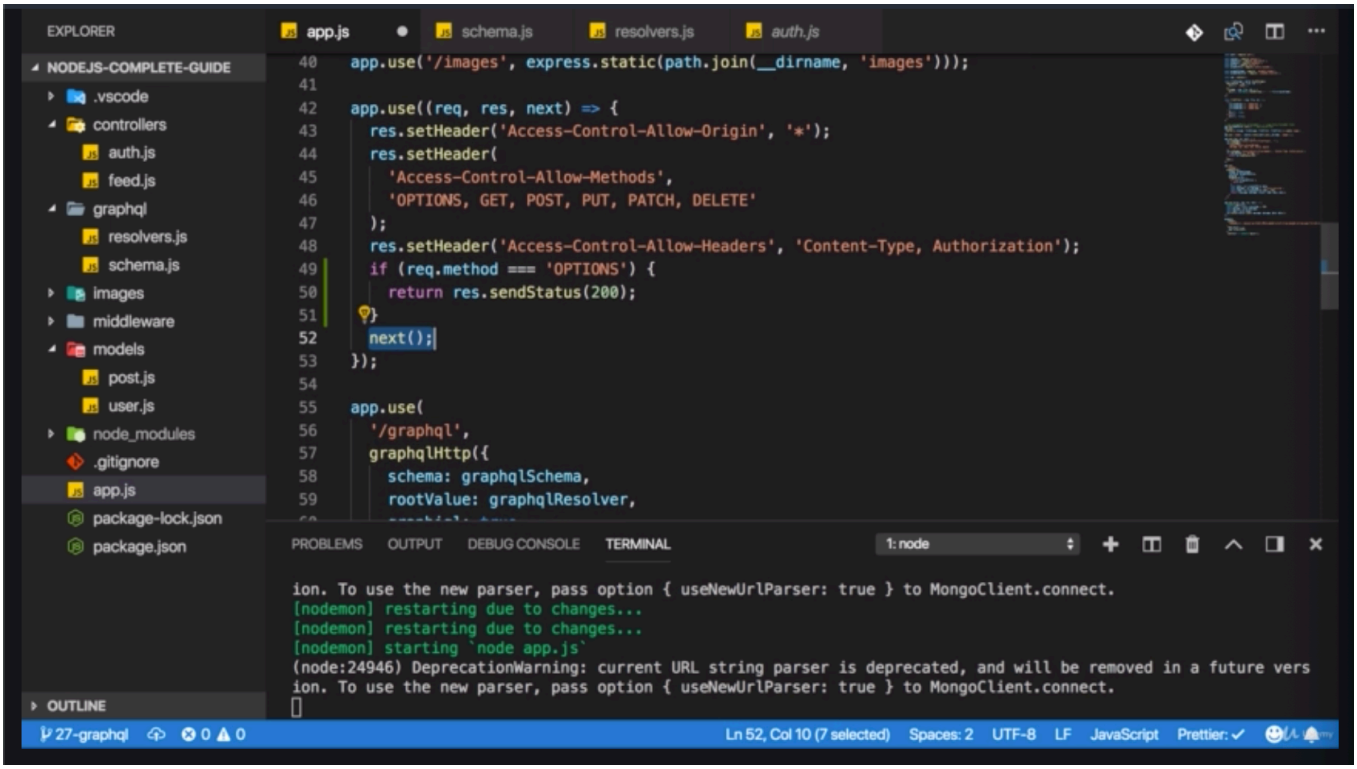



MessageNode
Login
Signup
YOUR E-MAIL
test@test.com
YOUR NAME
Max
PASSWORD
SIGNUP


MessageNode
YOUR E-MAIL
An Error Occurred
Failed to fetch
CANCEL
ACCEPT
Console
top
Filter
2 hidden
webpackHotDevClient.js:120
./src/pages/Feed/Feed.js
Line 157: 'post' is assigned a value but never used no-unused-vars
Failed to load resource: the :8080/graphql:1
server responded with a status of 405 (Method Not Allowed)
Failed to load http://localhost:808 signup:1
0/graphql: Response for preflight does not have HTTP ok status.
TypeError: Failed to fetch App.js:150

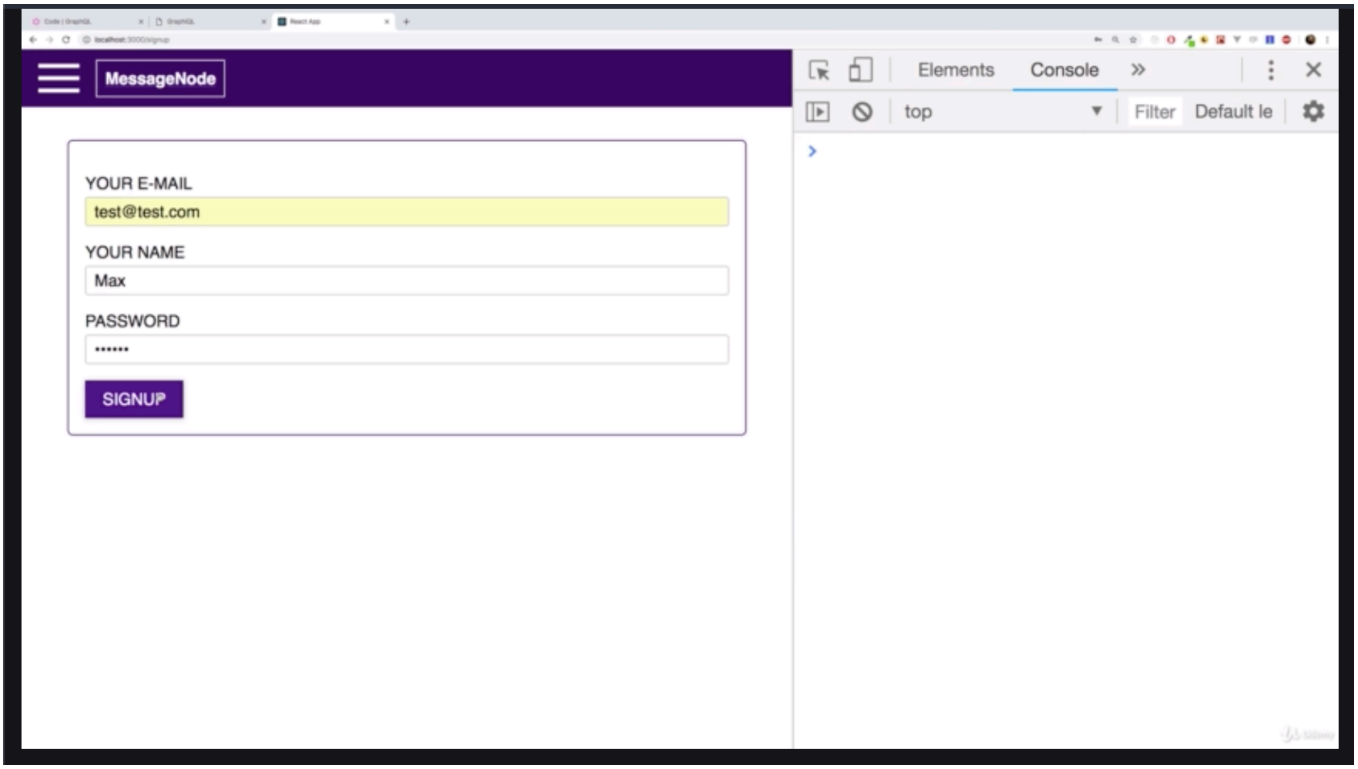


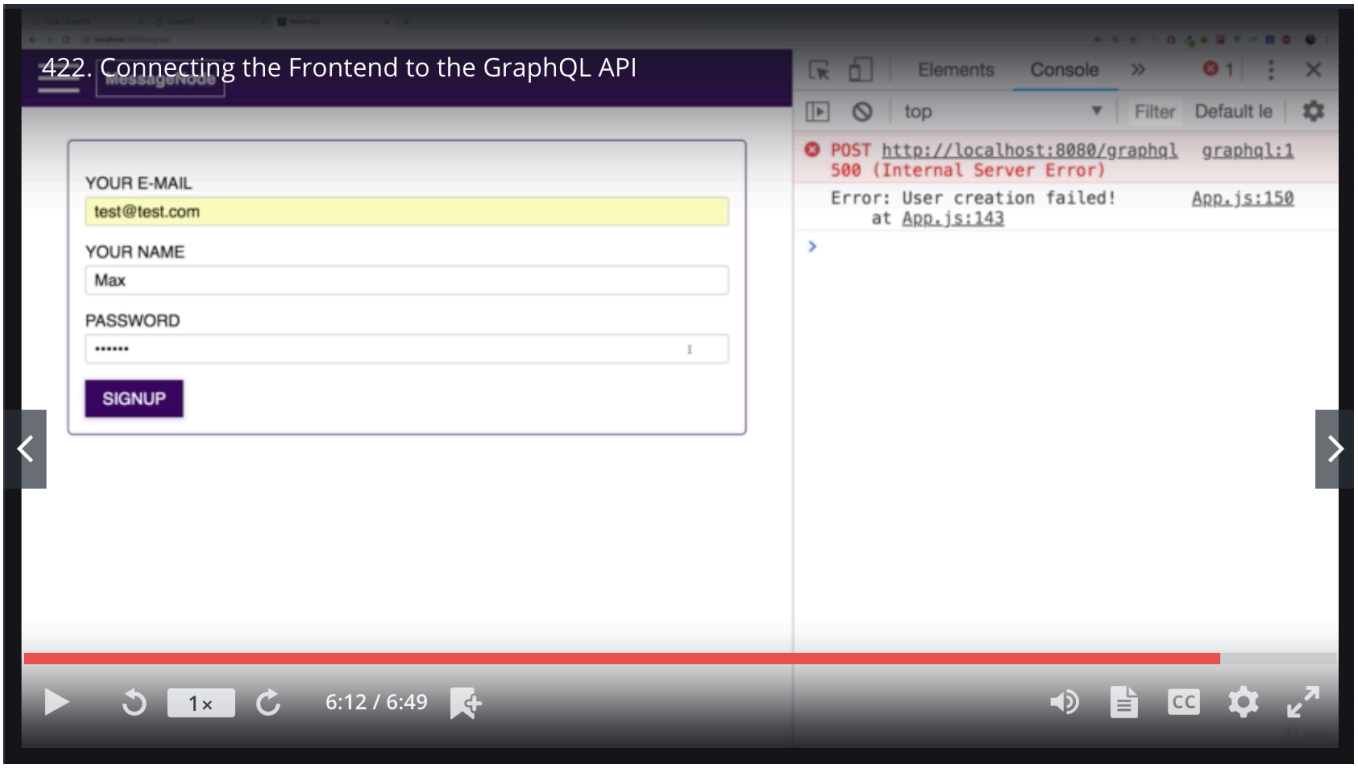
- this is a email address which is already taken. and therefore i should fail doing that. but this is a different error and you might expect an error where my method is not allowed and this can be a tricky thing to fix if you are brand new to craft.
- the reason for this error is we get this error as 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.



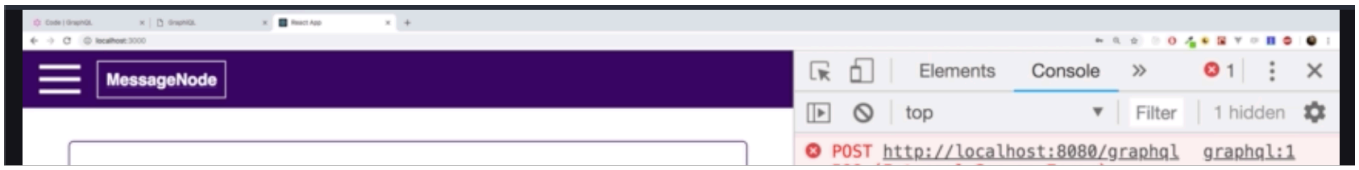
- so the solution is to go to that middleware where i set up my course headers and there i check if my req.method is equal to options and if it's all options request then, i will return res.sendStatus(200). so i will send an empty response with a status code of 200. i return so that 'next()' is not executed 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.

```
//App.js(f)

import React, { Component, Fragment } from 'react';
import { Route, Switch, Redirect, withRouter } from 'react-router-dom';

import Layout from './components/Layout/Layout';
import Backdrop from './components/Backdrop/Backdrop';
import Toolbar from './components/Toolbar/Toolbar';
import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
import ErrorHandler from './components/ErrorHandler/ErrorHandler';
import FeedPage from './pages/Feed/Feed';
import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
import LoginPage from './pages/Auth/Login';
import SignupPage from './pages/Auth/Signup';
import './App.css';

class App extends Component {
  state = {
    showBackdrop: false,
    showMobileNav: false,
    isAuth: false,
    token: null,
    userId: null,
    authLoading: false,
    error: null
  };

  componentDidMount() {
    const token = localStorage.getItem('token');
    const expiryDate = localStorage.getItem('expiryDate');
    if (!token || !expiryDate) {
      return;
```

```
    }
    if (new Date(expiryDate) <= new Date()) {
      this.logoutHandler();
      return;
    }
    const userId = localStorage.getItem('userId');
    const remainingMilliseconds =
      new Date(expiryDate).getTime() - new Date().getTime();
    this.setState({ isAuth: true, token: token, userId: userId });
    this.setAutoLogout(remainingMilliseconds);
  }

  mobileNavHandler = isOpen => {
    this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
  };

  backdropClickHandler = () => {
    this.setState({ showBackdrop: false, showMobileNav: false, error: null });
  };

  logoutHandler = () => {
    this.setState({ isAuth: false, token: null });
    localStorage.removeItem('token');
    localStorage.removeItem('expiryDate');
    localStorage.removeItem('userId');
  };

  loginHandler = (event, authData) => {
    event.preventDefault();
    this.setState({ authLoading: true });
    fetch('http://localhost:8080/auth/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        email: authData.email,
        password: authData.password
      })
    })
      .then(res => {
        if (res.status === 422) {
          throw new Error('Validation failed.');
        }
        if (res.status !== 200 && res.status !== 201) {
          console.log('Error!');
          throw new Error('Could not authenticate you!');
        }
        return res.json();
      })
      .then(resData => {
        console.log(resData);
        this.setState({
          isAuth: true,
          token: resData.token,
          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         isAuth: 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: `
124     mutation {
125       createUser(userInput: {email: "${authData.signupForm.email.value}", name:
"${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(
```

```
            "Validation failed. Make sure the email address isn't used yet!"
          );
        }
        if(resData.errors) {
          throw new Error('User creation failed!');
        }
        console.log(resData);
        this.setState({ isAuth: false, authLoading: false });
        this.props.history.replace('/');
      })
      .catch(err => {
        console.log(err);
        this.setState({
          isAuth: false,
          authLoading: false,
          error: err
        });
      });
  };

  setAutoLogout = milliseconds => {
    setTimeout(() => {
      this.logoutHandler();
    }, milliseconds);
  };

  errorHandler = () => {
    this.setState({ error: null });
  };

  render() {
    let routes = (
      <Switch>
        <Route
          path="/"
          exact
          render={props => (
            <LoginPage
              {...props}
              onLogin={this.loginHandler}
              loading={this.state.authLoading}
            />
          )}
        />
        <Route
          path="/signup"
          exact
          render={props => (
            <SignupPage
              {...props}
              onSignup={this.signupHandler}
              loading={this.state.authLoading}
            />
          )}
        />
        <Redirect to="/" />
```

```
      </Switch>
    );
    if (this.state.isAuth) {
      routes = (
        <Switch>
          <Route
            path="/"
            exact
            render={props => (
              <FeedPage userId={this.state.userId} token={this.state.token} />
            )}
          />
          <Route
            path="/:postId"
            render={props => (
              <SinglePostPage
                {...props}
                userId={this.state.userId}
                token={this.state.token}
              />
            )}
          />
          <Redirect to="/" />
        </Switch>
      );
    }
    return (
      <Fragment>
        {this.state.showBackdrop && (
          <Backdrop onClick={this.backdropClickHandler} />
        )}
        <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
        <Layout
          header={
            <Toolbar>
              <MainNavigation
                onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
                onLogout={this.logoutHandler}
                isAuth={this.state.isAuth}
              />
            </Toolbar>
          }
          mobileNav={
            <MobileNavigation
              open={this.state.showMobileNav}
              mobile
              onChooseItem={this.mobileNavHandler.bind(this, false)}
              onLogout={this.logoutHandler}
              isAuth={this.state.isAuth}
            />
          }
        />
        {routes}
      </Fragment>
    );
  }
```

```
}

export default withRouter(App);
```

```
//app.js(b)

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const multer = require('multer');

const graphqlHttp = require('express-graphql');

const graphSchema = require('./graphql/schema');
const graphResolver = require('./graphql/resolvers');

const app = express();

const fileStorage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'images');
  },
  filename: (req, file, cb) => {
    cb(null, new Date().toISOString() + '-' + file.originalname);
  }
});

const fileFilter = (req, file, cb) => {
  if(
    file.mimetype === 'image/png' ||
    file.mimetype === 'image/jpg' ||
    file.mimetype === 'image/jpeg'
    ) {
      cb(null, true);
    } else {
      cb(null, false);
    }
}

// app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
app.use(bodyParser.json()); // application/json
app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
app.use('/images', express.static(path.join(__dirname, 'images')));

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader(
    'Access-Control-Allow-Methods',
    'OPTIONS, GET, POST, PUT, PATCH, DELETE'
  );
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
  if(req.method === 'OPTIONS') {
    return res.sendStatus(200);
  }
```

```
  next();
});

app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver,
    graphiql: true,
    formatError(err) {
      if (!err.originalError) {
        return err;
      }
      const data = err.originalError.data;
      const message = err.message || 'An error occurred.';
      const code = err.originalError.code || 500
      return { message: message, status: code, data: data };
    }
  })
);

app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));
```

* Chapter 423: Adding A Login Query & Resolver

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

```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

/**at 'login' in type RootQuery'
 * i will return not a user
 * but some data that contains, for example, user_id, the token
 * because these are the 2 pieces of information i happened to need here
 * and that makes sense as a response
 * of course you could return the overall user object
 */
module.exports = buildSchema(`
    type Post {
        _id: ID!
```

```
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    type AuthData {
        token: String!
        userId: String!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    type RootQuery {
        login(email: String!, password: String!): AuthData!
    }

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

    schema {
        query: RootQuery
        mutation: RootMutation
    }
`);
```

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');
const jwt = require('jsonwebtoken');

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

module.exports = {
    createUser: async function({ userInput }, req) {
        //const email = args.userInput.email;
        const errors = [];
        if (!validator.isEmail(userInput.email)){
            errors.push({ message: 'E-Mail is invalid.' });
        }
```

```
        if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password, { min: 5 })){
            errors.push({message: 'Password Too Short!'});
        }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        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() };
    },
    /**'{ email, password }' is from ./graphql/schema.js(b) 'type RootQuery'*/
    login: async function({ email, password }) {
        const user = await User.findOne({email: email});
        if(!user) {
            const error = new Error('User not found.');
            error.code = 404;
            throw error;
        }
        const isEqual = await bcrypt.compare(password, user.password);
        if(!isEqual) {
            const error = new Error('Password is incorrect.');
            error.code = 401;
            throw error;
        }
        const token = jwt.sign({
            userId: user._id.toString(),
            email: user.email
        },
        'somesupersecretsecret',
        { expiresIn: '1h' })
        /**'token' on the left is from ./graphql/schema.js(b) 'type AuthData' */
        return { token: token, userId: user._id.toString() };
    }
}
```

* 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.

```
//App.js(f)

import React, { Component, Fragment } from 'react';
import { Route, Switch, Redirect, withRouter } from 'react-router-dom';

import Layout from './components/Layout/Layout';
import Backdrop from './components/Backdrop/Backdrop';
import Toolbar from './components/Toolbar/Toolbar';
import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
import ErrorHandler from './components/ErrorHandler/ErrorHandler';
import FeedPage from './pages/Feed/Feed';
import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
import LoginPage from './pages/Auth/Login';
import SignupPage from './pages/Auth/Signup';
import './App.css';

class App extends Component {
  state = {
    showBackdrop: false,
    showMobileNav: false,
    isAuth: false,
    token: null,
    userId: null,
    authLoading: false,
    error: null
  };

  componentDidMount() {
    const token = localStorage.getItem('token');
    const expiryDate = localStorage.getItem('expiryDate');
    if (!token || !expiryDate) {
      return;
    }
    if (new Date(expiryDate) <= new Date()) {
      this.logoutHandler();
      return;
    }
```

```
    const userId = localStorage.getItem('userId');
    const remainingMilliseconds =
      new Date(expiryDate).getTime() - new Date().getTime();
    this.setState({ isAuth: true, token: token, userId: userId });
    this.setAutoLogout(remainingMilliseconds);
  }

  mobileNavHandler = isOpen => {
    this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
  };

  backdropClickHandler = () => {
    this.setState({ showBackdrop: false, showMobileNav: false, error: null });
  };

  logoutHandler = () => {
    this.setState({ isAuth: false, token: null });
    localStorage.removeItem('token');
    localStorage.removeItem('expiryDate');
    localStorage.removeItem('userId');
  };

  loginHandler = (event, authData) => {
    event.preventDefault();
      /**for a query,
     * you don't repeat query
     * you start right away with the curly braces
     * as we did it in the graphiql interface
     *
     * 'login' is from ./graphql/schema.js(b) 'type RootQuery login'
     * */
    const graphqlQuery = {
      query: `
        {
          login(email: "${authData.email}", password: "${authData.password}") {
            token
            userId
          }
        }
      `
    };
    this.setState({ authLoading: true });
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors && resData.errors[0].status === 422) {
          throw new Error(
            "Validation failed. Make sure the email address isn't used yet!"
```

```
        );
      }
      if (resData.errors) {
        throw new Error('User login failed!');
      }
      console.log(resData);
      this.setState({
        isAuth: true,
        token: resData.data.login.token,
        authLoading: false,
        userId: resData.data.login.userId
      });
      localStorage.setItem('token', resData.data.login.token);
      localStorage.setItem('userId', resData.data.login.userId);
      const remainingMilliseconds = 60 * 60 * 1000;
      const expiryDate = new Date(
        new Date().getTime() + remainingMilliseconds
      );
      localStorage.setItem('expiryDate', expiryDate.toISOString());
      this.setAutoLogout(remainingMilliseconds);
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isAuth: false,
        authLoading: false,
        error: err
      });
    });
};

signupHandler = (event, authData) => {
  event.preventDefault();
  this.setState({ authLoading: true });
  const graphqlQuery = {
    query: `
      mutation {
        createUser(userInput: {email: "${
          authData.signupForm.email.value
        }", name:"${authData.signupForm.name.value}", password:"${
      authData.signupForm.password.value
    }"}) {
          _id
          email
        }
      }
    `
  };
  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
```

```
        return res.json();
      })
      .then(resData => {
        if (resData.errors && resData.errors[0].status === 422) {
          throw new Error(
            "Validation failed. Make sure the email address isn't used yet!"
          );
        }
        if (resData.errors) {
          throw new Error('User creation failed!');
        }
        console.log(resData);
        this.setState({ isAuth: false, authLoading: false });
        this.props.history.replace('/');
      })
      .catch(err => {
        console.log(err);
        this.setState({
          isAuth: false,
          authLoading: false,
          error: err
        });
      });
  };

  setAutoLogout = milliseconds => {
    setTimeout(() => {
      this.logoutHandler();
    }, milliseconds);
  };

  errorHandler = () => {
    this.setState({ error: null });
  };

  render() {
    let routes = (
      <Switch>
        <Route
          path="/"
          exact
          render={props => (
            <LoginPage
              {...props}
              onLogin={this.loginHandler}
              loading={this.state.authLoading}
            />
          )}
        />
        <Route
          path="/signup"
          exact
          render={props => (
            <SignupPage
              {...props}
              onSignup={this.signupHandler}
```

```
            loading={this.state.authLoading}
          />
        )}
      />
      <Redirect to="/" />
    </Switch>
  );
  if (this.state.isAuth) {
    routes = (
      <Switch>
        <Route
          path="/"
          exact
          render={props => (
            <FeedPage userId={this.state.userId} token={this.state.token} />
          )}
        />
        <Route
          path="/:postId"
          render={props => (
            <SinglePostPage
              {...props}
              userId={this.state.userId}
              token={this.state.token}
            />
          )}
        />
        <Redirect to="/" />
      </Switch>
    );
  }
  return (
    <Fragment>
      {this.state.showBackdrop && (
        <Backdrop onClick={this.backdropClickHandler} />
      )}
      <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
      <Layout
        header={
          <Toolbar>
            <MainNavigation
              onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
              onLogout={this.logoutHandler}
              isAuth={this.state.isAuth}
            />
          </Toolbar>
        }
        mobileNav={
          <MobileNavigation
            open={this.state.showMobileNav}
            mobile
            onChooseItem={this.mobileNavHandler.bind(this, false)}
            onLogout={this.logoutHandler}
            isAuth={this.state.isAuth}
          />
        }
```

```
        />
        {routes}
      </Fragment>
    );
  }
}

export default withRouter(App);

```

* 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.

```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    type AuthData {
        token: String!
        userId: String!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }
```

```
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.isLength(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,
```

```
            password: hashedPw
        });
        const createdUser = await user.save();
        return { ...createdUser._doc, _id: createdUser._id.toString() };
    },
    login: async function({ email, password }) {
        const user = await User.findOne({email: email});
        if(!user) {
            const error = new Error('User not found.');
            error.code = 404;
            throw error;
        }
        const isEqual = await bcrypt.compare(password, user.password);
        if(!isEqual) {
            const error = new Error('Password is incorrect.');
            error.code = 401;
            throw error;
        }
        const token = jwt.sign({
            userId: user._id.toString(),
            email: user.email
        },
        'somesupersecretsecret',
        { expiresIn: '1h' })
        return { token: token, userId: user._id.toString() };
    },
    /**'postInput' is from 'type RootMutation ' */
    createPost: async function({ postInput }, req){
        const errors = [];
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Title is invalid.' });
            }
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Content is invalid.' });
            }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        const post = new User({
            title: postInput.title,
            content: postInput.content,
            imageUrl: postInput.imageUrl
        });
        const createdPost = await post.save();
        //Add post to users' posts
        return {
            ...createdPost._doc,
```

```
            _id: createdPost._id.toString(),
            createdAt: createdPost.createdAt.toISOString(),
            updatedAt: createdPost.updatedAt.toISOString()
        };
    }
}
```

* Chapter 426: Extracting User Data From The Auth Token

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

```
//./middleware/auth.js(b)

const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.get('Authorization');
  if (!authHeader) {
    /**'req.isAuth = false'
     * so that i can handle this inside of my resolvers.
     */
    req.isAuth = false
    return next();
  }
  const token = authHeader.split(' ')[1];
  let decodedToken;
  try {
    decodedToken = jwt.verify(token, 'somesupersecretsecret');
  } catch (err) {
    req.isAuth = false;
    return next();
  }
  if (!decodedToken) {
    req.isAuth = false;
    return next();
  }
  req.userId = decodedToken.userId;
  req.isAuth = true;
  next();
};

```

```
//app.js(b)

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const multer = require('multer');

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: 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 middleare 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
```

```
app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver,
    graphiql: true,
    formatError(err) {
      if (!err.originalError) {
        return err;
      }
      const data = err.originalError.data;
      const message = err.message || 'An error occurred.';
      const code = err.originalError.code || 500
      return { message: message, status: code, data: data };
    }
  })
);

app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));
```

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');
const jwt = require('jsonwebtoken');

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

module.exports = {
    createUser: async function({ userInput }, req) {
        //const email = args.userInput.email;
        const errors = [];
        if (!validator.isEmail(userInput.email)){
            errors.push({ message: 'E-Mail is invalid.' });
        }
        if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password,
{ min: 5 })){
            errors.push({message: 'Password Too Short!'});
        }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
```

```
            throw error;
        }
        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() };
    },
    login: async function({ email, password }) {
        const user = await User.findOne({email: email});
        if(!user) {
            const error = new Error('User not found.');
            error.code = 404;
            throw error;
        }
        const isEqual = await bcrypt.compare(password, user.password);
        if(!isEqual) {
            const error = new Error('Password is incorrect.');
            error.code = 401;
            throw error;
        }
        const token = jwt.sign({
            userId: user._id.toString(),
            email: user.email
        },
        'somesupersecretsecret',
        { expiresIn: '1h' })
        return { token: token, userId: user._id.toString() };
    },
    /**'postInput' is from 'type RootMutation ' */
    createPost: async function({ postInput }, req){
        if(!req.isAuth){
            const error = new Error('Not authenticated!');
            error.code = 401;
            throw error;
        }
        const errors = [];
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Title is invalid.' });
            }
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Content is invalid.' });
```

```
        }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        const user = await User.findById(req.userId);
        if(!user){
            const error = new Error('Invalid user.');
            error.code = 401;
            throw error;
        }
        const post = new User({
            title: postInput.title,
            content: postInput.content,
            imageUrl: postInput.imageUrl,
            creator: user
        });
        const createdPost = await post.save();
        user.posts.push(createdPost);
        return {
            ...createdPost._doc,
            _id: createdPost._id.toString(),
            createdAt: createdPost.createdAt.toISOString(),
            updatedAt: createdPost.updatedAt.toISOString()
        };
    }
}
```

* 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

```
//./src/pages/Feed/Feed.js(f)

```

```
import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    fetch('http://localhost:8080/auth/status', {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch user status.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({ status: resData.status });
      })
      .catch(this.catchError);

    this.loadPosts();
  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
      page--;
      this.setState({ postPage: page });
    }
```

```
    fetch('http://localhost:8080/feed/posts?page=' + page, {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch posts.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({
          posts: resData.posts.map(post => {
            return {
              ...post,
              imagePath: post.imageUrl
            };
          }),
          totalPosts: resData.totalItems,
          postsLoading: false
        });
      })
      .catch(this.catchError);
  };

  statusUpdateHandler = event => {
    event.preventDefault();
    fetch('http://localhost:8080/auth/status', {
      method: 'PATCH',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        status: this.state.status
      })
    })
      .then(res => {
        if (res.status !== 200 && res.status !== 201) {
          throw new Error("Can't update status!");
        }
        return res.json();
      })
      .then(resData => {
        console.log(resData);
      })
      .catch(this.catchError);
  };

  newPostHandler = () => {
    this.setState({ isEditing: true });
  };

  startEditPostHandler = postId => {
    this.setState(prevState => {
```

```
      const loadedPost = { ...prevState.posts.find(p => p._id === postId) };

      return {
        isEditing: true,
        editPost: loadedPost
      };
    });
  };

  cancelEditHandler = () => {
    this.setState({ isEditing: false, editPost: null });
  };

  finishEditHandler = postData => {
    this.setState({
      editLoading: true
    });
    const formData = new FormData();
    formData.append('title', postData.title);
    formData.append('content', postData.content);
    formData.append('image', postData.image);

    let graphqlQuery = {
      query: `
        mutation {
          createPost(postInput: {title: "${postData.title}", content: "${
        postData.content
      }", imageUrl: "some url"}) {
            _id
            title
            content
            imageUrl
            creator {
              name
            }
            createdAt
          }
        }
      `
    };

    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      body: JSON.stringify(graphqlQuery),
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      }
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors && resData.errors[0].status === 422) {
          throw new Error(
            "Validation failed. Make sure the email address isn't used yet!"
```

```
        );
      }
      if (resData.errors) {
        throw new Error('User login failed!');
      }
      console.log(resData);
      const post = {
        _id: resData.data.createPost._id,
        title: resData.data.createPost.title,
        content: resData.data.createPost.content,
        creator: resData.data.createPost.creator,
        createdAt: resData.data.createPost.createdAt
      };
      this.setState(prevState => {
        return {
          isEditing: false,
          editPost: null,
          editLoading: false
        };
      });
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isEditing: false,
        editPost: null,
        editLoading: false,
        error: err
      });
    });
};

statusInputChangeHandler = (input, value) => {
  this.setState({ status: value });
};

deletePostHandler = postId => {
  this.setState({ postsLoading: true });
  fetch('http://localhost:8080/feed/post/' + postId, {
    method: 'DELETE',
    headers: {
      Authorization: 'Bearer ' + this.props.token
    }
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error('Deleting a post failed!');
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
      this.loadPosts();
      // this.setState(prevState => {
      //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
      //   return { posts: updatedPosts, postsLoading: false };
```

```
      // });
    })
    .catch(err => {
      console.log(err);
      this.setState({ postsLoading: false });
    });
};

errorHandler = () => {
  this.setState({ error: null });
};

catchError = error => {
  this.setState({ error: error });
};

render() {
  return (
    <Fragment>
      <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
      <FeedEdit
        editing={this.state.isEditing}
        selectedPost={this.state.editPost}
        loading={this.state.editLoading}
        onCancelEdit={this.cancelEditHandler}
        onFinishEdit={this.finishEditHandler}
      />
      <section className="feed__status">
        <form onSubmit={this.statusUpdateHandler}>
          <Input
            type="text"
            placeholder="Your status"
            control="input"
            onChange={this.statusInputChangeHandler}
            value={this.state.status}
          />
          <Button mode="flat" type="submit">
            Update
          </Button>
        </form>
      </section>
      <section className="feed__control">
        <Button mode="raised" design="accent" onClick={this.newPostHandler}>
          New Post
        </Button>
      </section>
      <section className="feed">
        {this.state.postsLoading && (
          <div style={{ textAlign: 'center', marginTop: '2rem' }}>
            <Loader />
          </div>
        )}
        {this.state.posts.length <= 0 && !this.state.postsLoading ? (
          <p style={{ textAlign: 'center' }}>No posts found.</p>
        ) : null}
        {!this.state.postsLoading && (
```

```
          <Paginator
            onPrevious={this.loadPosts.bind(this, 'previous')}
            onNext={this.loadPosts.bind(this, 'next')}
            lastPage={Math.ceil(this.state.totalPosts / 2)}
            currentPage={this.state.postPage}
          >
            {this.state.posts.map(post => (
              <Post
                key={post._id}
                id={post._id}
                author={post.creator.name}
                date={new Date(post.createdAt).toLocaleDateString('en-US')}
                title={post.title}
                image={post.imageUrl}
                content={post.content}
                onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                onDelete={this.deletePostHandler.bind(this, post._id)}
              />
            ))}
          </Paginator>
        )}
      </section>
    </Fragment>
  );
  }
}

export default Feed;

```

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

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

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');
const jwt = require('jsonwebtoken');

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

module.exports = {
    createUser: async function({ userInput }, req) {
        //const email = args.userInput.email;
        const errors = [];
        if (!validator.isEmail(userInput.email)){
            errors.push({ message: 'E-Mail is invalid.' });
        }
        if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password,
{ min: 5 })){
            errors.push({message: 'Password Too Short!'});
        }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        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() };
    },
    login: async function({ email, password }) {
        const user = await User.findOne({email: email});
        if(!user) {
            const error = new Error('User not found.');
            error.code = 404;
            throw error;
        }
        const isEqual = await bcrypt.compare(password, user.password);
        if(!isEqual) {
            const error = new Error('Password is incorrect.');
            error.code = 401;
            throw error;
        }
```

```
        const token = jwt.sign({
            userId: user._id.toString(),
            email: user.email
        },
        'somesupersecretsecret',
        { expiresIn: '1h' })
        return { token: token, userId: user._id.toString() };
    },
    /**'postInput' is from 'type RootMutation ' */
    createPost: async function({ postInput }, req){
        if(!req.isAuth){
            const error = new Error('Not authenticated!');
            error.code = 401;
            throw error;
        }
        const errors = [];
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Title is invalid.' });
            }
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Content is invalid.' });
            }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        const user = await User.findById(req.userId);
        if(!user){
            const error = new Error('Invalid user.');
            error.code = 401;
            throw error;
        }
        const post = new User({
            title: postInput.title,
            content: postInput.content,
            imageUrl: postInput.imageUrl,
            creator: user
        });
        const createdPost = await post.save();
        user.posts.push(createdPost);
        await user.save()
        return {
            ...createdPost._doc,
            _id: createdPost._id.toString(),
            createdAt: createdPost.createdAt.toISOString(),
            updatedAt: createdPost.updatedAt.toISOString()
        };
    }
```

```
}
```

```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    fetch('http://localhost:8080/auth/status', {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch user status.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({ status: resData.status });
      })
      .catch(this.catchError);

    this.loadPosts();
  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
```

```
      page--;
      this.setState({ postPage: page });
    }
    fetch('http://localhost:8080/feed/posts?page=' + page, {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch posts.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({
          posts: resData.posts.map(post => {
            return {
              ...post,
              imagePath: post.imageUrl
            };
          }),
          totalPosts: resData.totalItems,
          postsLoading: false
        });
      })
      .catch(this.catchError);
  };

  statusUpdateHandler = event => {
    event.preventDefault();
    fetch('http://localhost:8080/auth/status', {
      method: 'PATCH',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        status: this.state.status
      })
    })
      .then(res => {
        if (res.status !== 200 && res.status !== 201) {
          throw new Error("Can't update status!");
        }
        return res.json();
      })
      .then(resData => {
        console.log(resData);
      })
      .catch(this.catchError);
  };

  newPostHandler = () => {
    this.setState({ isEditing: true });
  };
```

```
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 => {
```

```
        if (resData.errors && resData.errors[0].status === 422) {
          throw new Error(
            "Validation failed. Make sure the email address isn't used yet!"
          );
        }
        if (resData.errors) {
          throw new Error('User login failed!');
        }
        console.log(resData);
        const post = {
          _id: resData.data.createPost._id,
          title: resData.data.createPost.title,
          content: resData.data.createPost.content,
          creator: resData.data.createPost.creator,
          createdAt: resData.data.createPost.createdAt
        };
        this.setState(prevState => {
          return {
            isEditing: false,
            editPost: null,
            editLoading: false
          };
        });
      })
      .catch(err => {
        console.log(err);
        this.setState({
          isEditing: false,
          editPost: null,
          editLoading: false,
          error: err
        });
      });
  };

  statusInputChangeHandler = (input, value) => {
    this.setState({ status: value });
  };

  deletePostHandler = postId => {
    this.setState({ postsLoading: true });
    fetch('http://localhost:8080/feed/post/' + postId, {
      method: 'DELETE',
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200 && res.status !== 201) {
          throw new Error('Deleting a post failed!');
        }
        return res.json();
      })
      .then(resData => {
        console.log(resData);
        this.loadPosts();
```

```
      // this.setState(prevState => {
      //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
      //   return { posts: updatedPosts, postsLoading: false };
      // });
    })
    .catch(err => {
      console.log(err);
      this.setState({ postsLoading: false });
    });
};

errorHandler = () => {
  this.setState({ error: null });
};

catchError = error => {
  this.setState({ error: error });
};

render() {
  return (
    <Fragment>
      <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
      <FeedEdit
        editing={this.state.isEditing}
        selectedPost={this.state.editPost}
        loading={this.state.editLoading}
        onCancelEdit={this.cancelEditHandler}
        onFinishEdit={this.finishEditHandler}
      />
      <section className="feed__status">
        <form onSubmit={this.statusUpdateHandler}>
          <Input
            type="text"
            placeholder="Your status"
            control="input"
            onChange={this.statusInputChangeHandler}
            value={this.state.status}
          />
          <Button mode="flat" type="submit">
            Update
          </Button>
        </form>
      </section>
      <section className="feed__control">
        <Button mode="raised" design="accent" onClick={this.newPostHandler}>
          New Post
        </Button>
      </section>
      <section className="feed">
        {this.state.postsLoading && (
          <div style={{ textAlign: 'center', marginTop: '2rem' }}>
            <Loader />
          </div>
        )}
        {this.state.posts.length <= 0 && !this.state.postsLoading ? (
```

```
          <p style={{ textAlign: 'center' }}>No posts found.</p>
        ) : null}
        {!this.state.postsLoading && (
          <Paginator
            onPrevious={this.loadPosts.bind(this, 'previous')}
            onNext={this.loadPosts.bind(this, 'next')}
            lastPage={Math.ceil(this.state.totalPosts / 2)}
            currentPage={this.state.postPage}
          >
            {this.state.posts.map(post => (
              <Post
                key={post._id}
                id={post._id}
                author={post.creator.name}
                date={new Date(post.createdAt).toLocaleDateString('en-US')}
                title={post.title}
                image={post.imageUrl}
                content={post.content}
                onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                onDelete={this.deletePostHandler.bind(this, post._id)}
              />
            ))}
          </Paginator>
        )}
      </section>
    </Fragment>
  );
 }
}

export default Feed;

```

* 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.

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');
const jwt = require('jsonwebtoken');

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

module.exports = {
    createUser: async function({ userInput }, req) {
```

```
        //const email = args.userInput.email;
        const errors = [];
        if (!validator.isEmail(userInput.email)){
            errors.push({ message: 'E-Mail is invalid.' });
        }
        if (validator.isEmpty(userInput.password) || !validator.isLength(userInput.password,
{ min: 5 })){
            errors.push({message: 'Password Too Short!'});
        }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        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() };
    },
    login: async function({ email, password }) {
        const user = await User.findOne({email: email});
        if(!user) {
            const error = new Error('User not found.');
            error.code = 404;
            throw error;
        }
        const isEqual = await bcrypt.compare(password, user.password);
        if(!isEqual) {
            const error = new Error('Password is incorrect.');
            error.code = 401;
            throw error;
        }
        const token = jwt.sign({
            userId: user._id.toString(),
            email: user.email
        },
        'somesupersecretsecret',
        { expiresIn: '1h' })
        return { token: token, userId: user._id.toString() };
    },
    createPost: async function({ postInput }, req){
        if(!req.isAuth){
            const error = new Error('Not authenticated!');
            error.code = 401;
            throw error;
        }
```

```
    const errors = [];
    if(
        validator.isEmpty(postInput.title) ||
        !validator.isLength(postInput.title, { min: 5 })
        ) {
            errors.push({ message: 'Title is invalid.' });
        }
    if(
        validator.isEmpty(postInput.title) ||
        !validator.isLength(postInput.title, { min: 5 })
        ) {
            errors.push({ message: 'Content is invalid.' });
        }
    if(errors.length > 0){
        const error = new Error('Invalid Input.');
        error.data = errors;
        error.code = 422;
        throw error;
    }
    const user = await User.findById(req.userId);
    if(!user){
        const error = new Error('Invalid user.');
        error.code = 401;
        throw error;
    }
    const post = new User({
        title: postInput.title,
        content: postInput.content,
        imageUrl: postInput.imageUrl,
        creator: user
    });
    const createdPost = await post.save();
    user.posts.push(createdPost);
    await user.save()
    return {
        ...createdPost._doc,
        _id: createdPost._id.toString(),
        createdAt: createdPost.createdAt.toISOString(),
        updatedAt: createdPost.updatedAt.toISOString()
    };
},
posts: async function(args, req){
    if(!req.isAuth){
        const error = new Error('Not authenticated!');
        error.code = 401;
        throw error;
    }
    const totalPosts = await Post.find().countDocuments();
    const posts = await Post.find()
        .sort({createdAt: -1})
        .populate('creator');
    return {posts: posts.map(p => {
        return {
            ...p._doc,
            _id: p._id.toString(),
            createdAt: p.createdAt.toISOString(),
```

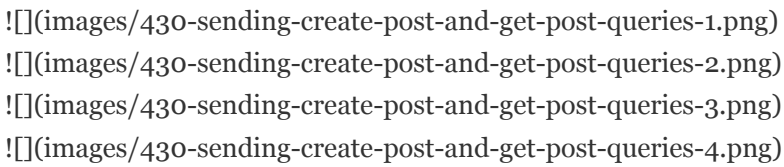
```
            updatedAt: p.updatedAt.toISOString()
        };
    }), totalPosts: totalPosts};
  }
}
```

```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    type PostData {
        posts: [Post!]!
        totalPosts: Int!
    }

    type AuthData {
        token: String!
        userId: String!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

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

    input PostInputData {
        title: String!
        content: String!
        imageUrl: String!
    }

```

```
    type RootMutation {
        createUser(userInput: UserInputData): User!
        createPost(postInput: PostInputData): Post!
    }

    schema {
        query: RootQuery
        mutation: RootMutation
    }
`);
```

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

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




```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    fetch('http://localhost:8080/auth/status', {
      headers: {
```

```
      Authorization: 'Bearer ' + this.props.token
    }
  })
    .then(res => {
      if (res.status !== 200) {
        throw new Error('Failed to fetch user status.');
      }
      return res.json();
    })
    .then(resData => {
      this.setState({ status: resData.status });
    })
    .catch(this.catchError);

  this.loadPosts();
}

loadPosts = direction => {
  if (direction) {
    this.setState({ postsLoading: true, posts: [] });
  }
  let page = this.state.postPage;
  if (direction === 'next') {
    page++;
    this.setState({ postPage: page });
  }
  if (direction === 'previous') {
    page--;
    this.setState({ postPage: page });
  }
  /**note that we see the entire flexibility of GraphQL
   * all doing its job here
   * we are fetching exactly that posts that we need
   * we don't fetch things like create, email, updated field, imageUrl
   * which don't need
   * we only fetch what we need
   * so we only fetch what we need
   * and we use graphQL to its fullest potential here.
   */
  const graphqlQuery = {
    query: `
      {
        posts {
          posts {
            _id
            title
            content
            creator {
              name
            }
            createdAt
          }
          totalPosts
        }
      }
    `
```

```
    }
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors) {
          throw new Error('Fetching posts failed!');
        }
        this.setState({
          posts: resData.data.posts.posts.map(post => {
            return {
              ...post,
              imagePath: post.imageUrl
            };
          }),
          totalPosts: resData.data.posts.totalPosts,
          postsLoading: false
        });
      })
      .catch(this.catchError);
  };

  statusUpdateHandler = event => {
    event.preventDefault();
    fetch('http://localhost:8080/auth/status', {
      method: 'PATCH',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        status: this.state.status
      })
    })
      .then(res => {
        if (res.status !== 200 && res.status !== 201) {
          throw new Error("Can't update status!");
        }
        return res.json();
      })
      .then(resData => {
        console.log(resData);
      })
      .catch(this.catchError);
  };

  newPostHandler = () => {
    this.setState({ isEditing: true });
```

```
};

startEditPostHandler = postId => {
  this.setState(prevState => {
    const loadedPost = { ...prevState.posts.find(p => p._id === postId) };

    return {
      isEditing: true,
      editPost: loadedPost
    };
  });
};

cancelEditHandler = () => {
  this.setState({ isEditing: false, editPost: null });
};

finishEditHandler = postData => {
  this.setState({
    editLoading: true
  });
  const formData = new FormData();
  formData.append('title', postData.title);
  formData.append('content', postData.content);
  formData.append('image', postData.image);

  let graphqlQuery = {
    query: `
      mutation {
        createPost(postInput: {title: "${postData.title}", content: "${
      postData.content
    }", imageUrl: "some url"}) {
          _id
          title
          content
          imageUrl
          creator {
            name
          }
          createdAt
        }
      }
    `
  };

  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    body: JSON.stringify(graphqlQuery),
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    }
  })
    .then(res => {
      return res.json();
    })
```

```
    .then(resData => {
      if (resData.errors && resData.errors[0].status === 422) {
        throw new Error(
          "Validation failed. Make sure the email address isn't used yet!"
        );
      }
      if (resData.errors) {
        throw new Error('User login failed!');
      }
      console.log(resData);
      const post = {
        _id: resData.data.createPost._id,
        title: resData.data.createPost.title,
        content: resData.data.createPost.content,
        creator: resData.data.createPost.creator,
        createdAt: resData.data.createPost.createdAt
      };
      this.setState(prevState => {
        let updatedPosts = [...prevState.posts];
        if (prevState.editPost) {
          const postIndex = prevState.posts.findIndex(
            p => p._id === prevState.editPost._id
          );
          updatedPosts[postIndex] = post;
        } else {
          updatedPosts.unshift(post);
        }
        return {
          posts: updatedPosts,
          isEditing: false,
          editPost: null,
          editLoading: false
        };
      });
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isEditing: false,
        editPost: null,
        editLoading: false,
        error: err
      });
    });
};

statusInputChangeHandler = (input, value) => {
  this.setState({ status: value });
};

deletePostHandler = postId => {
  this.setState({ postsLoading: true });
  fetch('http://localhost:8080/feed/post/' + postId, {
    method: 'DELETE',
    headers: {
      Authorization: 'Bearer ' + this.props.token
```

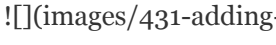
```
    }
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error('Deleting a post failed!');
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
      this.loadPosts();
      // this.setState(prevState => {
      //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
      //   return { posts: updatedPosts, postsLoading: false };
      // });
    })
    .catch(err => {
      console.log(err);
      this.setState({ postsLoading: false });
    });
};

errorHandler = () => {
  this.setState({ error: null });
};

catchError = error => {
  this.setState({ error: error });
};

render() {
  return (
    <Fragment>
      <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
      <FeedEdit
        editing={this.state.isEditing}
        selectedPost={this.state.editPost}
        loading={this.state.editLoading}
        onCancelEdit={this.cancelEditHandler}
        onFinishEdit={this.finishEditHandler}
      />
      <section className="feed__status">
        <form onSubmit={this.statusUpdateHandler}>
          <Input
            type="text"
            placeholder="Your status"
            control="input"
            onChange={this.statusInputChangeHandler}
            value={this.state.status}
          />
          <Button mode="flat" type="submit">
            Update
          </Button>
        </form>
      </section>
      <section className="feed__control">
```

```
          <Button mode="raised" design="accent" onClick={this.newPostHandler}>
            New Post
          </Button>
        </section>
        <section className="feed">
          {this.state.postsLoading && (
            <div style={{ textAlign: 'center', marginTop: '2rem' }}>
              <Loader />
            </div>
          )}
          {this.state.posts.length <= 0 && !this.state.postsLoading ? (
            <p style={{ textAlign: 'center' }}>No posts found.</p>
          ) : null}
          {!this.state.postsLoading && (
            <Paginator
              onPrevious={this.loadPosts.bind(this, 'previous')}
              onNext={this.loadPosts.bind(this, 'next')}
              lastPage={Math.ceil(this.state.totalPosts / 2)}
              currentPage={this.state.postPage}
            >
              {this.state.posts.map(post => (
                <Post
                  key={post._id}
                  id={post._id}
                  author={post.creator.name}
                  date={new Date(post.createdAt).toLocaleDateString('en-US')}
                  title={post.title}
                  image={post.imageUrl}
                  content={post.content}
                  onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                  onDelete={this.deletePostHandler.bind(this, post._id)}
                />
              ))}
            </Paginator>
          )}
        </section>
      </Fragment>
    );
  }
}

export default Feed;

```

* Chapter 431: Adding Pagination

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





```graphql
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    type PostData {
        posts: [Post!]!
        totalPosts: Int!
    }

    type AuthData {
        token: String!
        userId: String!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

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

```
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.isLength(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() };
```

```
    },
    login: async function({ email, password }) {
        const user = await User.findOne({email: email});
        if(!user) {
            const error = new Error('User not found.');
            error.code = 404;
            throw error;
        }
        const isEqual = await bcrypt.compare(password, user.password);
        if(!isEqual) {
            const error = new Error('Password is incorrect.');
            error.code = 401;
            throw error;
        }
        const token = jwt.sign({
            userId: user._id.toString(),
            email: user.email
        },
        'somesupersecretsecret',
        { expiresIn: '1h' })
        return { token: token, userId: user._id.toString() };
    },
    createPost: async function({ postInput }, req){
        if(!req.isAuth){
            const error = new Error('Not authenticated!');
            error.code = 401;
            throw error;
        }
        const errors = [];
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Title is invalid.' });
            }
        if(
            validator.isEmpty(postInput.title) ||
            !validator.isLength(postInput.title, { min: 5 })
            ) {
                errors.push({ message: 'Content is invalid.' });
            }
        if(errors.length > 0){
            const error = new Error('Invalid Input.');
            error.data = errors;
            error.code = 422;
            throw error;
        }
        const user = await User.findById(req.userId);
        if(!user){
            const error = new Error('Invalid user.');
            error.code = 401;
            throw error;
        }
        const post = new User({
            title: postInput.title,
            content: postInput.content,
```

```
            imageUrl: postInput.imageUrl,
            creator: user
        });
        const createdPost = await post.save();
        user.posts.push(createdPost);
        await user.save()
        return {
            ...createdPost._doc,
            _id: createdPost._id.toString(),
            createdAt: createdPost.createdAt.toISOString(),
            updatedAt: createdPost.updatedAt.toISOString()
        };
    },
    posts: async function({page}, req){
        if(!req.isAuth){
            const error = new Error('Not authenticated!');
            error.code = 401;
            throw error;
        }
        if(!page) {
            page = 1;
        }
        const perPage = 2;
        const totalPosts = await Post.find().countDocuments();
        const posts = await Post.find()
            .sort({createdAt: -1})
            /**if you are on page 2,
             * you have 2 - 1 = 1
             * if you skip 1 * 2,
             * so the first 2 items were on page 1
             */
            .skip((page - 1) * perPage)
            .limit(perPage)
            .populate('creator');
        return {posts: posts.map(p => {
            return {
                ...p._doc,
                _id: p._id.toString(),
                createdAt: p.createdAt.toISOString(),
                updatedAt: p.updatedAt.toISOString()
            };
        }), totalPosts: totalPosts};
    }
}
```

```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
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 = {
```

```
      query: `
        {
          posts(page: $(page)) {
            posts {
              _id
              title
              content
              creator {
                name
              }
              createdAt
            }
            totalPosts
          }
        }
      `
    }
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors) {
          throw new Error('Fetching posts failed!');
        }
        this.setState({
          posts: resData.data.posts.posts.map(post => {
            return {
              ...post,
              imagePath: post.imageUrl
            };
          }),
          totalPosts: resData.data.posts.totalPosts,
          postsLoading: false
        });
      })
      .catch(this.catchError);
  };

  statusUpdateHandler = event => {
    event.preventDefault();
    fetch('http://localhost:8080/auth/status', {
      method: 'PATCH',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        status: this.state.status
```

```
    })
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error("Can't update status!");
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
    })
    .catch(this.catchError);
};

newPostHandler = () => {
  this.setState({ isEditing: true });
};

startEditPostHandler = postId => {
  this.setState(prevState => {
    const loadedPost = { ...prevState.posts.find(p => p._id === postId) };

    return {
      isEditing: true,
      editPost: loadedPost
    };
  });
};

cancelEditHandler = () => {
  this.setState({ isEditing: false, editPost: null });
};

finishEditHandler = postData => {
  this.setState({
    editLoading: true
  });
  const formData = new FormData();
  formData.append('title', postData.title);
  formData.append('content', postData.content);
  formData.append('image', postData.image);

  let graphqlQuery = {
    query: `
      mutation {
        createPost(postInput: {title: "${postData.title}", content: "${
      postData.content
    }", imageUrl: "some url"}) {
          _id
          title
          content
          imageUrl
          creator {
            name
          }
          createdAt
```

```
      }
    }
  `
};

fetch('http://localhost:8080/graphql', {
  method: 'POST',
  body: JSON.stringify(graphqlQuery),
  headers: {
    Authorization: 'Bearer ' + this.props.token,
    'Content-Type': 'application/json'
  }
})
  .then(res => {
    return res.json();
  })
  .then(resData => {
    if (resData.errors && resData.errors[0].status === 422) {
      throw new Error(
        "Validation failed. Make sure the email address isn't used yet!"
      );
    }
    if (resData.errors) {
      throw new Error('User login failed!');
    }
    console.log(resData);
    const post = {
      _id: resData.data.createPost._id,
      title: resData.data.createPost.title,
      content: resData.data.createPost.content,
      creator: resData.data.createPost.creator,
      createdAt: resData.data.createPost.createdAt
    };
    this.setState(prevState => {
      let updatedPosts = [...prevState.posts];
      if (prevState.editPost) {
        const postIndex = prevState.posts.findIndex(
          p => p._id === prevState.editPost._id
        );
        updatedPosts[postIndex] = post;
      } else {
        /**this will remove one element
         * and add a new one at the beginning.
         */
        updatedPosts.pop();
        updatedPosts.unshift(post);
      }
      return {
        posts: updatedPosts,
        isEditing: false,
        editPost: null,
        editLoading: false
      };
    });
  })
  .catch(err => {
```

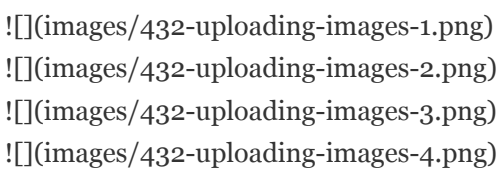
```
      console.log(err);
      this.setState({
        isEditing: false,
        editPost: null,
        editLoading: false,
        error: err
      });
    });
};

statusInputChangeHandler = (input, value) => {
  this.setState({ status: value });
};

deletePostHandler = postId => {
  this.setState({ postsLoading: true });
  fetch('http://localhost:8080/feed/post/' + postId, {
    method: 'DELETE',
    headers: {
      Authorization: 'Bearer ' + this.props.token
    }
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error('Deleting a post failed!');
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
      this.loadPosts();
      // this.setState(prevState => {
      //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
      //   return { posts: updatedPosts, postsLoading: false };
      // });
    })
    .catch(err => {
      console.log(err);
      this.setState({ postsLoading: false });
    });
};

errorHandler = () => {
  this.setState({ error: null });
};

catchError = error => {
  this.setState({ error: error });
};

render() {
  return (
    <Fragment>
      <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
      <FeedEdit
        editing={this.state.isEditing}
```

```
        selectedPost={this.state.editPost}
        loading={this.state.editLoading}
        onCancelEdit={this.cancelEditHandler}
        onFinishEdit={this.finishEditHandler}
      />
      <section className="feed__status">
        <form onSubmit={this.statusUpdateHandler}>
          <Input
            type="text"
            placeholder="Your status"
            control="input"
            onChange={this.statusInputChangeHandler}
            value={this.state.status}
          />
          <Button mode="flat" type="submit">
            Update
          </Button>
        </form>
      </section>
      <section className="feed__control">
        <Button mode="raised" design="accent" onClick={this.newPostHandler}>
          New Post
        </Button>
      </section>
      <section className="feed">
        {this.state.postsLoading && (
          <div style={{ textAlign: 'center', marginTop: '2rem' }}>
            <Loader />
          </div>
        )}
        {this.state.posts.length <= 0 && !this.state.postsLoading ? (
          <p style={{ textAlign: 'center' }}>No posts found.</p>
        ) : null}
        {!this.state.postsLoading && (
          <Paginator
            onPrevious={this.loadPosts.bind(this, 'previous')}
            onNext={this.loadPosts.bind(this, 'next')}
            lastPage={Math.ceil(this.state.totalPosts / 2)}
            currentPage={this.state.postPage}
          >
            {this.state.posts.map(post => (
              <Post
                key={post._id}
                id={post._id}
                author={post.creator.name}
                date={new Date(post.createdAt).toLocaleDateString('en-US')}
                title={post.title}
                image={post.imageUrl}
                content={post.content}
                onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                onDelete={this.deletePostHandler.bind(this, post._id)}
              />
            ))}
          </Paginator>
        )}
      </section>
```

```
      </Fragment>
    );
  }
}

export default Feed;

```

* Chapter 432: Uploading Images

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




```
//app.js(b)

const path = require('path');
const fs = require('fs');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const multer = require('multer');

const graphqlHttp = require('express-graphql');

const graphSchema = require('./graphql/schema');
const graphResolver = require('./graphql/resolvers');
const auth = require('./middleware/auth');

const app = express();

const fileStorage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'images');
  },
  filename: (req, file, cb) => {
    cb(null, new Date().toISOString() + '-' + file.originalname);
  }
});

const fileFilter = (req, file, cb) => {
  if(
    file.mimetype === 'image/png' ||
    file.mimetype === 'image/jpg' ||
    file.mimetype === 'image/jpeg'
```

```
    ) {
      cb(null, true);
    } else {
      cb(null, false);
    }
}

// app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
app.use(bodyParser.json()); // application/json
app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
app.use('/images', express.static(path.join(__dirname, 'images')));

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader(
    'Access-Control-Allow-Methods',
    'OPTIONS, GET, POST, PUT, PATCH, DELETE'
  );
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
  if(req.method === 'OPTIONS') {
    return res.sendStatus(200);
  }
  next();
});

app.use(auth);

app.put('/post-image', (req, res, next) => {
  if(!req.isAuth){
    throw new Error('Not authenticated!');
  }
  if(!req.file){
    return res.status(200).json({message: 'No file provided!'});
  }
  if(req.body.oldPath) {
    /**i will check for the existence of a body field
     * which is named 'oldPath'
     * which means that an oldPath was passed
     * with the incoming request in which case i wanna clear my old image
     * and i will pass in the oldPath
     * because we added a new image
     */
    clearImage(req.body.oldPath);
  }
  /**'req.file.path' is the path where multer stored the image
   * this is the path we can use the front-end
  */
  return res.status(201).json({message: 'File stored.', filePath: req.file.path});
});

app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver,
    graphiql: true,
    customFormatErrorFn(err) {
      if (!err.originalError) {
```

```
      return err;
    }
    const data = err.originalError.data;
    const message = err.message || 'An error occurred.';
    const code = err.originalError.code || 500
    return { message: message, status: code, data: data };
  }
 })
);

app.use((error, req, res, next) => {
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));

  const clearImage = filePath => {
    filePath = path.join(__dirname, '..', filePath);
    fs.unlink(filePath, err => console.log(err));
  };
```

```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

```

```
  componentDidMount() {
    fetch('http://localhost:8080/auth/status', {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch user status.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({ status: resData.status });
      })
      .catch(this.catchError);

    this.loadPosts();
  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
      page--;
      this.setState({ postPage: page });
    }
    const graphqlQuery = {
      query: `
        {
          posts(page: ${page}) {
            posts {
              _id
              title
              content
              imageUrl
              creator {
                name
              }
              createdAt
            }
            totalPosts
          }
        }
      `
    };
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
```

```
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Fetching posts failed!');
      }
      this.setState({
        posts: resData.data.posts.posts.map(post => {
          return {
            ...post,
            imagePath: post.imageUrl
          };
        }),
        totalPosts: resData.data.posts.totalPosts,
        postsLoading: false
      });
    })
    .catch(this.catchError);
};

statusUpdateHandler = event => {
  event.preventDefault();
  fetch('http://localhost:8080/auth/status', {
    method: 'PATCH',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      status: this.state.status
    })
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error("Can't update status!");
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
    })
    .catch(this.catchError);
};

newPostHandler = () => {
  this.setState({ isEditing: true });
};

startEditPostHandler = postId => {
  this.setState(prevState => {
    const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
```

```
    return {
      isEditing: true,
      editPost: loadedPost
    };
  });
};

cancelEditHandler = () => {
  this.setState({ isEditing: false, editPost: null });
};

finishEditHandler = postData => {
  this.setState({
    editLoading: true
  });
  const formData = new FormData();
  formData.append('image', postData.image);
  if(this.state.editPost) {
    formData.append('oldPath', this.state.editPost.imagePath);
  }
  fetch('http://localhost:8080/post-image', {
    /**method is 'PUT'
     * because on my app.js(b)
     * and i defined this to handle 'PUT' request
     */
    method: 'PUT',
    headers: {
      /**i should not set the content-type to application/json
       * otherwise it will be parsed as data
       * and that doesn't work for my binary data here.
       */
      Authorization: 'Bearer ' + this.props.token
    },
    body: formData
  })
  .then(res => res.json())
  .then(fileResData => {
    /**'filePath' is from app.js(b) */
    const imageUrl = fileResData.filePath;
    let graphqlQuery = {
      query: `
        mutation {
          createPost(postInput: {title: "${postData.title}", content: "${
        postData.content
      }", imageUrl: "${imageUrl}"}) {
            _id
            title
            content
            imageUrl
            creator {
              name
            }
            createdAt
          }
        }
```

```
      `
    };
    return fetch('http://localhost:8080/graphql', {
      method: 'POST',
      body: JSON.stringify(graphqlQuery),
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      }
    })
  }).then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors && resData.errors[0].status === 422) {
        throw new Error(
          "Validation failed. Make sure the email address isn't used yet!"
        );
      }
      if (resData.errors) {
        throw new Error('User login failed!');
      }
      console.log(resData);
      const post = {
        _id: resData.data.createPost._id,
        title: resData.data.createPost.title,
        content: resData.data.createPost.content,
        creator: resData.data.createPost.creator,
        createdAt: resData.data.createPost.createdAt,
        imagePath: resData.data.createPost.imageUrl
      };
      this.setState(prevState => {
        let updatedPosts = [...prevState.posts];
        if (prevState.editPost) {
          const postIndex = prevState.posts.findIndex(
            p => p._id === prevState.editPost._id
          );
          updatedPosts[postIndex] = post;
        } else {
          updatedPosts.pop();
          updatedPosts.unshift(post);
        }
        return {
          posts: updatedPosts,
          isEditing: false,
          editPost: null,
          editLoading: false
        };
      });
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isEditing: false,
        editPost: null,
        editLoading: false,
```

```
        error: err
      });
    });
};

statusInputChangeHandler = (input, value) => {
  this.setState({ status: value });
};

deletePostHandler = postId => {
  this.setState({ postsLoading: true });
  fetch('http://localhost:8080/feed/post/' + postId, {
    method: 'DELETE',
    headers: {
      Authorization: 'Bearer ' + this.props.token
    }
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error('Deleting a post failed!');
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
      this.loadPosts();
      // this.setState(prevState => {
      //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
      //   return { posts: updatedPosts, postsLoading: false };
      // });
    })
    .catch(err => {
      console.log(err);
      this.setState({ postsLoading: false });
    });
};

errorHandler = () => {
  this.setState({ error: null });
};

catchError = error => {
  this.setState({ error: error });
};

render() {
  return (
    <Fragment>
      <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
      <FeedEdit
        editing={this.state.isEditing}
        selectedPost={this.state.editPost}
        loading={this.state.editLoading}
        onCancelEdit={this.cancelEditHandler}
        onFinishEdit={this.finishEditHandler}
      />
```

```
        <section className="feed__status">
          <form onSubmit={this.statusUpdateHandler}>
            <Input
              type="text"
              placeholder="Your status"
              control="input"
              onChange={this.statusInputChangeHandler}
              value={this.state.status}
            />
            <Button mode="flat" type="submit">
              Update
            </Button>
          </form>
        </section>
        <section className="feed__control">
          <Button mode="raised" design="accent" onClick={this.newPostHandler}>
            New Post
          </Button>
        </section>
        <section className="feed">
          {this.state.postsLoading && (
            <div style={{ textAlign: 'center', marginTop: '2rem' }}>
              <Loader />
            </div>
          )}
          {this.state.posts.length <= 0 && !this.state.postsLoading ? (
            <p style={{ textAlign: 'center' }}>No posts found.</p>
          ) : null}
          {!this.state.postsLoading && (
            <Paginator
              onPrevious={this.loadPosts.bind(this, 'previous')}
              onNext={this.loadPosts.bind(this, 'next')}
              lastPage={Math.ceil(this.state.totalPosts / 2)}
              currentPage={this.state.postPage}
            >
              {this.state.posts.map(post => (
                <Post
                  key={post._id}
                  id={post._id}
                  author={post.creator.name}
                  date={new Date(post.createdAt).toLocaleDateString('en-US')}
                  title={post.title}
                  image={post.imageUrl}
                  content={post.content}
                  onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                  onDelete={this.deletePostHandler.bind(this, post._id)}
                />
              ))}
            </Paginator>
          )}
        </section>
      </Fragment>
    );
  }
}

```

```
export default Feed;

```

* Chapter 434: Viewing A Single Post

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


```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    type AuthData {
        token: String!
        userId: String!
    }

    type PostData {
        posts: [Post!]!
        totalPosts: Int!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    input PostInputData {
```

```
    title: String!
    content: String!
    imageUrl: String!
  }

  type RootQuery {
    login(email: String!, password: String!): AuthData!
    posts(page: Int): PostData!
    post(id: ID!): Post!
  }

  type RootMutation {
    createUser(userInput: UserInputData): User!
    createPost(postInput: PostInputData): Post!
  }

  schema {
    query: RootQuery
    mutation: RootMutation
  }
`);

```

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');
const jwt = require('jsonwebtoken');

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

module.exports = {
  createUser: async function({ userInput }, req) {
    //   const email = args.userInput.email;
    const errors = [];
    if (!validator.isEmail(userInput.email)) {
      errors.push({ message: 'E-Mail is invalid.' });
    }
    if (
      validator.isEmpty(userInput.password) ||
      !validator.isLength(userInput.password, { min: 5 })
    ) {
      errors.push({ message: 'Password too short!' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    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() };
  },
  login: async function({ email, password }) {
    const user = await User.findOne({ email: email });
    if (!user) {
      const error = new Error('User not found.');
      error.code = 401;
      throw error;
    }
    const isEqual = await bcrypt.compare(password, user.password);
    if (!isEqual) {
      const error = new Error('Password is incorrect.');
      error.code = 401;
      throw error;
    }
    const token = jwt.sign(
      {
        userId: user._id.toString(),
        email: user.email
      },
      'somesupersecretsecret',
      { expiresIn: '1h' }
    );
    return { token: token, userId: user._id.toString() };
  },
  createPost: async function({ postInput }, req) {
    if (!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
    const errors = [];
    if (
      validator.isEmpty(postInput.title) ||
      !validator.isLength(postInput.title, { min: 5 })
    ) {
      errors.push({ message: 'Title is invalid.' });
    }
    if (
      validator.isEmpty(postInput.content) ||
      !validator.isLength(postInput.content, { min: 5 })
    ) {
      errors.push({ message: 'Content is invalid.' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
```

```
    const user = await User.findById(req.userId);
    if (!user) {
      const error = new Error('Invalid user.');
      error.code = 401;
      throw error;
    }
    const post = new Post({
      title: postInput.title,
      content: postInput.content,
      imageUrl: postInput.imageUrl,
      creator: user
    });
    const createdPost = await post.save();
    user.posts.push(createdPost);
    await user.save();
    return {
      ...createdPost._doc,
      _id: createdPost._id.toString(),
      createdAt: createdPost.createdAt.toISOString(),
      updatedAt: createdPost.updatedAt.toISOString()
    };
  },
  posts: async function({ page }, req) {
    if (!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
    if (!page) {
      page = 1;
    }
    const perPage = 2;
    const totalPosts = await Post.find().countDocuments();
    const posts = await Post.find()
      .sort({ createdAt: -1 })
      .skip((page - 1) * perPage)
      .limit(perPage)
      .populate('creator');
    return {
      posts: posts.map(p => {
        return {
          ...p._doc,
          _id: p._id.toString(),
          createdAt: p.createdAt.toISOString(),
          updatedAt: p.updatedAt.toISOString()
        };
      }),
      totalPosts: totalPosts
    };
  },
  post: async function({ id }, req) {
    if (!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
```

```
    const post = await Post.findById(id).populate('creator');
    if (!post) {
      const error = new Error('No post found!');
      error.code = 404;
      throw error;
    }
    return {
      ...post._doc,
      _id: post._id.toString(),
      createdAt: post.createdAt.toISOString(),
      updatedAt: post.updatedAt.toISOString()
    };
  }
};

```

```
//./src/pages/Feed/SinglePost/SinglePost.js(f)

import React, { Component } from 'react';

import Image from '../../../components/Image/Image';
import './SinglePost.css';

class SinglePost extends Component {
  state = {
    title: '',
    author: '',
    date: '',
    image: '',
    content: ''
  };

  componentDidMount() {
    const postId = this.props.match.params.postId;
    const graphqlQuery = {
      query: `{
          post(id: "${postId}") {
            title
            content
            imageUrl
            creator {
              name
            }
            createdAt
          }
        }
      `
    };
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
```

```
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Fetching post failed!');
      }
      this.setState({
        title: resData.data.post.title,
        author: resData.data.post.creator.name,
        image: 'http://localhost:8080/' + resData.data.post.imageUrl,
        date: new Date(resData.data.post.createdAt).toLocaleDateString('en-US'),
        content: resData.data.post.content
      });
    })
    .catch(err => {
      console.log(err);
    });
  }

  render() {
    return (
      <section className="single-post">
        <h1>{this.state.title}</h1>
        <h2>
          Created by {this.state.author} on {this.state.date}
        </h2>
        <div className="single-post__image">
          <Image contain imageUrl={this.state.image} />
        </div>
        <p>{this.state.content}</p>
      </section>
    );
  }
}

export default SinglePost;

```

* Chapter 435: Updating Posts

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


```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

```

```
module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    type AuthData {
        token: String!
        userId: String!
    }

    type PostData {
        posts: [Post!]!
        totalPosts: Int!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    input PostInputData {
        title: String!
        content: String!
        imageUrl: String!
    }

    type RootQuery {
        login(email: String!, password: String!): AuthData!
        posts(page: Int): PostData!
        post(id: ID!): Post!
    }

    type RootMutation {
        createUser(userInput: UserInputData): User!
        createPost(postInput: PostInputData): Post!
        updatePost(id: ID!, postInput: PostInputData): Post!
    }

    schema {
        query: RootQuery
```

```
    mutation: RootMutation
  }
`);
```

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');
const jwt = require('jsonwebtoken');

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

module.exports = {
  createUser: async function({ userInput }, req) {
    //   const email = args.userInput.email;
    const errors = [];
    if (!validator.isEmail(userInput.email)) {
      errors.push({ message: 'E-Mail is invalid.' });
    }
    if (
      validator.isEmpty(userInput.password) ||
      !validator.isLength(userInput.password, { min: 5 })
    ) {
      errors.push({ message: 'Password too short!' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    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() };
  },
  login: async function({ email, password }) {
    const user = await User.findOne({ email: email });
    if (!user) {
      const error = new Error('User not found.');
      error.code = 401;
      throw error;
    }
    const isEqual = await bcrypt.compare(password, user.password);
    if (!isEqual) {
      const error = new Error('Password is incorrect.');
```

```
    error.code = 401;
    throw error;
  }
  const token = jwt.sign(
    {
      userId: user._id.toString(),
      email: user.email
    },
    'somesupersecretsecret',
    { expiresIn: '1h' }
  );
  return { token: token, userId: user._id.toString() };
},
createPost: async function({ postInput }, req) {
  if (!req.isAuth) {
    const error = new Error('Not authenticated!');
    error.code = 401;
    throw error;
  }
  const errors = [];
  if (
    validator.isEmpty(postInput.title) ||
    !validator.isLength(postInput.title, { min: 5 })
  ) {
    errors.push({ message: 'Title is invalid.' });
  }
  if (
    validator.isEmpty(postInput.content) ||
    !validator.isLength(postInput.content, { min: 5 })
  ) {
    errors.push({ message: 'Content is invalid.' });
  }
  if (errors.length > 0) {
    const error = new Error('Invalid input.');
    error.data = errors;
    error.code = 422;
    throw error;
  }
  const user = await User.findById(req.userId);
  if (!user) {
    const error = new Error('Invalid user.');
    error.code = 401;
    throw error;
  }
  const post = new Post({
    title: postInput.title,
    content: postInput.content,
    imageUrl: postInput.imageUrl,
    creator: user
  });
  const createdPost = await post.save();
  user.posts.push(createdPost);
  await user.save();
  return {
    ...createdPost._doc,
    _id: createdPost._id.toString(),
```

```
    createdAt: createdPost.createdAt.toISOString(),
    updatedAt: createdPost.updatedAt.toISOString()
  };
},
posts: async function({ page }, req) {
  if (!req.isAuth) {
    const error = new Error('Not authenticated!');
    error.code = 401;
    throw error;
  }
  if (!page) {
    page = 1;
  }
  const perPage = 2;
  const totalPosts = await Post.find().countDocuments();
  const posts = await Post.find()
    .sort({ createdAt: -1 })
    .skip((page - 1) * perPage)
    .limit(perPage)
    .populate('creator');
  return {
    posts: posts.map(p => {
      return {
        ...p._doc,
        _id: p._id.toString(),
        createdAt: p.createdAt.toISOString(),
        updatedAt: p.updatedAt.toISOString()
      };
    }),
    totalPosts: totalPosts
  };
},
post: async function({ id }, req) {
  if (!req.isAuth) {
    const error = new Error('Not authenticated!');
    error.code = 401;
    throw error;
  }
  const post = await Post.findById(id).populate('creator');
  if (!post) {
    const error = new Error('No post found!');
    error.code = 404;
    throw error;
  }
  return {
    ...post._doc,
    _id: post._id.toString(),
    createdAt: post.createdAt.toISOString(),
    updatedAt: post.updatedAt.toISOString()
  };
},
updatePost: async function({id, postInput}, req) {
    if(!req.isAuth) {
        const error = new Error('Not authenticated!');
        error.code = 401;
        throw error;
```

```
    }
    const post = await Post.findById(id).populate('creator');
    if(!post) {
        const error = new Error('No post found!');
        error.code = 404;
        throw error;
    }
    if(post.creator._id.toString() !== req.userId.toString()) {
      const error = new Error('Not authorized!');
      error.code = 403;
      throw error;
    }
    const errors = [];
    if (
      validator.isEmpty(postInput.title) ||
      !validator.isLength(postInput.title, { min: 5 })
    ) {
      errors.push({ message: 'Title is invalid.' });
    }
    if (
      validator.isEmpty(postInput.content) ||
      !validator.isLength(postInput.content, { min: 5 })
    ) {
      errors.push({ message: 'Content is invalid.' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    post.title = postInput.title;
    post.content = postInput.content;
    /**it has to be checked against a text
     * because it will be converted to text
     * by the way i'm sending it on the front-end
     */
    if(postInput.imageUrl !== 'undefined') {
      post.imageUrl = postInput.imageUrl;
    }
    const updatedPost = await post.save();
    return {
        ...updatedPost._doc,
        _id: updatedPost._id.toString(),
        createdAt: updatedPost.createdAt.toISOString(),
        updatedAt: updatedPost.updatedAt.toISOString()
    };
  }
};

```

```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
```

```
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    fetch('http://localhost:8080/auth/status', {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch user status.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({ status: resData.status });
      })
      .catch(this.catchError);

    this.loadPosts();
  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
      page--;
      this.setState({ postPage: page });
    }
    const graphqlQuery = {
      query: `
        {
          posts(page: ${page}) {
```

```
        posts {
          _id
          title
          content
          imageUrl
          creator {
            name
          }
          createdAt
        }
        totalPosts
      }
    }
  `
  };
  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Fetching posts failed!');
      }
      this.setState({
        posts: resData.data.posts.posts.map(post => {
          return {
            ...post,
            imagePath: post.imageUrl
          };
        }),
        totalPosts: resData.data.posts.totalPosts,
        postsLoading: false
      });
    })
    .catch(this.catchError);
};

statusUpdateHandler = event => {
  event.preventDefault();
  fetch('http://localhost:8080/auth/status', {
    method: 'PATCH',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      status: this.state.status
    })
  })
```

```
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error("Can't update status!");
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
    })
    .catch(this.catchError);
};

newPostHandler = () => {
  this.setState({ isEditing: true });
};

startEditPostHandler = postId => {
  this.setState(prevState => {
    const loadedPost = { ...prevState.posts.find(p => p._id === postId) };

    return {
      isEditing: true,
      editPost: loadedPost
    };
  });
};

cancelEditHandler = () => {
  this.setState({ isEditing: false, editPost: null });
};

finishEditHandler = postData => {
  this.setState({
    editLoading: true
  });
  const formData = new FormData();
  formData.append('image', postData.image);
  if(this.state.editPost) {
    formData.append('oldPath', this.state.editPost.imagePath);
  }
  fetch('http://localhost:8080/post-image', {
    method: 'PUT',
    headers: {
      Authorization: 'Bearer ' + this.props.token
    },
    body: formData
  })
  .then(res => res.json())
  .then(fileResData => {
    const imageUrl = fileResData.filePath;
    let graphqlQuery = {
      query: `
        mutation {
          createPost(postInput: {title: "${postData.title}", content: "${
        postData.content
      }", imageUrl: "${imageUrl}"}) {
```

```
          _id
          title
          content
          imageUrl
          creator {
            name
          }
          createdAt
        }
      }
    `
  };

  if(this.state.editPost){
    graphqlQuery = {
      query: `
      mutation {
        updatePost(
          id: "${this.state.editPost._id}",
          postInput: {
            title: "${postData.title}",
            content: "${postData.content}",
            imageUrl: "${imageUrl}"}) {
            _id
            title
            content
            imageUrl
            creator {
              name
            }
            createdAt
          }
        }
      `
    };
  }

  return fetch('http://localhost:8080/graphql', {
    method: 'POST',
    body: JSON.stringify(graphqlQuery),
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    }
  })
}).then(res => {
    return res.json();
  })
  .then(resData => {
    if (resData.errors && resData.errors[0].status === 422) {
      throw new Error(
        "Validation failed. Make sure the email address isn't used yet!"
      );
    }
    if (resData.errors) {
      throw new Error('User login failed!');
```

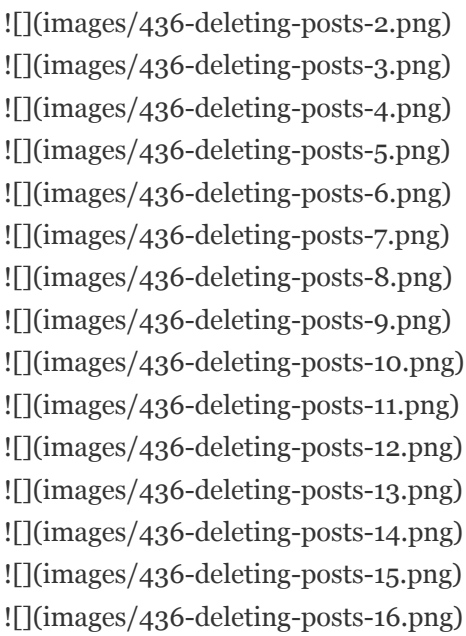
```
231       }
232       /**we can add a new variable 'resDataField'
233        * and by default, that should be 'createPost'
234        * because creating a post also is our default query up there.
235        */
236       let resDataField = 'createPost'
237       if (this.state.editPost) {
238         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: {
```

```
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200 && res.status !== 201) {
          throw new Error('Deleting a post failed!');
        }
        return res.json();
      })
      .then(resData => {
        console.log(resData);
        this.loadPosts();
        // this.setState(prevState => {
        //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
        //   return { posts: updatedPosts, postsLoading: false };
        // });
      })
      .catch(err => {
        console.log(err);
        this.setState({ postsLoading: false });
      });
  };

  errorHandler = () => {
    this.setState({ error: null });
  };

  catchError = error => {
    this.setState({ error: error });
  };

  render() {
    return (
      <Fragment>
        <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
        <FeedEdit
          editing={this.state.isEditing}
          selectedPost={this.state.editPost}
          loading={this.state.editLoading}
          onCancelEdit={this.cancelEditHandler}
          onFinishEdit={this.finishEditHandler}
        />
        <section className="feed__status">
          <form onSubmit={this.statusUpdateHandler}>
            <Input
              type="text"
              placeholder="Your status"
              control="input"
              onChange={this.statusInputChangeHandler}
              value={this.state.status}
            />
            <Button mode="flat" type="submit">
              Update
            </Button>
          </form>
        </section>
```

```
          <section className="feed__control">
            <Button mode="raised" design="accent" onClick={this.newPostHandler}>
              New Post
            </Button>
          </section>
          <section className="feed">
            {this.state.postsLoading && (
              <div style={{ textAlign: 'center', marginTop: '2rem' }}>
                <Loader />
              </div>
            )}
            {this.state.posts.length <= 0 && !this.state.postsLoading ? (
              <p style={{ textAlign: 'center' }}>No posts found.</p>
            ) : null}
            {!this.state.postsLoading && (
              <Paginator
                onPrevious={this.loadPosts.bind(this, 'previous')}
                onNext={this.loadPosts.bind(this, 'next')}
                lastPage={Math.ceil(this.state.totalPosts / 2)}
                currentPage={this.state.postPage}
              >
                {this.state.posts.map(post => (
                  <Post
                    key={post._id}
                    id={post._id}
                    author={post.creator.name}
                    date={new Date(post.createdAt).toLocaleDateString('en-US')}
                    title={post.title}
                    image={post.imageUrl}
                    content={post.content}
                    onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                    onDelete={this.deletePostHandler.bind(this, post._id)}
                  />
                ))}
              </Paginator>
            )}
          </section>
        </Fragment>
      );
    }
}

export default Feed;

```

* 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)




```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }

    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
```

```
    status: String!
    posts: [Post!]!
}

type AuthData {
    token: String!
    userId: String!
}

type PostData {
    posts: [Post!]!
    totalPosts: Int!
}

input UserInputData {
    email: String!
    name: String!
    password: String!
}

input PostInputData {
    title: String!
    content: String!
    imageUrl: String!
}

type RootQuery {
    login(email: String!, password: String!): AuthData!
    posts(page: Int): PostData!
    post(id: ID!): Post!
}

type RootMutation {
    createUser(userInput: UserInputData): User!
    createPost(postInput: PostInputData): Post!
    updatePost(id: ID!, postInput: PostInputData): Post!
    deletePost(id: ID!): Boolean
}

schema {
    query: RootQuery
    mutation: RootMutation
}
`);

```

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
const validator = require('validator');
const jwt = require('jsonwebtoken');

const User = require('../models/user');
const Post = require('../models/post');
const { clearImage } = require('../util/file');

module.exports = {
```

```
  createUser: async function({ userInput }, req) {
    //   const email = args.userInput.email;
    const errors = [];
    if (!validator.isEmail(userInput.email)) {
      errors.push({ message: 'E-Mail is invalid.' });
    }
    if (
      validator.isEmpty(userInput.password) ||
      !validator.isLength(userInput.password, { min: 5 })
    ) {
      errors.push({ message: 'Password too short!' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    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() };
  },
  login: async function({ email, password }) {
    const user = await User.findOne({ email: email });
    if (!user) {
      const error = new Error('User not found.');
      error.code = 401;
      throw error;
    }
    const isEqual = await bcrypt.compare(password, user.password);
    if (!isEqual) {
      const error = new Error('Password is incorrect.');
      error.code = 401;
      throw error;
    }
    const token = jwt.sign(
      {
        userId: user._id.toString(),
        email: user.email
      },
      'somesupersecretsecret',
      { expiresIn: '1h' }
    );
    return { token: token, userId: user._id.toString() };
  },
  createPost: async function({ postInput }, req) {
```

```
    if (!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
    const errors = [];
    if (
      validator.isEmpty(postInput.title) ||
      !validator.isLength(postInput.title, { min: 5 })
    ) {
      errors.push({ message: 'Title is invalid.' });
    }
    if (
      validator.isEmpty(postInput.content) ||
      !validator.isLength(postInput.content, { min: 5 })
    ) {
      errors.push({ message: 'Content is invalid.' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    const user = await User.findById(req.userId);
    if (!user) {
      const error = new Error('Invalid user.');
      error.code = 401;
      throw error;
    }
    const post = new Post({
      title: postInput.title,
      content: postInput.content,
      imageUrl: postInput.imageUrl,
      creator: user
    });
    const createdPost = await post.save();
    user.posts.push(createdPost);
    await user.save();
    return {
      ...createdPost._doc,
      _id: createdPost._id.toString(),
      createdAt: createdPost.createdAt.toISOString(),
      updatedAt: createdPost.updatedAt.toISOString()
    };
  },
  posts: async function({ page }, req) {
    if (!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
    if (!page) {
      page = 1;
    }
    const perPage = 2;
```

```
    const totalPosts = await Post.find().countDocuments();
    const posts = await Post.find()
      .sort({ createdAt: -1 })
      .skip((page - 1) * perPage)
      .limit(perPage)
      .populate('creator');
    return {
      posts: posts.map(p => {
        return {
          ...p._doc,
          _id: p._id.toString(),
          createdAt: p.createdAt.toISOString(),
          updatedAt: p.updatedAt.toISOString()
        };
      }),
      totalPosts: totalPosts
    };
  },
  post: async function({ id }, req) {
    if (!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
    const post = await Post.findById(id).populate('creator');
    if (!post) {
      const error = new Error('No post found!');
      error.code = 404;
      throw error;
    }
    return {
      ...post._doc,
      _id: post._id.toString(),
      createdAt: post.createdAt.toISOString(),
      updatedAt: post.updatedAt.toISOString()
    };
  },
  updatePost: async function({id, postInput}, req) {
      if(!req.isAuth) {
          const error = new Error('Not authenticated!');
          error.code = 401;
          throw error;
      }
      const post = await Post.findById(id).populate('creator');
      if(!post) {
          const error = new Error('No post found!');
          error.code = 404;
          throw error;
      }
      if(post.creator._id.toString() !== req.userId.toString()) {
        const error = new Error('Not authorized!');
        error.code = 403;
        throw error;
      }
      const errors = [];
      if (
```

```
      validator.isEmpty(postInput.title) ||
      !validator.isLength(postInput.title, { min: 5 })
    ) {
      errors.push({ message: 'Title is invalid.' });
    }
    if (
      validator.isEmpty(postInput.content) ||
      !validator.isLength(postInput.content, { min: 5 })
    ) {
      errors.push({ message: 'Content is invalid.' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    post.title = postInput.title;
    post.content = postInput.content;
    if(postInput.imageUrl !== 'undefined') {
      post.imageUrl = postInput.imageUrl;
    }
    const updatedPost = await post.save();
    return {
      ...updatedPost._doc,
      _id: updatedPost._id.toString(),
      createdAt: updatedPost.createdAt.toISOString(),
      updatedAt: updatedPost.updatedAt.toISOString()
    };
  },
  deletePost: async function({ id }, req) {
    if(!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
    const post = await Post.findById(id);
    if(!post) {
      const error = new Error('No post found!');
      error.code = 404;
      throw error;
    }
    /**since i don't populate my creator field here,
     * the creator here is a not an object with _id
     * but creator itself is already the ID
     * because that is how it's stored in a post.
     * and we need to call populate to change that and the result
     * we are not doing that here
     * so we only access creator to directly get the ID of the user who created the post
     */
    if(post.creator.toString() !== req.userId.toString()) {
      const error = new Error('Not authorized!');
      error.code = 403;
      throw error;
    }
    clearImage(post.imageUrl);
```

```
    await Post.findByIdAndRemove(id);
    const user = await User.findById(req.userId);
    user.posts.pull(id);
    await user.save();
    /**in ./graphql/schema.js(b),
     * i find that i wanna return boolean
     */
    return true;
  }
};

```

```
//./util/file.js(b)

const path = require('path');
const fs = require('fs');

const clearImage = filePath => {
    filePath = path.join(__dirname, '..', filePath);
    fs.unlink(filePath, err => console.log(err));
};

exports.clearImage = clearImage;
```

```
//app.js(b)

const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const multer = require('multer');

const graphqlHttp = require('express-graphql');

const graphSchema = require('./graphql/schema');
const graphResolver = require('./graphql/resolvers');
const auth = require('./middleware/auth');
const { clearImage } = require('./util/file');

const app = express();

const fileStorage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'images');
  },
  filename: (req, file, cb) => {
    cb(null, new Date().toISOString() + '-' + file.originalname);
  }
});

const fileFilter = (req, file, cb) => {
  if(
    file.mimetype === 'image/png' ||
    file.mimetype === 'image/jpg' ||
    file.mimetype === 'image/jpeg'
    ) {
```

```
    cb(null, true);
    } else {
    cb(null, false);
    }
}

// app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
app.use(bodyParser.json()); // application/json
app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
app.use('/images', express.static(path.join(__dirname, 'images')));

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader(
    'Access-Control-Allow-Methods',
    'OPTIONS, GET, POST, PUT, PATCH, DELETE'
  );
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
  if(req.method === 'OPTIONS') {
    return res.sendStatus(200);
  }
  next();
});

app.use(auth);

app.put('/post-image', (req, res, next) => {
  if(!req.isAuth){
    throw new Error('Not authenticated!');
  }
  if(!req.file){
    return res.status(200).json({message: 'No file provided!'});
  }
  if(req.body.oldPath) {
    clearImage(req.body.oldPath);
  }
  return res.status(201).json({message: 'File stored.', filePath: req.file.path});
});

app.use('/graphql', graphqlHttp({
    schema: graphSchema,
    rootValue: graphResolver,
    graphiql: true,
    customFormatErrorFn(err) {
      if (!err.originalError) {
        return err;
      }
      const data = err.originalError.data;
      const message = err.message || 'An error occurred.';
      const code = err.originalError.code || 500
      return { message: message, status: code, data: data };
    }
  })
);

app.use((error, req, res, next) => {
```

```
  console.log(error);
  const status = error.statusCode || 500;
  const message = error.message;
  const data = error.data;
  res.status(status).json({ message: message, data: data });
});

mongoose
  .connect(
    'mongodb+srv://maximilian:rldnjs12@cluster0-z3vlk.mongodb.net/message?retryWrites=true'
    )
  .then(result => {
    app.listen(8080);
  })
  .catch(err => console.log(err));
```

```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    fetch('http://localhost:8080/auth/status', {
      headers: {
        Authorization: 'Bearer ' + this.props.token
      }
    })
      .then(res => {
        if (res.status !== 200) {
          throw new Error('Failed to fetch user status.');
        }
        return res.json();
      })
      .then(resData => {
        this.setState({ status: resData.status });
      })
      .catch(this.catchError);
```

```
    this.loadPosts();
  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
      page--;
      this.setState({ postPage: page });
    }
    const graphqlQuery = {
      query: `
        {
          posts(page: ${page}) {
            posts {
              _id
              title
              content
              imageUrl
              creator {
                name
              }
              createdAt
            }
            totalPosts
          }
        }
      `
    };
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors) {
          throw new Error('Fetching posts failed!');
        }
        this.setState({
          posts: resData.data.posts.posts.map(post => {
            return {
              ...post,
              imagePath: post.imageUrl
```

```
        };
      }),
      totalPosts: resData.data.posts.totalPosts,
      postsLoading: false
    });
  })
  .catch(this.catchError);
};

statusUpdateHandler = event => {
  event.preventDefault();
  fetch('http://localhost:8080/auth/status', {
    method: 'PATCH',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      status: this.state.status
    })
  })
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error("Can't update status!");
      }
      return res.json();
    })
    .then(resData => {
      console.log(resData);
    })
    .catch(this.catchError);
};

newPostHandler = () => {
  this.setState({ isEditing: true });
};

startEditPostHandler = postId => {
  this.setState(prevState => {
    const loadedPost = { ...prevState.posts.find(p => p._id === postId) };

    return {
      isEditing: true,
      editPost: loadedPost
    };
  });
};

cancelEditHandler = () => {
  this.setState({ isEditing: false, editPost: null });
};

finishEditHandler = postData => {
  this.setState({
    editLoading: true
  });
```

```
    const formData = new FormData();
    formData.append('image', postData.image);
    if(this.state.editPost) {
      formData.append('oldPath', this.state.editPost.imagePath);
    }
    fetch('http://localhost:8080/post-image', {
      method: 'PUT',
      headers: {
        Authorization: 'Bearer ' + this.props.token
      },
      body: formData
    })
    .then(res => res.json())
    .then(fileResData => {
      const imageUrl = fileResData.filePath;
      let graphqlQuery = {
        query: `
          mutation {
            createPost(postInput: {title: "${postData.title}", content: "${
          postData.content
        }", imageUrl: "${imageUrl}"}) {
              _id
              title
              content
              imageUrl
              creator {
                name
              }
              createdAt
            }
          }
        `
      };

      if(this.state.editPost){
        graphqlQuery = {
          query: `
          mutation {
            updatePost(
              id: "${this.state.editPost._id}",
              postInput: {
                title: "${postData.title}",
                content: "${postData.content}",
                imageUrl: "${imageUrl}"}) {
                _id
                title
                content
                imageUrl
                creator {
                  name
                }
                createdAt
              }
            }
          `
        };
```

```
    }

    return fetch('http://localhost:8080/graphql', {
      method: 'POST',
      body: JSON.stringify(graphqlQuery),
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      }
    })
  }).then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors && resData.errors[0].status === 422) {
        throw new Error(
          "Validation failed. Make sure the email address isn't used yet!"
        );
      }
      if (resData.errors) {
        throw new Error('User login failed!');
      }
      let resDataField = 'createPost'
      if (this.state.editPost) {
        resDataField = 'updatePost';
      }
      const post = {
        _id: resData.data[resDataField]._id,
        title: resData.data[resDataField].title,
        content: resData.data[resDataField].content,
        creator: resData.data[resDataField].creator,
        createdAt: resData.data[resDataField].createdAt,
        imagePath: resData.data[resDataField].imageUrl
      };
      this.setState(prevState => {
        let updatedPosts = [...prevState.posts];
        if (prevState.editPost) {
          const postIndex = prevState.posts.findIndex(
            p => p._id === prevState.editPost._id
          );
          updatedPosts[postIndex] = post;
        } else {
          updatedPosts.pop();
          updatedPosts.unshift(post);
        }
        return {
          posts: updatedPosts,
          isEditing: false,
          editPost: null,
          editLoading: false
        };
      });
    })
    .catch(err => {
      console.log(err);
      this.setState({
```

```
        isEditing: false,
        editPost: null,
        editLoading: false,
        error: err
      });
    });
};

statusInputChangeHandler = (input, value) => {
  this.setState({ status: value });
};

deletePostHandler = postId => {
  this.setState({ postsLoading: true });
  const graphqlQuery = {
    /**we have no nested object here
     * so we couldn't even get any nested or detailed data
     * we only get true or false
     */
    query: `
      mutation {
        deletePost(id: "${postId}")
      }
    `
  };
  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if(resData.errors) {
        throw new Error('Deleting the post failed!');
      }
      console.log(resData);
      this.loadPosts();
      // this.setState(prevState => {
      //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
      //   return { posts: updatedPosts, postsLoading: false };
      // });
    })
    .catch(err => {
      console.log(err);
      this.setState({ postsLoading: false });
    });
};

errorHandler = () => {
  this.setState({ error: null });
};
```

```
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}
```

```
            date={new Date(post.createdAt).toLocaleDateString('en-US')}
            title={post.title}
            image={post.imageUrl}
            content={post.content}
            onStartEdit={this.startEditPostHandler.bind(this, post._id)}
            onDelete={this.deletePostHandler.bind(this, post._id)}
          />
        ))}
      </Paginator>
    )}
  </section>
 </Fragment>
 );
 }
}

export default Feed;

```

* 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


```
//./graphql/schema.js(b)

const { buildSchema } = require('graphql');

module.exports = buildSchema(`
    type Post {
        _id: ID!
        title: String!
        content: String!
        imageUrl: String!
        creator: User!
        createdAt: String!
        updatedAt: String!
    }
```

```
    type User {
        _id: ID!
        name: String!
        email: String!
        password: String
        status: String!
        posts: [Post!]!
    }

    type AuthData {
        token: String!
        userId: String!
    }

    type PostData {
        posts: [Post!]!
        totalPosts: Int!
    }

    input UserInputData {
        email: String!
        name: String!
        password: String!
    }

    input PostInputData {
        title: String!
        content: String!
        imageUrl: String!
    }

    type RootQuery {
        login(email: String!, password: String!): AuthData!
        posts(page: Int): PostData!
        post(id: ID!): Post!
        user: User!
    }

    type RootMutation {
        createUser(userInput: UserInputData): User!
        createPost(postInput: PostInputData): Post!
        updatePost(id: ID!, postInput: PostInputData): Post!
        deletePost(id: ID!): Boolean
        updateStatus(status: String!): User!
    }

    schema {
        query: RootQuery
        mutation: RootMutation
    }
`);

```

```
//./graphql/resolvers.js(b)

const bcrypt = require('bcryptjs');
```

```
const validator = require('validator');
const jwt = require('jsonwebtoken');

const User = require('../models/user');
const Post = require('../models/post');
const { clearImage } = require('../util/file');

module.exports = {
  createUser: async function({ userInput }, req) {
    //   const email = args.userInput.email;
    const errors = [];
    if (!validator.isEmail(userInput.email)) {
      errors.push({ message: 'E-Mail is invalid.' });
    }
    if (
      validator.isEmpty(userInput.password) ||
      !validator.isLength(userInput.password, { min: 5 })
    ) {
      errors.push({ message: 'Password too short!' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    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() };
  },
  login: async function({ email, password }) {
    const user = await User.findOne({ email: email });
    if (!user) {
      const error = new Error('User not found.');
      error.code = 401;
      throw error;
    }
    const isEqual = await bcrypt.compare(password, user.password);
    if (!isEqual) {
      const error = new Error('Password is incorrect.');
      error.code = 401;
      throw error;
    }
    const token = jwt.sign(
      {
        userId: user._id.toString(),
```

```
      email: user.email
    },
    'somesupersecretsecret',
    { expiresIn: '1h' }
  );
  return { token: token, userId: user._id.toString() };
},
createPost: async function({ postInput }, req) {
  if (!req.isAuth) {
    const error = new Error('Not authenticated!');
    error.code = 401;
    throw error;
  }
  const errors = [];
  if (
    validator.isEmpty(postInput.title) ||
    !validator.isLength(postInput.title, { min: 5 })
  ) {
    errors.push({ message: 'Title is invalid.' });
  }
  if (
    validator.isEmpty(postInput.content) ||
    !validator.isLength(postInput.content, { min: 5 })
  ) {
    errors.push({ message: 'Content is invalid.' });
  }
  if (errors.length > 0) {
    const error = new Error('Invalid input.');
    error.data = errors;
    error.code = 422;
    throw error;
  }
  const user = await User.findById(req.userId);
  if (!user) {
    const error = new Error('Invalid user.');
    error.code = 401;
    throw error;
  }
  const post = new Post({
    title: postInput.title,
    content: postInput.content,
    imageUrl: postInput.imageUrl,
    creator: user
  });
  const createdPost = await post.save();
  user.posts.push(createdPost);
  await user.save();
  return {
    ...createdPost._doc,
    _id: createdPost._id.toString(),
    createdAt: createdPost.createdAt.toISOString(),
    updatedAt: createdPost.updatedAt.toISOString()
  };
},
posts: async function({ page }, req) {
  if (!req.isAuth) {
```

```
    const error = new Error('Not authenticated!');
    error.code = 401;
    throw error;
  }
  if (!page) {
    page = 1;
  }
  const perPage = 2;
  const totalPosts = await Post.find().countDocuments();
  const posts = await Post.find()
    .sort({ createdAt: -1 })
    .skip((page - 1) * perPage)
    .limit(perPage)
    .populate('creator');
  return {
    posts: posts.map(p => {
      return {
        ...p._doc,
        _id: p._id.toString(),
        createdAt: p.createdAt.toISOString(),
        updatedAt: p.updatedAt.toISOString()
      };
    }),
    totalPosts: totalPosts
  };
},
post: async function({ id }, req) {
  if (!req.isAuth) {
    const error = new Error('Not authenticated!');
    error.code = 401;
    throw error;
  }
  const post = await Post.findById(id).populate('creator');
  if (!post) {
    const error = new Error('No post found!');
    error.code = 404;
    throw error;
  }
  return {
    ...post._doc,
    _id: post._id.toString(),
    createdAt: post.createdAt.toISOString(),
    updatedAt: post.updatedAt.toISOString()
  };
},
updatePost: async function({id, postInput}, req) {
    if(!req.isAuth) {
        const error = new Error('Not authenticated!');
        error.code = 401;
        throw error;
    }
    const post = await Post.findById(id).populate('creator');
    if(!post) {
        const error = new Error('No post found!');
        error.code = 404;
        throw error;
```

```
    }
    if(post.creator._id.toString() !== req.userId.toString()) {
      const error = new Error('Not authorized!');
      error.code = 403;
      throw error;
    }
    const errors = [];
    if (
      validator.isEmpty(postInput.title) ||
      !validator.isLength(postInput.title, { min: 5 })
    ) {
      errors.push({ message: 'Title is invalid.' });
    }
    if (
      validator.isEmpty(postInput.content) ||
      !validator.isLength(postInput.content, { min: 5 })
    ) {
      errors.push({ message: 'Content is invalid.' });
    }
    if (errors.length > 0) {
      const error = new Error('Invalid input.');
      error.data = errors;
      error.code = 422;
      throw error;
    }
    post.title = postInput.title;
    post.content = postInput.content;
    if(postInput.imageUrl !== 'undefined') {
      post.imageUrl = postInput.imageUrl;
    }
    const updatedPost = await post.save();
    return {
      ...updatedPost._doc,
      _id: updatedPost._id.toString(),
      createdAt: updatedPost.createdAt.toISOString(),
      updatedAt: updatedPost.updatedAt.toISOString()
    };
  },
  deletePost: async function({ id }, req) {
    if(!req.isAuth) {
      const error = new Error('Not authenticated!');
      error.code = 401;
      throw error;
    }
    const post = await Post.findById(id);
    if(!post) {
      const error = new Error('No post found!');
      error.code = 404;
      throw error;
    }
    if(post.creator.toString() !== req.userId.toString()) {
      const error = new Error('Not authorized!');
      error.code = 403;
      throw error;
    }
    clearImage(post.imageUrl);
```

```
    await Post.findByIdAndRemove(id);
    const user = await User.findById(req.userId);
    user.posts.pull(id);
    await user.save();
    return true;
  },
  user: async function(args, req) {
    if(!req.isAuth) {
        const error = new Error('Not authenticated!');
        error.code = 401;
        throw error;
    }
    const user = await User.findById(req.userId);
    if(!user) {
            const error = new Error('Not authenticated!');
            error.code = 401;
            throw error;
    }
    return {...user._doc, _id: user._id.toString()}
  },
  updateStatus: async function({status}, req) {
    if(!req.isAuth) {
        const error = new Error('Not authenticated!');
        error.code = 401;
        throw error;
    }
    const user = await User.findById(req.userId);
    if(!user) {
        const error = new Error('Not authenticated!');
        error.code = 401;
        throw error;
    }
    user.status = status;
    await user.save();
    return {
        ...user._doc,
        _id: user._id.toString()
    }
  }
};

```

```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
```

```
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    const graphqlQuery = {
      query: `
      {
        user {
          status
        }
      }
      `
    }
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors) {
          throw new Error('Fetching status failed!');
        }
        this.setState({ status: resData.data.user.status });
      })
      .catch(this.catchError);

    this.loadPosts();
  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
      page--;
      this.setState({ postPage: page });
    }
    const graphqlQuery = {
```

```
    query: `
      {
        posts(page: ${page}) {
          posts {
            _id
            title
            content
            imageUrl
            creator {
              name
            }
            createdAt
          }
          totalPosts
        }
      }
    `
  };
  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Fetching posts failed!');
      }
      this.setState({
        posts: resData.data.posts.posts.map(post => {
          return {
            ...post,
            imagePath: post.imageUrl
          };
        }),
        totalPosts: resData.data.posts.totalPosts,
        postsLoading: false
      });
    })
    .catch(this.catchError);
};

statusUpdateHandler = event => {
  event.preventDefault();
  const graphqlQuery = {
    query: `
      mutation {
        updateStatus(status: "${this.state.status}") {
          status
        }
      }
```

```
      `
    }
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors) {
          throw new Error('Fetching posts failed!');
        }
        console.log(resData);
      })
      .catch(this.catchError);
  };

  newPostHandler = () => {
    this.setState({ isEditing: true });
  };

  startEditPostHandler = postId => {
    this.setState(prevState => {
      const loadedPost = { ...prevState.posts.find(p => p._id === postId) };

      return {
        isEditing: true,
        editPost: loadedPost
      };
    });
  };

  cancelEditHandler = () => {
    this.setState({ isEditing: false, editPost: null });
  };

  finishEditHandler = postData => {
    this.setState({
      editLoading: true
    });
    const formData = new FormData();
    formData.append('image', postData.image);
    if(this.state.editPost) {
      formData.append('oldPath', this.state.editPost.imagePath);
    }
    fetch('http://localhost:8080/post-image', {
      method: 'PUT',
      headers: {
        Authorization: 'Bearer ' + this.props.token
      },
      body: formData
```

```
    })
    .then(res => res.json())
    .then(fileResData => {
      const imageUrl = fileResData.filePath;
      let graphqlQuery = {
        query: `
          mutation {
            createPost(postInput: {title: "${postData.title}", content: "${
          postData.content
        }", imageUrl: "${imageUrl}"}) {
              _id
              title
              content
              imageUrl
              creator {
                name
              }
              createdAt
            }
          }
        `
      };

      if(this.state.editPost){
        graphqlQuery = {
          query: `
          mutation {
            updatePost(
              id: "${this.state.editPost._id}",
              postInput: {
                title: "${postData.title}",
                content: "${postData.content}",
                imageUrl: "${imageUrl}"}) {
                _id
                title
                content
                imageUrl
                creator {
                  name
                }
                createdAt
              }
            }
          `
        };
      }

      return fetch('http://localhost:8080/graphql', {
        method: 'POST',
        body: JSON.stringify(graphqlQuery),
        headers: {
          Authorization: 'Bearer ' + this.props.token,
          'Content-Type': 'application/json'
        }
      })
    }).then(res => {
```

```
      return res.json();
    })
    .then(resData => {
      if (resData.errors && resData.errors[0].status === 422) {
        throw new Error(
          "Validation failed. Make sure the email address isn't used yet!"
        );
      }
      if (resData.errors) {
        throw new Error('User login failed!');
      }
      let resDataField = 'createPost'
      if (this.state.editPost) {
        resDataField = 'updatePost';
      }
      const post = {
        _id: resData.data[resDataField]._id,
        title: resData.data[resDataField].title,
        content: resData.data[resDataField].content,
        creator: resData.data[resDataField].creator,
        createdAt: resData.data[resDataField].createdAt,
        imagePath: resData.data[resDataField].imageUrl
      };
      this.setState(prevState => {
        let updatedPosts = [...prevState.posts];
        if (prevState.editPost) {
          const postIndex = prevState.posts.findIndex(
            p => p._id === prevState.editPost._id
          );
          updatedPosts[postIndex] = post;
        } else {
          if (prevState.posts.length >= 2) {
            updatedPosts.pop();
          }
          updatedPosts.unshift(post);
        }
        return {
          posts: updatedPosts,
          isEditing: false,
          editPost: null,
          editLoading: false
        };
    });
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isEditing: false,
        editPost: null,
        editLoading: false,
        error: err
      });
    });
};

statusInputChangeHandler = (input, value) => {
```

```
    this.setState({ status: value });
  };

  deletePostHandler = postId => {
    this.setState({ postsLoading: true });
    const graphqlQuery = {
      /**we have no nested object here
       * so we couldn't even get any nested or detailed data
       * we only get true or false
       */
      query: `
        mutation {
          deletePost(id: "${postId}")
        }
      `
    };
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if(resData.errors) {
          throw new Error('Deleting the post failed!');
        }
        console.log(resData);
        this.loadPosts();
        // this.setState(prevState => {
        //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
        //   return { posts: updatedPosts, postsLoading: false };
        // });
      })
      .catch(err => {
        console.log(err);
        this.setState({ postsLoading: false });
      });
  };

  errorHandler = () => {
    this.setState({ error: null });
  };

  catchError = error => {
    this.setState({ error: error });
  };

  render() {
    return (
      <Fragment>
        <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
```

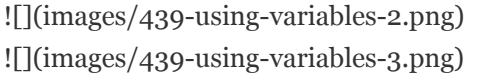
```
        <FeedEdit
          editing={this.state.isEditing}
          selectedPost={this.state.editPost}
          loading={this.state.editLoading}
          onCancelEdit={this.cancelEditHandler}
          onFinishEdit={this.finishEditHandler}
        />
        <section className="feed__status">
          <form onSubmit={this.statusUpdateHandler}>
            <Input
              type="text"
              placeholder="Your status"
              control="input"
              onChange={this.statusInputChangeHandler}
              value={this.state.status}
            />
            <Button mode="flat" type="submit">
              Update
            </Button>
          </form>
        </section>
        <section className="feed__control">
          <Button mode="raised" design="accent" onClick={this.newPostHandler}>
            New Post
          </Button>
        </section>
        <section className="feed">
          {this.state.postsLoading && (
            <div style={{ textAlign: 'center', marginTop: '2rem' }}>
              <Loader />
            </div>
          )}
          {this.state.posts.length <= 0 && !this.state.postsLoading ? (
            <p style={{ textAlign: 'center' }}>No posts found.</p>
          ) : null}
          {!this.state.postsLoading && (
            <Paginator
              onPrevious={this.loadPosts.bind(this, 'previous')}
              onNext={this.loadPosts.bind(this, 'next')}
              lastPage={Math.ceil(this.state.totalPosts / 2)}
              currentPage={this.state.postPage}
            >
              {this.state.posts.map(post => (
                <Post
                  key={post._id}
                  id={post._id}
                  author={post.creator.name}
                  date={new Date(post.createdAt).toLocaleDateString('en-US')}
                  title={post.title}
                  image={post.imageUrl}
                  content={post.content}
                  onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                  onDelete={this.deletePostHandler.bind(this, post._id)}
                />
              ))}
            </Paginator>
```

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

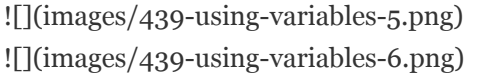
* Chapter 439: Using Variables

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

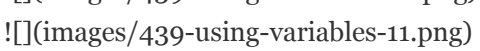






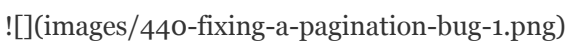
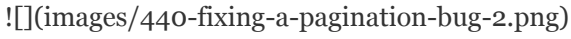
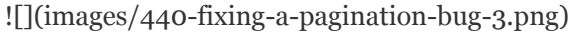
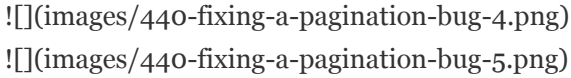
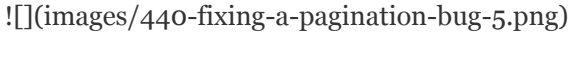




* Chapter 440: Fixing A Pagination Bug

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

```
//./src/pages/Feed/Feed.js(f)

import React, { Component, Fragment } from 'react';

import Post from '../../components/Feed/Post/Post';
import Button from '../../components/Button/Button';
import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
import Input from '../../components/Form/Input/Input';
import Paginator from '../../components/Paginator/Paginator';
import Loader from '../../components/Loader/Loader';
import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
import './Feed.css';

class Feed extends Component {
  state = {
    isEditing: false,
    posts: [],
    totalPosts: 0,
    editPost: null,
    status: '',
    postPage: 1,
    postsLoading: true,
    editLoading: false
  };

  componentDidMount() {
    const graphqlQuery = {
      query: `
        {
          user {
            status
          }
        }
      `
    };
    fetch('http://localhost:8080/graphql', {
```

```
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors) {
          throw new Error('Fetching status failed!');
        }
        this.setState({ status: resData.data.user.status });
      })
      .catch(this.catchError);

    this.loadPosts();
  }

  loadPosts = direction => {
    if (direction) {
      this.setState({ postsLoading: true, posts: [] });
    }
    let page = this.state.postPage;
    if (direction === 'next') {
      page++;
      this.setState({ postPage: page });
    }
    if (direction === 'previous') {
      page--;
      this.setState({ postPage: page });
    }
    /**you also remember that
     * for mutations we had to add mutation here
     * but for queries, we didn't have to add 'query' here
     * and if you would have added a query like this,
     *
     *    query: `query {posts{}()}`
     *
     * we would have got an error
     * now i will add it
     * because i will add something else
     * i will give this query a name
     * that doesn't make a difference.
     * it doesn't make it behave differently
     * it will help in error messages
     * but that name also allows us to do something else
     * so here we are getting all posts
     * here i will named as 'FetchPosts'
     * with this name assigned,
     * i can add parentheses after this name to define
     * which variables this query will use
     * and we create such a variable with a dollar sign
     * but then no curly braces
```

```
    * and then the name of the variable you wanna use
    * and that name is up to you
    * i will name it 'page'
    *
    * 'query FetchPosts($page: Int)'
    * this graphql syntax will be parsed on the server
    * this is not javascript code that runs on a client
    * this will only tell our graphql server
    * that we have a query which will use an internal variable
    * now i'm saying internal to a question is where do we use that '$page'
    * we use it in the place where we have dynamic variable value
    * and that would be where we currently inject our value.
    */
    const graphqlQuery = {
      query: `
        query FetchPosts($page: Int) {
          posts(page: $page) {
            posts {
              _id
              title
              content
              imageUrl
              creator {
                name
              }
              createdAt
            }
            totalPosts
          }
        }
      `,
      /*** the only question is
       * how do we get our page variable in javascript
       * into page variable in graphql '$page'
       * we add a 2nd property to that query object we are creating
       * 2nd property is 'variable' which has to be named 'variables'
       * and 1st property name has to be 'query'
       *
       * 'variables' is the object
       * where we can assign values to the variables we pass into our 'query'
       * it should be just 'page' without the $
       * and then my javascript variable or value
       * 'page' on the left is page in '$page'
       * 'page' on the right is page the javascript variable in 'page++' or 'page--'
       *
       */
      variables: {
        page: page
      }
    };
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      },
```

```
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Fetching posts failed!');
      }
      this.setState({
        posts: resData.data.posts.posts.map(post => {
          return {
            ...post,
            imagePath: post.imageUrl
          };
        }),
        totalPosts: resData.data.posts.totalPosts,
        postsLoading: false
      });
    })
    .catch(this.catchError);
};

statusUpdateHandler = event => {
  event.preventDefault();
  const graphqlQuery = {
    query: `
      mutation UpdateUserStatus($userStatus: String!) {
        updateStatus(status: $userStatus) {
          status
        }
      }
    `,
    variables: {
      userStatus: this.state.status
    }
  };
  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Fetching posts failed!');
      }
      console.log(resData);
    })
    .catch(this.catchError);
};
```

```
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
```

```
        }
      }
    `,
      variables: {
        title: postData.title,
        content: postData.content,
        imageUrl: imageUrl
      }
    };

    if (this.state.editPost) {
      graphqlQuery = {
        query: `
          mutation UpdateExistingPost($postId: ID!, $title: String!, $content: String!, $imageUrl: String!) {
            updatePost(id: $postId, postInput: {title: $title, content: $content, imageUrl: $imageUrl}) {
              _id
              title
              content
              imageUrl
              creator {
                name
              }
              createdAt
            }
          }
        `,
        variables: {
          postId: this.state.editPost._id,
          title: postData.title,
          content: postData.content,
          imageUrl: imageUrl
        }
      };
    }

    return fetch('http://localhost:8080/graphql', {
      method: 'POST',
      body: JSON.stringify(graphqlQuery),
      headers: {
        Authorization: 'Bearer ' + this.props.token,
        'Content-Type': 'application/json'
      }
    });
  })
  .then(res => {
    return res.json();
  })
  .then(resData => {
    if (resData.errors && resData.errors[0].status === 422) {
      throw new Error(
        "Validation failed. Make sure the email address isn't used yet!"
      );
    }
    if (resData.errors) {
```

```
        throw new Error('User login failed!');
      }
      let resDataField = 'createPost';
      if (this.state.editPost) {
        resDataField = 'updatePost';
      }
      const post = {
        _id: resData.data[resDataField]._id,
        title: resData.data[resDataField].title,
        content: resData.data[resDataField].content,
        creator: resData.data[resDataField].creator,
        createdAt: resData.data[resDataField].createdAt,
        imagePath: resData.data[resDataField].imageUrl
      };
/**i need to make sure
 * that i increase the amount of total posts in our post
 * in our state in react here where we manage how many posts we have on that page,
 *
 *    totalPosts: resData.data.posts.totalPosts
 *
 * i set total posts to a different amount of posts
 * when i fetch all posts
 * where i get that total posts information.
 * when adding a new element,
 * i know that total posts will be the old total posts + 1
 *
 */
      this.setState(prevState => {
        let updatedPosts = [...prevState.posts];
        let updatedTotalPosts = prevState.totalPosts;
        if (prevState.editPost) {
          const postIndex = prevState.posts.findIndex(
            p => p._id === prevState.editPost._id
          );
          updatedPosts[postIndex] = post;
        } else {
          updatedTotalPosts++;
          if (prevState.posts.length >= 2) {
            updatedPosts.pop();
          }
          updatedPosts.unshift(post);
        }
        return {
          posts: updatedPosts,
          isEditing: false,
          editPost: null,
          editLoading: false,
          totalPosts: updatedTotalPosts
        };
      });
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isEditing: false,
        editPost: null,
```

```
        editLoading: false,
        error: err
      });
    });
};

statusInputChangeHandler = (input, value) => {
  this.setState({ status: value });
};

deletePostHandler = postId => {
  this.setState({ postsLoading: true });
  const graphqlQuery = {
    query: `
      mutation {
        deletePost(id: "${postId}")
      }
    `
  };
  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    headers: {
      Authorization: 'Bearer ' + this.props.token,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Deleting the post failed!');
      }
      console.log(resData);
      this.loadPosts();
      // this.setState(prevState => {
      //   const updatedPosts = prevState.posts.filter(p => p._id !== postId);
      //   return { posts: updatedPosts, postsLoading: false };
      // });
    })
    .catch(err => {
      console.log(err);
      this.setState({ postsLoading: false });
    });
};

errorHandler = () => {
  this.setState({ error: null });
};

catchError = error => {
  this.setState({ error: error });
};

render() {
```

```
    return (
      <Fragment>
        <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
        <FeedEdit
          editing={this.state.isEditing}
          selectedPost={this.state.editPost}
          loading={this.state.editLoading}
          onCancelEdit={this.cancelEditHandler}
          onFinishEdit={this.finishEditHandler}
        />
        <section className="feed__status">
          <form onSubmit={this.statusUpdateHandler}>
            <Input
              type="text"
              placeholder="Your status"
              control="input"
              onChange={this.statusInputChangeHandler}
              value={this.state.status}
            />
            <Button mode="flat" type="submit">
              Update
            </Button>
          </form>
        </section>
        <section className="feed__control">
          <Button mode="raised" design="accent" onClick={this.newPostHandler}>
            New Post
          </Button>
        </section>
        <section className="feed">
          {this.state.postsLoading && (
            <div style={{ textAlign: 'center', marginTop: '2rem' }}>
              <Loader />
            </div>
          )}
          {this.state.posts.length <= 0 && !this.state.postsLoading ? (
            <p style={{ textAlign: 'center' }}>No posts found.</p>
          ) : null}
          {!this.state.postsLoading && (
            <Paginator
              onPrevious={this.loadPosts.bind(this, 'previous')}
              onNext={this.loadPosts.bind(this, 'next')}
              lastPage={Math.ceil(this.state.totalPosts / 2)}
              currentPage={this.state.postPage}
            >
              {this.state.posts.map(post => (
                <Post
                  key={post._id}
                  id={post._id}
                  author={post.creator.name}
                  date={new Date(post.createdAt).toLocaleDateString('en-US')}
                  title={post.title}
                  image={post.imageUrl}
                  content={post.content}
                  onStartEdit={this.startEditPostHandler.bind(this, post._id)}
                  onDelete={this.deletePostHandler.bind(this, post._id)}
```

```
            />
          ))}
        </Paginator>
      )}
    </section>
  </Fragment>
  );
 }
}

export default Feed;
```

```
//./src/pages/Feed/SinglePost/SinglePost.js(f)

import React, { Component } from 'react';

import Image from '../../../components/Image/Image';
import './SinglePost.css';

class SinglePost extends Component {
  state = {
    title: '',
    author: '',
    date: '',
    image: '',
    content: ''
  };

  componentDidMount() {
    const postId = this.props.match.params.postId;
    const graphqlQuery = {
      /**i also have a query
       * where i can check the value
       * when i get a single post
       * so i will give it as a name by adding 'query'
       * and the name 'FetchSinglePost'
       */
      query: `query FetchSinglePost($postId: ID!) {
          post(id: $postId) {
            title
            content
            imageUrl
            creator {
              name
            }
            createdAt
          }
        }
      `,
      variables: {
        postId: postId
      }
    };
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        Authorization: 'Bearer ' + this.props.token,
```

```
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors) {
        throw new Error('Fetching post failed!');
      }
      this.setState({
        title: resData.data.post.title,
        author: resData.data.post.creator.name,
        image: 'http://localhost:8080/' + resData.data.post.imageUrl,
        date: new Date(resData.data.post.createdAt).toLocaleDateString('en-US'),
        content: resData.data.post.content
      });
    })
    .catch(err => {
      console.log(err);
    });
  }

  render() {
    return (
      <section className="single-post">
        <h1>{this.state.title}</h1>
        <h2>
          Created by {this.state.author} on {this.state.date}
        </h2>
        <div className="single-post__image">
          <Image contain imageUrl={this.state.image} />
        </div>
        <p>{this.state.content}</p>
      </section>
    );
  }
}

export default SinglePost;

```

```
//App.js(f)

import React, { Component, Fragment } from 'react';
import { Route, Switch, Redirect, withRouter } from 'react-router-dom';

import Layout from './components/Layout/Layout';
import Backdrop from './components/Backdrop/Backdrop';
import Toolbar from './components/Toolbar/Toolbar';
import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
import ErrorHandler from './components/ErrorHandler/ErrorHandler';
import FeedPage from './pages/Feed/Feed';
import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
import LoginPage from './pages/Auth/Login';
```

```
import SignupPage from './pages/Auth/Signup';
import './App.css';

class App extends Component {
  state = {
    showBackdrop: false,
    showMobileNav: false,
    isAuth: false,
    token: null,
    userId: null,
    authLoading: false,
    error: null
  };

  componentDidMount() {
    const token = localStorage.getItem('token');
    const expiryDate = localStorage.getItem('expiryDate');
    if (!token || !expiryDate) {
      return;
    }
    if (new Date(expiryDate) <= new Date()) {
      this.logoutHandler();
      return;
    }
    const userId = localStorage.getItem('userId');
    const remainingMilliseconds =
      new Date(expiryDate).getTime() - new Date().getTime();
    this.setState({ isAuth: true, token: token, userId: userId });
    this.setAutoLogout(remainingMilliseconds);
  }

  mobileNavHandler = isOpen => {
    this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
  };

  backdropClickHandler = () => {
    this.setState({ showBackdrop: false, showMobileNav: false, error: null });
  };

  logoutHandler = () => {
    this.setState({ isAuth: false, token: null });
    localStorage.removeItem('token');
    localStorage.removeItem('expiryDate');
    localStorage.removeItem('userId');
  };

  loginHandler = (event, authData) => {
    event.preventDefault();
    const graphqlQuery = {
      query: `
        query UserLogin($email: String!, $password: String!) {
          login(email: $email, password: $password) {
            token
            userId
          }
        }
```

```
      `,
      variables: {
        email: authData.email,
        password: authData.password
      }
    };
    this.setState({ authLoading: true });
    fetch('http://localhost:8080/graphql', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(graphqlQuery)
    })
      .then(res => {
        return res.json();
      })
      .then(resData => {
        if (resData.errors && resData.errors[0].status === 422) {
          throw new Error(
            "Validation failed. Make sure the email address isn't used yet!"
          );
        }
        if (resData.errors) {
          throw new Error('User login failed!');
        }
        console.log(resData);
        this.setState({
          isAuth: true,
          token: resData.data.login.token,
          authLoading: false,
          userId: resData.data.login.userId
        });
        localStorage.setItem('token', resData.data.login.token);
        localStorage.setItem('userId', resData.data.login.userId);
        const remainingMilliseconds = 60 * 60 * 1000;
        const expiryDate = new Date(
          new Date().getTime() + remainingMilliseconds
        );
        localStorage.setItem('expiryDate', expiryDate.toISOString());
        this.setAutoLogout(remainingMilliseconds);
      })
      .catch(err => {
        console.log(err);
        this.setState({
          isAuth: false,
          authLoading: false,
          error: err
        });
      });
  };

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

```
    query: `
      mutation CreateNewUser($email: String!, $name: String!, $password: String!) {
        createUser(userInput: {email: $email, name: $name, password: $password}) {
          _id
          email
        }
      }
    `,
    variables: {
      email: authData.signupForm.email.value,
      name: authData.signupForm.name.value,
      password: authData.signupForm.password.value
    }
  };
  fetch('http://localhost:8080/graphql', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(graphqlQuery)
  })
    .then(res => {
      return res.json();
    })
    .then(resData => {
      if (resData.errors && resData.errors[0].status === 422) {
        throw new Error(
          "Validation failed. Make sure the email address isn't used yet!"
        );
      }
      if (resData.errors) {
        throw new Error('User creation failed!');
      }
      console.log(resData);
      this.setState({ isAuth: false, authLoading: false });
      this.props.history.replace('/');
    })
    .catch(err => {
      console.log(err);
      this.setState({
        isAuth: false,
        authLoading: false,
        error: err
      });
    });
};

setAutoLogout = milliseconds => {
  setTimeout(() => {
    this.logoutHandler();
  }, milliseconds);
};

errorHandler = () => {
  this.setState({ error: null });
};
```

```
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.isAuth) {
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 && (
```

```
          <Backdrop onClick={this.backdropClickHandler} />
        )}
        <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
        <Layout
          header={
            <Toolbar>
              <MainNavigation
                onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
                onLogout={this.logoutHandler}
                isAuth={this.state.isAuth}
              />
            </Toolbar>
          }
          mobileNav={
            <MobileNavigation
              open={this.state.showMobileNav}
              mobile
              onChooseItem={this.mobileNavHandler.bind(this, false)}
              onLogout={this.logoutHandler}
              isAuth={this.state.isAuth}
            />
          }
        />
        {routes}
      </Fragment>
    );
  }
}

export default withRouter(App);

```

* Chapter 441: Wrap Up