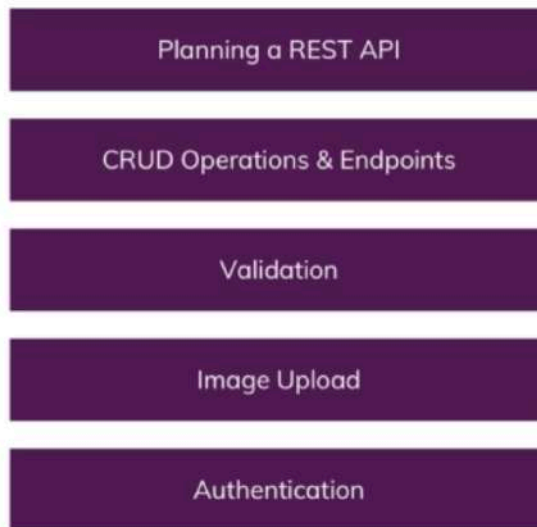


25. Working With REST APIs - The Practical Application

* Chapter 365: Module Introduction



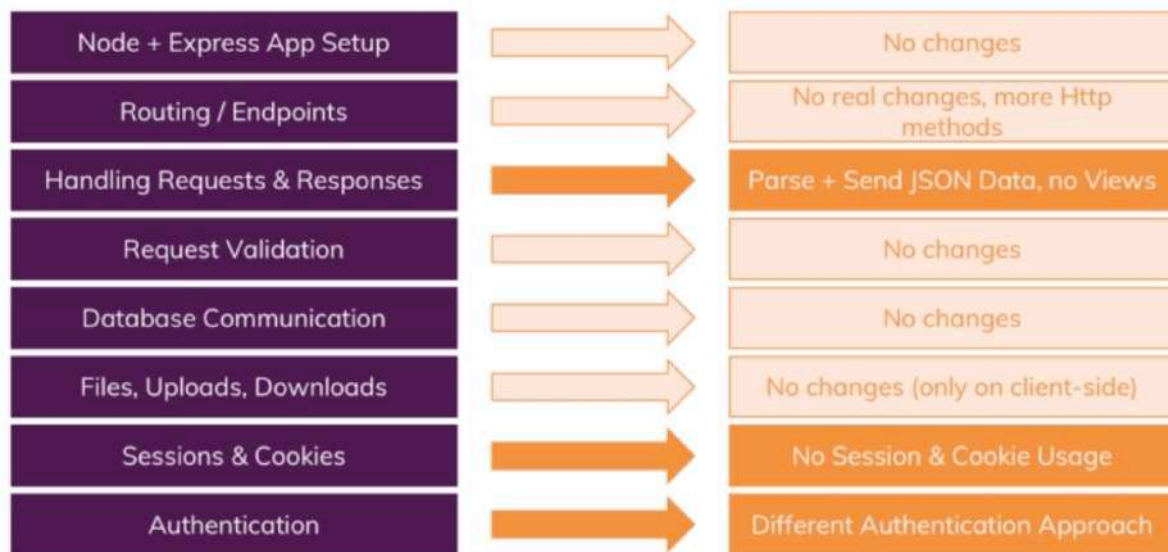
What's In This Module?



365

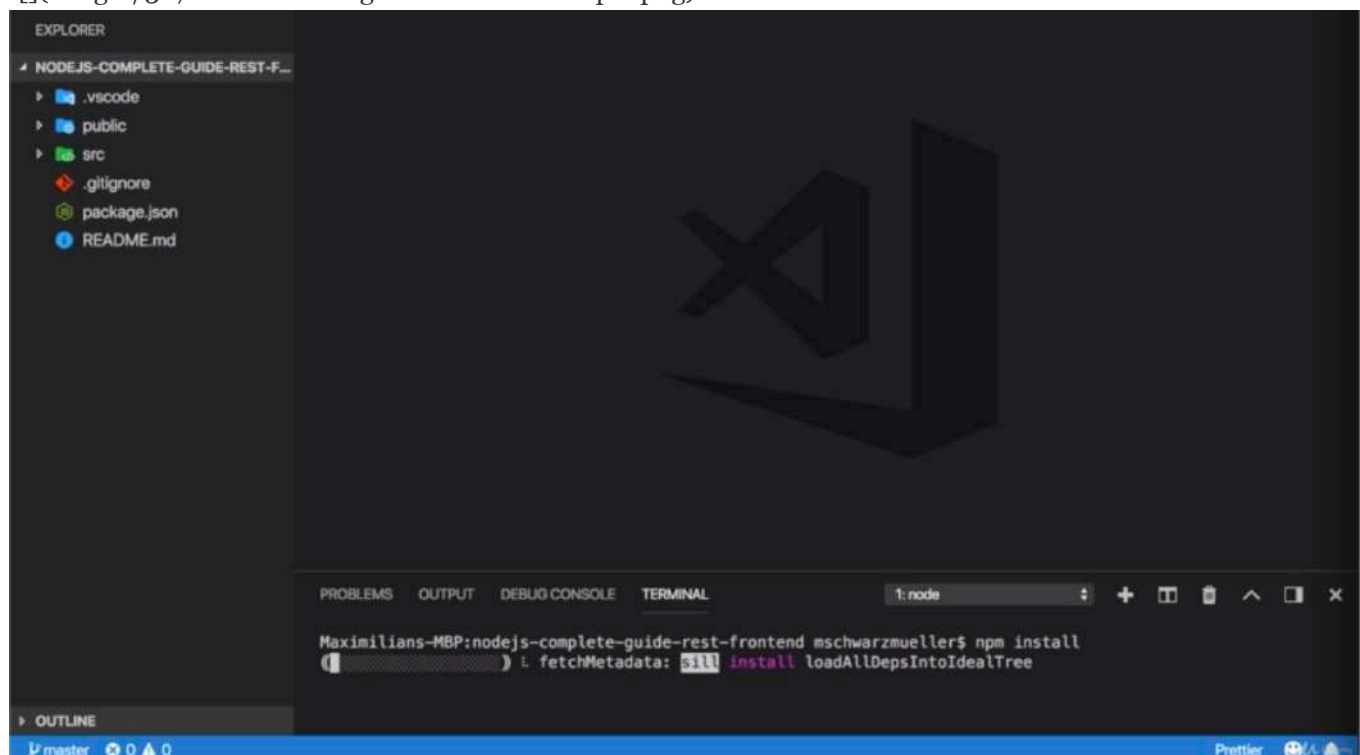
* Chapter 366: REST APIs & The Rest Of The Course

REST & The Rest of the Course

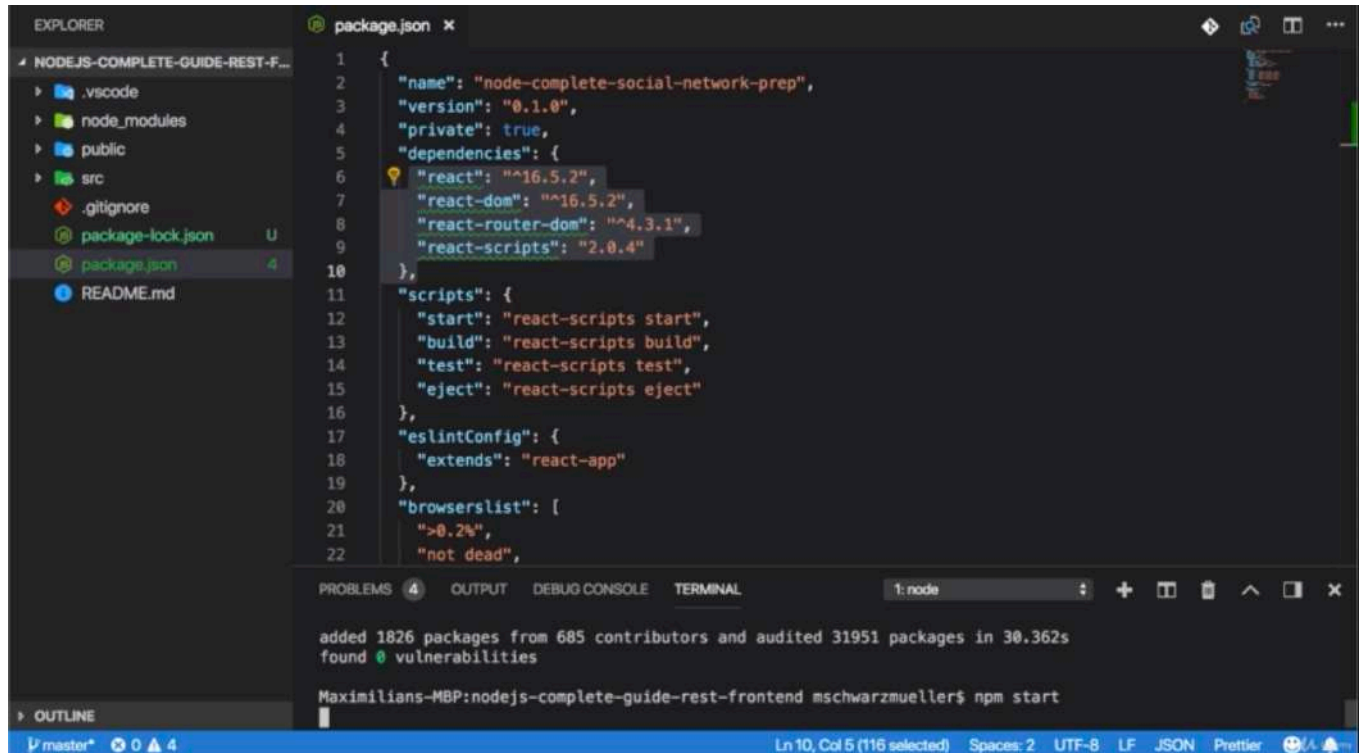


- for Sessions and Cookies, we have changes because we will not use sessions and cookies anymore with REST API. the reason for that is you learned these RESTful principle or these REST API principles and one of them was that each request is treated separately, it is looked at independently from previous requests, so we have no connection between the client and server, we have no shared connection history to be precise and therefore we manage no sessions on the server because REST API doesn't care about the clients or whether that client connected to the API before. therefore Authentication will also have to change.

* Chapter 367: Understanding The Frontend Setup



- you will start with attached project in Udemy. folder name is '01-frontend-fetching-list-of-posts' so 'npm install' to install node_modules.



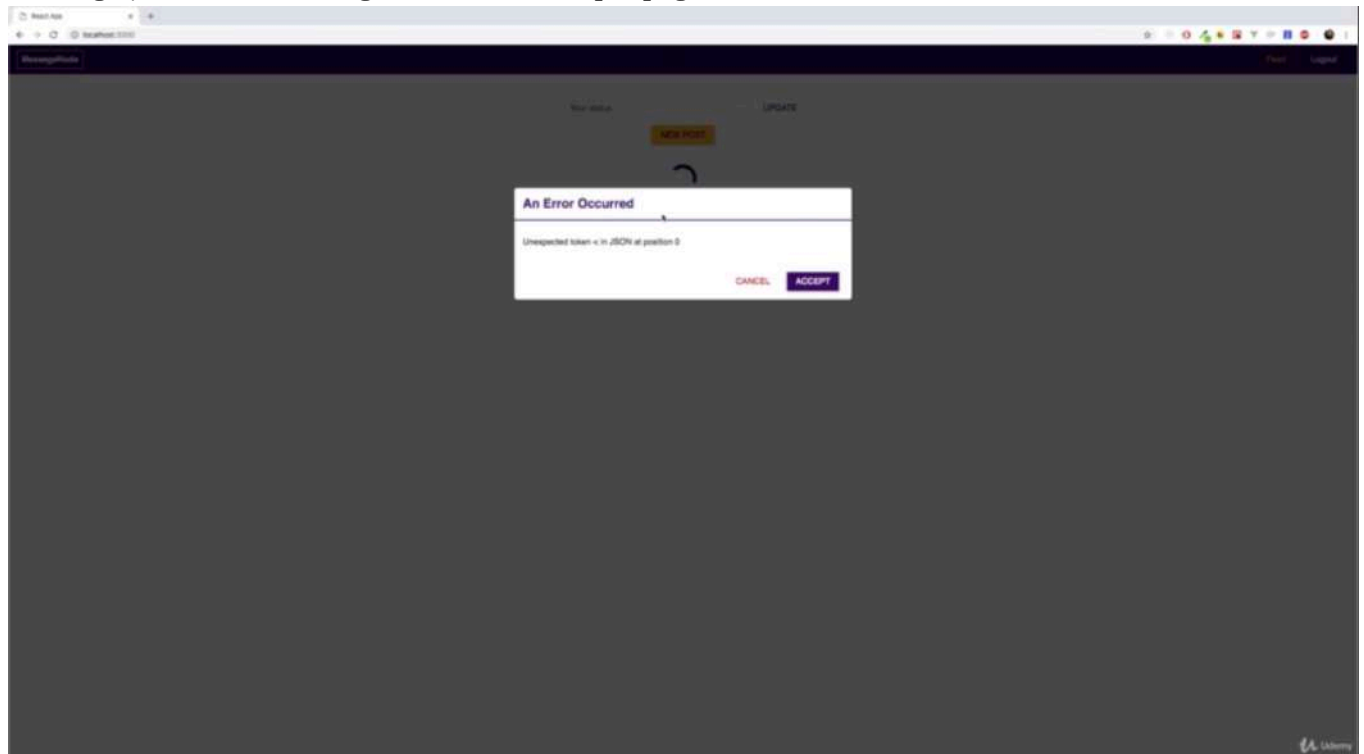
The screenshot shows a VS Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'NODEJS-COMPLETE-GUIDE-REST-F...' with files like '.vscode', 'node_modules', 'public', 'src', '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The 'package.json' file is open in the editor, showing the following content:

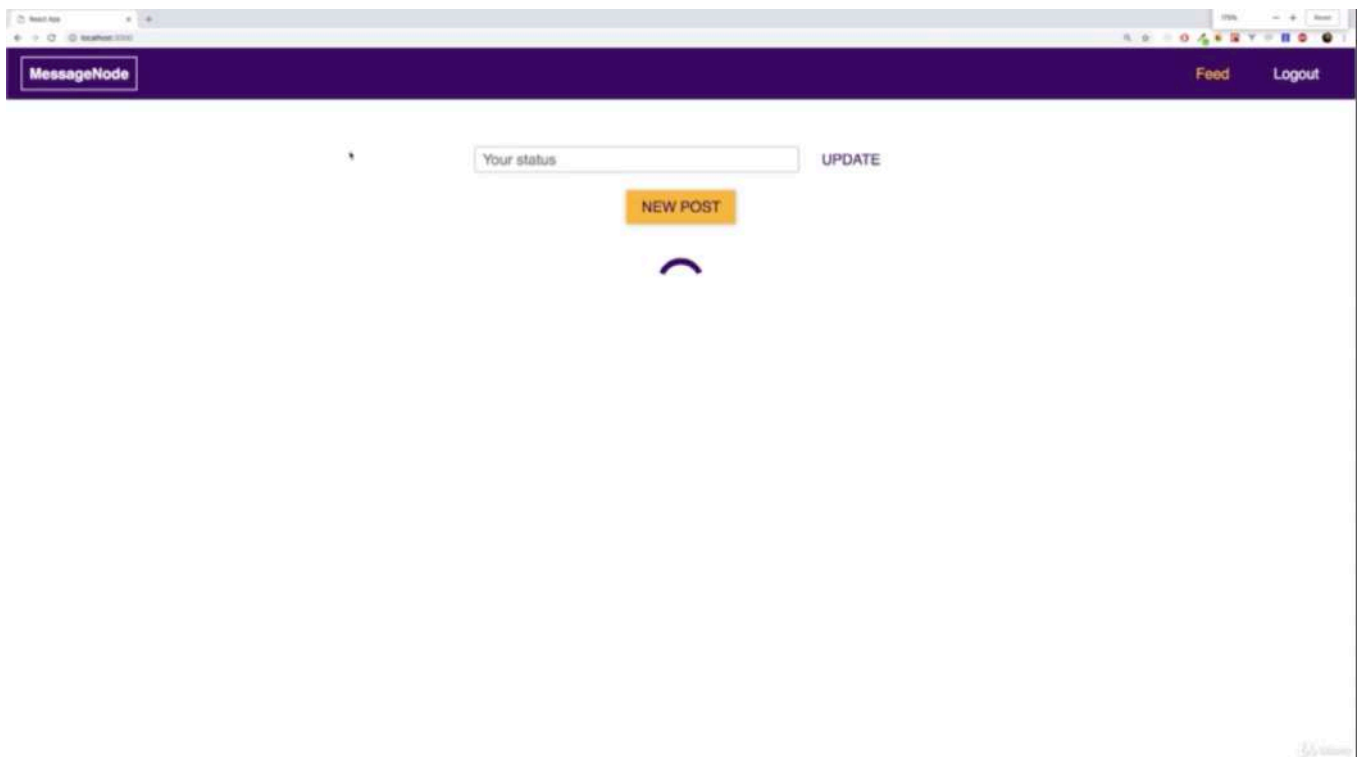
```
1 {
2   "name": "node-complete-social-network-prep",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "react": "^16.5.2",
7     "react-dom": "^16.5.2",
8     "react-router-dom": "^4.3.1",
9     "react-scripts": "2.0.4"
10  },
11  "scripts": {
12    "start": "react-scripts start",
13    "build": "react-scripts build",
14    "test": "react-scripts test",
15    "eject": "react-scripts eject"
16  },
17  "eslintConfig": {
18    "extends": "react-app"
19  },
20  "browserslist": [
21    ">0.2%",
22    "not dead",
```

The terminal at the bottom shows the output of the 'npm start' command:

```
Maximilians-MBP:nodejs-complete-guide-rest-frontend mschwarzmueller$ npm start
added 1826 packages from 685 contributors and audited 31951 packages in 30.362s
found 0 vulnerabilities
```

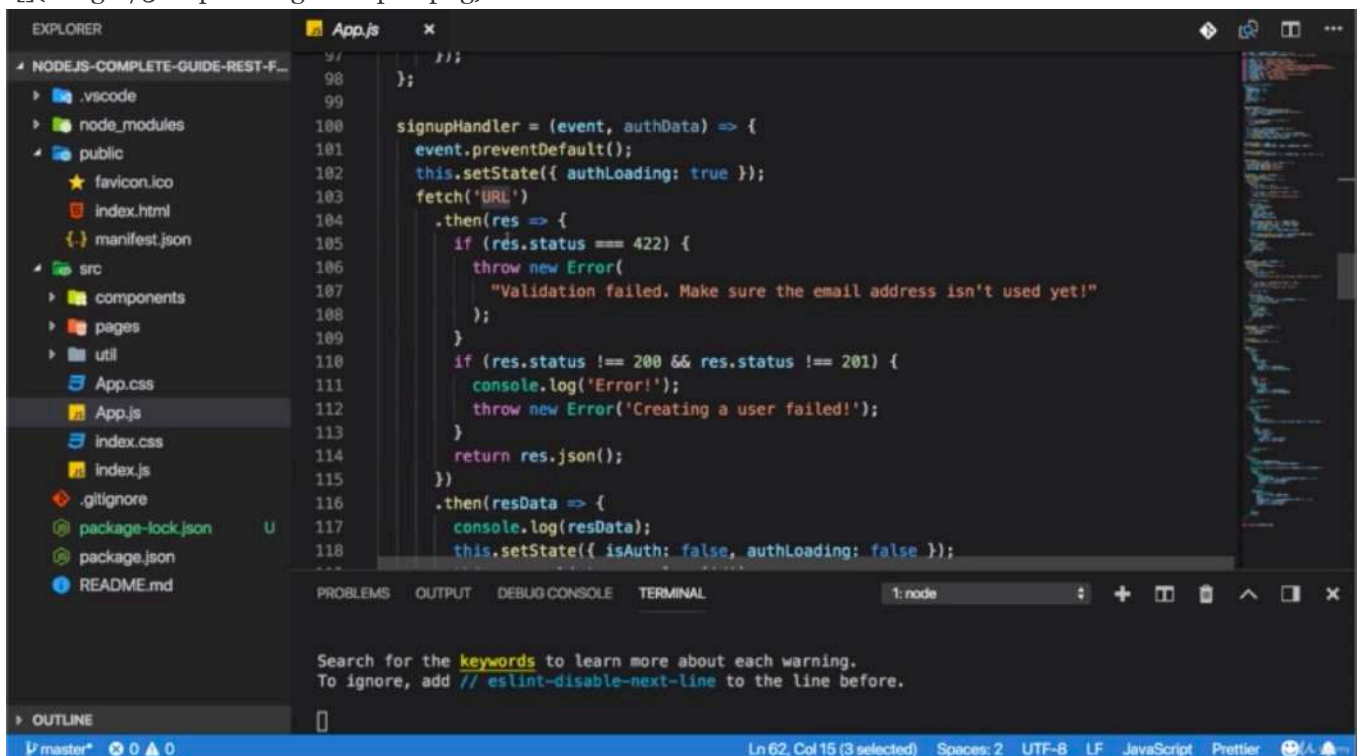
- once you ran all of that, you can run 'npm start' to start a development server which is now actually a node.js server. but it's not related to the node server we will build. it's not related to our backend. this is only a dummy development server which serves the build version of our frontend application. so it serves a simple HTML file in public folder.





- this interface is fully rendered through React.js.
- right now nothing is working because i got no backend attached. this is also the reason why we have an error message right at the start but we will get rid of that throughout this module.
- this is the frontend and this is now the frontend i wanna connect to my backend.

* Chapter 368: Planning The API



- the app.js file is our entry file where our react application starts or where it starts rendering the first screens.

* Chapter 369: Fetching Lists Of Posts

* backend = (b)

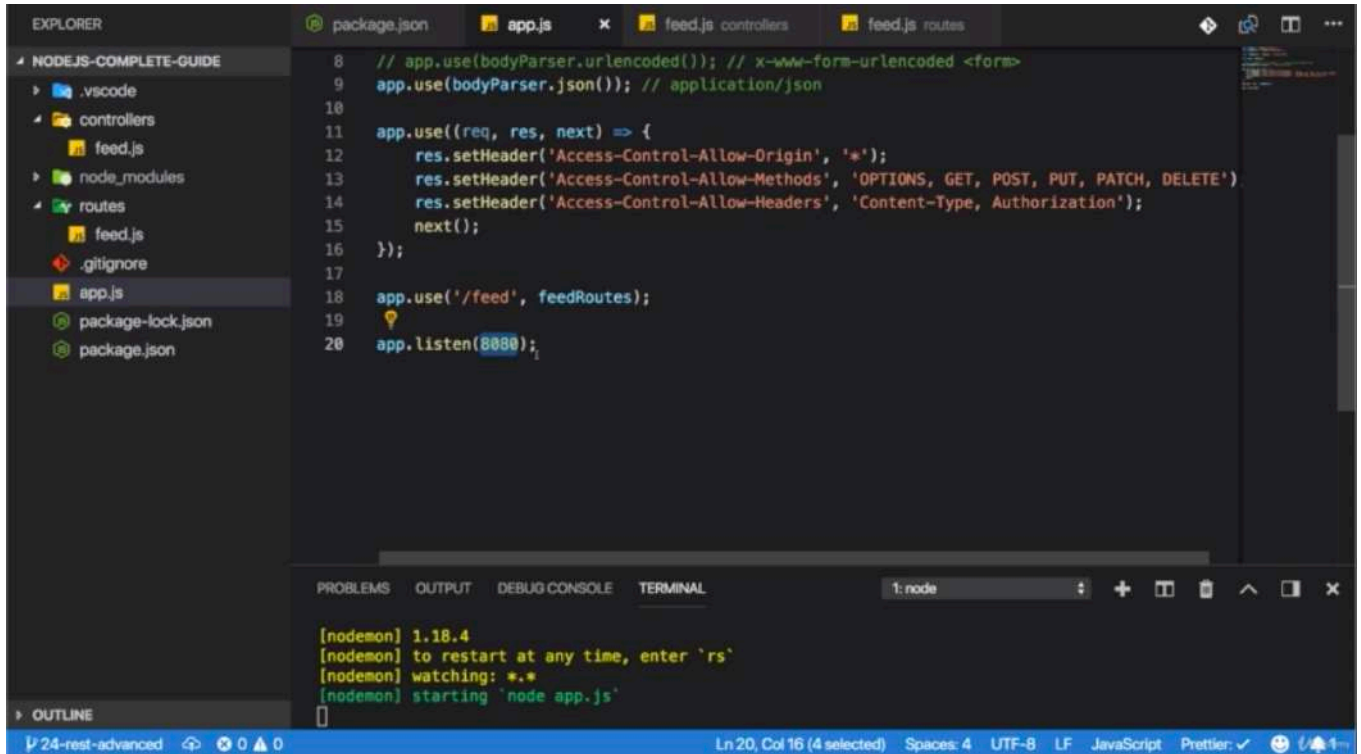
* frontend = (f)

1. update

- ./controllers/feed.js(b)

- ./images/duck.jpg

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



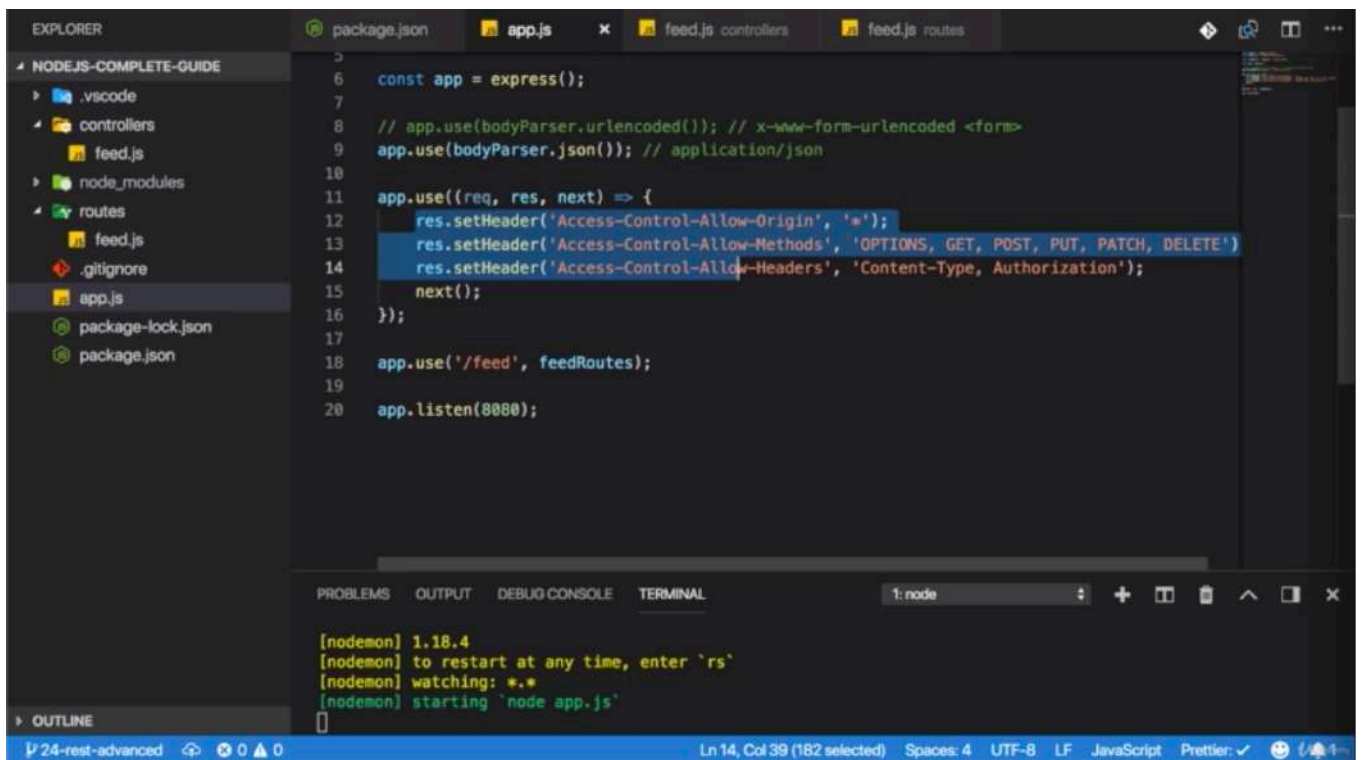
The screenshot shows a VS Code editor with the following components:

- EXPLORER:** A file tree on the left showing the project structure: `NODEJS-COMPLETE-GUIDE`, `.vscode`, `controllers` (containing `feed.js`), `node_modules`, `routes` (containing `feed.js`), `.gitignore`, `app.js`, `package-lock.json`, and `package.json`.
- EDITOR:** The `app.js` file is open, showing the following code:

```
8 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
9 app.use(bodyParser.json()); // application/json
10
11 app.use((req, res, next) => {
12   res.setHeader('Access-Control-Allow-Origin', '*');
13   res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET, POST, PUT, PATCH, DELETE');
14   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
15   next();
16 });
17
18 app.use('/feed', feedRoutes);
19
20 app.listen(8080);
```
- TERMINAL:** A terminal window at the bottom shows the output of a nodemon command:

```
[nodemon] 1.18.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
```

- i'm using different ports. i'm starting the application server on port 8080, my frontend application automatically takes port 3000 and this simulates that these 2 ends of my application are served by different servers which is a common scenario since frontend-only applications like React can be served on so-called static hosts which are optimized for applications that only consist of HTML, javascript, CSS. and hence you might have 2 different servers even if you created both the backend and frontend.



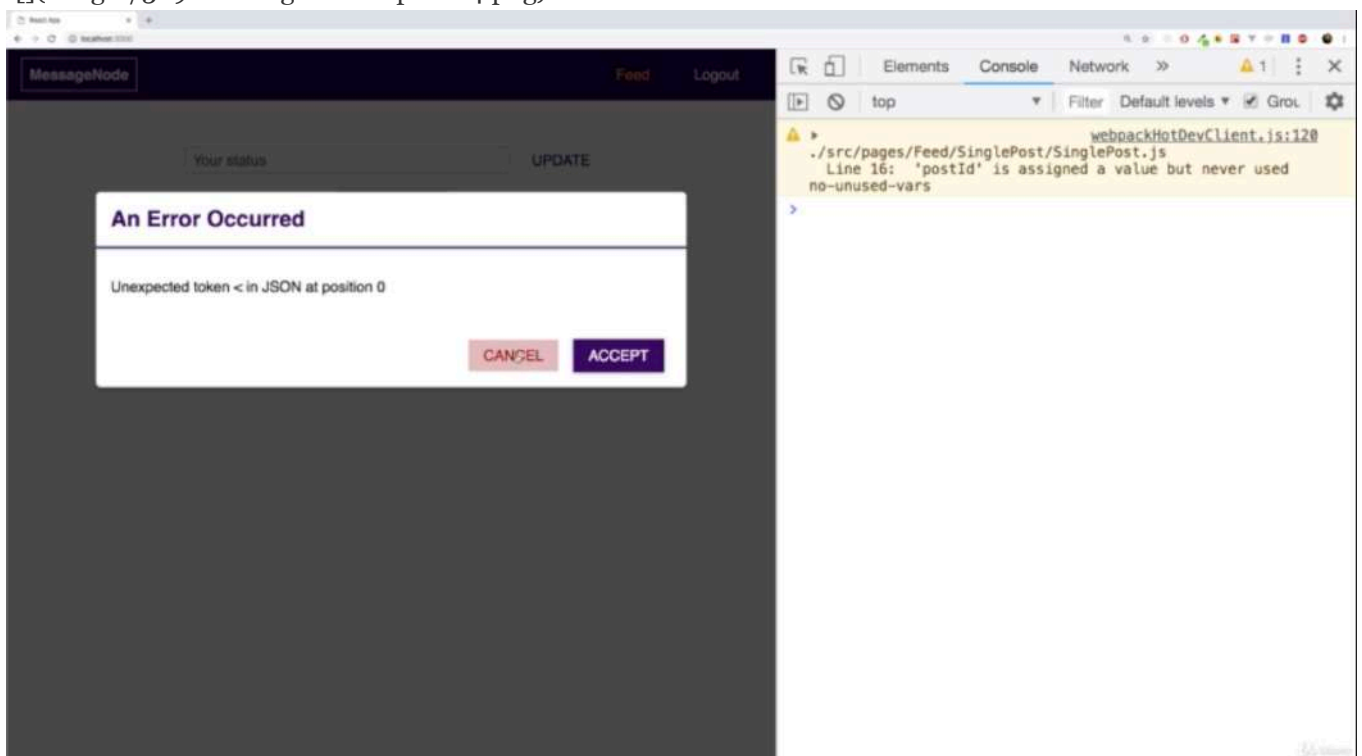
The screenshot shows a VS Code editor with the following components:

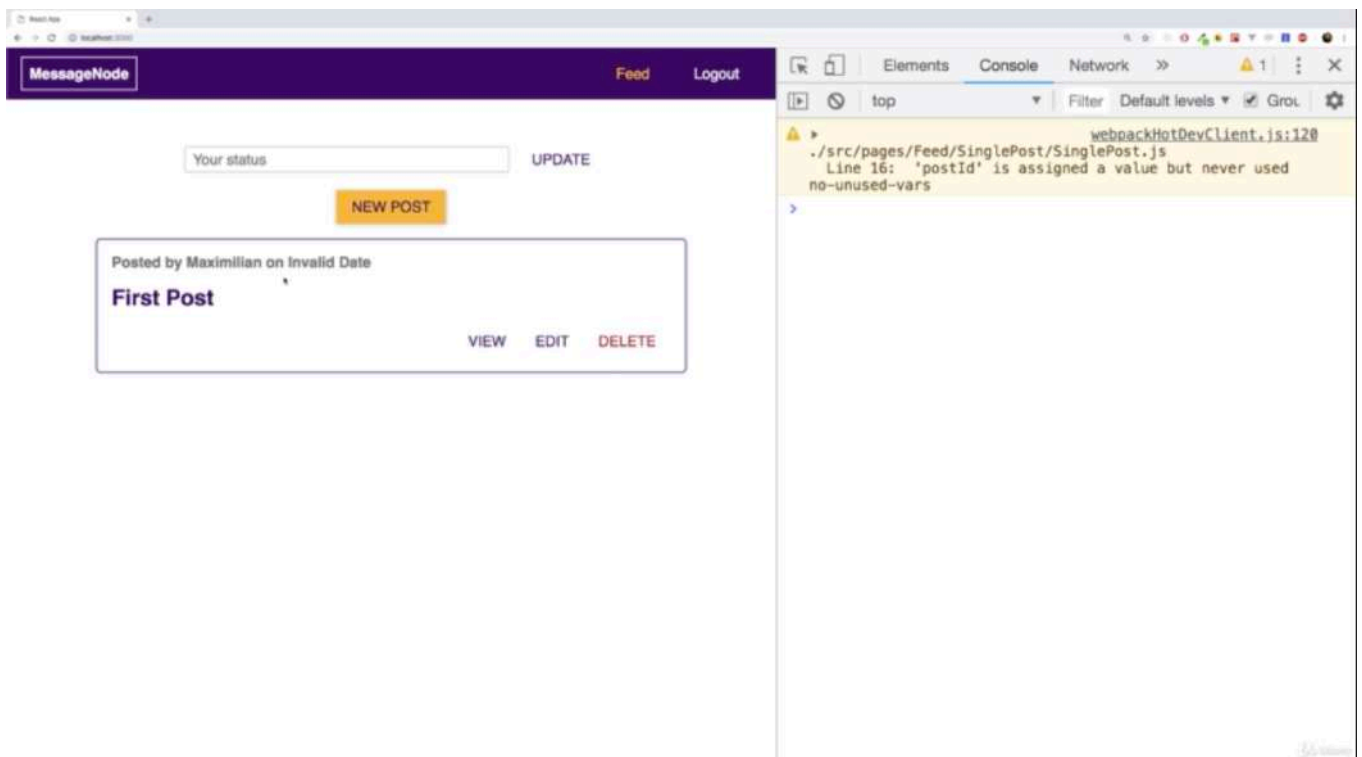
- EXPLORER:** A file tree on the left showing the project structure: `NODEJS-COMPLETE-GUIDE`, `.vscode`, `controllers`, `feed.js`, `node_modules`, `routes`, `feed.js`, `.gitignore`, `app.js`, `package-lock.json`, and `package.json`.
- EDITOR:** The `app.js` file is open, showing the following code:

```
5
6 const app = express();
7
8 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
9 app.use(bodyParser.json()); // application/json
10
11 app.use((req, res, next) => {
12   res.setHeader('Access-Control-Allow-Origin', '*');
13   res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET, POST, PUT, PATCH, DELETE');
14   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
15   next();
16 });
17
18 app.use('/feed', feedRoutes);
19
20 app.listen(8080);
```
- TERMINAL:** The terminal at the bottom shows the output of running the application:

```
[nodemon] 1.18.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
```

- we have different ports and therefore we have different domains and therefore we need our course headers otherwise nothing would work.





```

1  //./controllers/feed.js(b)
2
3  exports.getPosts = (req, res, next) => {
4    res.status(200).json({
5      posts: [
6        {
7          _id: '1',
8          title: 'First Post',
9          content: 'This is the first post!',
10         imageUrl: 'images/duck.jpg',
11         creator: {
12           name: 'Maximilian'
13         },
14         createdAt: new Date()
15       }
16     ]
17   });
18 };
19
20 exports.createPost = (req, res, next) => {
21   const title = req.body.title;
22   const content = req.body.content;
23   console.log(title, content);
24   //Create post in db
25   res.status(201).json({
26     message: 'Post created successfully!',
27     post: { id: new Date().toISOString(), title: title, content: content }
28   });
29 };

```

```

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

```



```

6 import Button from '../../components/Button/Button';
7 import FeedEdit from '../../components/Feed/FeedEdit/FeedEdit';
8 import Input from '../../components/Form/Input/Input';
9 import Paginator from '../../components/Paginator/Paginator';
10 import Loader from '../../components/Loader/Loader';
11 import ErrorHandler from '../../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('URL')
28       .then(res => {
29         if (res.status !== 200) {
30           throw new Error('Failed to fetch user status.');

```



```

62     }
63     return res.json();
64   })
65   .then(resData => {
66     this.setState({
67       posts: resData.posts,
68       totalPosts: resData.totalItems,
69       postsLoading: false
70     });
71   })
72   .catch(this.catchError);
73 };
74
75 statusUpdateHandler = event => {
76   event.preventDefault();
77   fetch('URL')
78     .then(res => {
79       if (res.status !== 200 && res.status !== 201) {
80         throw new Error("Can't update status!");
81       }
82       return res.json();
83     })
84     .then(resData => {
85       console.log(resData);
86     })
87     .catch(this.catchError);
88 };
89
90 newPostHandler = () => {
91   this.setState({ isEditing: true });
92 };
93
94 startEditPostHandler = postId => {
95   this.setState(prevState => {
96     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
97
98     return {
99       isEditing: true,
100       editPost: loadedPost
101     };
102   });
103 };
104
105 cancelEditHandler = () => {
106   this.setState({ isEditing: false, editPost: null });
107 };
108
109 finishEditHandler = postData => {
110   this.setState({
111     editLoading: true
112   });
113   // Set up data (with image!)
114   let url = 'URL';
115   if (this.state.editPost) {
116     url = 'URL';
117   }

```

```

118
119     fetch(url)
120     .then(res => {
121         if (res.status !== 200 && res.status !== 201) {
122             throw new Error('Creating or editing a post failed!');
123         }
124         return res.json();
125     })
126     .then(resData => {
127         const post = {
128             _id: resData.post._id,
129             title: resData.post.title,
130             content: resData.post.content,
131             creator: resData.post.creator,
132             createdAt: resData.post.createdAt
133         };
134         this.setState(prevState => {
135             let updatedPosts = [...prevState.posts];
136             if (prevState.editPost) {
137                 const postIndex = prevState.posts.findIndex(
138                     p => p._id === prevState.editPost._id
139                 );
140                 updatedPosts[postIndex] = post;
141             } else if (prevState.posts.length < 2) {
142                 updatedPosts = prevState.posts.concat(post);
143             }
144             return {
145                 posts: updatedPosts,
146                 isEditing: false,
147                 editPost: null,
148                 editLoading: false
149             };
150         });
151     })
152     .catch(err => {
153         console.log(err);
154         this.setState({
155             isEditing: false,
156             editPost: null,
157             editLoading: false,
158             error: err
159         });
160     });
161 };
162
163 statusInputChangeHandler = (input, value) => {
164     this.setState({ status: value });
165 };
166
167 deletePostHandler = postId => {
168     this.setState({ postsLoading: true });
169     fetch('URL')
170     .then(res => {
171         if (res.status !== 200 && res.status !== 201) {
172             throw new Error('Deleting a post failed!');
173         }

```

```

174     return res.json();
175   })
176   .then(resData => {
177     console.log(resData);
178     this.setState(prevState => {
179       const updatedPosts = prevState.posts.filter(p => p._id !== postId);
180       return { posts: updatedPosts, postsLoading: false };
181     });
182   })
183   .catch(err => {
184     console.log(err);
185     this.setState({ postsLoading: false });
186   });
187 };
188
189 errorHandler = () => {
190   this.setState({ error: null });
191 };
192
193 catchError = error => {
194   this.setState({ error: error });
195 };
196
197 render() {
198   return (
199     <Fragment>
200       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
201       <FeedEdit
202         editing={this.state.isEditing}
203         selectedPost={this.state.editPost}
204         loading={this.state.editLoading}
205         onCancelEdit={this.cancelEditHandler}
206         onFinishEdit={this.finishEditHandler}
207       />
208       <section className="feed__status">
209         <form onSubmit={this.statusUpdateHandler}>
210           <Input
211             type="text"
212             placeholder="Your status"
213             control="input"
214             onChange={this.statusInputChangeHandler}
215             value={this.state.status}
216           />
217           <Button mode="flat" type="submit">
218             Update
219           </Button>
220         </form>
221       </section>
222       <section className="feed__control">
223         <Button mode="raised" design="accent" onClick={this.newPostHandler}>
224           New Post
225         </Button>
226       </section>
227       <section className="feed">
228         {this.state.postsLoading && (
229           <div style={{ textAlign: 'center', marginTop: '2rem' }}>

```

```

230     <Loader />
231   </div>
232   })
233   {this.state.posts.length <= 0 && !this.state.postsLoading ? (
234     <p style={{ textAlign: 'center' }}>No posts found.</p>
235   ) : null}
236   {!this.state.postsLoading && (
237     <Paginator
238       onPrevious={this.loadPosts.bind(this, 'previous')}
239       onNext={this.loadPosts.bind(this, 'next')}
240       lastPage={Math.ceil(this.state.totalPosts / 2)}
241       currentPage={this.state.postPage}
242     >
243       {this.state.posts.map(post => (
244         <Post
245           key={post._id}
246           id={post._id}
247           author={post.creator}
248           date={new Date(post.createdAt).toLocaleDateString('en-US')}
249           title={post.title}
250           image={post.imageUrl}
251           content={post.content}
252           onStartEdit={this.startEditPostHandler.bind(this, post._id)}
253           onDelete={this.deletePostHandler.bind(this, post._id)}
254         />
255       ))}
256     </Paginator>
257   )}
258   </section>
259 </Fragment>
260 );
261 }
262 }
263
264 export default Feed;
265

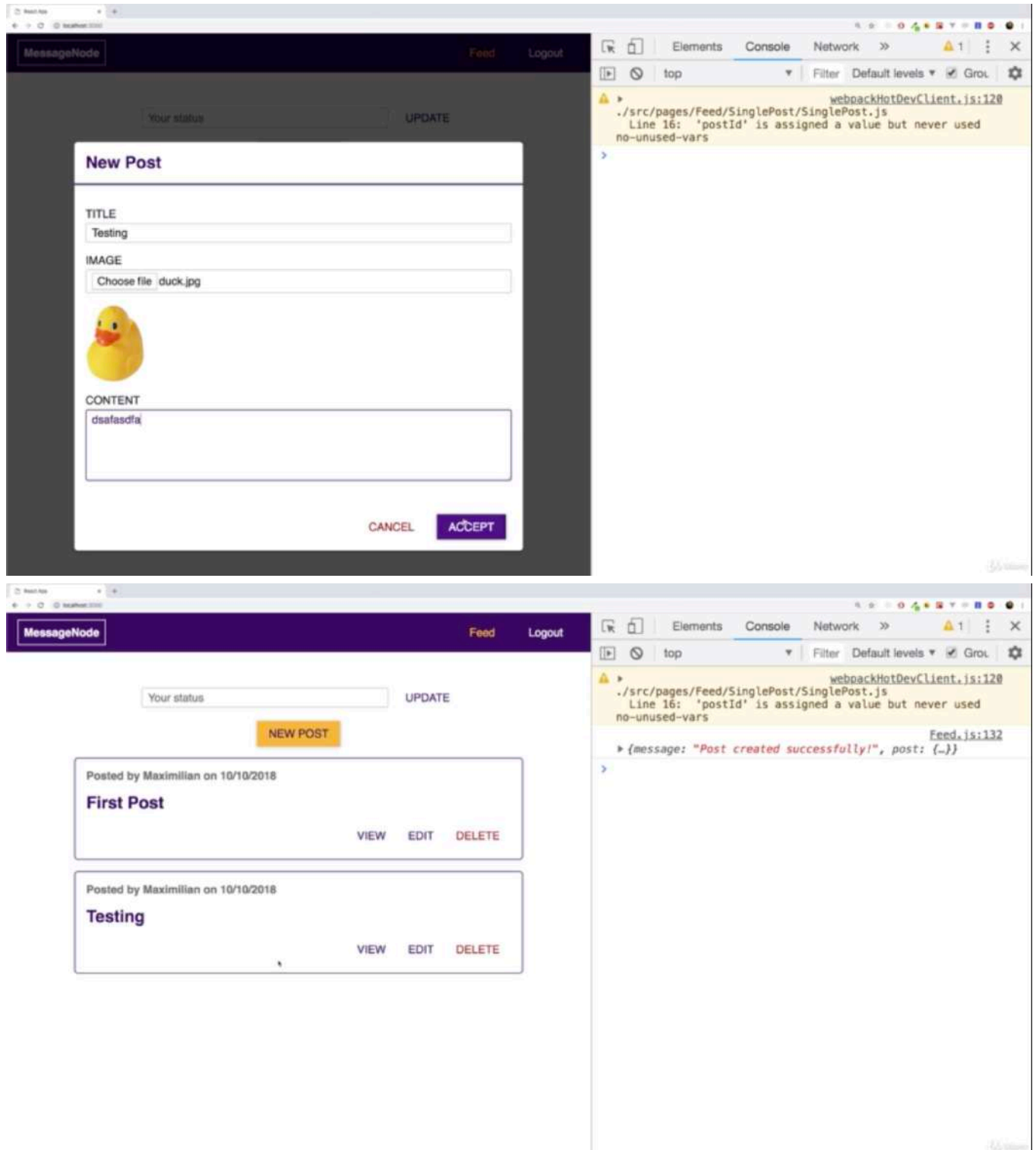
```

//images/duck.jpg



* Chapter 370: Adding A Create Post Endpoint

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



- 2 things are missing. server-side validation. we only got client-side validation but that can always be tricked because users can see your browser side javascript code. they could basically find ways around that. so it's not safe mechanism.

```
1 //./controllers/feed.js(b)
2
3 exports.getPosts = (req, res, next) => {
4   res.status(200).json({
5     posts: [
6       {
7         _id: '1',
```

```

8     title: 'First Post',
9     content: 'This is the first post!',
10    imageUrl: 'images/duck.jpg',
11    creator: {
12      name: 'Maximilian'
13    },
14    createdAt: new Date()
15  }
16 ]
17 });
18 };
19
20 exports.createPost = (req, res, next) => {
21   const title = req.body.title;
22   const content = req.body.content;
23   console.log(title, content);
24   //Create post in db
25   res.status(201).json({
26     message: 'Post created successfully!',
27     post: {
28       _id: new Date().toISOString(),
29       title: title,
30       content: content,
31       creator: { name: 'Maximilian' },
32       createdAt: new Date()
33     }
34   });
35 };

```

```

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

  /**this route i just talked about in the REST API
   * this should fetch all these posts.
   */
  fetch('http://localhost:8080/feed/posts')
    .then(res => {
      if (res.status !== 200) {
        throw new Error('Failed to fetch posts.');
```

```

      }
      return res.json();
    })
    .then(resData => {
      this.setState({
        posts: resData.posts,
        totalPosts: resData.totalItems,
        postsLoading: false
      });
    })
    .catch(this.catchError);
};

statusUpdateHandler = event => {
  event.preventDefault();
  fetch('URL')
    .then(res => {
      if (res.status !== 200 && res.status !== 201) {
        throw new Error("Can't update status!");
      }
      return res.json();
    })
    .then(resData => {
```

```

85         console.log(resData);
86     })
87     .catch(this.catchError);
88 };
89
90 newPostHandler = () => {
91     this.setState({ isEditing: true });
92 };
93
94 startEditPostHandler = postId => {
95     this.setState(prevState => {
96         const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
97
98         return {
99             isEditing: true,
100             editPost: loadedPost
101         };
102     });
103 };
104
105 cancelEditHandler = () => {
106     this.setState({ isEditing: false, editPost: null });
107 };
108
109 finishEditHandler = postData => {
110     this.setState({
111         editLoading: true
112     });
113     // Set up data (with image!)
114     let url = 'http://localhost:8080/feed/post';
115     let method = 'POST'
116     if (this.state.editPost) {
117         url = 'URL';
118     }
119
120     fetch(url, {
121         method: method,
122         headers: {
123             'Content-Type': 'application/json'
124         },
125         body: JSON.stringify({
126             title: postData.title,
127             content: postData.content
128         })
129     })
130     .then(res => {
131         if (res.status !== 200 && res.status !== 201) {
132             throw new Error('Creating or editing a post failed!');
133         }
134         return res.json();
135     })
136     .then(resData => {
137         console.log(resData)
138         const post = {
139             _id: resData.post._id,
140             title: resData.post.title,

```

```

141     content: resData.post.content,
142     creator: resData.post.creator,
143     createdAt: resData.post.createdAt
144   };
145   this.setState(prevState => {
146     let updatedPosts = [...prevState.posts];
147     if (prevState.editPost) {
148       const postIndex = prevState.posts.findIndex(
149         p => p._id === prevState.editPost._id
150       );
151       updatedPosts[postIndex] = post;
152     } else if (prevState.posts.length < 2) {
153       updatedPosts = prevState.posts.concat(post);
154     }
155     return {
156       posts: updatedPosts,
157       isEditing: false,
158       editPost: null,
159       editLoading: false
160     };
161   });
162 })
163 .catch(err => {
164   console.log(err);
165   this.setState({
166     isEditing: false,
167     editPost: null,
168     editLoading: false,
169     error: err
170   });
171 });
172 };
173
174 statusInputChangeHandler = (input, value) => {
175   this.setState({ status: value });
176 };
177
178 deletePostHandler = postId => {
179   this.setState({ postsLoading: true });
180   fetch('URL')
181     .then(res => {
182       if (res.status !== 200 && res.status !== 201) {
183         throw new Error('Deleting a post failed!');
184       }
185       return res.json();
186     })
187     .then(resData => {
188       console.log(resData);
189       this.setState(prevState => {
190         const updatedPosts = prevState.posts.filter(p => p._id !== postId);
191         return { posts: updatedPosts, postsLoading: false };
192       });
193     })
194     .catch(err => {
195       console.log(err);
196       this.setState({ postsLoading: false });

```

```

197     });
198   };
199
200   errorHandler = () => {
201     this.setState({ error: null });
202   };
203
204   catchError = error => {
205     this.setState({ error: error });
206   };
207
208   render() {
209     return (
210       <Fragment>
211         <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
212         <FeedEdit
213           editing={this.state.isEditing}
214           selectedPost={this.state.editPost}
215           loading={this.state.editLoading}
216           onCancelEdit={this.cancelEditHandler}
217           onFinishEdit={this.finishEditHandler}
218         />
219         <section className="feed__status">
220           <form onSubmit={this.statusUpdateHandler}>
221             <Input
222               type="text"
223               placeholder="Your status"
224               control="input"
225               onChange={this.statusInputChangeHandler}
226               value={this.state.status}
227             />
228             <Button mode="flat" type="submit">
229               Update
230             </Button>
231           </form>
232         </section>
233         <section className="feed__control">
234           <Button mode="raised" design="accent" onClick={this.newPostHandler}>
235             New Post
236           </Button>
237         </section>
238         <section className="feed">
239           {this.state.postsLoading && (
240             <div style={{ textAlign: 'center', marginTop: '2rem' }}>
241               <Loader />
242             </div>
243           )}
244           {this.state.posts.length <= 0 && !this.state.postsLoading ? (
245             <p style={{ textAlign: 'center' }}>No posts found.</p>
246           ) : null}
247           {!this.state.postsLoading && (
248             <Paginator
249               onPrevious={this.loadPosts.bind(this, 'previous')}
250               onNext={this.loadPosts.bind(this, 'next')}
251               lastPage={Math.ceil(this.state.totalPosts / 2)}
252               currentPage={this.state.postPage}

```

```

253 >
254 {this.state.posts.map(post => (
255   <Post
256     key={post._id}
257     id={post._id}
258     author={post.creator}
259     date={new Date(post.createdAt).toLocaleDateString('en-US')}
260     title={post.title}
261     image={post.imageUrl}
262     content={post.content}
263     onStartEdit={this.startEditPostHandler.bind(this, post._id)}
264     onDelete={this.deletePostHandler.bind(this, post._id)}
265   />
266 ))}
267 </Paginator>
268 )}
269 </section>
270 </Fragment>
271 );
272 }
273 }
274
275 export default Feed;
276

```

* Chapter 371: Adding Server Side Validation

1. update
 - ./routes/feed.js(b)
 - ./controllers/feed.js(b)

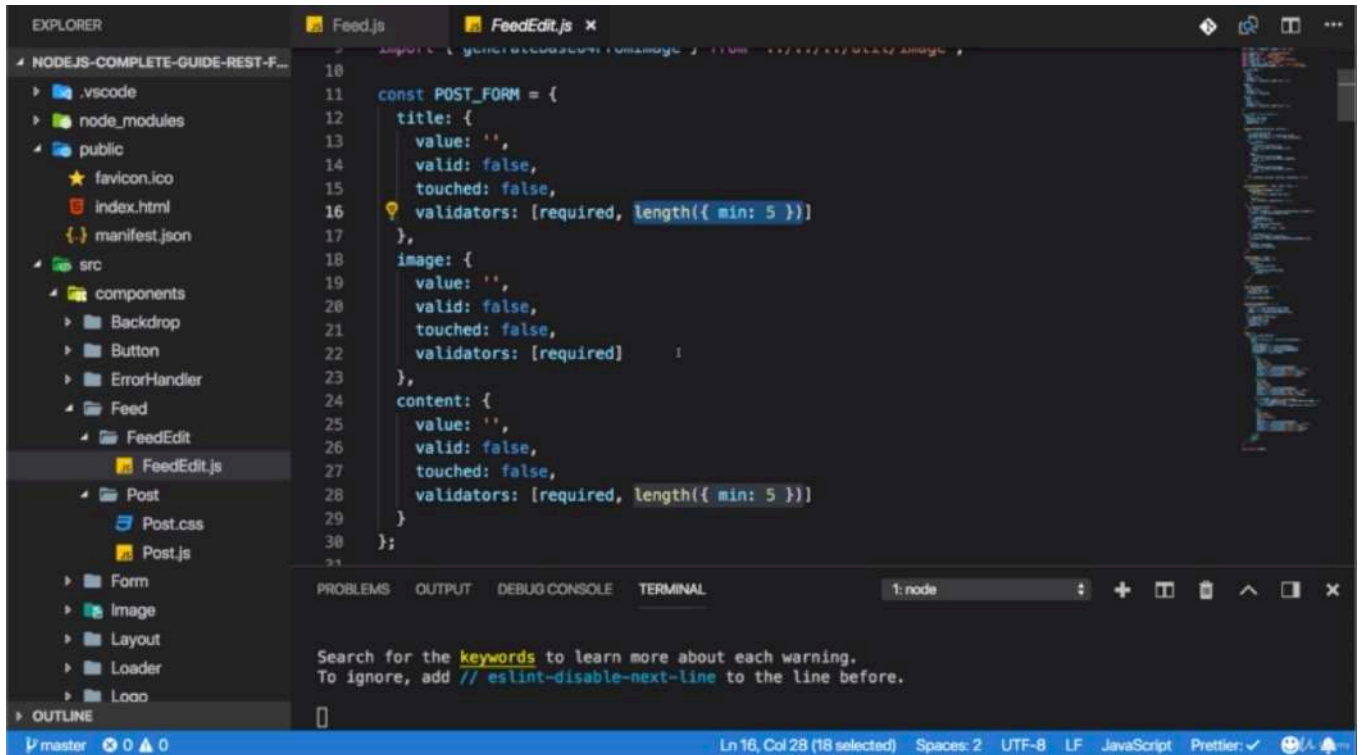

```

EXPLORER
  NODEJS-COMPLETE-GUIDE
    .vscode
    controllers
      feed.js
    Images
      duck.jpg
    node_modules
    routes
      feed.js
      .gitignore
      app.js
      package-lock.json
      package.json
  package.json
  app.js
  feed.js controllers
  feed.js routes

15   });
16   };
17
18   exports.createPost = (req, res, next) => {
19     const title = req.body.title;
20     const content = req.body.content;
21     // Create post in db
22     res.status(201).json({
23       message: 'Post created successfully!',
24       post: {
25         _id: new Date().toISOString(),
26         title: title,
27         content: content,
28         creator: { name: 'Maximilian' },
29         createdAt: new Date()
30       }
31     });
32   };
33
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: node
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$
Maximilians-MBP:nodejs-complete-guide mschwarzmueller$ npm install --save express-validator

```

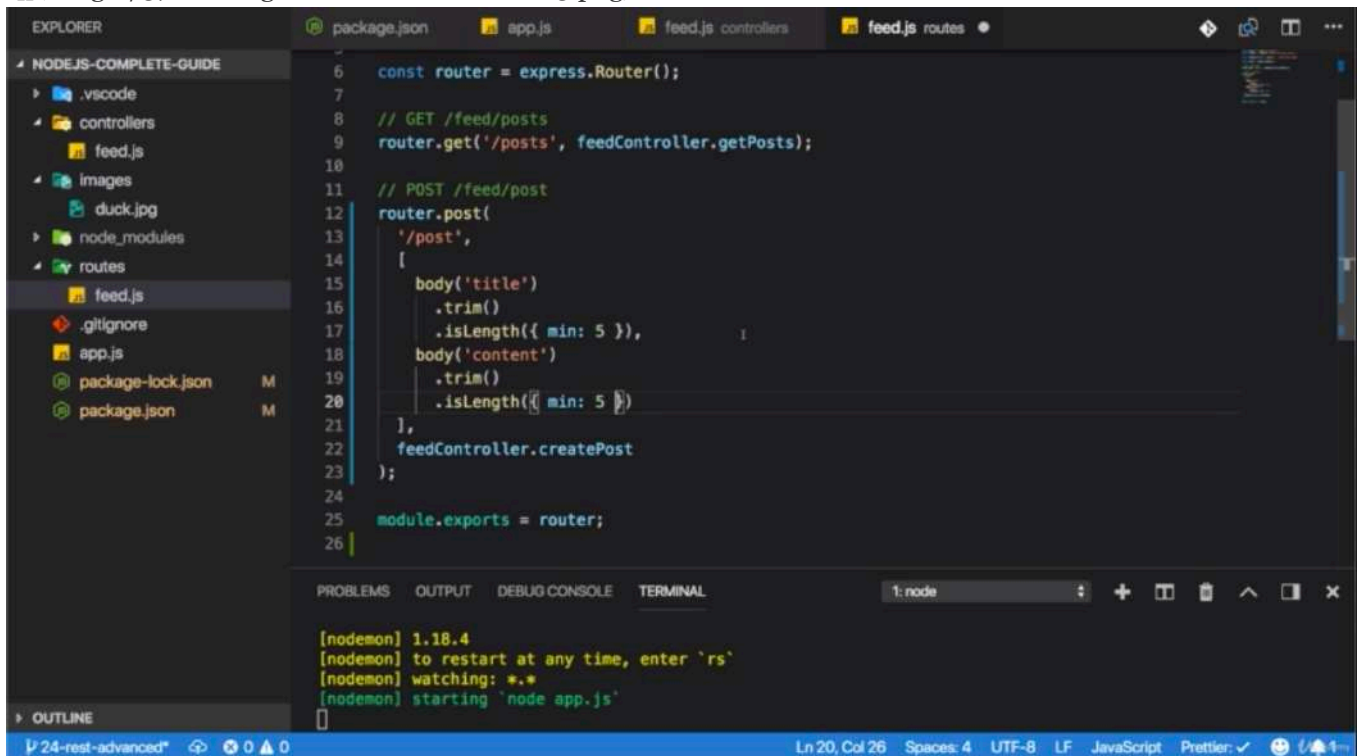
- 'npm install --save express-validator' is the package that helps us with implementing validation into our node express applications.



The screenshot shows a VS Code editor with the Explorer sidebar on the left. The Explorer shows a project structure with folders like .vscode, node_modules, public, and src. The src folder contains components, and the components folder contains FeedEdit.js. The main editor displays the content of FeedEdit.js, which defines a POST form with validation rules for title and content. The title and content fields are required and must be at least 5 characters long. The image field is required but has no length validation. The content field is required and must be at least 5 characters long. The status bar at the bottom shows the file is at line 16, column 28.

```
10
11 const POST_FORM = {
12   title: {
13     value: '',
14     valid: false,
15     touched: false,
16     validators: [required, length({ min: 5 })]
17   },
18   image: {
19     value: '',
20     valid: false,
21     touched: false,
22     validators: [required]
23   },
24   content: {
25     value: '',
26     valid: false,
27     touched: false,
28     validators: [required, length({ min: 5 })]
29   }
30 };
31
```

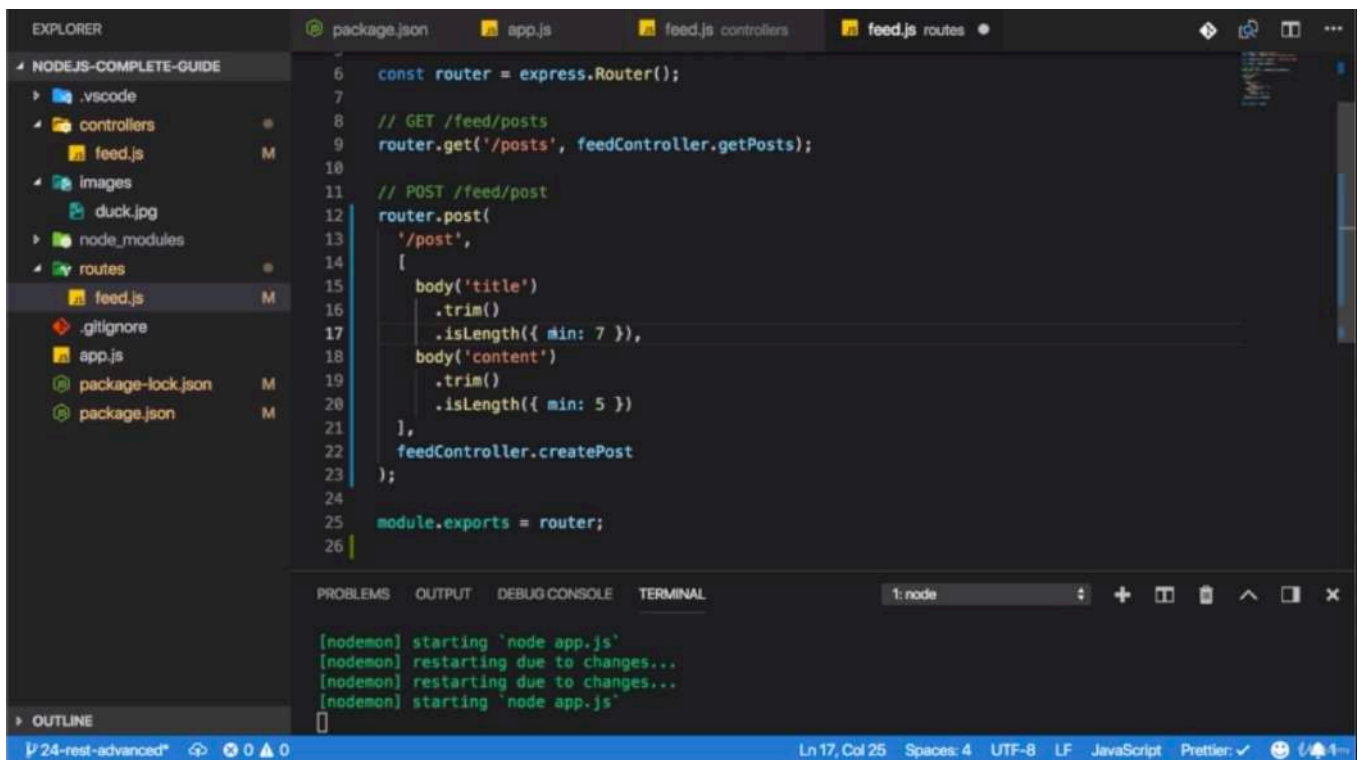
- in the ./src/components/Feed/FeedEdit/FeedEdit.js(f) file, i'm validating this to have minimum length of 5 characters and to be non-empty which is the case if it is at least 5 characters long. that's a little side note on what i'm doing on the frontend.



The screenshot shows a VS Code editor with the Explorer sidebar on the left. The Explorer shows a project structure with folders like .vscode, controllers, feed.js, images, node_modules, and routes. The main editor displays the content of app.js, which sets up an Express.js router. The router has a GET route for /feed/posts and a POST route for /feed/post. The POST route uses the feedController.createPost method. The status bar at the bottom shows the file is at line 20, column 26.

```
6 const router = express.Router();
7
8 // GET /feed/posts
9 router.get('/posts', feedController.getPosts);
10
11 // POST /feed/post
12 router.post(
13   '/post',
14   [
15     body('title')
16       .trim()
17       .isLength({ min: 5 }),
18     body('content')
19       .trim()
20       .isLength({ min: 5 })
21   ],
22   feedController.createPost
23 );
24
25 module.exports = router;
26
```

- and the backend should meed that same pattern

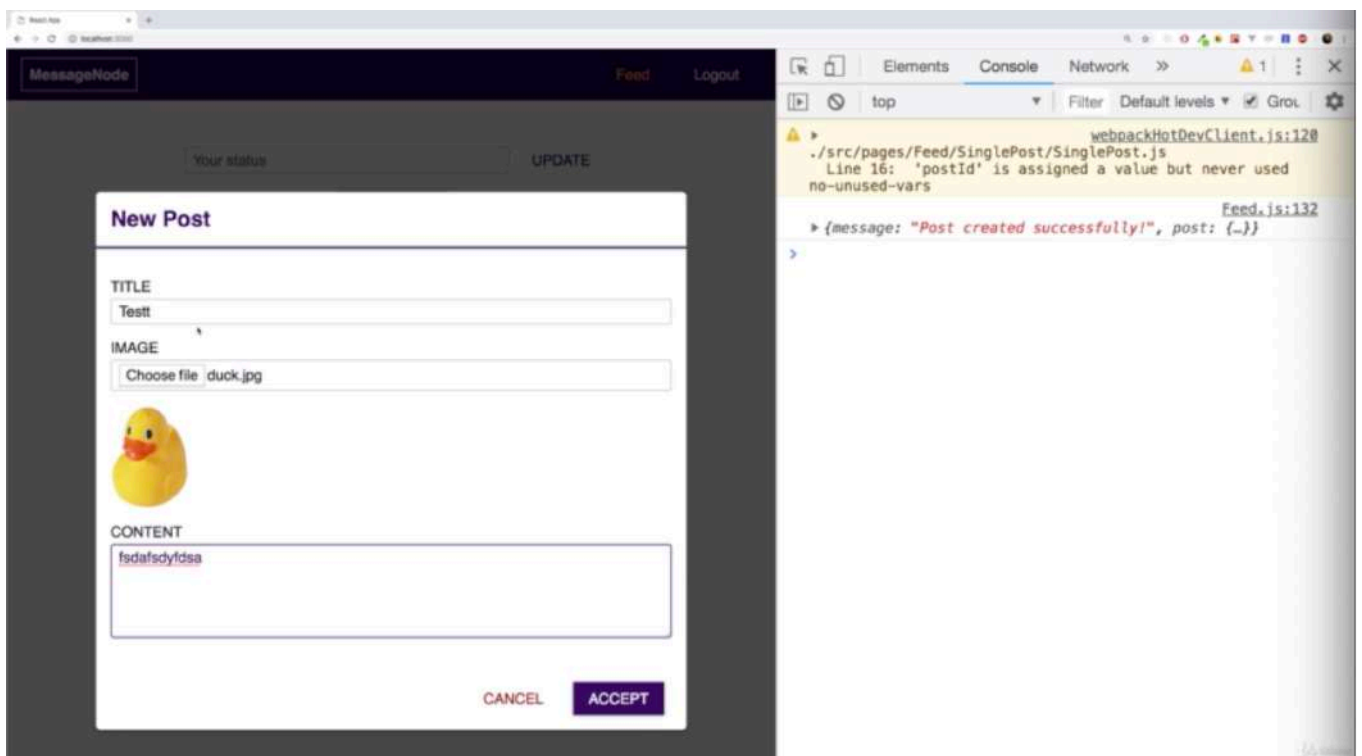


```
6 const router = express.Router();
7
8 // GET /feed/posts
9 router.get('/posts', feedController.getPosts);
10
11 // POST /feed/post
12 router.post(
13   '/post',
14   [
15     body('title')
16       .trim()
17       .isLength({ min: 7 }),
18     body('content')
19       .trim()
20       .isLength({ min: 5 })
21   ],
22   feedController.createPost
23 );
24
25 module.exports = router;
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

```
[nodemon] starting 'node app.js'
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```



- if we increase the title minimum length to 7 on the server, if I go back to the frontend and I enter a title which is only 5 characters long and therefore long enough for the client side validation but too short for the server side validation which is a mismatch you don't want to have in a real app but it's great for testing.

MessageNode

FeedLogout

Your status

UPDATE

An Error Occurred

Creating or editing a post failed!

CANCELACCEPT

Posted by Maximilian on 10/10/2018

Testing

VIEWEDITDELETE

ElementsConsoleNetwork

top

FilterDefault levelsGro

webpackHotDevClient.js:120
./src/pages/Feed/SinglePost/SinglePost.js
Line 16: 'postId' is assigned a value but never used
no-unused-vars

Feed.js:132
{message: "Post created successfully!", post: {...}}

POST http://localhost:8080/feed/post 422 post:1
(Unprocessable Entity)

Error: Creating or editing a post failed! Feed.js:159
at Feed.js:127

MessageNode

FeedLogout

Your status

UPDATE

An Error Occurred

Creating or editing a post failed!

CANCELACCEPT

Posted by Maximilian on 10/10/2018

Testing

VIEWEDITDELETE

ElementsConsoleNetwork

View:Group by framePresen

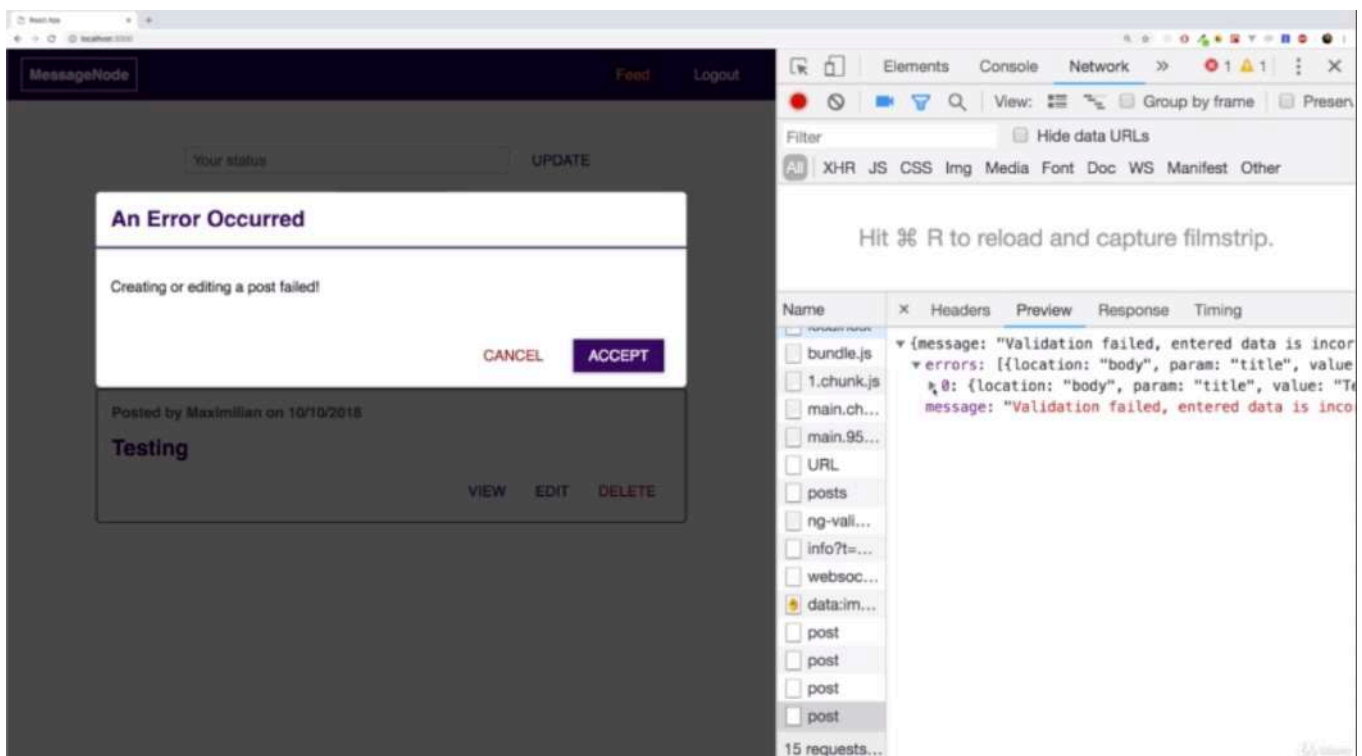
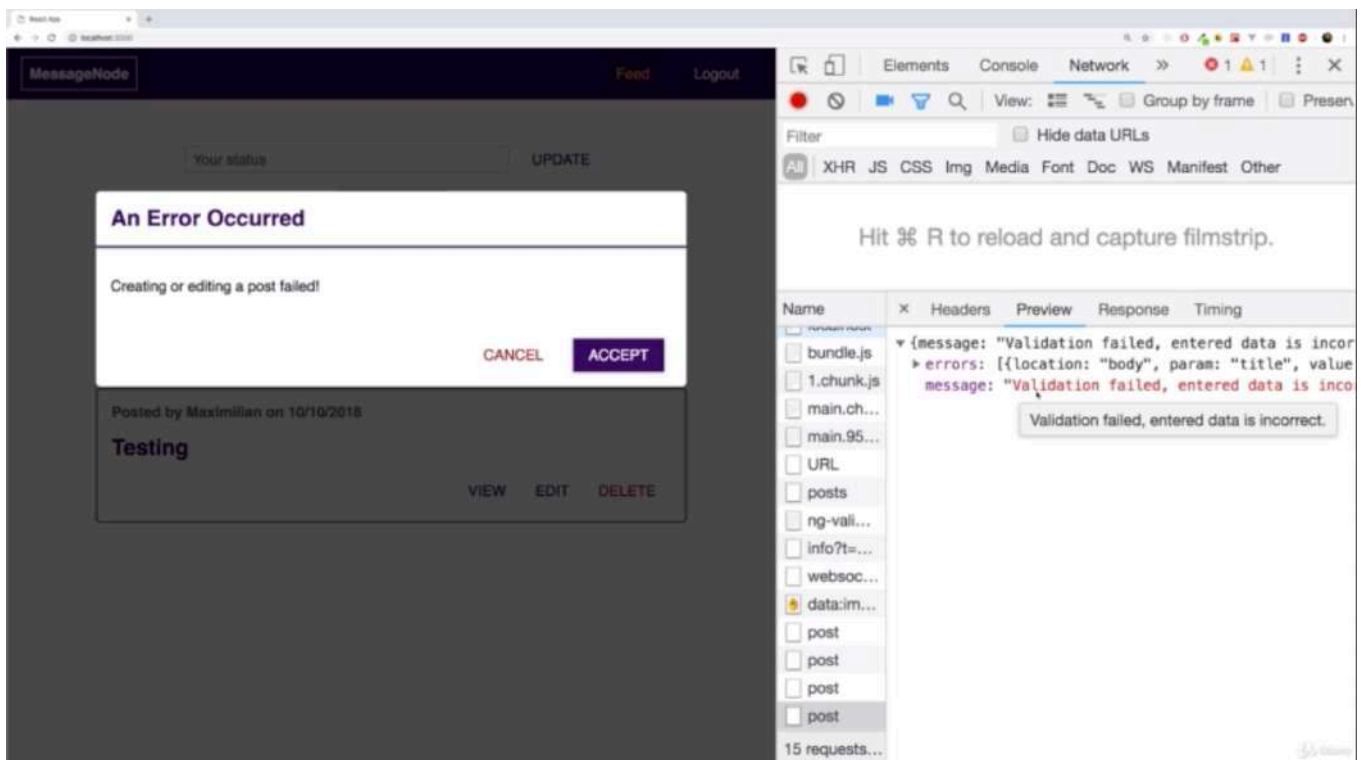
FilterHide data URLs

AllXHRJS CSS Img Media Font Doc WS Manifest Other

Hit ⌘ R to reload and capture filmstrip.

| Name | Status | Type | Size | Waterfall |
|--------------------|--------|--------|--------------|-----------|
| bundle.js | 200 | scr... | 6.5 KB | |
| 1.chunk.js | 200 | scr... | 389 KB | |
| main.chunk.js | 200 | scr... | 16.7 KB | |
| main.95d7631a... | 200 | scr... | 3.3 KB | |
| URL | 200 | fetch | 1.1 KB | |
| posts | 200 | fetch | 554 B | |
| ng-validate.js | 200 | scr... | (from dis... | |
| info?t=1539156... | 200 | xhr | 367 B | |
| websocket | 101 | we... | 0 B | |
| data:image/jpeg... | 200 | jpeg | (from me... | |
| post | 200 | fetch | 381 B | |
| post | 201 | json | 572 B | |
| post | 200 | fetch | 381 B | |
| post | 422 | json | 536 B | |

15 requests1.4 min



- that is the error message i'm showing on the frontend.
- but if you have a look at our console.log in the dev tools, we see a 422 error here and if we click on that we are taken to the network tab, we see that red request and if we inspect that, we see our response and that has that message we sent on the server side 'validation failed and so on' and it has this errors array wherer we see that the title failed. so you see that the server side validation is working

```

1 //./controllers/feed.js(b)
2
3 const { validationResult } = require('express-validator/check')
4
5 exports.getPosts = (req, res, next) => {
6   res.status(200).json({
7     posts: [
8       {

```

```

9      _id: '1',
10     title: 'First Post',
11     content: 'This is the first post!',
12     imageUrl: 'images/duck.jpg',
13     creator: {
14       name: 'Maximilian'
15     },
16     createdAt: new Date()
17   }
18 ]
19 });
20 };
21
22 exports.createPost = (req, res, next) => {
23   const errors = validationResult(req);
24   if(!errors.isEmpty()){
25     return res.status(422).json({
26       message: 'Validation Failed, Entered data is incorrect.',
27       errors: errors.array()
28     })
29   }
30   const title = req.body.title;
31   const content = req.body.content;
32   console.log(title, content);
33   //Create post in db
34   res.status(201).json({
35     message: 'Post created successfully!',
36     post: {
37       _id: new Date().toISOString(),
38       title: title,
39       content: content,
40       creator: { name: 'Maximilian' },
41       createdAt: new Date()
42     }
43   });
44 };

```

```

1  //./routes/feed.js(b)
2
3  const express = require('express');
4  /**'check' is the sub-package
5   * and that something is 'check' method to check the headers, the query parameters and so
6   * or the 'body' method to check the request body
7   */
8  const { body } = require('express-validator/check');
9
10 const feedController = require('../controllers/feed');
11
12 const router = express.Router();
13
14 // GET /feed/posts
15 router.get('/posts', feedController.getPosts);
16
17 // POST /feed/post
18 router.post('/post', [
19   body('title').trim().isLength({min: 5}),

```

```
20     body('content').trim().isLength({min: 5})
21 ], feedController.createPost);
22
23 module.exports = router;
```

* Chapter 372: Setting Up A Post Model

- 1. update
 - app.js(b)
 - ./models/post.js(b)

The screenshot displays a VS Code workspace for a Node.js application. The Explorer sidebar on the left lists the project files: package.json, app.js, feed.js, controllers, images, node_modules, routes, .gitignore, and package-lock.json. The main editor area shows the content of feed.js, which defines a feedController and a router. The router has a GET endpoint for /feed/posts and a POST endpoint for /feed/post. The REST client at the bottom shows a GET request to http://localhost:3000/feed/posts with a response of an array of two objects. The status bar at the bottom indicates the file is at line 19, column 14.

```

1 //app.js(b)
2
3 const express = require('express');
4 const bodyParser = require('body-parser');
5 const mongoose = require('mongoose');
6
7
8 const feedRoutes = require('./routes/feed')
9
10 const app = express();
11
12 app.use(bodyParser.json())
13
14 app.use((req, res, next) => {
15     res.setHeader('Access-Control-Allow-Origin', '*');
16     res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
17     res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization')
18     next()
19 })
20
21 app.use('/feed', feedRoutes)
22
23 /**connecting with MongoDB Atlas */
24 mongoose
25     .connect(
26         'mongodb+srv://maximilian:rlndnjs12@cluster0-z3v1k.mongodb.net/message?
retryWrites=true'
27     )
28     .then(result => {
29         app.listen(8080);
30     })
31     .catch(err => console.log(err));

```

```

1 //./models/post.js(b)
2
3 const mongoose = require('mongoose');
4 const Schema = mongoose.Schema;
5
6 const postSchema = new Schema({
7     title: {
8         type: String
9     },
10    imageUrl: {
11        type: String,
12        required: true
13    },
14    content: {
15        type: String,
16        required: true
17    },
18    creator: {
19        type: Object,
20        required: String
21    }
22 })
23 /**we can pass an option to the schema constructor
24 * which we haven't seen before,

```

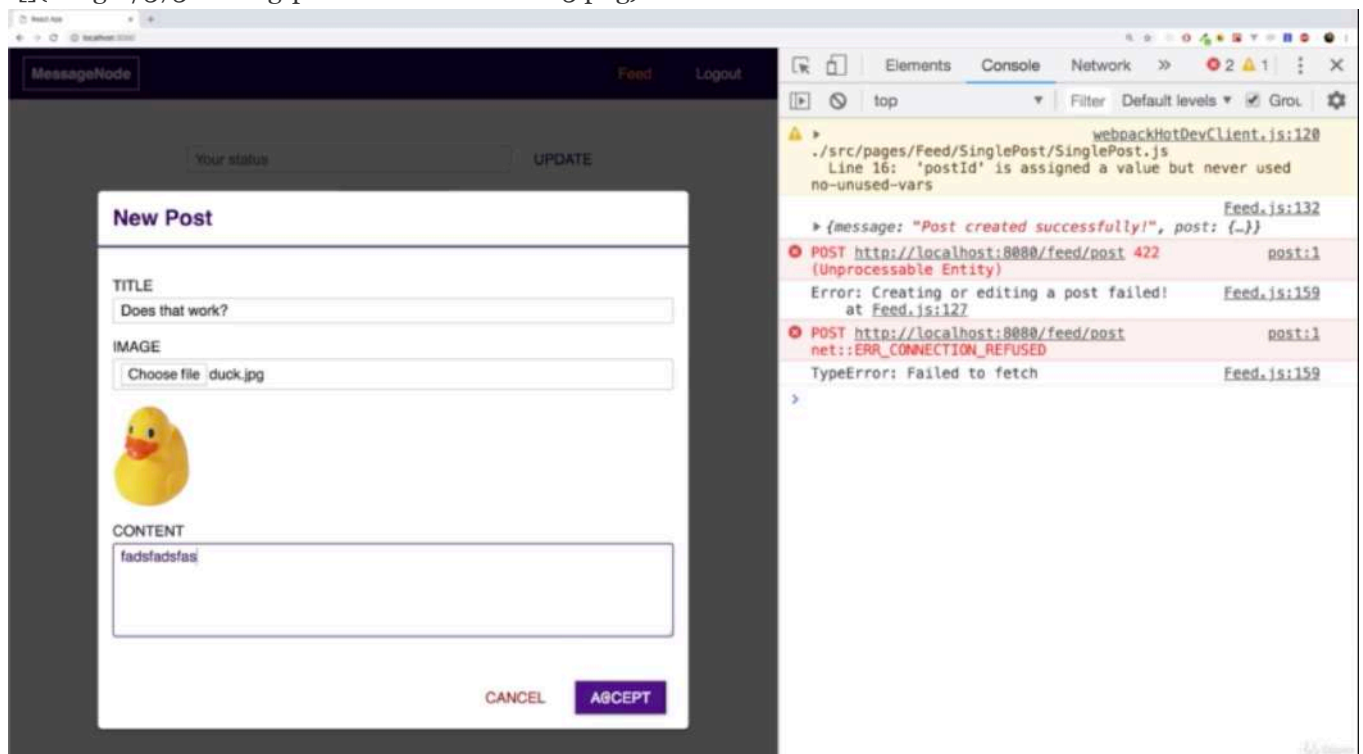
```

24 * we can configure this
25 * and in that object, i'm passing as a 2nd argument to the schema constructor,
26 * i can add the 'timestamps' key
27 * and set this to true
28 * and mongoose will then automatically add timestamps
29 * when a new version is added to the database,
30 * when a new object is added to the database.
31 */
32 },
33 { timestamps: true }
34 );
35
36 /**we don't export the schema but a model-based on the schema
37 * i will name it 'Post'
38 * and therefore this will create a Post database
39 * and i will export my postSchema here.
40 */
41 module.exports = mongoose.model('Post', postSchema)

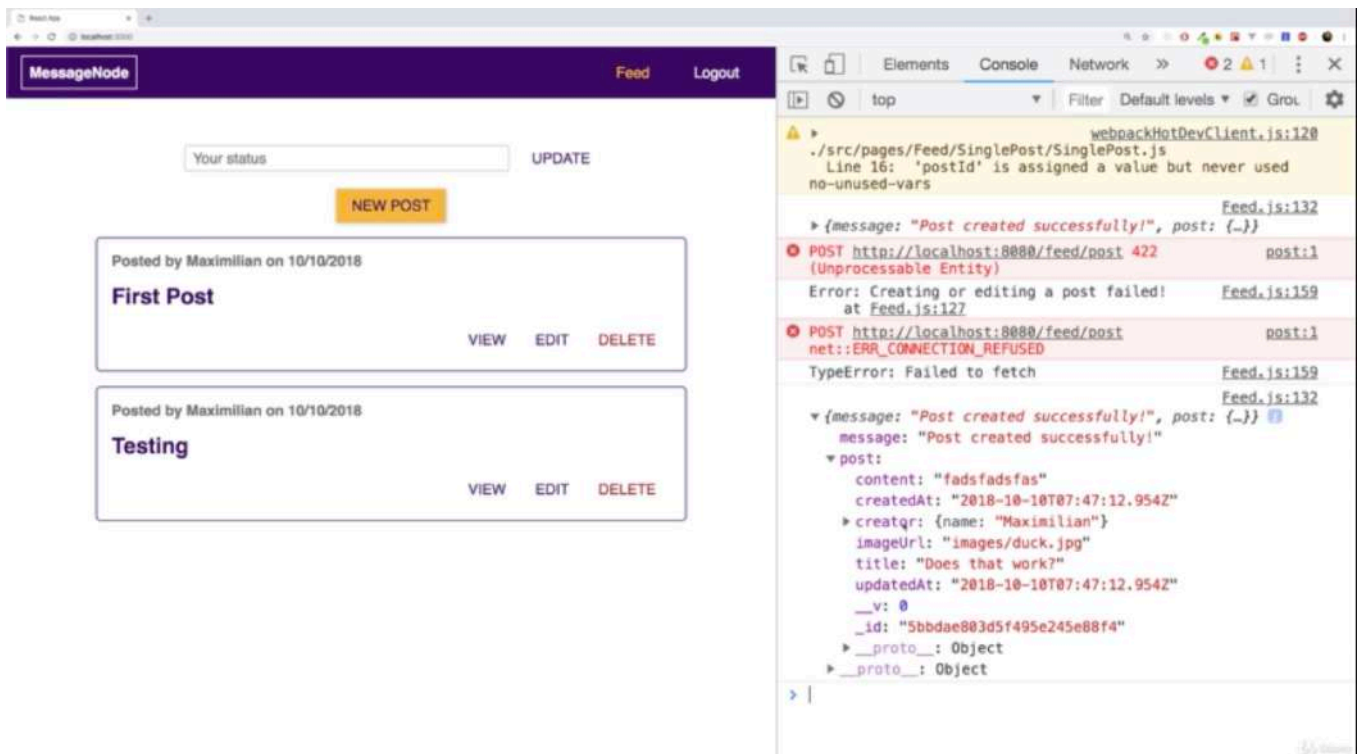
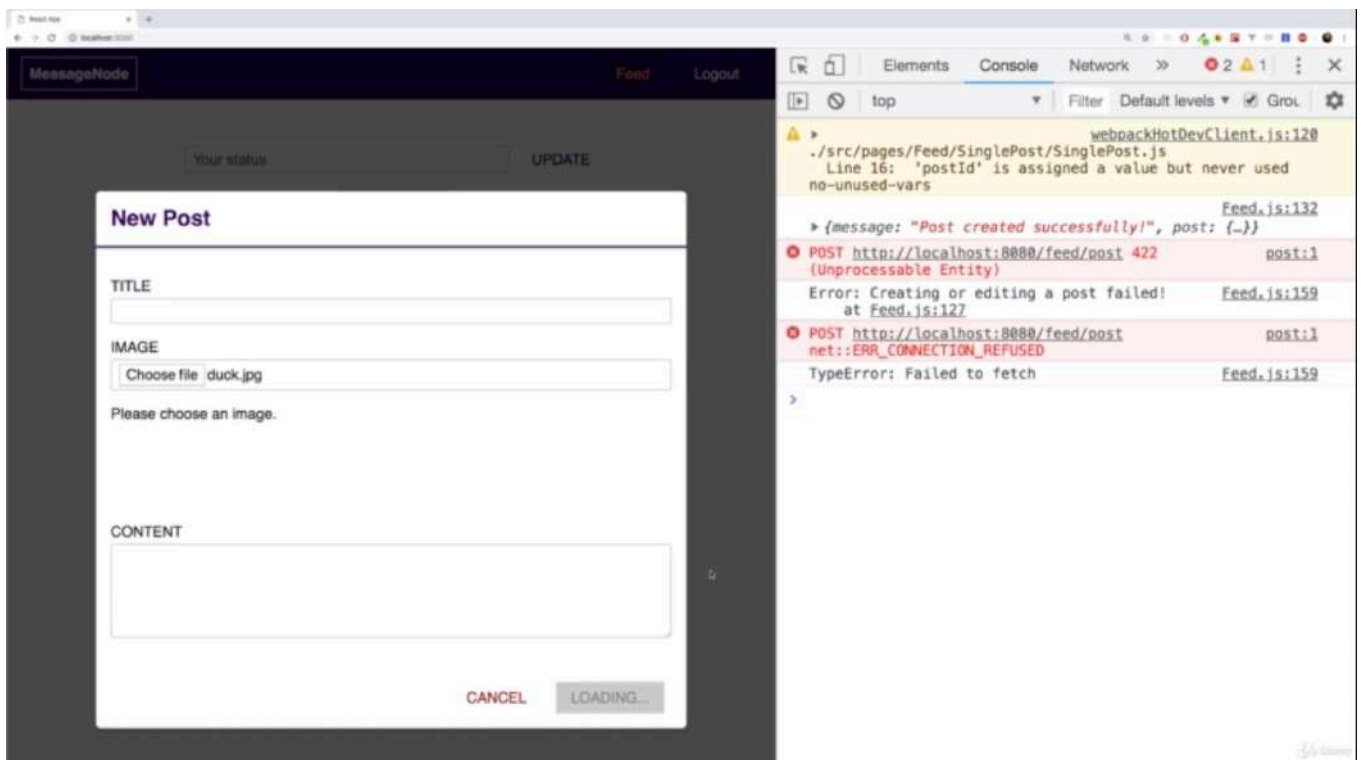
```

* Chapter 373: Storing Posts In The Database

1. update
- ./controllers/feed.js(b)



The screenshot shows a web application interface on the left and a browser console on the right. The web application has a dark theme with a top navigation bar containing 'MessageNode', 'Feed', and 'Logout'. Below the navigation bar is a form titled 'New Post' with fields for 'TITLE' (containing 'Does that work?'), 'IMAGE' (with a file input showing 'duck.jpg' and a yellow duck icon), and 'CONTENT' (containing 'fadsfadsfas'). There are 'UPDATE', 'CANCEL', and 'ACCEPT' buttons. The browser console on the right shows several error messages: a warning about an unused variable 'postId' in 'SinglePost.js', a success message 'Post created successfully!', and two 'POST' requests to 'http://localhost:8080/feed/post' that failed with 'Unprocessable Entity' and 'net::ERR_CONNECTION_REFUSED' errors.



```

1  ../controllers/feed.js(b)
2
3  const { validationResult } = require('express-validator/check');
4
5  const Post = require('../models/post');
6
7  exports.getPosts = (req, res, next) => {
8    res.status(200).json({
9      posts: [
10       {
11         _id: '1',
12         title: 'First Post',
13         content: 'This is the first post!',
14         imageUrl: 'images/duck.jpg',

```



```

15     creator: {
16       name: 'Maximilian'
17     },
18     createdAt: new Date()
19   }
20 ]
21 });
22 };
23
24 exports.createPost = (req, res, next) => {
25   const errors = validationResult(req);
26   if (!errors.isEmpty()) {
27     return res.status(422).json({
28       message: 'Validation failed, entered data is incorrect.',
29       errors: errors.array()
30     });
31   }
32   const title = req.body.title;
33   const content = req.body.content;
34   const post = new Post({
35     /**i don't need to set createdAt
36     * because mongoose will do that for me
37     * thanks to that timestamps option in ./models/post.js file
38     *
39     * i don't need to set _id
40     * because mongoose will create that for me as well
41     */
42     title: title,
43     content: content,
44     imageUrl: 'images/duck.jpg',
45     creator: { name: 'Maximilian' }
46   });
47   post
48     .save()
49     .then(result => {
50       res.status(201).json({
51         message: 'Post created successfully!',
52         post: result
53       });
54     })
55     .catch(err => {
56       console.log(err);
57     });
58   };
59

```

* Chapter 374: Static Images & Error Handling

1. update
 - app.js(b)
 - ./controllers/feed.js
 - ./routes/feed.js

The image shows a development environment with VS Code and a web browser. In VS Code, the `feed.js` file in the `routes` directory is open, showing a REST client for a POST endpoint. The client is configured with a title, image URL, and content, and a success message. The `package.json` file is also visible in the Explorer. The terminal shows the application running with Node.js and Nodemon. The web browser shows the application's interface, which includes a "New Post" form with fields for title, image, and content. The form is currently displaying the values from the REST client. The browser's console shows the REST client's response, which is a success message and a post object.

```
const feedController = require('../controllers/feed');

const router = express.Router();

// GET /feed/posts
router.get('/posts', feedController.getPosts);

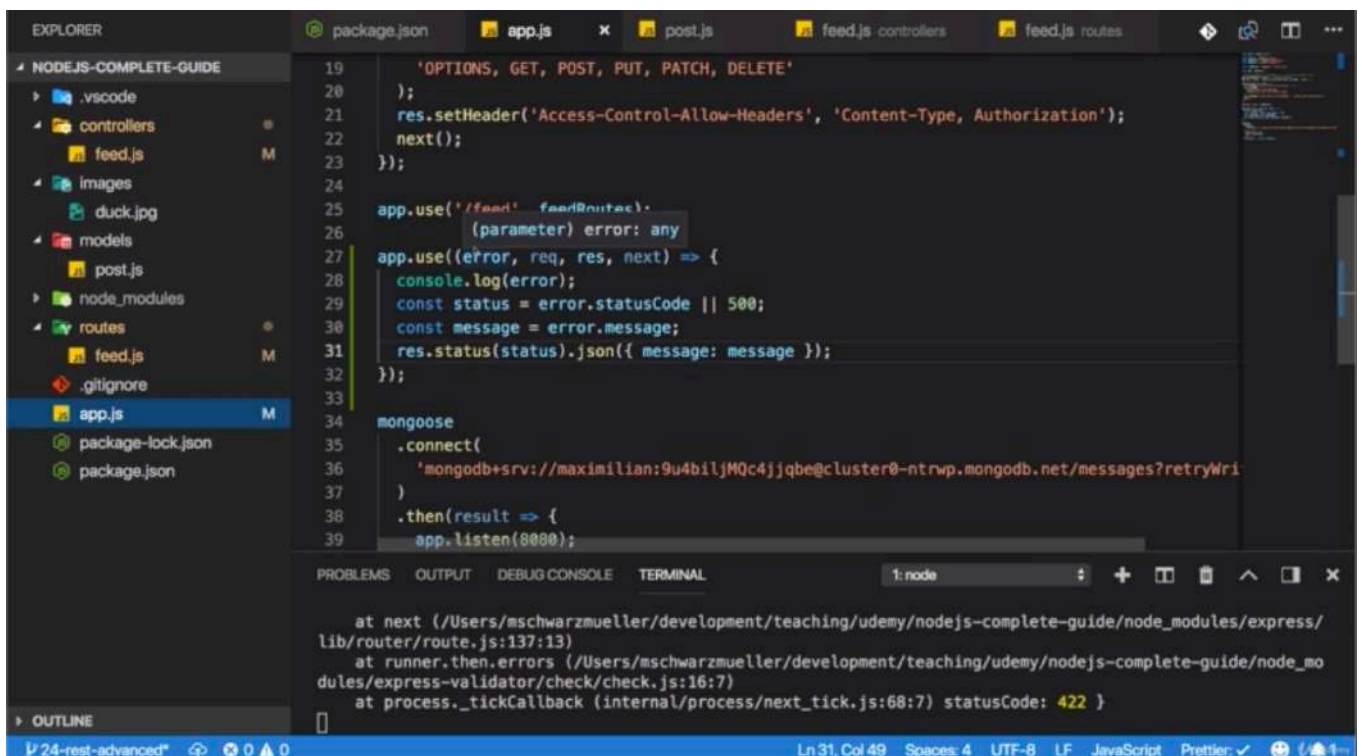
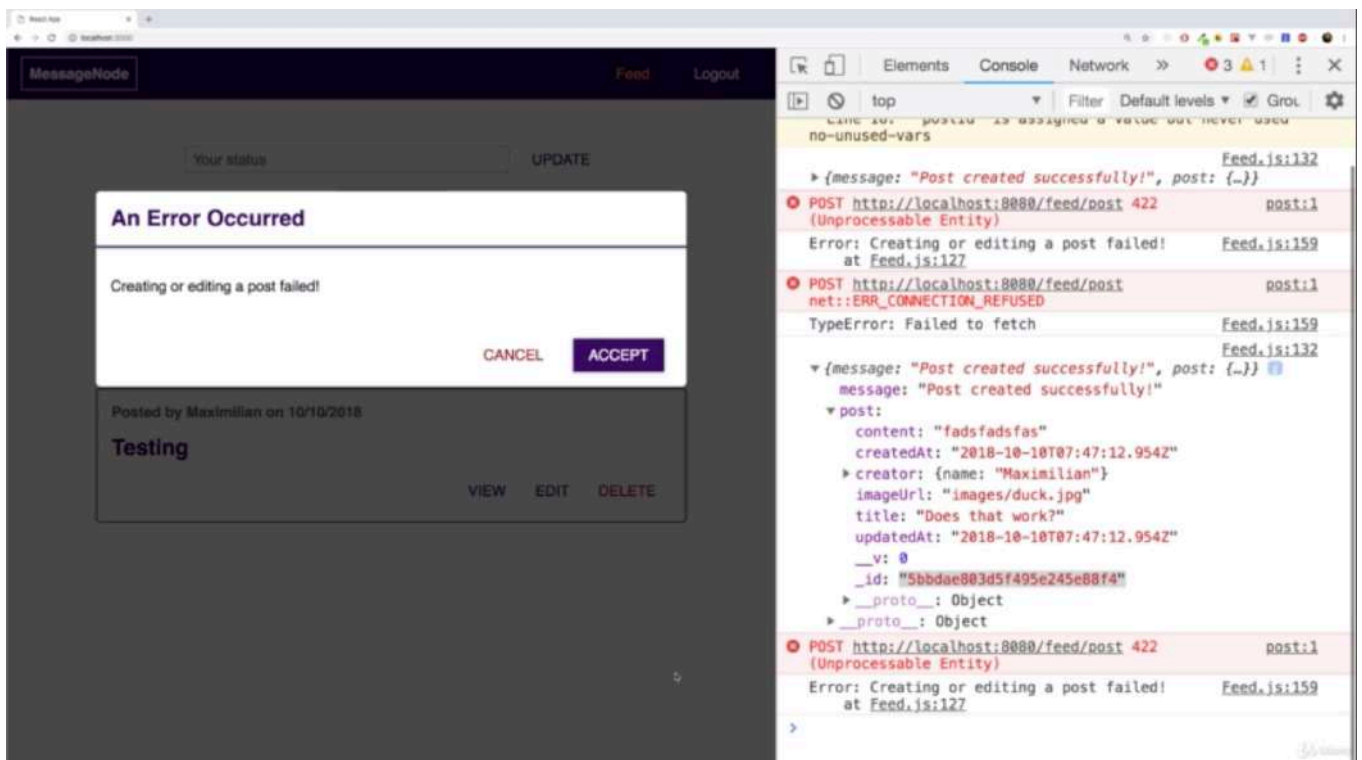
// POST /feed/post
router.post(
  '/post',
  [
    body('title')
      .trim()
      .isLength({ min: 7 }),
    body('content')
      .trim()
      .isLength({ min: 5 })
  ],
  feedController.createPost
);
```

Terminal output:

```
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
(node:24282) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
```

REST Client response:

```
{
  message: "Post created successfully!",
  post: {
    content: "fadsfadsfas",
    createdAt: "2018-10-10T07:47:12.954Z",
    creator: { name: "Maximilian" },
    imageUrl: "images/duck.jpg",
    title: "Does that work?",
    updatedAt: "2018-10-10T07:47:12.954Z",
    __v: 0,
    _id: "5bdae803d5f495e245e88f4",
    __proto__: Object
  }
}
```



- if i fill in title which is not long enough, we still get that error being thrown with the right status code. 422.

```

1 //app.js(b)
2
3 const path = require('path')
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8
9
10 const feedRoutes = require('./routes/feed')
11
12 const app = express();
13

```

```

14
15 app.use(bodyParser.json()) //application/json
16 /**i will use a middleware built into express,
17  * the static middleware which i use by calling the static function
18  *
19  * let's import the 'path' package,
20  * the core path package node provides
21  * andn then we can use 'path.join' to construct an absolute path to that images folder
22  * '__dirname' is available globally in node.js
23  * which gives us access to the directory path to that file, to the app.js file
24  * and in that same location as this app.js file,
25  * we find the images folder.
26  *
27  */
28 app.use('/images', express.static(path.join(__dirname, 'images')))
29
30 app.use((req, res, next) => {
31     res.setHeader('Access-Control-Allow-Origin', '*');
32     res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
33     res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization')
34     next()
35 })
36
37 app.use('/feed', feedRoutes);
38
39 /**this will be executed whenever an error is thrown or forwarded with next*/
40 app.use((error, req, res, next) => {
41     console.log(error);
42     const status = error.statusCode || 500;
43     const message = error.message;
44     res.status(status).json({message: message})
45 })
46
47 mongoose
48     .connect(
49         'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/message?
retryWrites=true'
50     )
51     .then(result => {
52         app.listen(8080);
53     })
54     .catch(err => console.log(err));

```



```

1 //./controllers/feed.js(b)
2
3 const { validationResult } = require('express-validator/check');
4
5 const Post = require('../models/post');
6
7 exports.getPosts = (req, res, next) => {
8     res.status(200).json({
9         posts: [
10             {
11                 _id: '1',
12                 title: 'First Post',
13                 content: 'This is the first post!',
14                 imageUrl: 'images/duck.jpg',

```

```

15     creator: {
16       name: 'Maximilian'
17     },
18     createdAt: new Date()
19   }
20 ]
21 });
22 };
23
24 exports.createPost = (req, res, next) => {
25   const errors = validationResult(req);
26   if (!errors.isEmpty()) {
27     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

*/*what does throwing an error do here?*

** since i'm not doing this in an asynchronous code snippet or anything like that,*

** it will automatically exit the function execution here*

** and instead try to reach the next error handling function or error handling*

middleware

** provided in the express application*

**/*

throw error;

}

const title = req.body.title;

const content = req.body.content;

const post = new Post({

title: title,

content: content,

imageUrl: 'images/duck.jpg',

creator: { name: 'Maximilian' }

});

post

.save()

.then(result => {

res.status(201).json({

message: 'Post created successfully!',

post: result

});

})

.catch(err => {

*/*we also got another possible error*

** and that is in here*

** if something goes wrong with storing the post,*

** i don't wanna log the error*

** instead i will check if my error has a status code field*

** which it will not have*

** but theoretically if i had more complex code*

** where i throw my own errors,*

** there might be some error that has it*

**/*

if(err.statusCode){

err.statusCode = 500;

}

*/*since i'm inside of promise chain,*

** so inside of an async code snippet,*

** throwing an error will not do the trick.*

```

70      * this will not reach the next error handling middleware
71      * this will now go and reach the next error handling express middleware
72      *
73      */
74      next(err)
75    });
76 };
77

```

```

1  //./routes/feed.js(b)
2
3  const express = require('express');
4  const { body } = require('express-validator/check');
5
6  const feedController = require('../controllers/feed');
7
8  const router = express.Router();
9
10 // GET /feed/posts
11 router.get('/posts', feedController.getPosts);
12
13 // POST /feed/post
14 router.post('/post', [
15   body('title').trim().isLength({min: 7}),
16   body('content').trim().isLength({min: 5})
17 ], feedController.createPost);
18
19 module.exports = router;

```

* Chapter 375: Fetching A Single Post

1. update
 - ./routes/feed.js(b)
 - ./controllers/feed.js(b)
 - ./src/pages/Feed/SinglePost/SinglePost.js(f)

React App

localhost:3000

MessageNodeFeedLogout

UPDATE

NEW POST

Posted by Maximilian on 10/10/2018

Does that work?

VIEW

EDIT

DELETE

Elements

Console

Network

>>

1

X

top

Filter

Default levels

Grou

Settings

webpackHotDevClient.js:120

./src/pages/Feed/SinglePost/SinglePost.js

Line 16: 'postId' is assigned a value but never used

no-unused-vars

SyntaxError: Unexpected token < in JSON SinglePost.js:33

at position 0

>

React App

localhost:3000

MessageNodeFeedLogout

UPDATE

NEW POST

Posted by Maximilian on 10/10/2018

Does that work?

VIEW

EDIT

DELETE

Elements

Console

Network

>>

1

X

top

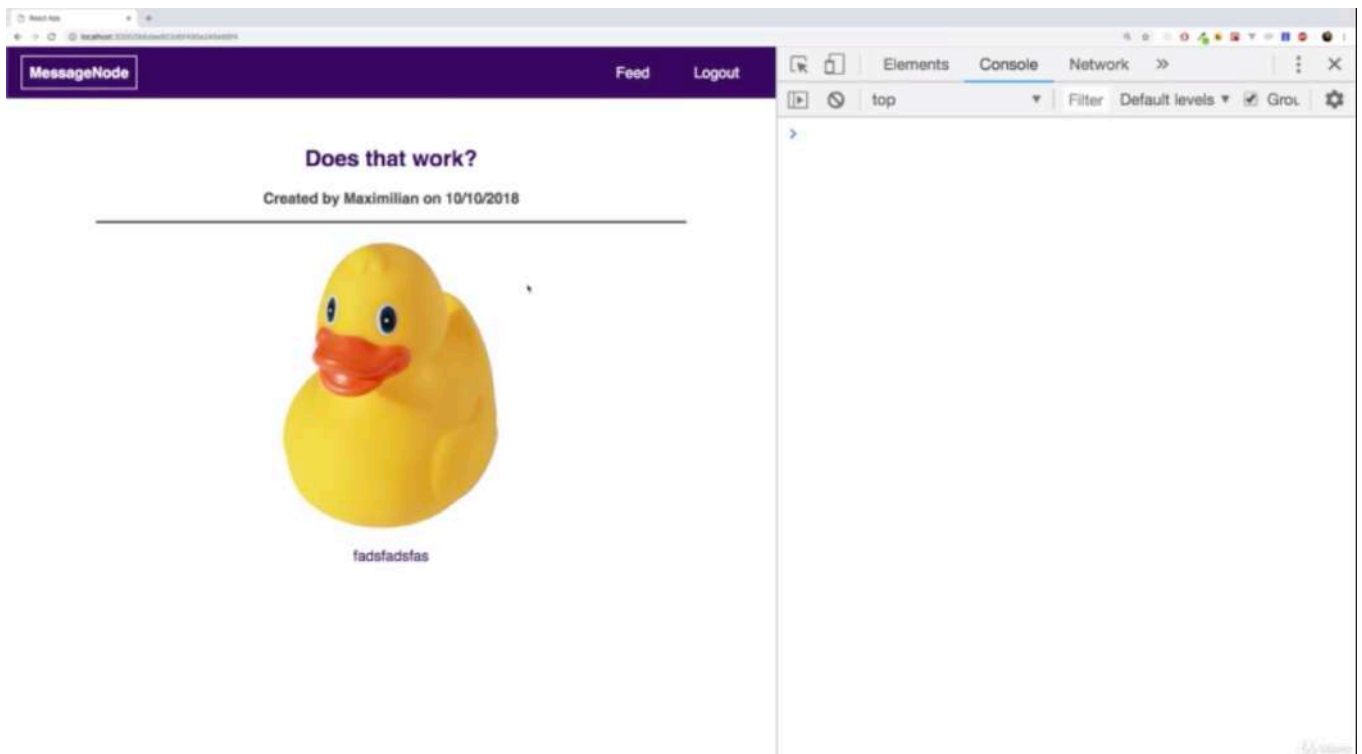
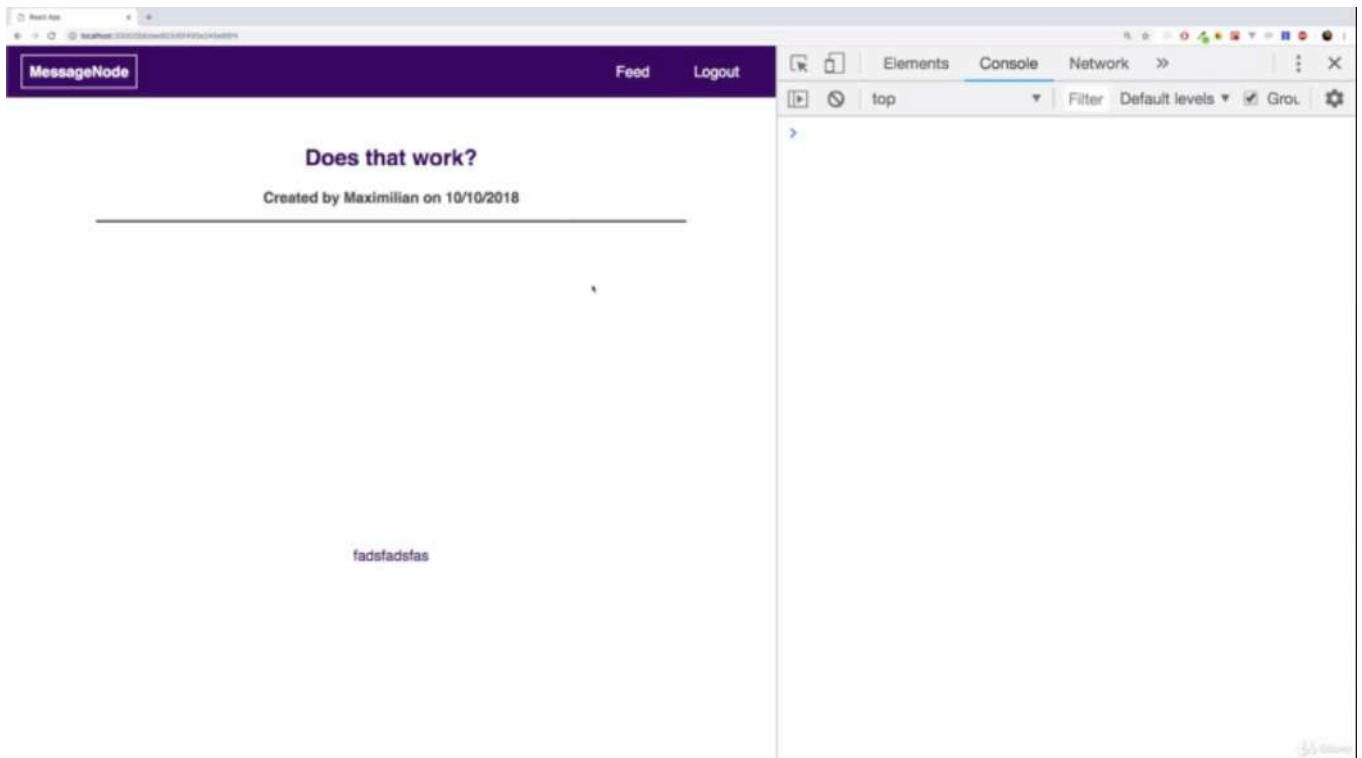
Filter

Default levels

Grou

Settings

>



```
1 //../routes/feed.js(b)
2
3 const express = require('express');
4 const { body } = require('express-validator/check');
5
6 const feedController = require('../controllers/feed');
7
8 const router = express.Router();
9
10 // GET /feed/posts
11 router.get('/posts', feedController.getPosts);
12
13 // POST /feed/post
14 router.post(
```

```

15   '/post',
16   [
17     body('title')
18       .trim()
19       .isLength({ min: 7 }),
20     body('content')
21       .trim()
22       .isLength({ min: 5 })
23   ],
24   feedController.createPost
25 );
26
27 router.get('/post/:postId', feedController.getPost);
28
29 module.exports = router;
30

```

```

1  //../controllers/feed.js(b)
2
3  const { validationResult } = require('express-validator/check');
4
5  const Post = require('../models/post');
6
7  exports.getPosts = (req, res, next) => {
8    Post.find()
9      .then(posts => {
10       res
11         .status(200)
12         .json({ message: 'Fetched posts successfully.', posts: posts });
13     })
14     .catch(err => {
15       if (!err.statusCode) {
16         err.statusCode = 500;
17       }
18       next(err);
19     });
20 };
21
22 exports.createPost = (req, res, next) => {
23   const errors = validationResult(req);
24   if (!errors.isEmpty()) {
25     const error = new Error('Validation failed, entered data is incorrect.');
```

```

26     error.statusCode = 422;
27     throw error;
28   }
29   const title = req.body.title;
30   const content = req.body.content;
31   const post = new Post({
32     title: title,
33     content: content,
34     imageUrl: 'images/duck.jpg',
35     creator: { name: 'Maximilian' }
36   });
37   post
38     .save()
39     .then(result => {
40       res.status(201).json({

```

```

41     message: 'Post created successfully!',
42     post: result
43   });
44 }
45 .catch(err => {
46   if (!err.statusCode) {
47     err.statusCode = 500;
48   }
49   next(err);
50 });
51 };
52
53 exports.getPost = (req, res, next) => {
54   const postId = req.params.postId;
55   Post.findById(postId)
56   /**this can be confusing
57    * because i'm inside of '.then()' block
58    * and you learned that
59    * you should use '.next()' in there
60    * but if you throw an error inside of '.then()' block,
61    * the next '.catch()' block will be reached
62    * and that error will be passed as an error to the catch block.
63   */
64   .then(post => {
65     if (!post) {
66       const error = new Error('Could not find post. ');
67       error.statusCode = 404;
68       throw error;
69     }
70     res.status(200).json({ message: 'Post fetched.', post: post });
71   })
72   .catch(err => {
73     if (!err.statusCode) {
74       err.statusCode = 500;
75     }
76     next(err);
77   });
78 };
79

```

```

1  //./src/pages/Feed/SinglePost/SinglePost.js
2
3  import React, { Component } from 'react';
4
5  import Image from '../../components/Image/Image';
6  import './SinglePost.css';
7
8  class SinglePost extends Component {
9    state = {
10      title: '',
11      author: '',
12      date: '',
13      image: '',
14      content: ''
15    };
16
17    componentDidMount() {

```

```

18     const postId = this.props.match.params.postId;
19     fetch('http://localhost:8080/feed/post/' + postId)
20       .then(res => {
21         if (res.status !== 200) {
22           throw new Error('Failed to fetch status');
23         }
24         return res.json();
25       })
26       .then(resData => {
27         this.setState({
28           title: resData.post.title,
29           author: resData.post.creator.name,
30           image: 'http://localhost:8080/' + resData.post.imageUrl,
31           date: new Date(resData.post.createdAt).toLocaleDateString('en-US'),
32           content: resData.post.content
33         });
34       })
35       .catch(err => {
36         console.log(err);
37       });
38   }
39
40   render() {
41     return (
42       <section className="single-post">
43         <h1>{this.state.title}</h1>
44         <h2>
45           Created by {this.state.author} on {this.state.date}
46         </h2>
47         <div className="single-post__image">
48           <Image contain imageUrl={this.state.image} />
49         </div>
50         <p>{this.state.content}</p>
51       </section>
52     );
53   }
54 }
55
56 export default SinglePost;
57

```

* Chapter 377: Uploading Images

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

-
-
-
-
-
-

EXPLORER

- NODEJS-COMPLETE-GUIDE
 - .vscode
 - controllers
 - feed.js
 - images
 - duck.jpg
 - models
 - post.js
 - node_modules
 - routes
 - feed.js
 - .gitignore
 - app.js
 - package-lock.json
 - package.json

```
51 exports.getPost = (req, res, next) => {
52   const postId = req.params.postId;
53   Post.findById(postId)
54     .then(post => {
55     if (!post) {
56       const error = new Error('Could not find post. ');
57       error.statusCode = 404;
58       throw error;
59     }
60     res.status(200).json({ message: 'Post fetched.', post: post });
61   })
62   .catch(err => {
63     if (!err.statusCode) {
64       err.statusCode = 500;
65     }
66     next(err);
67   });
68 };
69
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

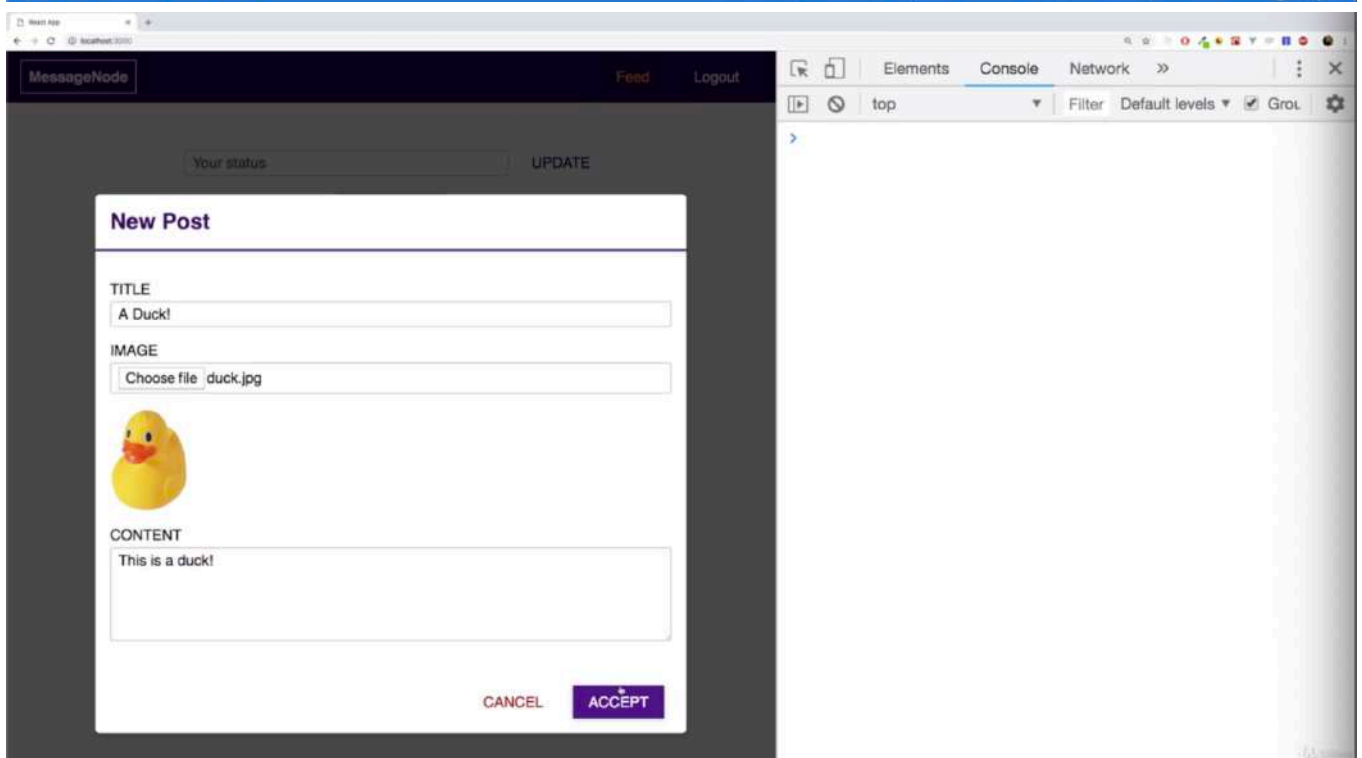
1: node

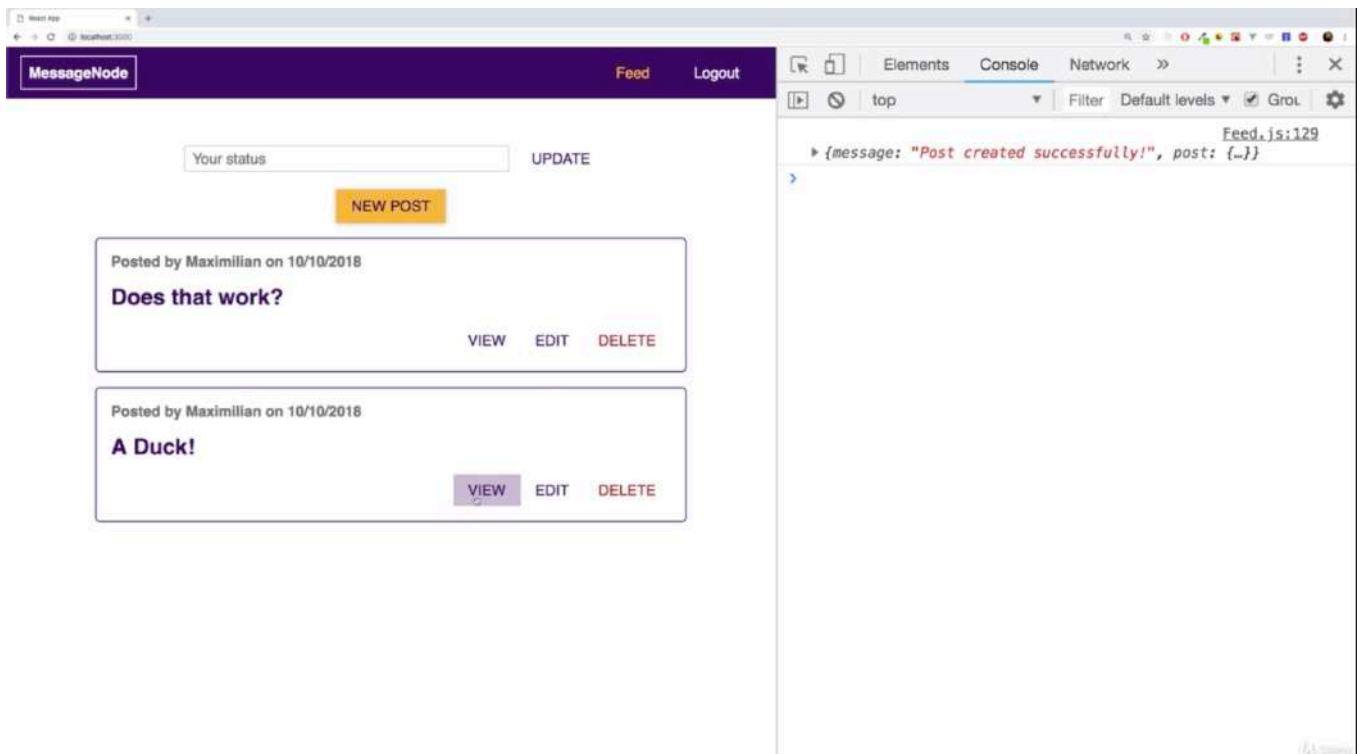
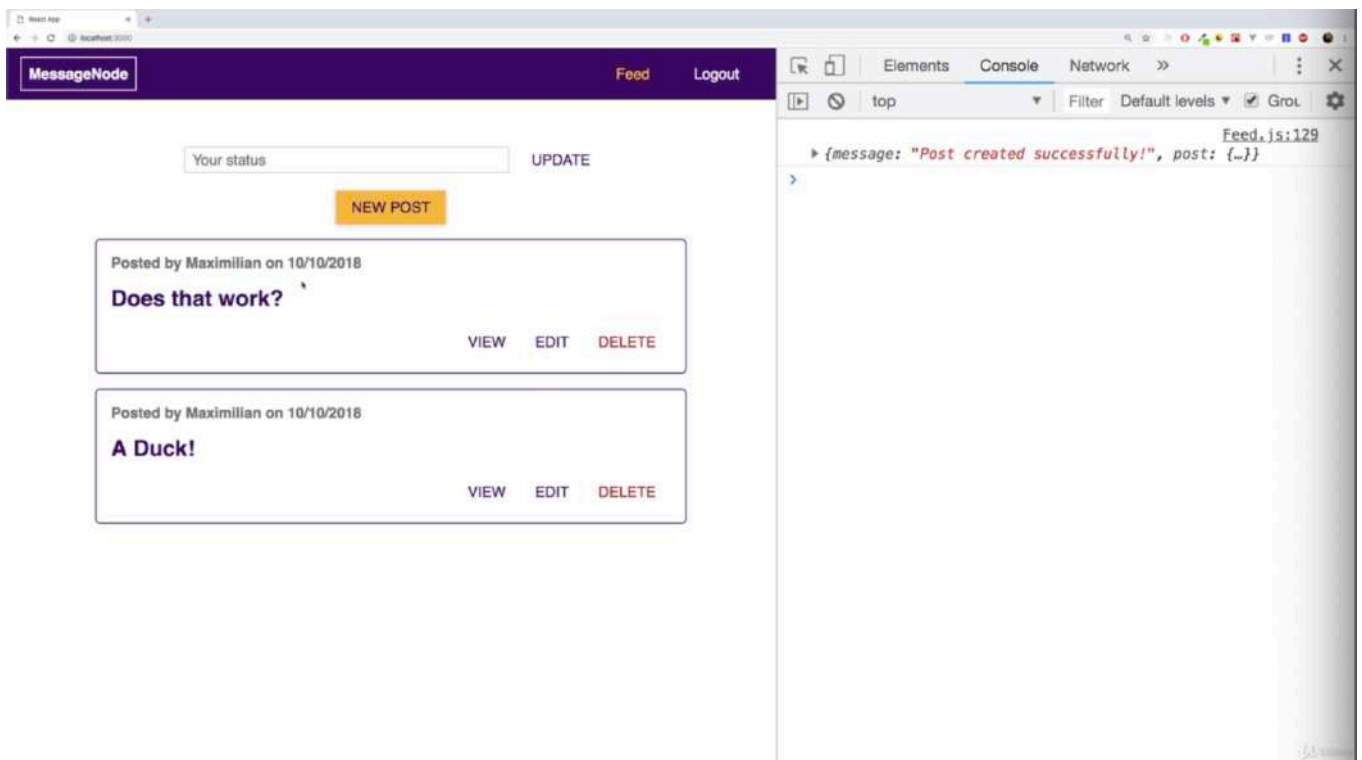
Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$
Maximilians-MBP:nodejs-complete-guide mschwarzmuellers\$ npm install --save multer
(...) i rollbackFailedOptional: verb npm-session 3caec49d08664314

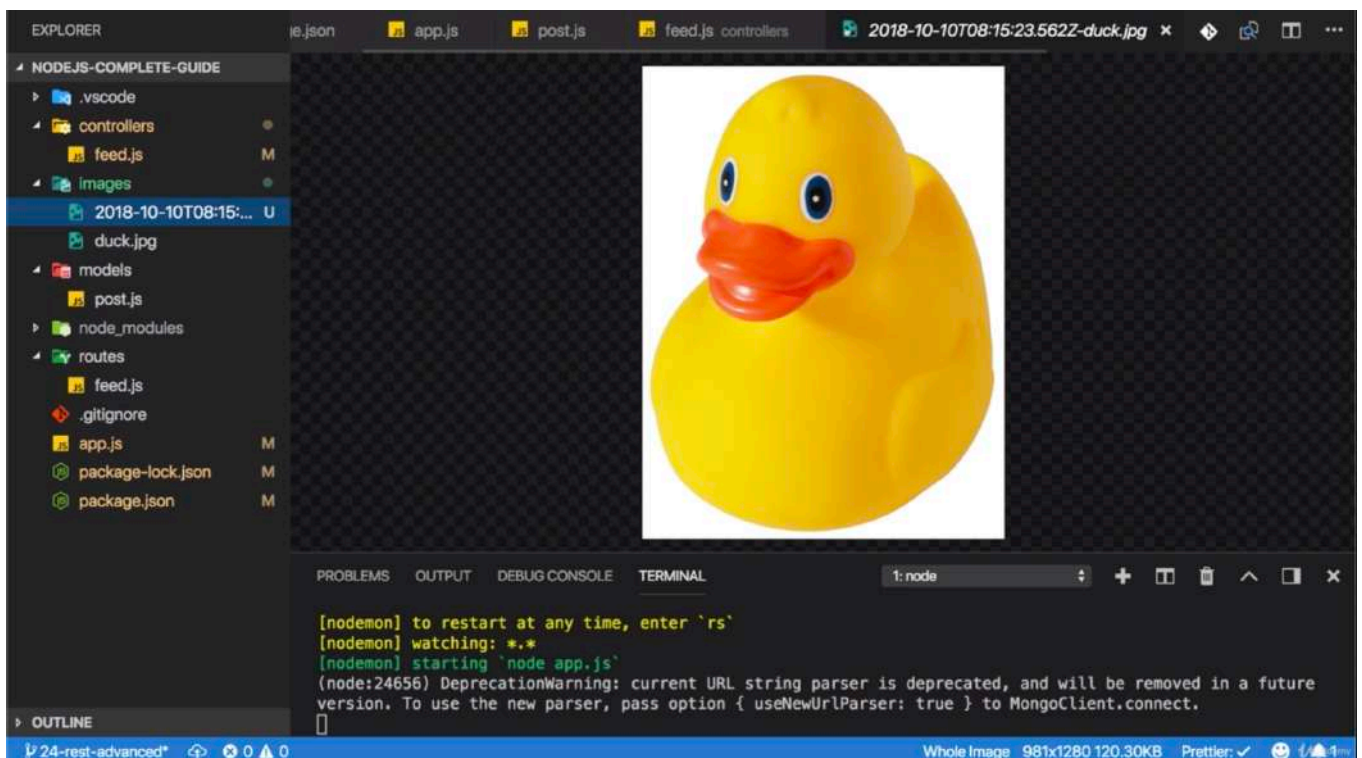
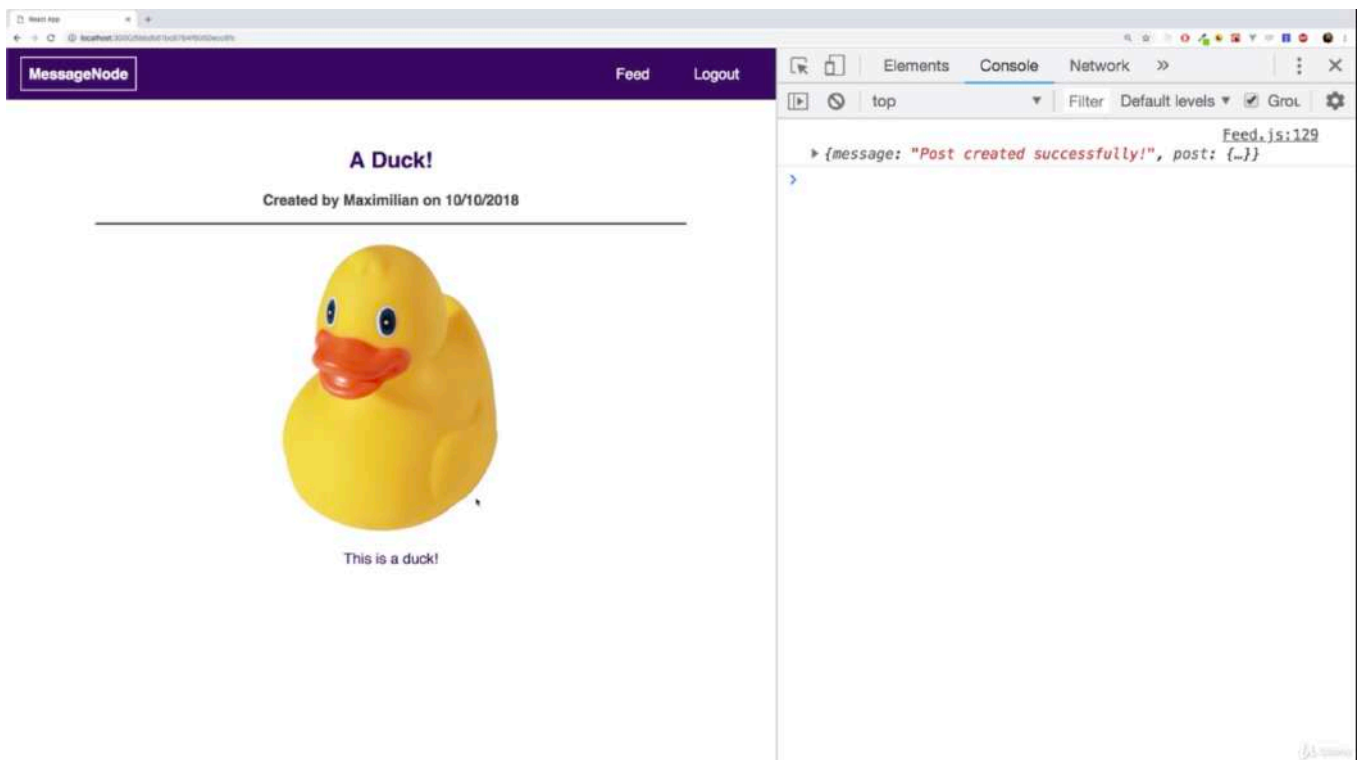
OUTLINE

24-rest-advanced 0 0 0

Ln 60, Col 66 (10 selected) Spaces: 4 UTF-8 LF JavaScript Prettier: ✓







```

1 //app.js(b)
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9
10 const feedRoutes = require('./routes/feed');
11
12 const app = express();
13
14 const fileStorage = multer.diskStorage({

```

```

15 destination: (req, file, cb) => {
16     /**got no error so 'null'
17     * and the destination is images pointing at 'images' folder here.
18     */
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 /**on that multer function,
41 * i will then call single image to inform multer
42 * that i will extract a single file stored in some field named 'image' in the incoming
43 requests.
44 * every incoming request is parsed for that file or such files
45 * with that, multer is registered,
46 * now we can use the file in our controller
47 * where we create a new Post.
48 */
49 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
50 app.use('/images', express.static(path.join(__dirname, 'images')));
51
52 app.use((req, res, next) => {
53     res.setHeader('Access-Control-Allow-Origin', '*');
54     res.setHeader(
55         'Access-Control-Allow-Methods',
56         'OPTIONS, GET, POST, PUT, PATCH, DELETE'
57     );
58     res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
59     next();
60 });
61
62 app.use('/feed', feedRoutes);
63
64 app.use((error, req, res, next) => {
65     console.log(error);
66     const status = error.statusCode || 500;
67     const message = error.message;
68     res.status(status).json({ message: message });
69 });

```

```

70 mongoose
71   .connect(
72     'mongodb+srv://maximilian:rldnjs12@cluster0-z3vkl.mongodb.net/message?retryWrites=true'
73   )
74   .then(result => {
75     app.listen(8080);
76   })
77   .catch(err => console.log(err));

1  //./controllers/feed.js(b)
2
3  const { validationResult } = require('express-validator/check');
4
5  const Post = require('../models/post');
6
7  exports.getPosts = (req, res, next) => {
8    Post.find()
9      .then(posts => {
10       res
11         .status(200)
12         .json({ message: 'Fetched posts successfully.', posts: posts });
13     })
14     .catch(err => {
15       if (!err.statusCode) {
16         err.statusCode = 500;
17       }
18       next(err);
19     });
20 };
21
22 exports.createPost = (req, res, next) => {
23   const errors = validationResult(req);
24   if (!errors.isEmpty()) {
25     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

```

26     error.statusCode = 422;
27     throw error;
28   }
29   if(!req.file){
30     const error = new Error('No image provided.');
```

error.statusCode = 422;

```

31     error.statusCode = 422;
32     throw error;
33   }
34   /**there the path variable which multer generates
35    * which holds the path to the file as it was stored on my server.
36    */
37   const imageUrl = req.file.path;
38   const title = req.body.title;
39   const content = req.body.content;
40   const post = new Post({
41     title: title,
42     content: content,
43     imageUrl: imageUrl,
44     creator: { name: 'Maximilian' }
45   });
46   post
47     .save()
48     .then(result => {
```

```

49     res.status(201).json({
50       message: 'Post created successfully!',
51       post: result
52     });
53   })
54   .catch(err => {
55     if (!err.statusCode) {
56       err.statusCode = 500;
57     }
58     next(err);
59   });
60 };
61
62 exports.getPost = (req, res, next) => {
63   const postId = req.params.postId;
64   Post.findById(postId)
65     .then(post => {
66       if (!post) {
67         const error = new Error('Could not find post.');
68         error.statusCode = 404;
69         throw error;
70       }
71       res.status(200).json({ message: 'Post fetched.', post: post });
72     })
73     .catch(err => {
74       if (!err.statusCode) {
75         err.statusCode = 500;
76       }
77       next(err);
78     });
79 };
80

```

```

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

```

```

25
26 componentDidMount() {
27     fetch('URL')
28         .then(res => {
29             if (res.status !== 200) {
30                 throw new Error('Failed to fetch user status.');
```

```

31             }
32             return res.json();
33         })
34         .then(resData => {
35             this.setState({ status: resData.status });
36         })
37         .catch(this.catchError);
38
39     this.loadPosts();
40 }
41
42 loadPosts = direction => {
43     if (direction) {
44         this.setState({ postsLoading: true, posts: [] });
45     }
46     let page = this.state.postPage;
47     if (direction === 'next') {
48         page++;
49         this.setState({ postPage: page });
50     }
51     if (direction === 'previous') {
52         page--;
53         this.setState({ postPage: page });
54     }
55     fetch('http://localhost:8080/feed/posts')
56         .then(res => {
57             if (res.status !== 200) {
58                 throw new Error('Failed to fetch posts.');
```

```

59             }
60             return res.json();
61         })
62         .then(resData => {
63             this.setState({
64                 posts: resData.posts,
65                 totalPosts: resData.totalItems,
66                 postsLoading: false
67             });
68         })
69         .catch(this.catchError);
70 };
71
72 statusUpdateHandler = event => {
73     event.preventDefault();
74     fetch('URL')
75         .then(res => {
76             if (res.status !== 200 && res.status !== 201) {
77                 throw new Error("Can't update status!");
78             }
79             return res.json();
80         })

```

```

81     .then(resData => {
82         console.log(resData);
83     })
84     .catch(this.catchError);
85 };
86
87 newPostHandler = () => {
88     this.setState({ isEditing: true });
89 };
90
91 startEditPostHandler = postId => {
92     this.setState(prevState => {
93         const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
94
95         return {
96             isEditing: true,
97             editPost: loadedPost
98         };
99     });
100 };
101
102 cancelEditHandler = () => {
103     this.setState({ isEditing: false, editPost: null });
104 };
105
106 finishEditHandler = postData => {
107     this.setState({
108         editLoading: true
109     });
110     /**we can create a new Form data object with a built-in object
111     * that browser side javascript offeres, the form data object.
112     */
113     const formData = new FormData()
114     /**first argument to append is the name of the field,
115     * second argument is the actual data.
116     *
117     * now i got from data prepared that now has mixed content, text,file
118     */
119     formData.append('title', postData.title);
120     formData.append('content', postData.content);
121     formData.append('image', postData.image);
122     let url = 'http://localhost:8080/feed/post';
123     let method = 'POST';
124     if (this.state.editPost) {
125         url = 'URL';
126     }
127
128     fetch(url, {
129         method: method,
130         /**we won't use JSON data anymore
131         * because JSON data is only text,
132         * so only data that can be represented as a text
133         * which a file can't be or not easily,
134         * it will be very big quickly
135         * and very big files are a hugh issue.
136         * so we can't use JSON for data

```



```

137     * where we have both a file or normal text data.
138     * instead we will use form data.
139     *
140     * when we used a form with this multipart form anchor tag
141     * which we added to the form HTML element.
142     * but we will not use anything to any form element,
143     * we will do it all in javascript instead.
144     */
145
146     /**form data will automatically set the headers.
147     * that is kind of convenience
148     */
149     body: formData
150
151 })
152
153 .then(res => {
154     if (res.status !== 200 && res.status !== 201) {
155         throw new Error('Creating or editing a post failed!');
156     }
157     return res.json();
158 })
159
160 .then(resData => {
161     console.log(resData);
162     const post = {
163         _id: resData.post._id,
164         title: resData.post.title,
165         content: resData.post.content,
166         creator: resData.post.creator,
167         createdAt: resData.post.createdAt
168     };
169     this.setState(prevState => {
170         let updatedPosts = [...prevState.posts];
171         if (prevState.editPost) {
172             const postIndex = prevState.posts.findIndex(
173                 p => p._id === prevState.editPost._id
174             );
175             updatedPosts[postIndex] = post;
176         } else if (prevState.posts.length < 2) {
177             updatedPosts = prevState.posts.concat(post);
178         }
179         return {
180             posts: updatedPosts,
181             isEditing: false,
182             editPost: null,
183             editLoading: false
184         };
185     });
186 })
187
188 .catch(err => {
189     console.log(err);
190     this.setState({
191         isEditing: false,
192         editPost: null,
193         editLoading: false,
194         error: err
195     });
196 })

```

```

193     });
194 };
195
196 statusInputChangeHandler = (input, value) => {
197     this.setState({ status: value });
198 };
199
200 deletePostHandler = postId => {
201     this.setState({ postsLoading: true });
202     fetch('URL')
203         .then(res => {
204             if (res.status !== 200 && res.status !== 201) {
205                 throw new Error('Deleting a post failed!');
206             }
207             return res.json();
208         })
209         .then(resData => {
210             console.log(resData);
211             this.setState(prevState => {
212                 const updatedPosts = prevState.posts.filter(p => p._id !== postId);
213                 return { posts: updatedPosts, postsLoading: false };
214             });
215         })
216         .catch(err => {
217             console.log(err);
218             this.setState({ postsLoading: false });
219         });
220 };
221
222 errorHandler = () => {
223     this.setState({ error: null });
224 };
225
226 catchError = error => {
227     this.setState({ error: error });
228 };
229
230 render() {
231     return (
232         <Fragment>
233             <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
234             <FeedEdit
235                 editing={this.state.isEditing}
236                 selectedPost={this.state.editPost}
237                 loading={this.state.editLoading}
238                 onCancelEdit={this.cancelEditHandler}
239                 onFinishEdit={this.finishEditHandler}
240             />
241             <section className="feed__status">
242                 <form onSubmit={this.statusUpdateHandler}>
243                     <Input
244                         type="text"
245                         placeholder="Your status"
246                         control="input"
247                         onChange={this.statusInputChangeHandler}
248                         value={this.state.status}

```

```

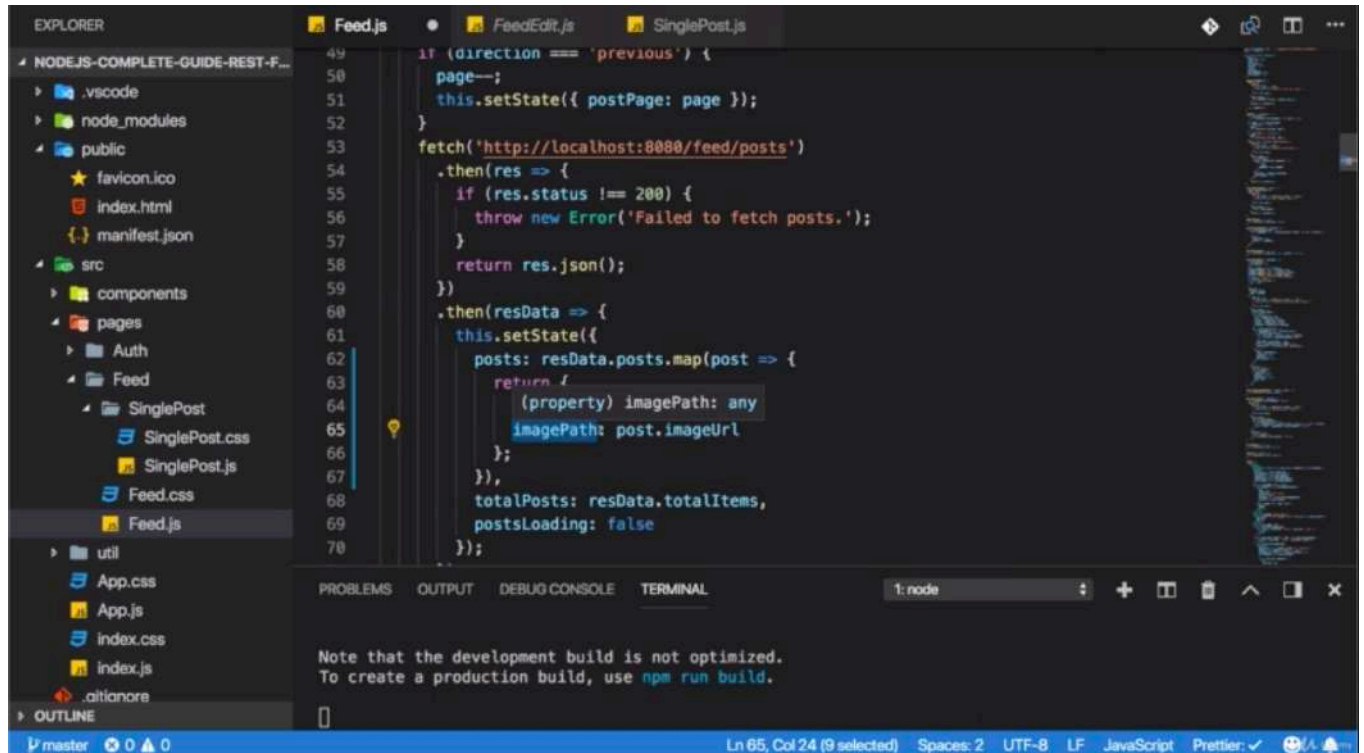
249     />
250     <Button mode="flat" type="submit">
251       Update
252     </Button>
253   </form>
254 </section>
255 <section className="feed__control">
256   <Button mode="raised" design="accent" onClick={this.newPostHandler}>
257     New Post
258   </Button>
259 </section>
260 <section className="feed">
261   {this.state.postsLoading && (
262     <div style={{ textAlign: 'center', marginTop: '2rem' }}>
263       <Loader />
264     </div>
265   )}
266   {this.state.posts.length <= 0 && !this.state.postsLoading ? (
267     <p style={{ textAlign: 'center' }}>No posts found.</p>
268   ) : null}
269   {!this.state.postsLoading && (
270     <Paginator
271       onPrevious={this.loadPosts.bind(this, 'previous')}
272       onNext={this.loadPosts.bind(this, 'next')}
273       lastPage={Math.ceil(this.state.totalPosts / 2)}
274       currentPage={this.state.postPage}
275     >
276       {this.state.posts.map(post => (
277         <Post
278           key={post._id}
279           id={post._id}
280           author={post.creator}
281           date={new Date(post.createdAt).toLocaleDateString('en-US')}
282           title={post.title}
283           image={post.imageUrl}
284           content={post.content}
285           onStartEdit={this.startEditPostHandler.bind(this, post._id)}
286           onDelete={this.deletePostHandler.bind(this, post._id)}
287         </Post>
288       ))}
289     </Paginator>
290   )}
291 </section>
292 </Fragment>
293 );
294 }
295 }
296
297 export default Feed;
298

```

* Chapter 378: Updating Posts

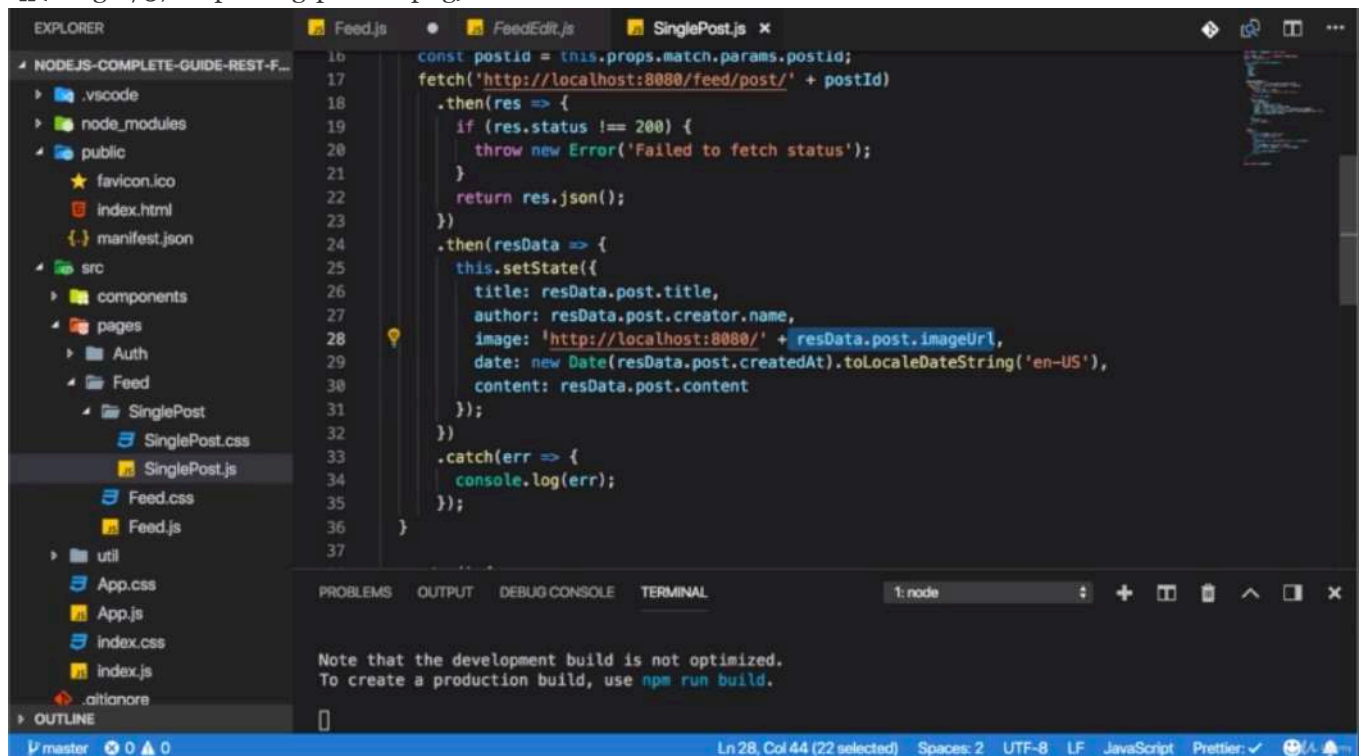
1. update
 - ./routes/feed.js(b)
 - ./controllers/feed.js(b)

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



```
49 if (direction === 'previous') {
50   page--;
51   this.setState({ postPage: page });
52 }
53 fetch('http://localhost:8080/feed/posts')
54   .then(res => {
55     if (res.status !== 200) {
56       throw new Error('Failed to fetch posts.');
```

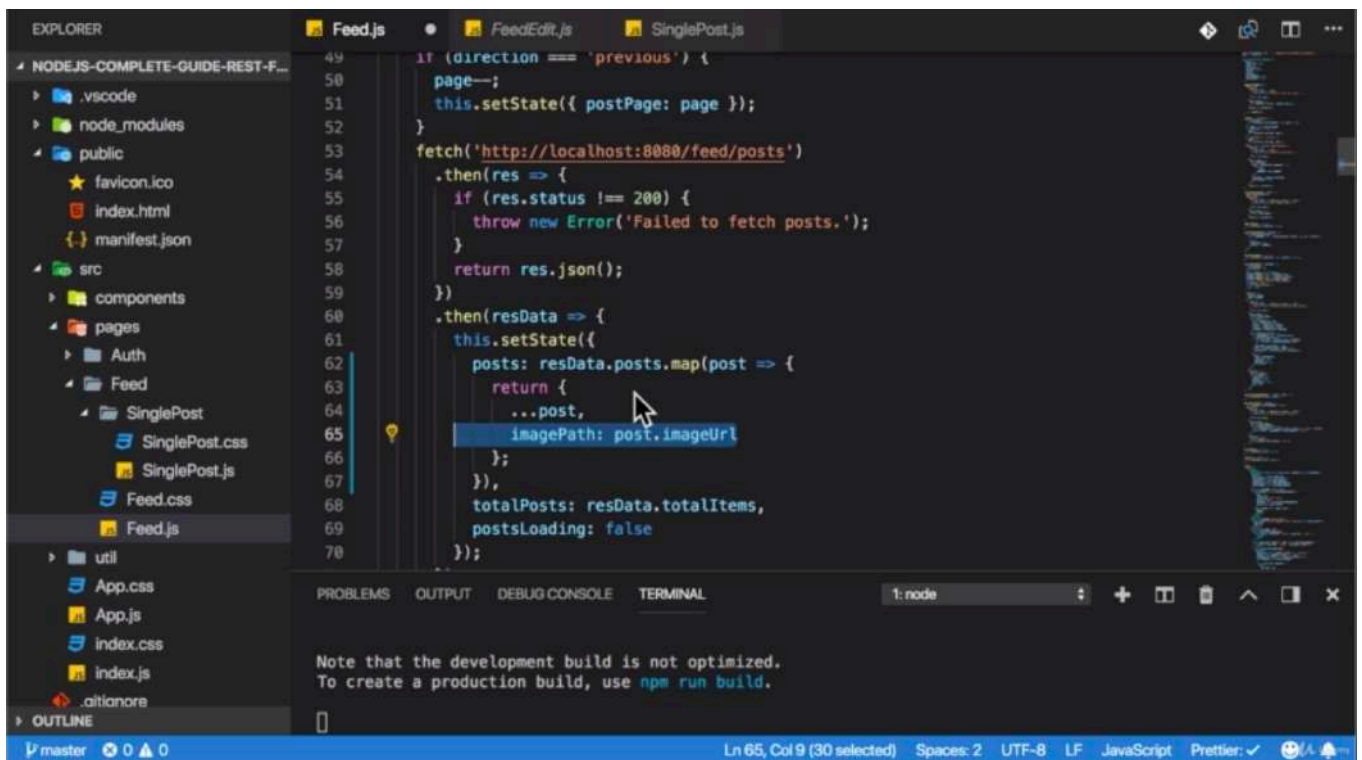
- 'imageUrl' is from ./models/post.js file in backend. i'm storing the original path 'imagePath'.



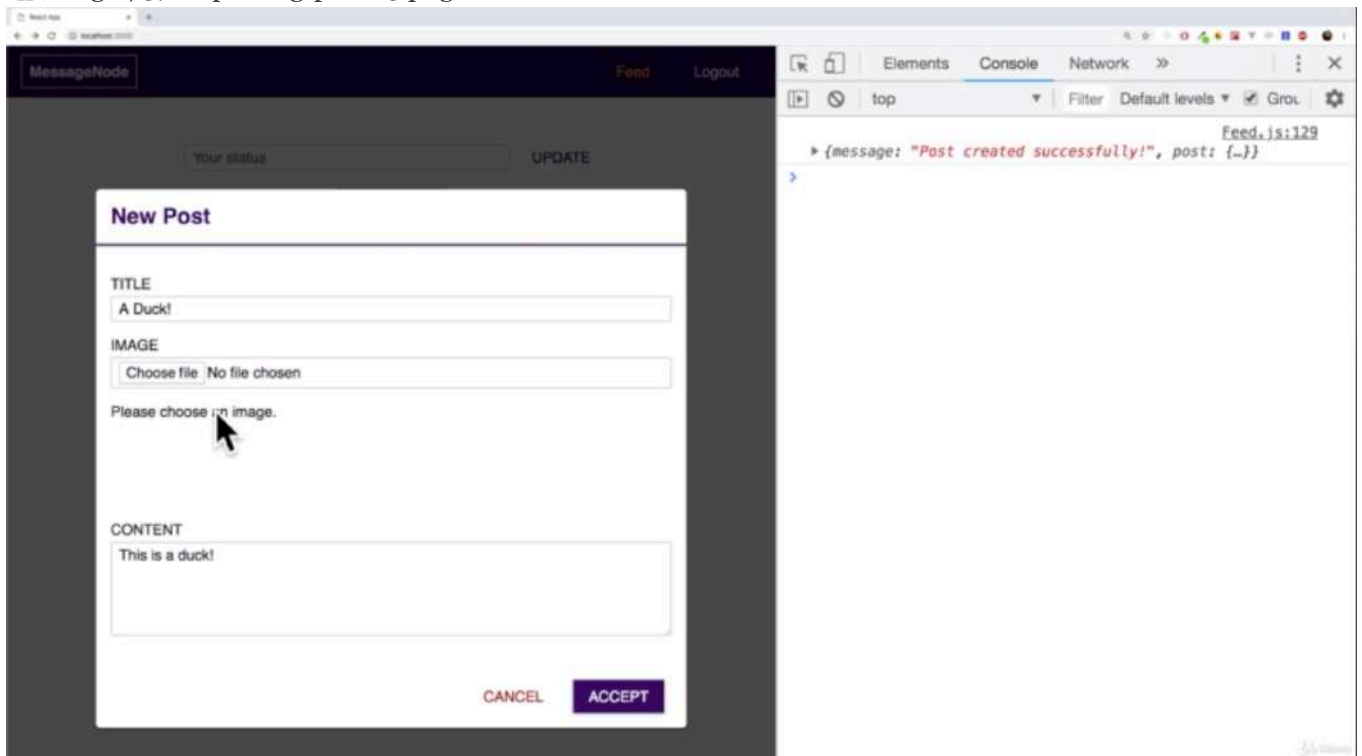
```
16 const postId = this.props.match.params.postId;
17 fetch('http://localhost:8080/feed/post/' + postId)
18   .then(res => {
19     if (res.status !== 200) {
20       throw new Error('Failed to fetch status.');
```

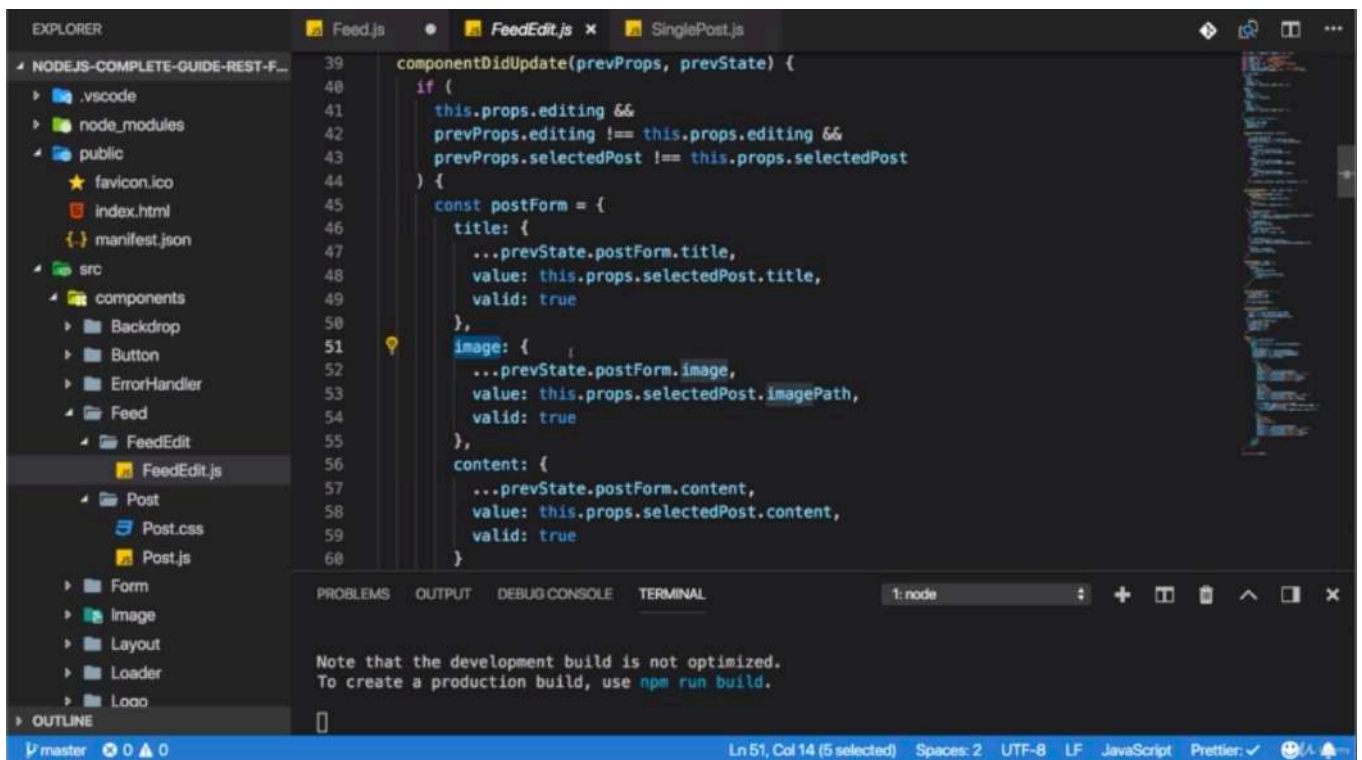
- because when i'm viewing a single post for example, i extract imageUrl and i append my URL to the domain like below in ./src/pages/Feed/SinglePost/SinglePost.js file,

image: 'http://localhost:8080/' + resData.post.imageUrl



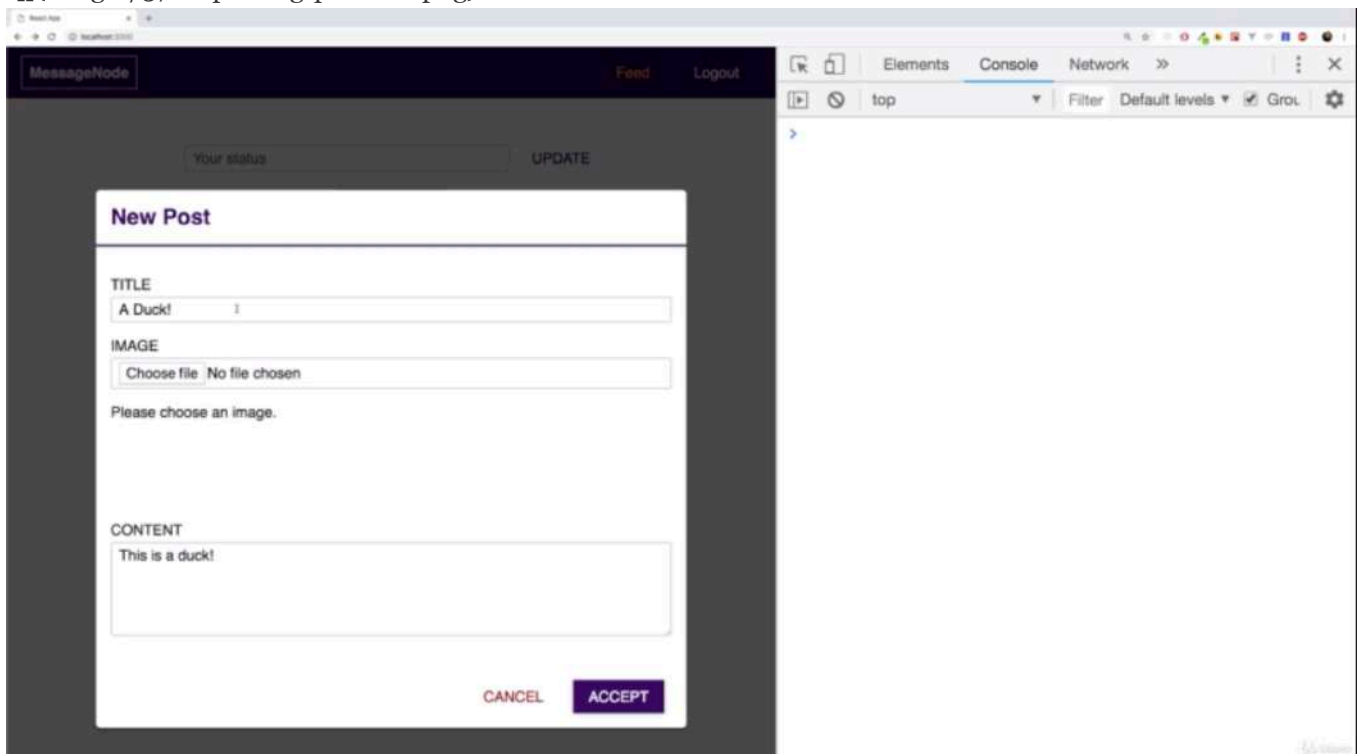
- now the path should be a path without a domain.

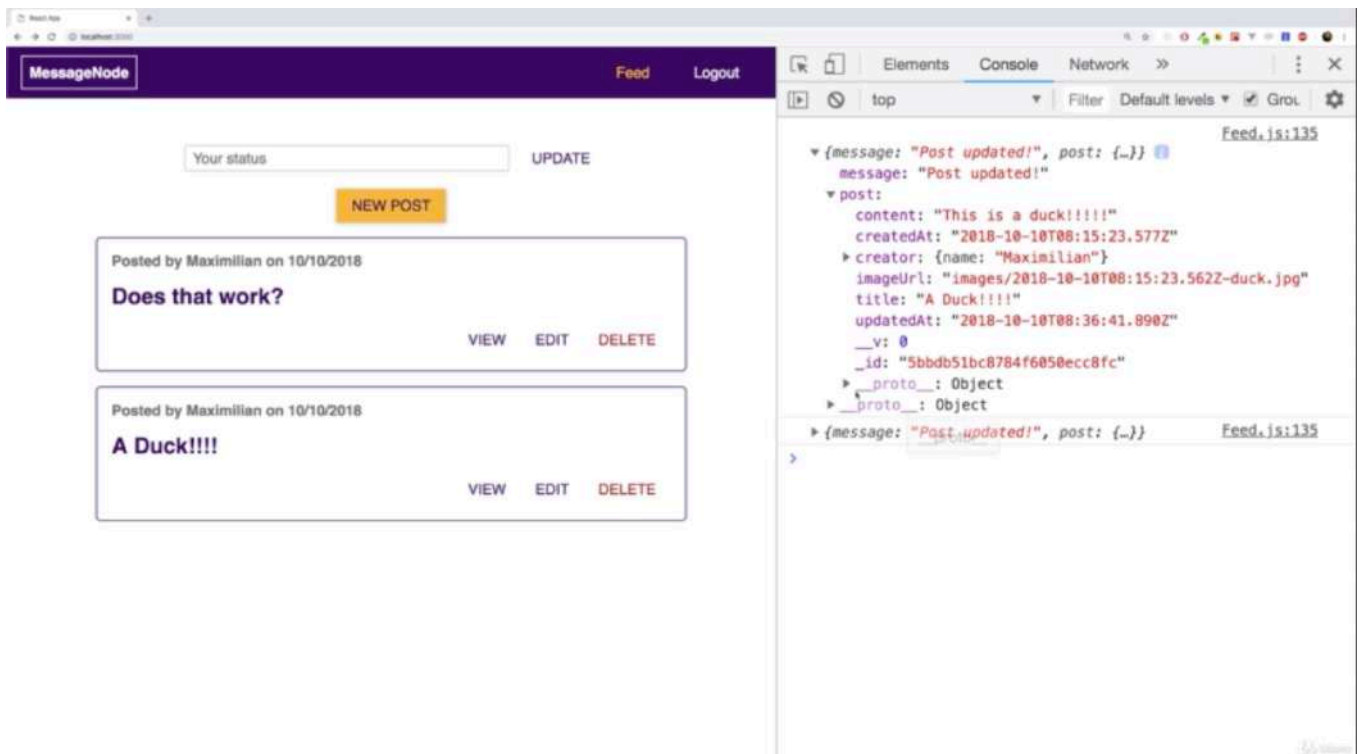
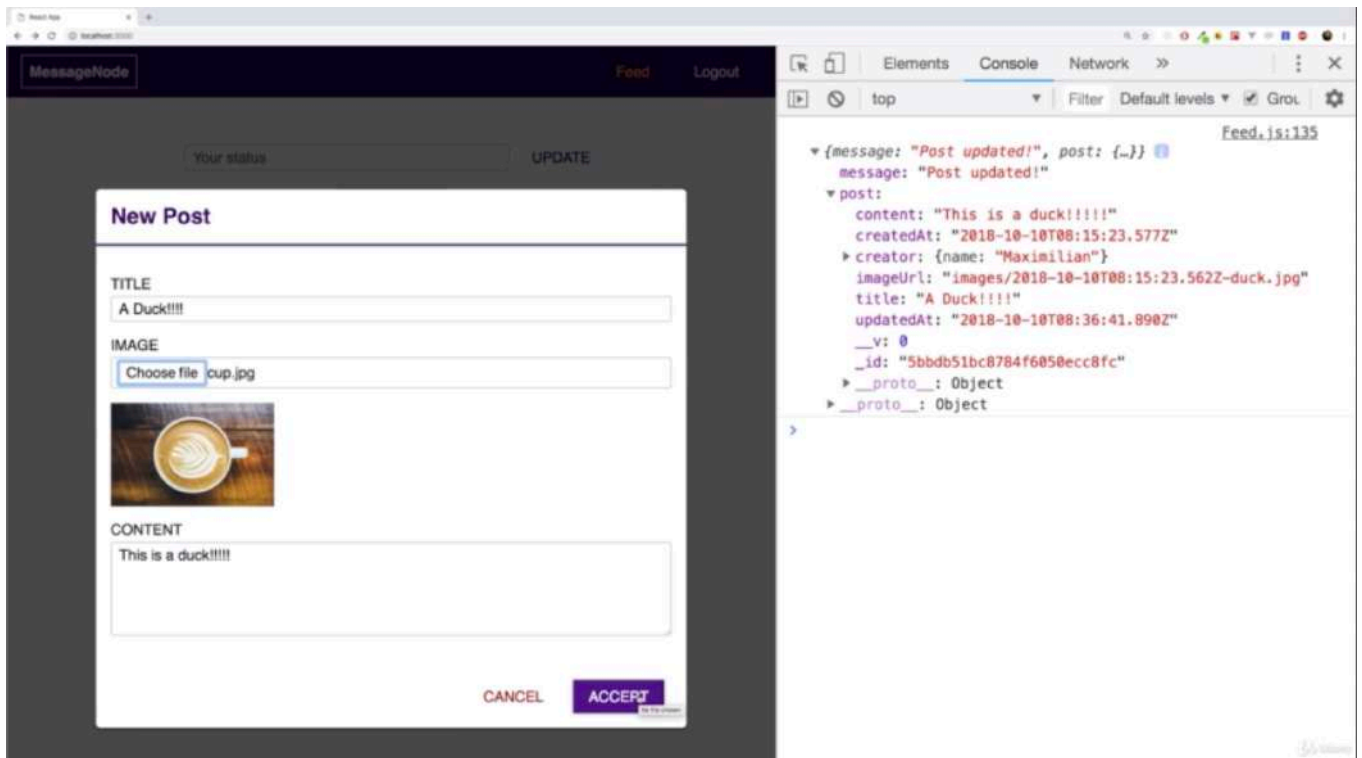


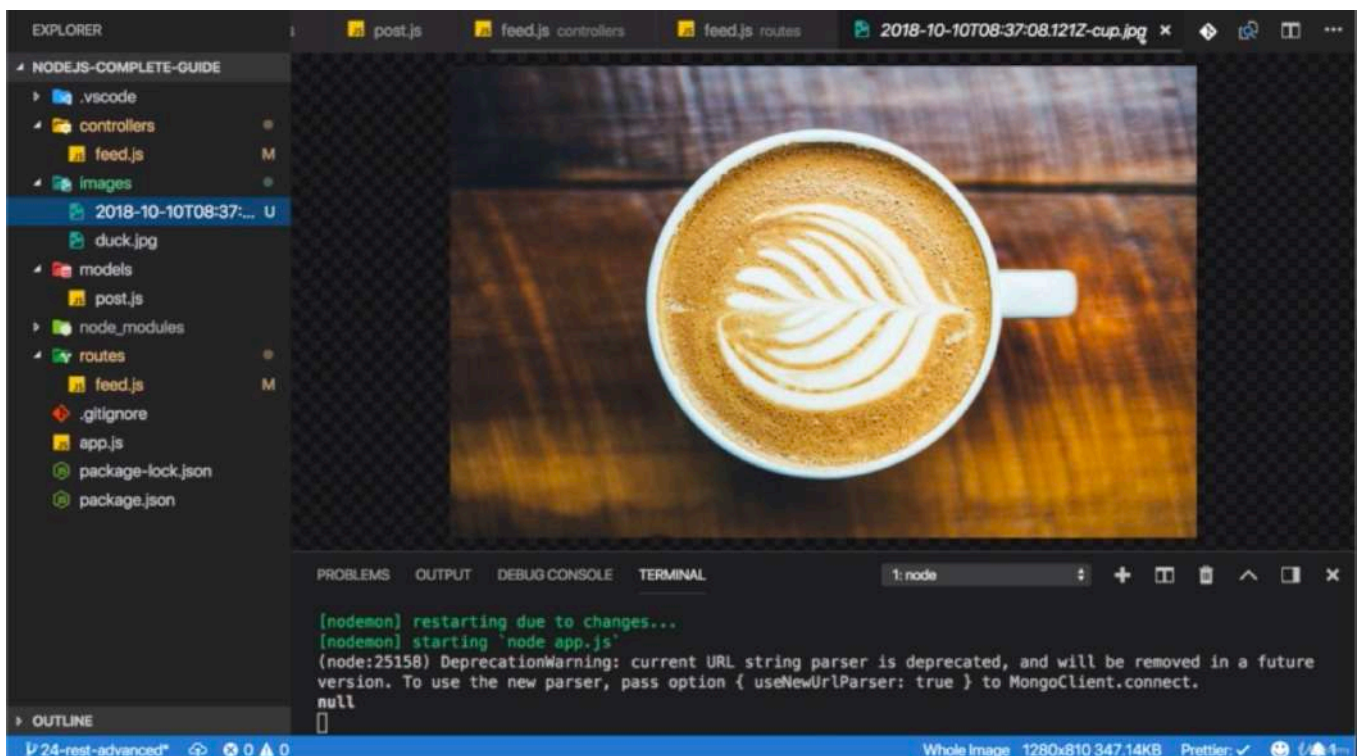
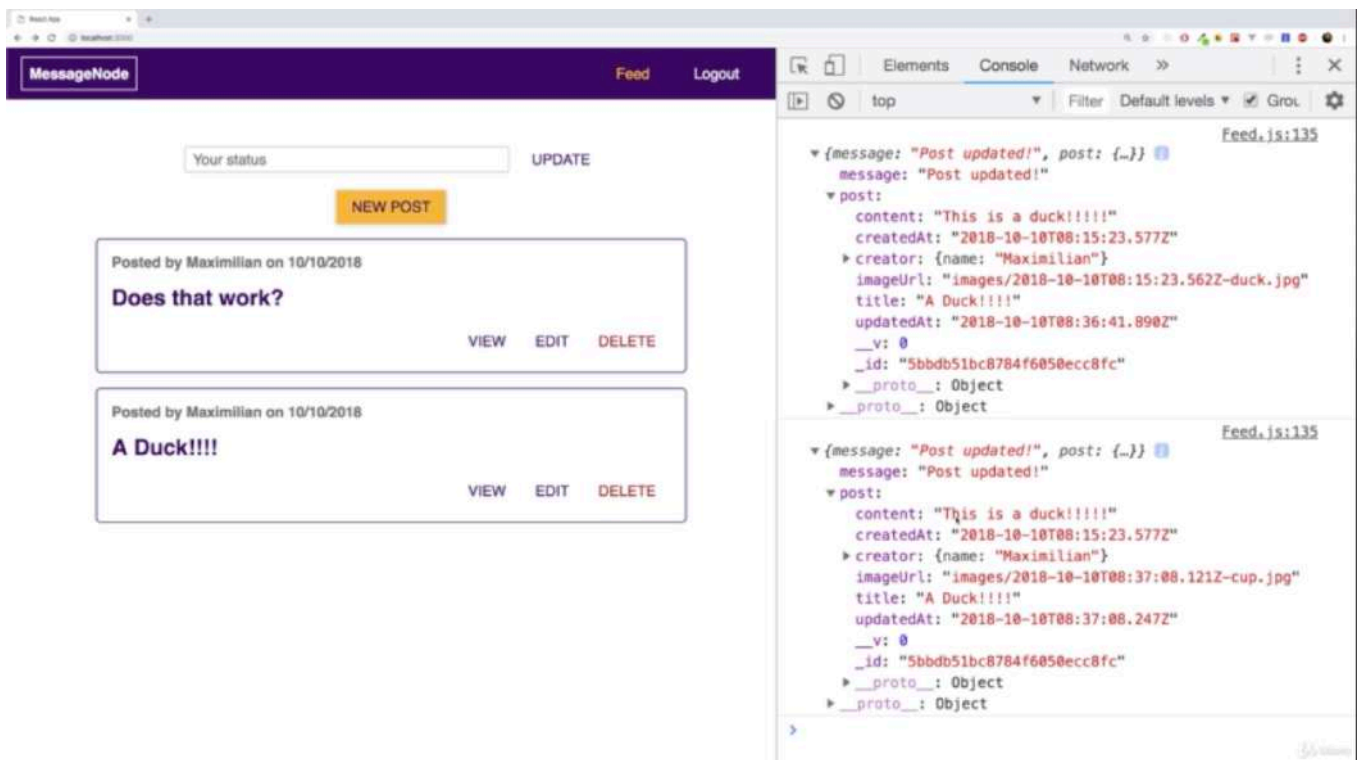


```
39 componentDidUpdate(prevProps, prevState) {
40
41   if (
42     this.props.editing &&
43     prevProps.editing !== this.props.editing &&
44     prevProps.selectedPost !== this.props.selectedPost
45   ) {
46     const postForm = {
47       title: {
48         ...prevState.postForm.title,
49         value: this.props.selectedPost.title,
50         valid: true
51       },
52       image: {
53         ...prevState.postForm.image,
54         value: this.props.selectedPost.imagePath,
55         valid: true
56       },
57       content: {
58         ...prevState.postForm.content,
59         value: this.props.selectedPost.content,
60         valid: true
61       }
62     };
63     this.setState({ postForm });
64   }
65 }
```

- because i'm keeping that here 'imagePath: post.imageUrl' and this will get reused later when we edit this through the edit model, behind the scene, this path will be stored and if i don't choose a new file here, this path will be submitted with my edited post and there, the path will be stored in a property named image which you can see in the ./src/components/Feed/FeedEdit/FeedEdit.js file







- if we edit it, changed. and on the backend, i have coffee mug now.

```

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

```

```

14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('URL')
28       .then(res => {
29         if (res.status !== 200) {
30           throw new Error('Failed to fetch user status.');
```

31 }
32 return res.json();
33 })
34 .then(resData => {
35 this.setState({ status: resData.status });
36 })
37 .catch(this.catchError);
38
39 this.loadPosts();
40 }
41
42 loadPosts = direction => {
43 if (direction) {
44 this.setState({ postsLoading: true, posts: [] });
45 }
46 let page = this.state.postPage;
47 if (direction === 'next') {
48 page++;
49 this.setState({ postPage: page });
50 }
51 if (direction === 'previous') {
52 page--;
53 this.setState({ postPage: page });
54 }
55 fetch('http://localhost:8080/feed/posts')
56 .then(res => {
57 if (res.status !== 200) {
58 throw new Error('Failed to fetch posts.');

59 }
60 return res.json();
61 })
62 .then(resData => {
63 this.setState({
64 posts: resData.posts.map(post => {
65 return {
66 ...post,
67 /**'imageUrl' is from ./models/post.js file in backend
68 * i'm storing the original path 'imagePath'
69 * because when i'm viewing a single post for example,

```

70      * i extract imageUrl and i append my URL to the domain like below
71      * in ./src/pages/Feed/SinglePost/SinglePost.js file,
72      *
73      *   image: 'http://localhost:8080/' + resData.post.imageUrl
74      *
75      * now the path should be a path without a domain.
76      * because i'm keeping that here 'imagePath: post.imageUrl'
77      * and this will get reused later when we edit through the edit model.
78      * behind the scene, this path will be stored
79      * and if i don't choose a new file here,
80      * this path will be submitted with my edited post
81      * and there, the path will be stored in a property named image
82      * which you can see in the
83      * ./src/components/Feed/FeedEdit/FeedEdit.js file
84      */
85      imagePath: post.imageUrl
86    };
87  }),
88  totalPosts: resData.totalItems,
89  postsLoading: false
90  });
91  })
92  .catch(this.catchError);
93  };
94
95  statusUpdateHandler = event => {
96    event.preventDefault();
97    fetch('URL')
98      .then(res => {
99      if (res.status !== 200 && res.status !== 201) {
100        throw new Error("Can't update status!");
101      }
102      return res.json();
103    })
104      .then(resData => {
105        console.log(resData);
106      })
107      .catch(this.catchError);
108  };
109
110  newPostHandler = () => {
111    this.setState({ isEditing: true });
112  };
113
114  startEditPostHandler = postId => {
115    this.setState(prevState => {
116      const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
117
118      return {
119        isEditing: true,
120        editPost: loadedPost
121      };
122    });
123  };
124
125  cancelEditHandler = () => {

```

```

126     this.setState({ isEditing: false, editPost: null });
127 };
128
129 finishEditHandler = postData => {
130     this.setState({
131         editLoading: true
132     });
133     const formData = new FormData()
134     formData.append('title', postData.title);
135     formData.append('content', postData.content);
136     formData.append('image', postData.image);
137     let url = 'http://localhost:8080/feed/post';
138     let method = 'POST';
139     if (this.state.editPost) {
140         url = 'http://localhost:8080/feed/post/' + this.state.editPost._id;
141         method = 'PUT';
142     }
143
144     fetch(url, {
145         method: method,
146         body: formData
147     })
148
149     .then(res => {
150         if (res.status !== 200 && res.status !== 201) {
151             throw new Error('Creating or editing a post failed!');
152         }
153         return res.json();
154     })
155     .then(resData => {
156         console.log(resData);
157         const post = {
158             _id: resData.post._id,
159             title: resData.post.title,
160             content: resData.post.content,
161             creator: resData.post.creator,
162             createdAt: resData.post.createdAt
163         };
164         this.setState(prevState => {
165             let updatedPosts = [...prevState.posts];
166             if (prevState.editPost) {
167                 const postIndex = prevState.posts.findIndex(
168                     p => p._id === prevState.editPost._id
169                 );
170                 updatedPosts[postIndex] = post;
171             } else if (prevState.posts.length < 2) {
172                 updatedPosts = prevState.posts.concat(post);
173             }
174             return {
175                 posts: updatedPosts,
176                 isEditing: false,
177                 editPost: null,
178                 editLoading: false
179             };
180         });
181     })

```

```

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

```



```

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

```

```

294 export default Feed;
295
1  //./routes/feed.js(b)
2
3  const express = require('express');
4  const { body } = require('express-validator/check');
5
6  const feedController = require('../controllers/feed');
7
8  const router = express.Router();
9
10 // GET /feed/posts
11 router.get('/posts', feedController.getPosts);
12
13 // POST /feed/post
14 router.post(
15   '/post',
16   [
17     body('title')
18       .trim()
19       .isLength({ min: 5 }),
20     body('content')
21       .trim()
22       .isLength({ min: 5 })
23   ],
24   feedController.createPost
25 );
26
27 router.get('/post/:postId', feedController.getPost);
28
29 /**i wanna be able to edit posts now
30  * and for that, i will use a new HTTP method
31  * editing a post is like replacing it,
32  * replacing the old post with a new one,
33  * we will keep the old Id but that is it.
34  * since we will replace a resource.
35  * i will use the put method
36  * with normal browser forms,
37  * you are not able to send it,
38  * through asynchronous requests triggered by javascript
39  * you are however.
40  *
41  * the important thing about PUT requests
42  * and the same is true for PATCH requests,
43  * is that they also have a request body, just like POST requests.
44  * request body will hold the actual post data i wanna use
45  * and that i wanna use to replace my existing post.
46  */
47
48 /**i will not validate my image
49  * because that is done directly in the controller action
50  * and i don't need to do any other validation for now.
51  */
52 router.put('/post/:postId', [
53   body('title')
54     .trim()

```

```

55     .isLength({ min: 5 }),
56     body('content')
57     .trim()
58     .isLength({ min: 5 })
59   ], feedController.updatePost
60 );
61
62 module.exports = router;
63

```

```

1  //./controllers/feed.js(b)
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9
10 exports.getPosts = (req, res, next) => {
11   Post.find()
12     .then(posts => {
13       res
14         .status(200)
15         .json({ message: 'Fetched posts successfully.', posts: posts });
16     })
17     .catch(err => {
18       if (!err.statusCode) {
19         err.statusCode = 500;
20       }
21       next(err);
22     });
23 };
24
25 exports.createPost = (req, res, next) => {
26   const errors = validationResult(req);
27   if (!errors.isEmpty()) {
28     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

throw error;

}

if(!req.file){

const error = new Error('No image provided.');

error.statusCode = 422;

throw error;

}

/**there the path variable which multer generates
 * which holds the path to the file as it was stored on my server.
 */

```

40 const imageUrl = req.file.path;
41 const title = req.body.title;
42 const content = req.body.content;
43 const post = new Post({
44   title: title,
45   content: content,
46   imageUrl: imageUrl,
47   creator: { name: 'Maximilian' }

```

```

48 });
49 post
50   .save()
51   .then(result => {
52     res.status(201).json({
53       message: 'Post created successfully!',
54       post: result
55     });
56   })
57   .catch(err => {
58     if (!err.statusCode) {
59       err.statusCode = 500;
60     }
61     next(err);
62   });
63 };
64
65 exports.getPost = (req, res, next) => {
66   const postId = req.params.postId;
67   Post.findById(postId)
68     .then(post => {
69       if (!post) {
70         const error = new Error('Could not find post.');
```

* and it's just some text in the req.body
 * that would be the case if no new file was added.
 * if no new file was picked.
 * then my frontend code has all the logic to take the existing URL
 * and keep that.
 *
 * but we might have picked a file
 * and in this case, request file will be set
 * and i can set imageUrl equal to req.file.path

```

71         error.statusCode = 404;
72         throw error;
73       }
74       res.status(200).json({ message: 'Post fetched.', post: post });
75     })
76     .catch(err => {
77       if (!err.statusCode) {
78         err.statusCode = 500;
79       }
80       next(err);
81     });
82 };
83
84 exports.updatePost = (req, res, next) => {
85   const postId = req.params.postId;
86   const errors = validationResult(req);
87   if (!errors.isEmpty()) {
88     const error = new Error('Validation failed, entered data is incorrect.');
```

```

104 * this is alternative.
105 *
106 * after all, at least one of the 2 should be set.
107 */
108
109 /**'image' from frontend. */
110 let imageUrl = req.body.image;
111 if(req.file) {
112     imageUrl = req.file.path
113 }
114 if(!imageUrl) {
115     const error = new Error('No file picked. ');
116     error.statusCode = 422;
117     throw error;
118 }
119 Post.findById(postId)
120     .then(post => {
121         if (!post) {
122             const error = new Error('Could not find post. ');
123             error.statusCode = 404;
124             throw error;
125         }
126         if(imageUrl !== post.image) {
127             clearImage(post.imageUrl);
128         }
129         post.title = title;
130         post.imageUrl = imageUrl;
131         post.content = content;
132         return post.save();
133     })
134     .then(result => {
135         res.status(200).json({message: 'Post updated!', post: result});
136     })
137     .catch(err => {
138         if (!err.statusCode) {
139             err.statusCode = 500;
140         }
141         next(err);
142     })
143 };
144
145 const clearImage = filePath => {
146     /**we should go up one folder to be in the root folder
147     * and then we look for whatever file path we got here
148     * images and then the image name would be the case in our application
149     *
150     * and i can then use 'unlink' function to delete that file
151     * by passing the file path to it
152     * and we can also log any error message
153     *
154     * i wanna trigger that clearImage function
155     * when i uploaded a new image
156     *
157     */
158     filePath = path.join(__dirname, '..', filePath)
159     fs.unlink(filePath, err => console.log(err));

```

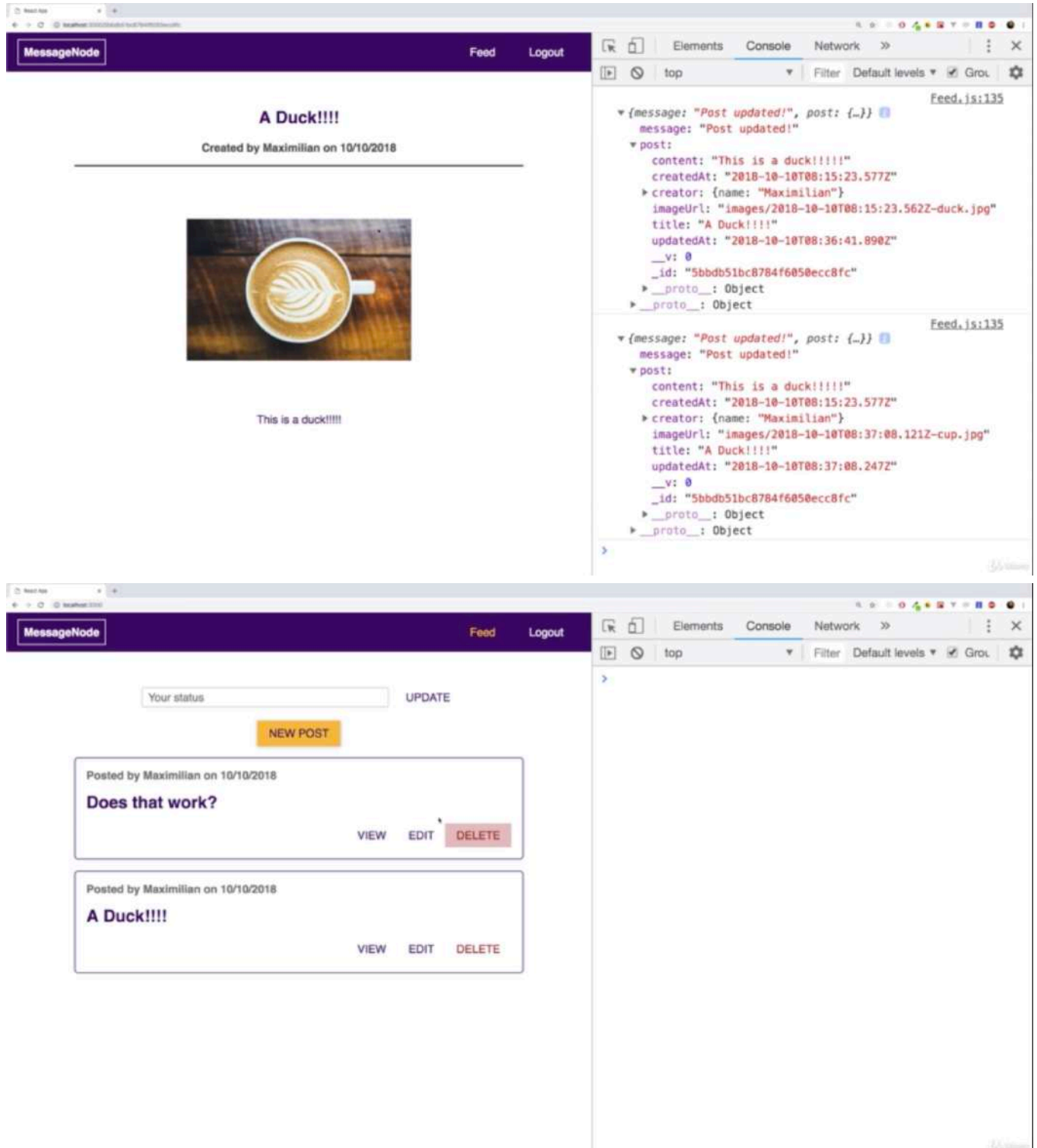
160 }

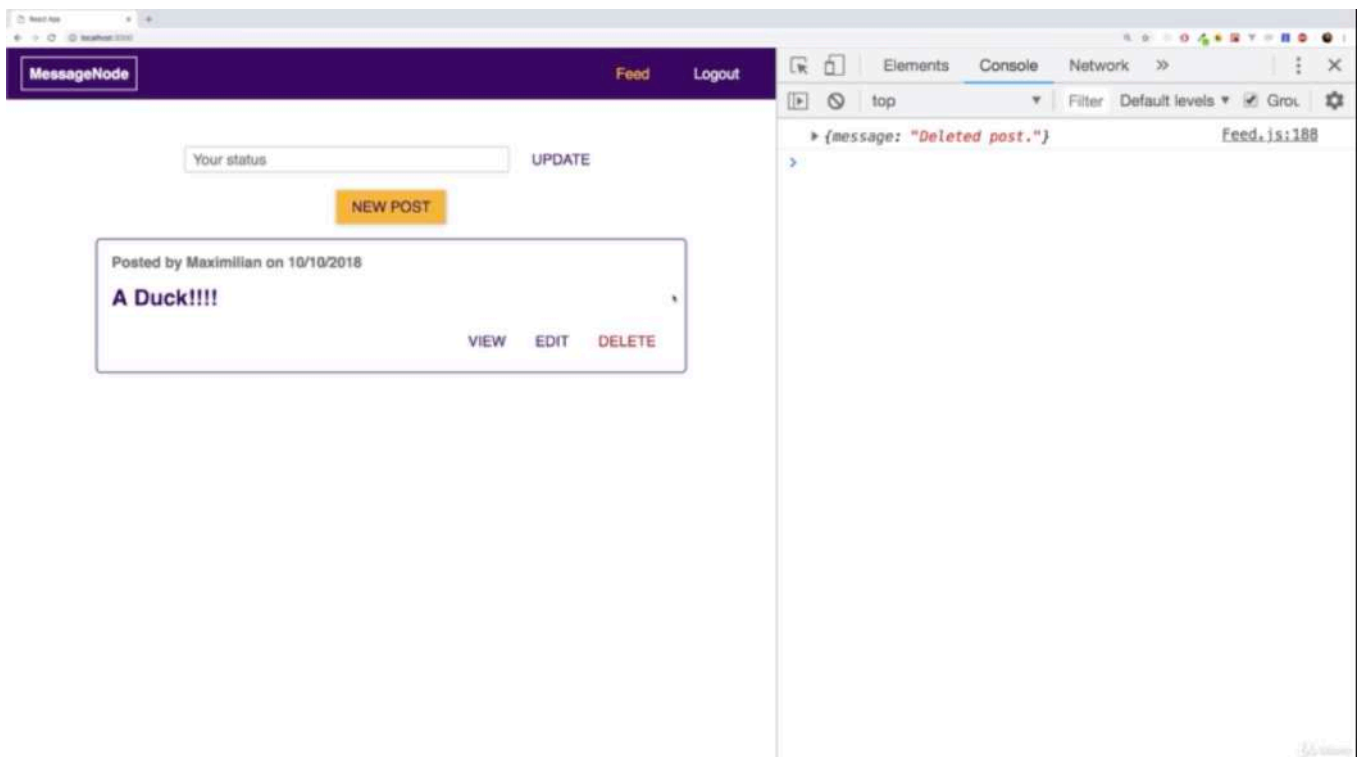
161

* Chapter 379: Deleting Posts

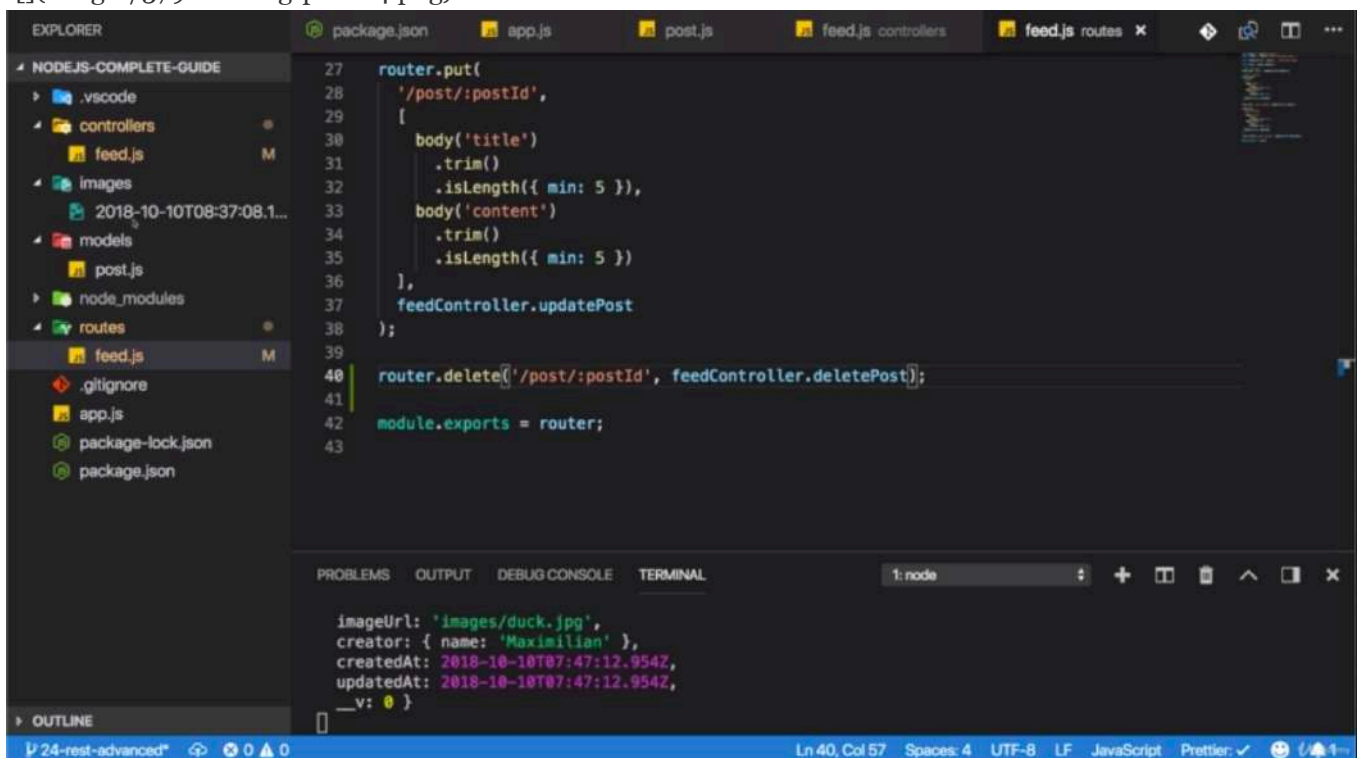
1. update

- ./routes/feed.js(b)
- ./controllers/feed.js(b)
- ./src/pages/Feed/Feed.js(f)





- the picture duck is gone.



```

1  //../controllers/feed.js(b)
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9
10 exports.getPosts = (req, res, next) => {
11   Post.find()
12     .then(posts => {

```



```

13     res
14     .status(200)
15     .json({ message: 'Fetched posts successfully.', posts: posts });
16   })
17   .catch(err => {
18     if (!err.statusCode) {
19       err.statusCode = 500;
20     }
21     next(err);
22   });
23 };
24
25 exports.createPost = (req, res, next) => {
26   const errors = validationResult(req);
27   if (!errors.isEmpty()) {
28     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

throw error;

```

31   }
32   if(!req.file){
33     const error = new Error('No image provided.');
```

error.statusCode = 422;

throw error;

```

36   }
37   /**there the path variable which multer generates
38    * which holds the path to the file as it was stored on my server.
39    */
40   const imageUrl = req.file.path;
41   const title = req.body.title;
42   const content = req.body.content;
43   const post = new Post({
44     title: title,
45     content: content,
46     imageUrl: imageUrl,
47     creator: { name: 'Maximilian' }
48   });
49   post
50     .save()
51     .then(result => {
52       res.status(201).json({
53         message: 'Post created successfully!',
54         post: result
55       });
56     })
57     .catch(err => {
58       if (!err.statusCode) {
59         err.statusCode = 500;
60       }
61       next(err);
62     });
63 };
64
65 exports.getPost = (req, res, next) => {
66   const postId = req.params.postId;
67   Post.findById(postId)
68     .then(post => {

```

```

69     if (!post) {
70         const error = new Error('Could not find post.');
```

71 error.statusCode = 404;
72 throw error;
73 }
74 res.status(200).json({ message: 'Post fetched.', post: post });
75 })
76 .catch(err => {
77 if (!err.statusCode) {
78 err.statusCode = 500;
79 }
80 next(err);
81 });
82 };
83
84 exports.updatePost = (req, res, next) => {
85 const postId = req.params.postId;
86 const errors = validationResult(req);
87 if (!errors.isEmpty()) {
88 const error = new Error('Validation failed, entered data is incorrect.');

89 error.statusCode = 422;
90 throw error;
91 }
92 const title = req.body.title;
93 const content = req.body.content;
94 let imageUrl = req.body.image;
95 if(req.file) {
96 imageUrl = req.file.path
97 }
98 if(!imageUrl) {
99 const error = new Error('No file picked.');

100 error.statusCode = 422;
101 throw error;
102 }
103 Post.findById(postId)
104 .then(post => {
105 if (!post) {
106 const error = new Error('Could not find post.');

107 error.statusCode = 404;
108 throw error;
109 }
110 if(imageUrl !== post.image) {
111 clearImage(post.imageUrl);
112 }
113 post.title = title;
114 post.imageUrl = imageUrl;
115 post.content = content;
116 return post.save();
117 })
118 .then(result => {
119 res.status(200).json({message: 'Post updated!', post: result});
120 })
121 .catch(err => {
122 if (!err.statusCode) {
123 err.statusCode = 500;
124 }

```

125     next(err);
126   })
127 };
128
129 exports.deletePost = (req, res, next) => {
130   const postId = req.params.postId;
131   /**why would i find it by Id first?
132    * i can also use findByIdAndRemove
133    * i will use this eventually
134    * but later we edit a user,
135    * i wanna verify whether that user created the post before i delete it.
136    */
137   Post.findById(postId)
138     .then(post => {
139     if (!post) {
140       const error = new Error('Could not find post. ');
141       error.statusCode = 404;
142       throw error;
143     }
144     //Check logged in user
145     clearImage(post.imageUrl);
146     return Post.findByIdAndRemove(postId);
147   })
148     .then(result => {
149     console.log(result);
150     res.status(200).json({message: 'Deleted post.'});
151   })
152     .catch(err => {
153     if (!err.statusCode) {
154       err.statusCode = 500;
155     }
156     next(err);
157   });
158 }
159
160 const clearImage = filePath => {
161   filePath = path.join(__dirname, '..', filePath)
162   fs.unlink(filePath, err => console.log(err));
163 }
164

```

```

1  //./routes/feed.js(b)
2
3  const express = require('express');
4  const { body } = require('express-validator/check');
5
6  const feedController = require('../controllers/feed');
7
8  const router = express.Router();
9
10 // GET /feed/posts
11 router.get('/posts', feedController.getPosts);
12
13 // POST /feed/post
14 router.post(
15   '/post',
16   [

```

```

17     body('title')
18         .trim()
19         .isLength({ min: 5 }),
20     body('content')
21         .trim()
22         .isLength({ min: 5 })
23 ],
24     feedController.createPost
25 );
26
27 router.get('/post/:postId', feedController.getPost);
28
29 /**i wanna be able to edit posts now
30 * and for that, i will use a new HTTP method
31 * editing a post is like replacing it,
32 * replacing the old post with a new one,
33 * we will keep the old Id but that is it.
34 * since we will replace a resource.
35 * i will use the put method
36 * with normal browser forms,
37 * you are not able to send it,
38 * through asynchronous requests triggered by javascript
39 * you are however.
40 *
41 * the important thing about PUT requests
42 * and the same is true for PATCH requests,
43 * is that they also have a request body, just like POST requests.
44 * request body will hold the actual post data i wanna use
45 * and that i wanna use to replace my existing post.
46 */
47
48 /**i will not validate my image
49 * because that is done directly in the controller action
50 * and i don't need to do any other validation for now.
51 */
52 router.put('/post/:postId', [
53     body('title')
54         .trim()
55         .isLength({ min: 5 }),
56     body('content')
57         .trim()
58         .isLength({ min: 5 })
59 ], feedController.updatePost
60 );
61
62 router.delete('/post/:postId', feedController.deletePost);
63
64 module.exports = router;
65

```

```

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

```

```

8 import Input from '../components/Form/Input/Input';
9 import Paginator from '../components/Paginator/Paginator';
10 import Loader from '../components/Loader/Loader';
11 import ErrorHandler from '../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('URL')
28       .then(res => {
29         if (res.status !== 200) {
30           throw new Error('Failed to fetch user status.');

```

```

64     posts: resData.posts.map(post => {
65         return {
66             ...post,
67             imagePath: post.imageUrl
68         };
69     }),
70     totalPosts: resData.totalItems,
71     postsLoading: false
72 });
73 })
74 .catch(this.catchError);
75 };
76
77 statusUpdateHandler = event => {
78     event.preventDefault();
79     fetch('URL')
80     .then(res => {
81         if (res.status !== 200 && res.status !== 201) {
82             throw new Error("Can't update status!");
83         }
84         return res.json();
85     })
86     .then(resData => {
87         console.log(resData);
88     })
89     .catch(this.catchError);
90 };
91
92 newPostHandler = () => {
93     this.setState({ isEditing: true });
94 };
95
96 startEditPostHandler = postId => {
97     this.setState(prevState => {
98         const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
99
100     return {
101         isEditing: true,
102         editPost: loadedPost
103     };
104 });
105 };
106
107 cancelEditHandler = () => {
108     this.setState({ isEditing: false, editPost: null });
109 };
110
111 finishEditHandler = postData => {
112     this.setState({
113         editLoading: true
114     });
115     const formData = new FormData()
116     formData.append('title', postData.title);
117     formData.append('content', postData.content);
118     formData.append('image', postData.image);
119     let url = 'http://localhost:8080/feed/post';

```

```

120 let method = 'POST';
121 if (this.state.editPost) {
122   url = 'http://localhost:8080/feed/post/' + this.state.editPost._id;
123   method = 'PUT';
124 }
125
126 fetch(url, {
127   method: method,
128   body: formData
129 })
130
131 .then(res => {
132   if (res.status !== 200 && res.status !== 201) {
133     throw new Error('Creating or editing a post failed!');
134   }
135   return res.json();
136 })
137 .then(resData => {
138   console.log(resData);
139   const post = {
140     _id: resData.post._id,
141     title: resData.post.title,
142     content: resData.post.content,
143     creator: resData.post.creator,
144     createdAt: resData.post.createdAt
145   };
146   this.setState(prevState => {
147     let updatedPosts = [...prevState.posts];
148     if (prevState.editPost) {
149       const postIndex = prevState.posts.findIndex(
150         p => p._id === prevState.editPost._id
151       );
152       updatedPosts[postIndex] = post;
153     } else if (prevState.posts.length < 2) {
154       updatedPosts = prevState.posts.concat(post);
155     }
156     return {
157       posts: updatedPosts,
158       isEditing: false,
159       editPost: null,
160       editLoading: false
161     };
162   });
163 })
164 .catch(err => {
165   console.log(err);
166   this.setState({
167     isEditing: false,
168     editPost: null,
169     editLoading: false,
170     error: err
171   });
172 });
173 };
174
175 statusInputChangeHandler = (input, value) => {

```



```

176     this.setState({ status: value });
177 };
178
179 deletePostHandler = postId => {
180     this.setState({ postsLoading: true });
181     fetch('http://localhost:8080/feed/post/' + postId, {
182         method: 'DELETE'
183     })
184     .then(res => {
185         if (res.status !== 200 && res.status !== 201) {
186             throw new Error('Deleting a post failed!');
187         }
188         return res.json();
189     })
190     .then(resData => {
191         console.log(resData);
192         this.setState(prevState => {
193             const updatedPosts = prevState.posts.filter(p => p._id !== postId);
194             return { posts: updatedPosts, postsLoading: false };
195         });
196     })
197     .catch(err => {
198         console.log(err);
199         this.setState({ postsLoading: false });
200     });
201 };
202
203 errorHandler = () => {
204     this.setState({ error: null });
205 };
206
207 catchError = error => {
208     this.setState({ error: error });
209 };
210
211 render() {
212     return (
213         <Fragment>
214             <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
215             <FeedEdit
216                 editing={this.state.isEditing}
217                 selectedPost={this.state.editPost}
218                 loading={this.state.editLoading}
219                 onCancelEdit={this.cancelEditHandler}
220                 onFinishEdit={this.finishEditHandler}
221             />
222             <section className="feed__status">
223                 <form onSubmit={this.statusUpdateHandler}>
224                     <Input
225                         type="text"
226                         placeholder="Your status"
227                         control="input"
228                         onChange={this.statusInputChangeHandler}
229                         value={this.state.status}
230                     />
231                     <Button mode="flat" type="submit">

```

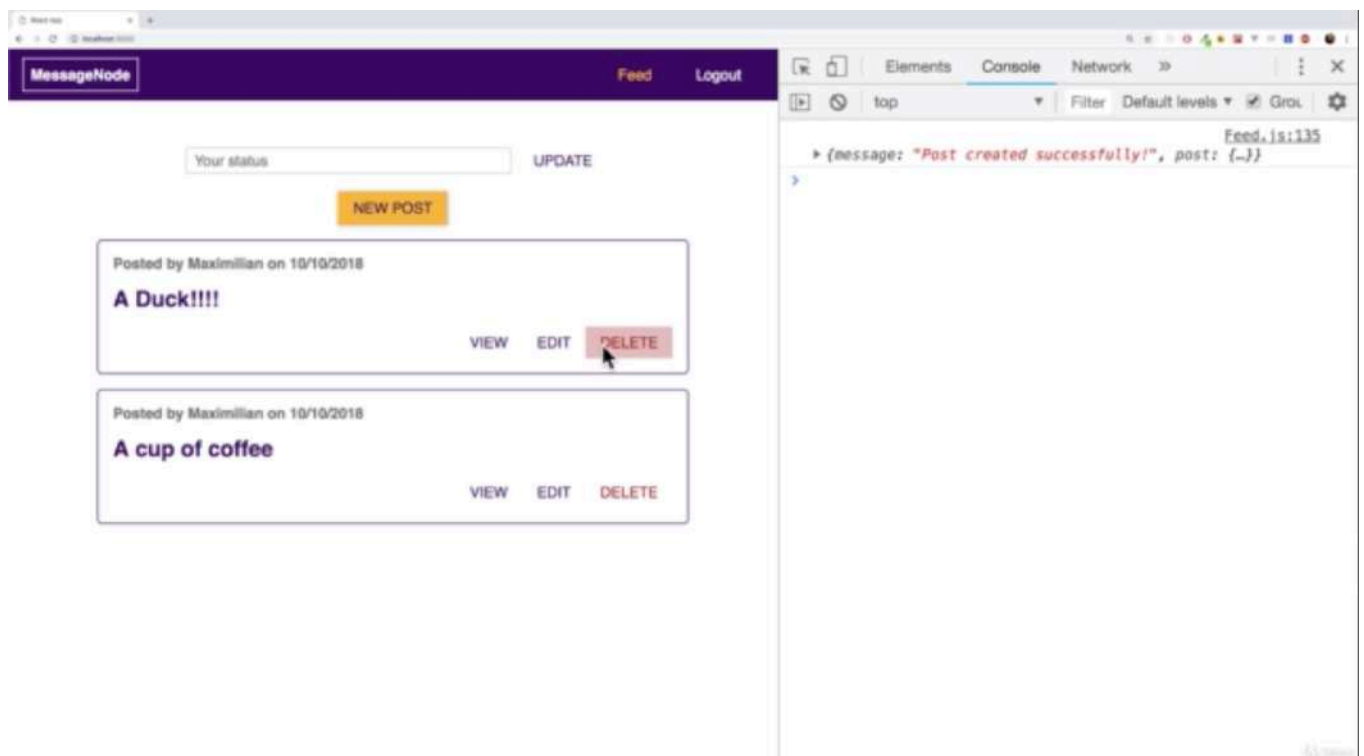
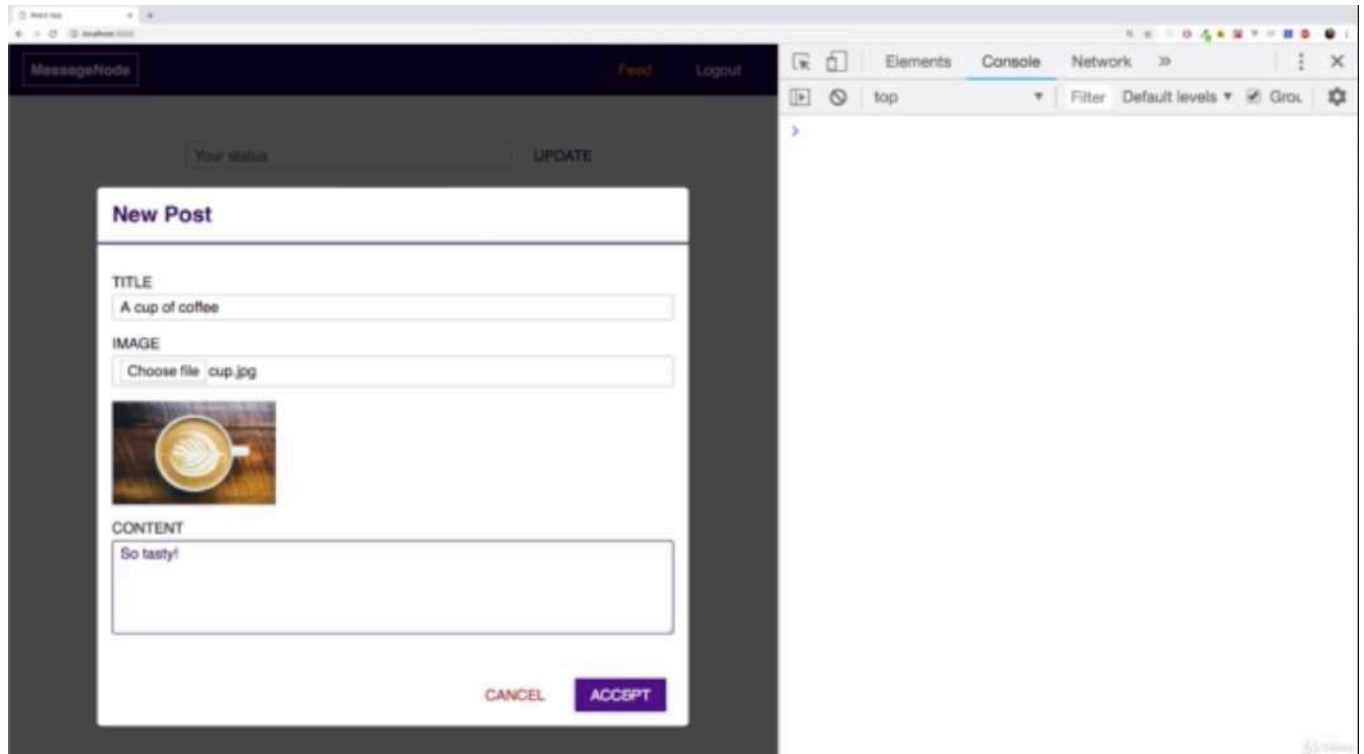
```

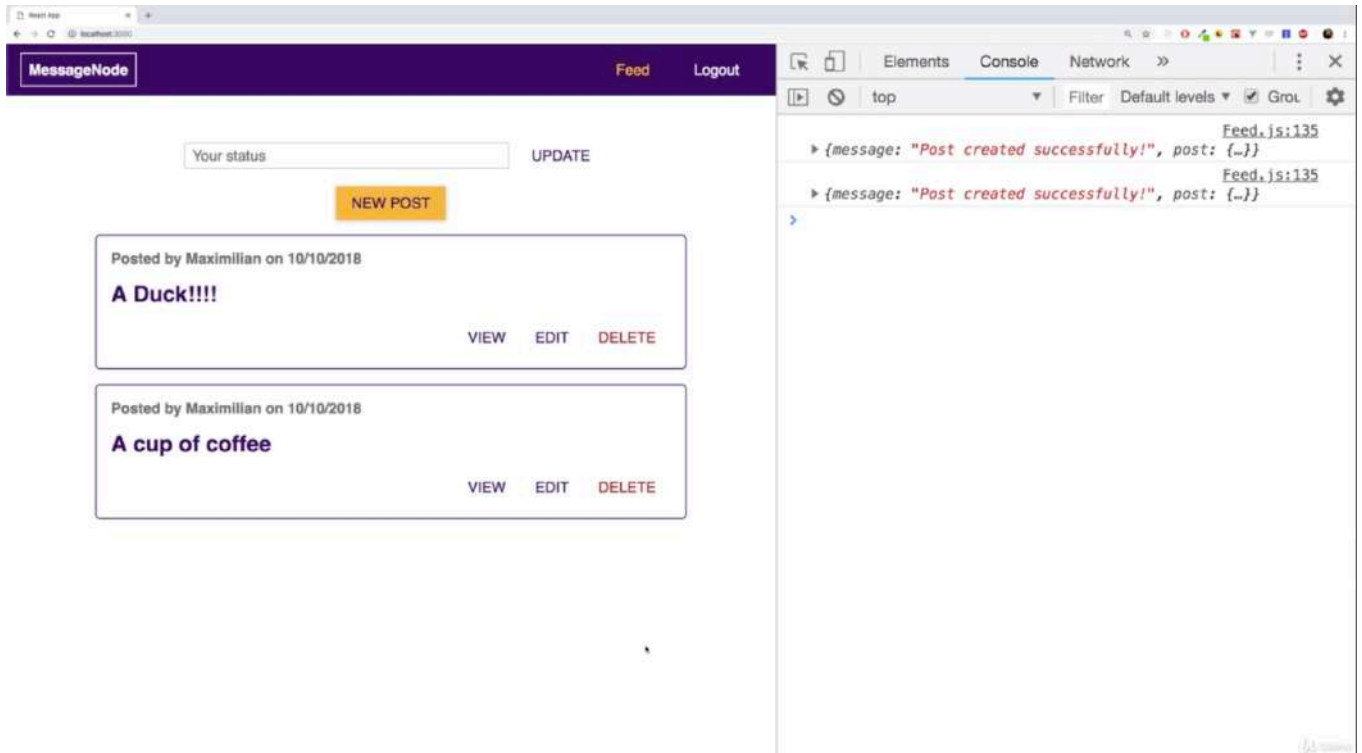
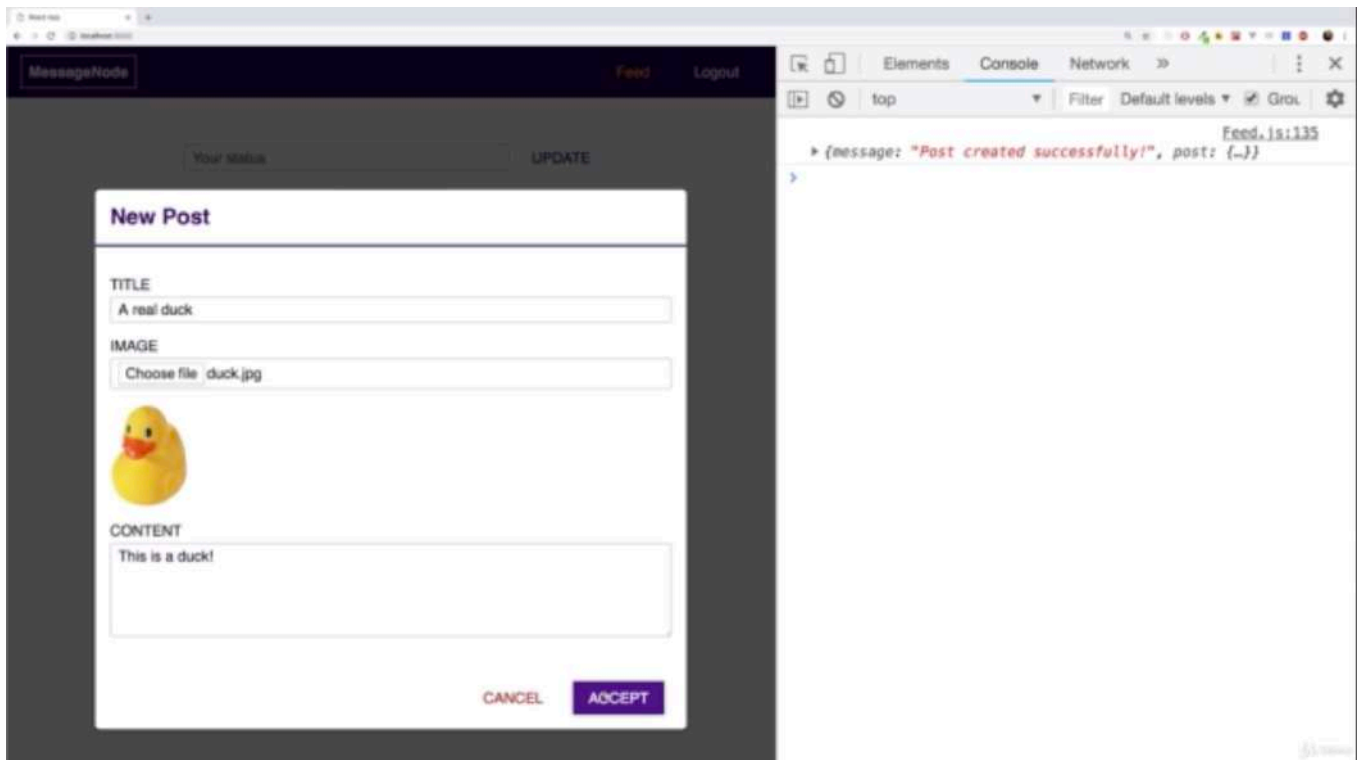
232         Update
233     </Button>
234 </form>
235 </section>
236 <section className="feed__control">
237     <Button mode="raised" design="accent" onClick={this.newPostHandler}>
238         New Post
239     </Button>
240 </section>
241 <section className="feed">
242     {this.state.postsLoading && (
243         <div style={{ textAlign: 'center', marginTop: '2rem' }}>
244             <Loader />
245         </div>
246     )}
247     {this.state.posts.length <= 0 && !this.state.postsLoading ? (
248         <p style={{ textAlign: 'center' }}>No posts found.</p>
249     ) : null}
250     {!this.state.postsLoading && (
251         <Paginator
252             onPrevious={this.loadPosts.bind(this, 'previous')}
253             onNext={this.loadPosts.bind(this, 'next')}
254             lastPage={Math.ceil(this.state.totalPosts / 2)}
255             currentPage={this.state.postPage}
256         >
257             {this.state.posts.map(post => (
258                 <Post
259                     key={post._id}
260                     id={post._id}
261                     author={post.creator}
262                     date={new Date(post.createdAt).toLocaleDateString('en-US')}
263                     title={post.title}
264                     image={post.imageUrl}
265                     content={post.content}
266                     onStartEdit={this.startEditPostHandler.bind(this, post._id)}
267                     onDelete={this.deletePostHandler.bind(this, post._id)}
268                 />
269             ))}
270         </Paginator>
271     )}
272 </section>
273 </Fragment>
274 );
275 }
276 }
277
278 export default Feed;

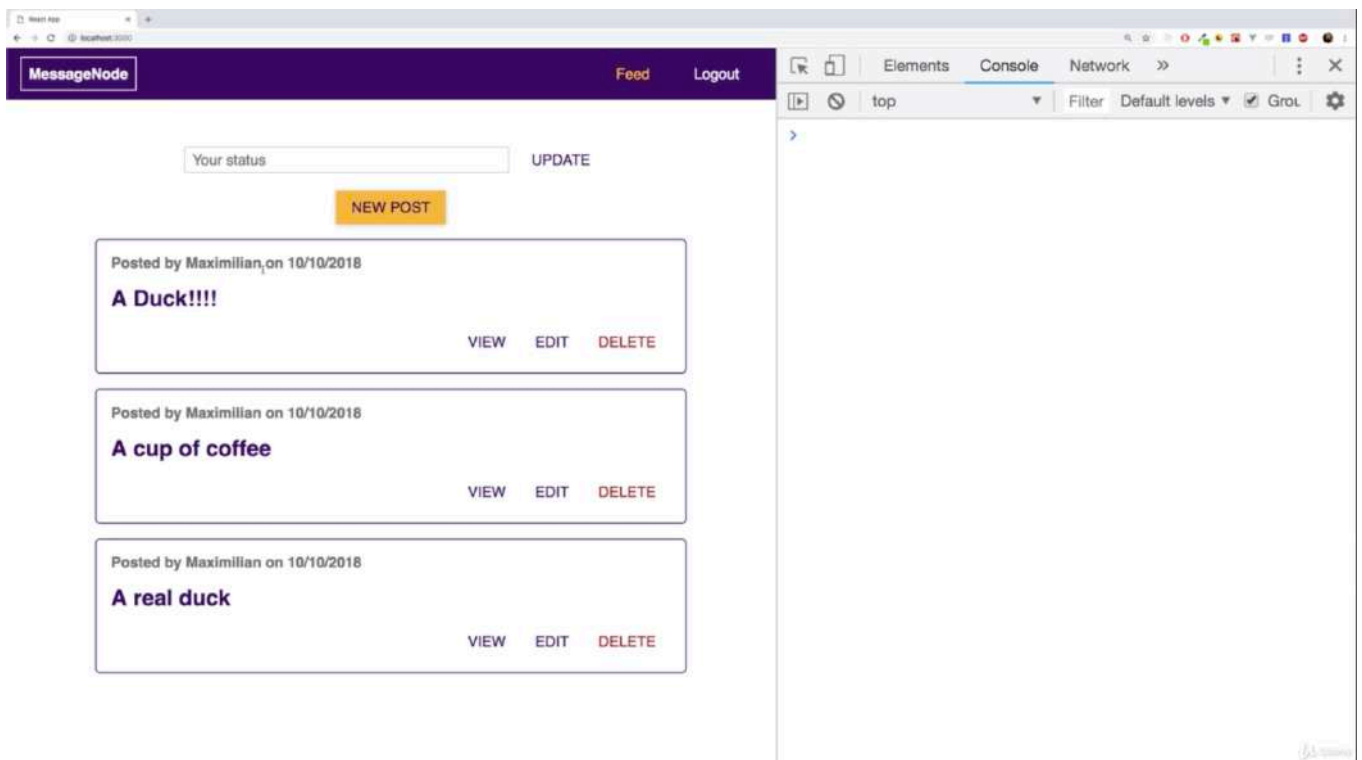
```

* Chapter 380: Adding Pagination

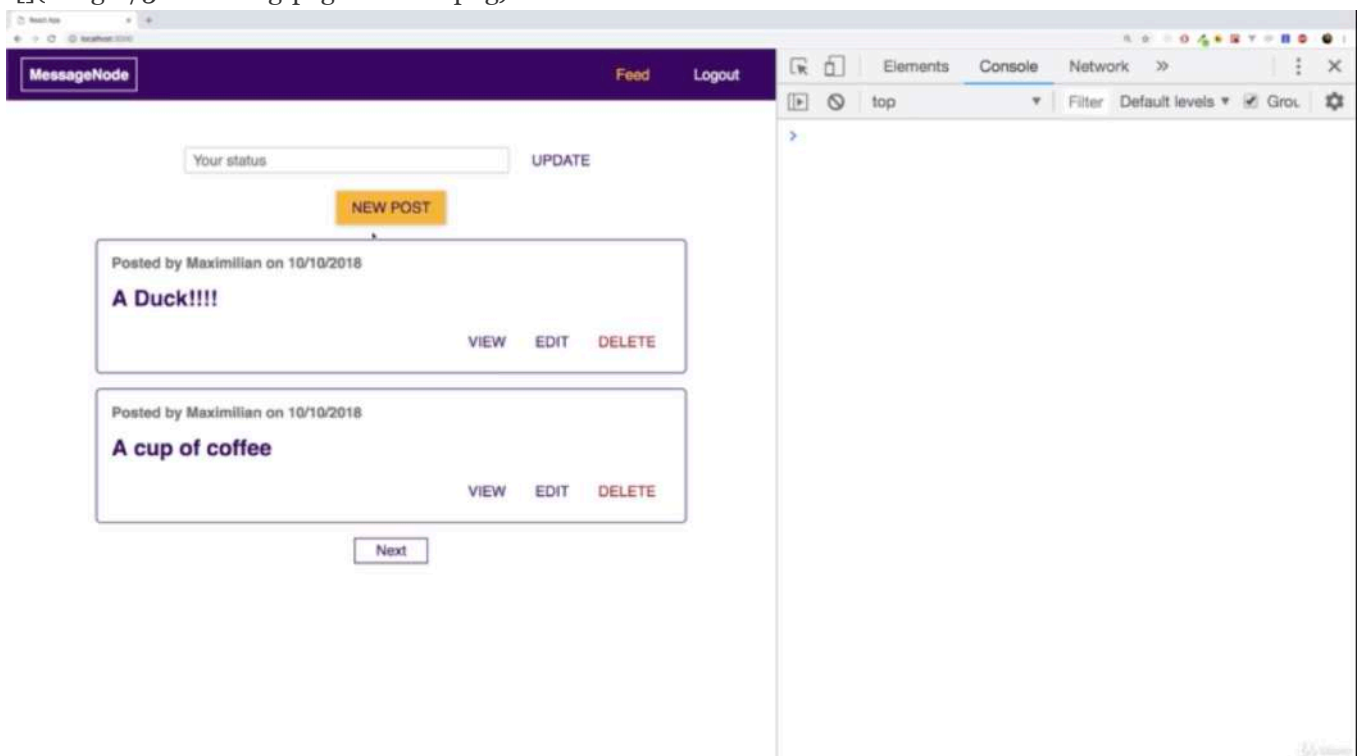
1. update
 - ./src/pages/Feed/Feed.js(f)
 - ./controllers/feed.js(b)

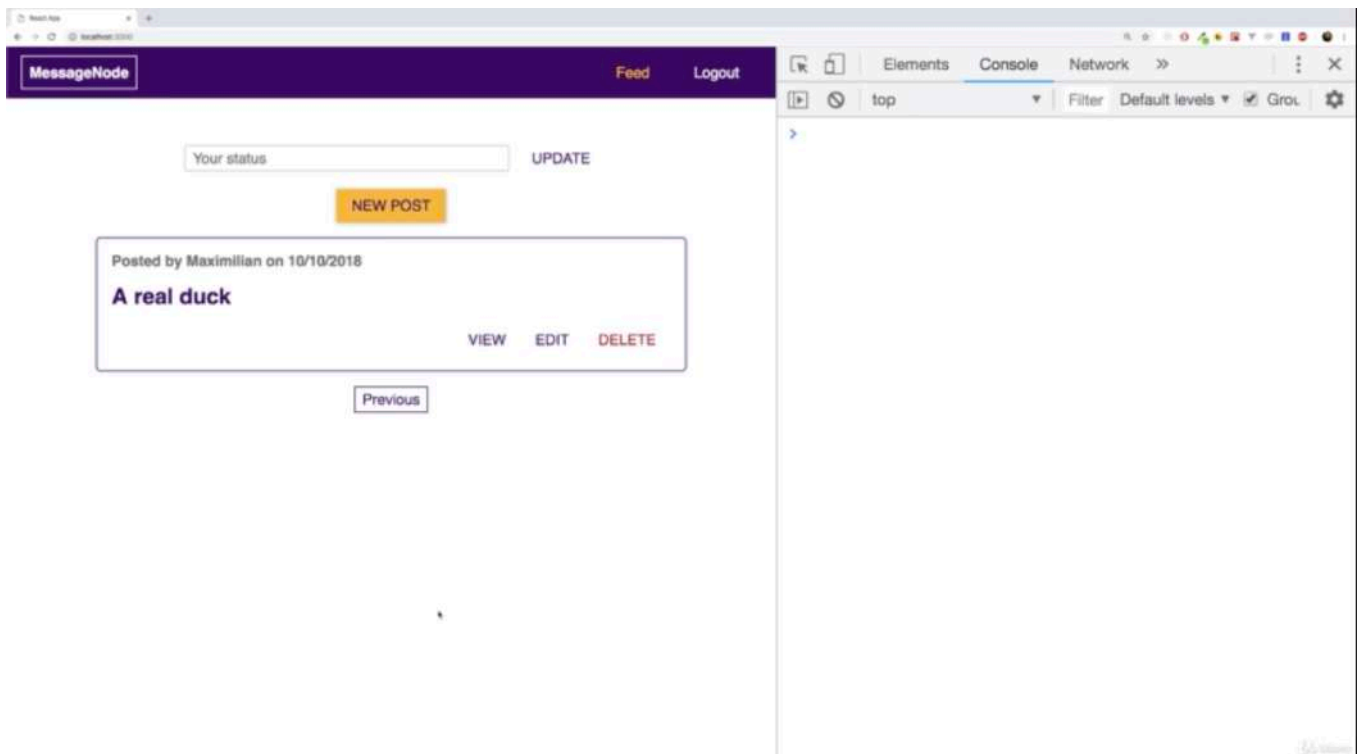
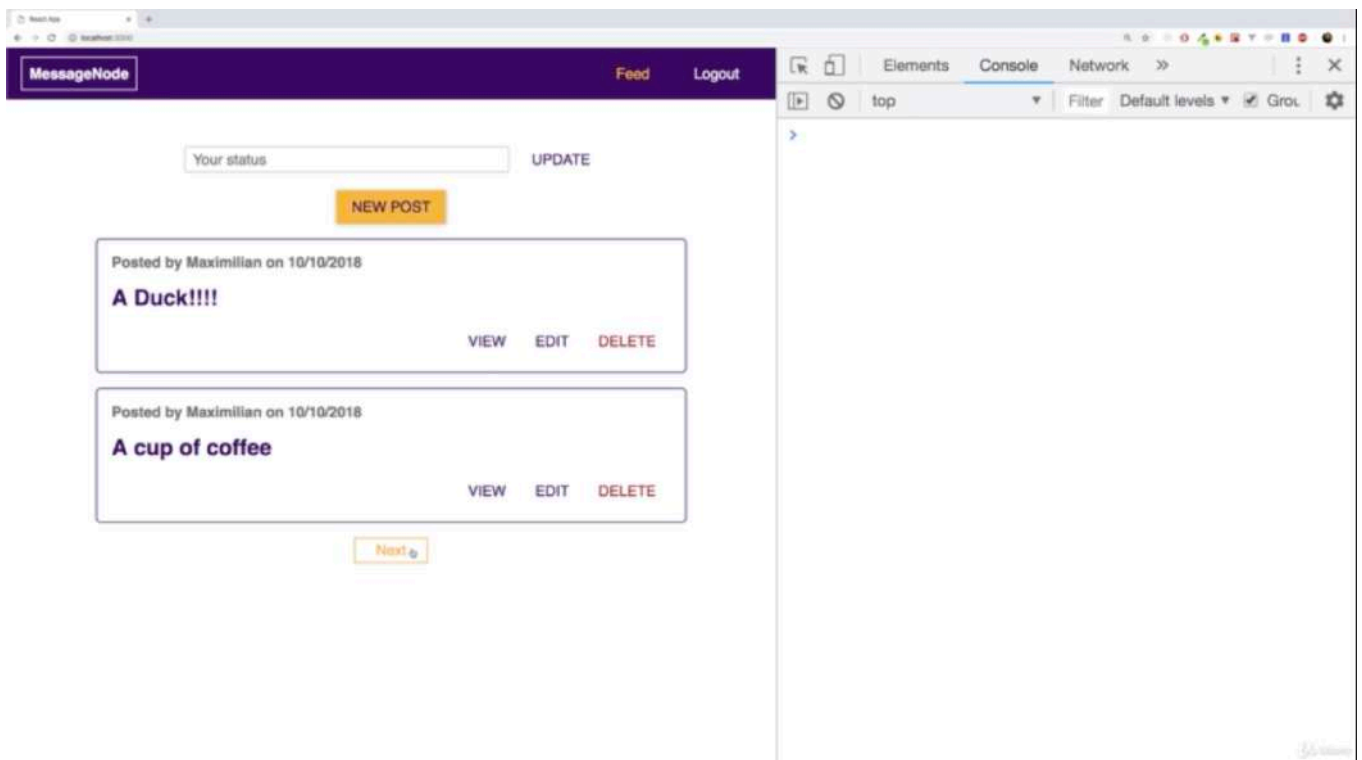






- i wanna limit showing data which is called pagination.





```

1  ../controllers/feed.js(b)
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9
10 exports.getPosts = (req, res, next) => {
11   const currentPage = req.query.page || 1;
12   /**i will set the per page value to 2
13    * and i do this because i have the same value in the frontend.
14    */

```

```

15  const perPage = 2;
16  let totalItems;
17  Post.find()
18    .countDocuments()
19    .then(count => {
20      totalItems = count;
21      /**we can skip a certain amount of items with the '.skip()' method*/
22      return Post.find()
23        .skip((currentPage - 1) * perPage)
24        .limit(perPage)
25    })
26    .then(posts => {
27      res
28        .status(200)
29        .json({
30          message: 'Fetched posts successfully.',
31          posts: posts,
32          totalItems: totalItems
33        });
34    })
35    .catch(err => {
36      if (!err.statusCode) {
37        err.statusCode = 500;
38      }
39      next(err);
40    });
41 };
42
43 exports.createPost = (req, res, next) => {
44   const errors = validationResult(req);
45   if (!errors.isEmpty()) {
46     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

throw error;

}

if(!req.file){

const error = new Error('No image provided.');

error.statusCode = 422;

throw error;

}

const imageUrl = req.file.path;

const title = req.body.title;

const content = req.body.content;

const post = new Post({

title: title,

content: content,

imageUrl: imageUrl,

creator: { name: 'Maximilian' }

});

post

.save()

.then(result => {

res.status(201).json({

message: 'Post created successfully!',

post: result


```

71     });
72   })
73   .catch(err => {
74     if (!err.statusCode) {
75       err.statusCode = 500;
76     }
77     next(err);
78   });
79 };
80
81 exports.getPost = (req, res, next) => {
82   const postId = req.params.postId;
83   Post.findById(postId)
84     .then(post => {
85       if (!post) {
86         const error = new Error('Could not find post.');
```

error.statusCode = 404;

throw error;

}

res.status(200).json({ message: 'Post fetched.', post: post });

})

.catch(err => {

if (!err.statusCode) {

err.statusCode = 500;

}

next(err);

});

};

99

100 exports.updatePost = (req, res, next) => {

101 const postId = req.params.postId;

102 const errors = validationResult(req);

103 if (!errors.isEmpty()) {

104 const error = new Error('Validation failed, entered data is incorrect.');

105 error.statusCode = 422;

106 throw error;

107 }

108 const title = req.body.title;

109 const content = req.body.content;

110 let imageUrl = req.body.image;

111 if(req.file) {

112 imageUrl = req.file.path

113 }

114 if(!imageUrl) {

115 const error = new Error('No file picked.');

116 error.statusCode = 422;

117 throw error;

118 }

119 Post.findById(postId)

120 .then(post => {

121 if (!post) {

122 const error = new Error('Could not find post.');

123 error.statusCode = 404;

124 throw error;

125 }

126 if(imageUrl !== post.image) {

```

127     clearImage(post.imageUrl);
128   }
129   post.title = title;
130   post.imageUrl = imageUrl;
131   post.content = content;
132   return post.save();
133 })
134 .then(result => {
135   res.status(200).json({message: 'Post updated!', post: result});
136 })
137 .catch(err => {
138   if (!err.statusCode) {
139     err.statusCode = 500;
140   }
141   next(err);
142 })
143 };
144
145 exports.deletePost = (req, res, next) => {
146   const postId = req.params.postId;
147   Post.findById(postId)
148     .then(post => {
149       if (!post) {
150         const error = new Error('Could not find post.');
```

```

151         error.statusCode = 404;
152         throw error;
153       }
154       //Check logged in user
155       clearImage(post.imageUrl);
156       return Post.findByIdAndRemove(postId);
157     })
158     .then(result => {
159       console.log(result);
160       res.status(200).json({message: 'Deleted post.'});
161     })
162     .catch(err => {
163       if (!err.statusCode) {
164         err.statusCode = 500;
165       }
166       next(err);
167     });
168 }
169
170 const clearImage = filePath => {
171   filePath = path.join(__dirname, '..', filePath)
172   fs.unlink(filePath, err => console.log(err));
173 }
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```

9 import Paginator from '../components/Paginator/Paginator';
10 import Loader from '../components/Loader/Loader';
11 import ErrorHandler from '../components/ErrorHandler/ErrorHandler';
12 import './Feed.css';
13
14 class Feed extends Component {
15   state = {
16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24   };
25
26   componentDidMount() {
27     fetch('URL')
28       .then(res => {
29         if (res.status !== 200) {
30           throw new Error('Failed to fetch user status.');

```

```

65         return {
66             ...post,
67             imagePath: post.imageUrl
68         };
69     }),
70     totalPosts: resData.totalItems,
71     postsLoading: false
72 });
73 })
74 .catch(this.catchError);
75 };
76
77 statusUpdateHandler = event => {
78     event.preventDefault();
79     fetch('URL')
80     .then(res => {
81         if (res.status !== 200 && res.status !== 201) {
82             throw new Error("Can't update status!");
83         }
84         return res.json();
85     })
86     .then(resData => {
87         console.log(resData);
88     })
89     .catch(this.catchError);
90 };
91
92 newPostHandler = () => {
93     this.setState({ isEditing: true });
94 };
95
96 startEditPostHandler = postId => {
97     this.setState(prevState => {
98         const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
99
100         return {
101             isEditing: true,
102             editPost: loadedPost
103         };
104     });
105 };
106
107 cancelEditHandler = () => {
108     this.setState({ isEditing: false, editPost: null });
109 };
110
111 finishEditHandler = postData => {
112     this.setState({
113         editLoading: true
114     });
115     const formData = new FormData()
116     formData.append('title', postData.title);
117     formData.append('content', postData.content);
118     formData.append('image', postData.image);
119     let url = 'http://localhost:8080/feed/post';
120     let method = 'POST';

```

```

121     if (this.state.editPost) {
122         url = 'http://localhost:8080/feed/post/' + this.state.editPost._id;
123         method = 'PUT';
124     }
125
126     fetch(url, {
127         method: method,
128         body: formData
129     })
130
131     .then(res => {
132         if (res.status !== 200 && res.status !== 201) {
133             throw new Error('Creating or editing a post failed!');
134         }
135         return res.json();
136     })
137     .then(resData => {
138         console.log(resData);
139         const post = {
140             _id: resData.post._id,
141             title: resData.post.title,
142             content: resData.post.content,
143             creator: resData.post.creator,
144             createdAt: resData.post.createdAt
145         };
146         this.setState(prevState => {
147             let updatedPosts = [...prevState.posts];
148             if (prevState.editPost) {
149                 const postIndex = prevState.posts.findIndex(
150                     p => p._id === prevState.editPost._id
151                 );
152                 updatedPosts[postIndex] = post;
153             } else if (prevState.posts.length < 2) {
154                 updatedPosts = prevState.posts.concat(post);
155             }
156             return {
157                 posts: updatedPosts,
158                 isEditing: false,
159                 editPost: null,
160                 editLoading: false
161             };
162         });
163     })
164     .catch(err => {
165         console.log(err);
166         this.setState({
167             isEditing: false,
168             editPost: null,
169             editLoading: false,
170             error: err
171         });
172     });
173 };
174
175 statusInputChangeHandler = (input, value) => {
176     this.setState({ status: value });

```

```

177 };
178
179 deletePostHandler = postId => {
180   this.setState({ postsLoading: true });
181   fetch('http://localhost:8080/feed/post/' + postId, {
182     method: 'DELETE'
183   })
184     .then(res => {
185       if (res.status !== 200 && res.status !== 201) {
186         throw new Error('Deleting a post failed!');
187       }
188       return res.json();
189     })
190     .then(resData => {
191       console.log(resData);
192       this.setState(prevState => {
193         const updatedPosts = prevState.posts.filter(p => p._id !== postId);
194         return { posts: updatedPosts, postsLoading: false };
195       });
196     })
197     .catch(err => {
198       console.log(err);
199       this.setState({ postsLoading: false });
200     });
201 };
202
203 errorHandler = () => {
204   this.setState({ error: null });
205 };
206
207 catchError = error => {
208   this.setState({ error: error });
209 };
210
211 render() {
212   return (
213     <Fragment>
214       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
215       <FeedEdit
216         editing={this.state.isEditing}
217         selectedPost={this.state.editPost}
218         loading={this.state.editLoading}
219         onCancelEdit={this.cancelEditHandler}
220         onFinishEdit={this.finishEditHandler}
221       />
222       <section className="feed__status">
223         <form onSubmit={this.statusUpdateHandler}>
224           <Input
225             type="text"
226             placeholder="Your status"
227             control="input"
228             onChange={this.statusInputChangeHandler}
229             value={this.state.status}
230           />
231           <Button mode="flat" type="submit">
232             Update

```

```

233     </Button>
234   </form>
235 </section>
236 <section className="feed__control">
237   <Button mode="raised" design="accent" onClick={this.newPostHandler}>
238     New Post
239   </Button>
240 </section>
241 <section className="feed">
242   {this.state.postsLoading && (
243     <div style={{ textAlign: 'center', marginTop: '2rem' }}>
244       <Loader />
245     </div>
246   )}
247   {this.state.posts.length <= 0 && !this.state.postsLoading ? (
248     <p style={{ textAlign: 'center' }}>No posts found.</p>
249   ) : null}
250   {!this.state.postsLoading && (
251     <Paginator
252       onPrevious={this.loadPosts.bind(this, 'previous')}
253       onNext={this.loadPosts.bind(this, 'next')}
254       lastPage={Math.ceil(this.state.totalPosts / 2)}
255       currentPage={this.state.postPage}
256     >
257       {this.state.posts.map(post => (
258         <Post
259           key={post._id}
260           id={post._id}
261           author={post.creator}
262           date={new Date(post.createdAt).toLocaleDateString('en-US')}
263           title={post.title}
264           image={post.imageUrl}
265           content={post.content}
266           onStartEdit={this.startEditPostHandler.bind(this, post._id)}
267           onDelete={this.deletePostHandler.bind(this, post._id)}
268         </>
269       ))}
270     </Paginator>
271   )}
272 </section>
273 </Fragment>
274 );
275 }
276 }
277
278 export default Feed;
279

```

* Chapter 381: Adding A User Model

1. update
 - ./models/user.js(b)
 - ./routes/auth.js(b)
 - app.js(b)

```
1 //./models/user.js(b)
```



```

2
3 const mongoose = require('mongoose');
4 const Schema = mongoose.Schema;
5
6 const userSchema = new Schema({
7   email: {
8     type: String,
9     required: true
10  },
11  password: {
12    type: String,
13    required: true
14  },
15  name: {
16    type: String,
17    required: true
18  },
19  status: {
20    type: String,
21    required: true
22  },
23  /**each object in the array will be of Type 'Schema.Types.ObjectId'
24   * because this will be reference to a post
25   * and therefore i add this ref key and add the post model.
26   * so i will store references to Post in my users or on my users.
27   */
28  posts: [{
29    type: Schema.Types.ObjectId,
30    ref: 'Post'
31  }]
32 });
33
34 module.exports = mongoose.model('User', userSchema);

```

```

1 //./routes/auth.js(b)
2
3 const express = require('express');
4
5 const router = express.Router();
6
7 /**let's go for a PUT route
8  * because we create a user once.
9  */
10 router.put('/singup');
11
12 module.exports = router;

```

```

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

```

```

11 const authRoutes = require('./routes/auth');
12
13 const app = express();
14
15 const fileStorage = multer.diskStorage({
16   destination: (req, file, cb) => {
17     cb(null, 'images');
18   },
19   filename: (req, file, cb) => {
20     cb(null, new Date().toISOString() + '-' + file.originalname);
21   }
22 });
23
24 const fileFilter = (req, file, cb) => {
25   if(
26     file.mimetype === 'image/png' ||
27     file.mimetype === 'image/jpg' ||
28     file.mimetype === 'image/jpeg'
29   ) {
30     cb(null, true);
31   } else {
32     cb(null, false);
33   }
34 }
35
36 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
37 app.use(bodyParser.json()); // application/json
38 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
39 app.use('/images', express.static(path.join(__dirname, 'images')));
40
41 app.use((req, res, next) => {
42   res.setHeader('Access-Control-Allow-Origin', '*');
43   res.setHeader(
44     'Access-Control-Allow-Methods',
45     'OPTIONS, GET, POST, PUT, PATCH, DELETE'
46   );
47   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
48   next();
49 });
50
51 app.use('/feed', feedRoutes);
52 app.use('/auth', authRoutes);
53
54 app.use((error, req, res, next) => {
55   console.log(error);
56   const status = error.statusCode || 500;
57   const message = error.message;
58   res.status(status).json({ message: message });
59 });
60
61 mongoose
62   .connect(
63     'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/message?retryWrites=true'
64   )
65   .then(result => {
66     app.listen(8080);

```

```
67   })
68   .catch(err => console.log(err));
```

* Chapter 382: Adding User Signup Validation

- 1. update
- ./controllers/auth.js(b)
- ./routes/auth.js(b)
- app.js(b)

```
1  //./controllers/auth.js(b)
2
3  const { validationResult } = require('express-validator/check');
4
5  const User = require('../models/user');
6
7  exports.signup = (req, res, next) => {
8      const errors = validationResult(req);
9      if(!errors.isEmpty()){
10         const error = new Error('Validation failed');
11         error.statusCode = 422;
12         /**error.data = errors.array()'
13         * this would allow me to keep my errors
14         * which were retrieved by that validation package
15         */
16         error.data = errors.array();
17         throw error;
18     }
19     const email = req.body.email;
20     const name = req.body.name;
21     const password = req.body.password;
22
23 }
```

```
1  //./routes/auth.js(b)
2
3  const express = require('express');
4  const { body } = require('express-validator/check');
5
6  const User = require('../models/user');
7  const authController = require('../controllers/auth');
8
9  const router = express.Router();
10
11  /**let's go for a PUT route
12   * because we create a user once.
13   */
14  router.put(
15      '/signup',
16      [
17          body('email')
18              .isEmail()
19              .withMessage('Please enter a valid email.')
20              /**we can also add our own custom validator
21               * to check whether the email address already exists.
22               *
23               * value and then an object from which we can extract the request
```

```

24     * as a property with this destructuring syntax
25     */
26     .custom((value, { req }) => {
27         return User.findOne({ email: value }).then(userDoc => {
28             if (userDoc) {
29                 /**if thta is set,
30                  * then i will reject a promise,
31                  * so i will return Promise.reject()
32                  * and that will cause the validation
33                  * all other scenarios will cause it to succeed
34                  * and then i will return 'E-Mail address already exists!'
35                  */
36                 return Promise.reject('E-Mail address already exists!');
37             }
38         });
39     })
40     .normalizeEmail(),
41     body('password')
42     .trim()
43     .isLength({ min: 5 }),
44     body('name')
45     .trim()
46     .not()
47     .isEmpty()
48 ],
49 authController.signup
50 );
51
52 router.post('/login', authController.login);
53
54 module.exports = router;
55

```

```

1 //app.js(b)
2
3 const path = require('path');
4
5 const express = require('express');
6 const bodyParser = require('body-parser');
7 const mongoose = require('mongoose');
8 const multer = require('multer');
9
10 const feedRoutes = require('./routes/feed');
11 const authRoutes = require('./routes/auth');
12
13 const app = express();
14
15 const fileStorage = multer.diskStorage({
16     destination: (req, file, cb) => {
17         cb(null, 'images');
18     },
19     filename: (req, file, cb) => {
20         cb(null, new Date().toISOString() + '-' + file.originalname);
21     }
22 });
23
24 const fileFilter = (req, file, cb) => {

```

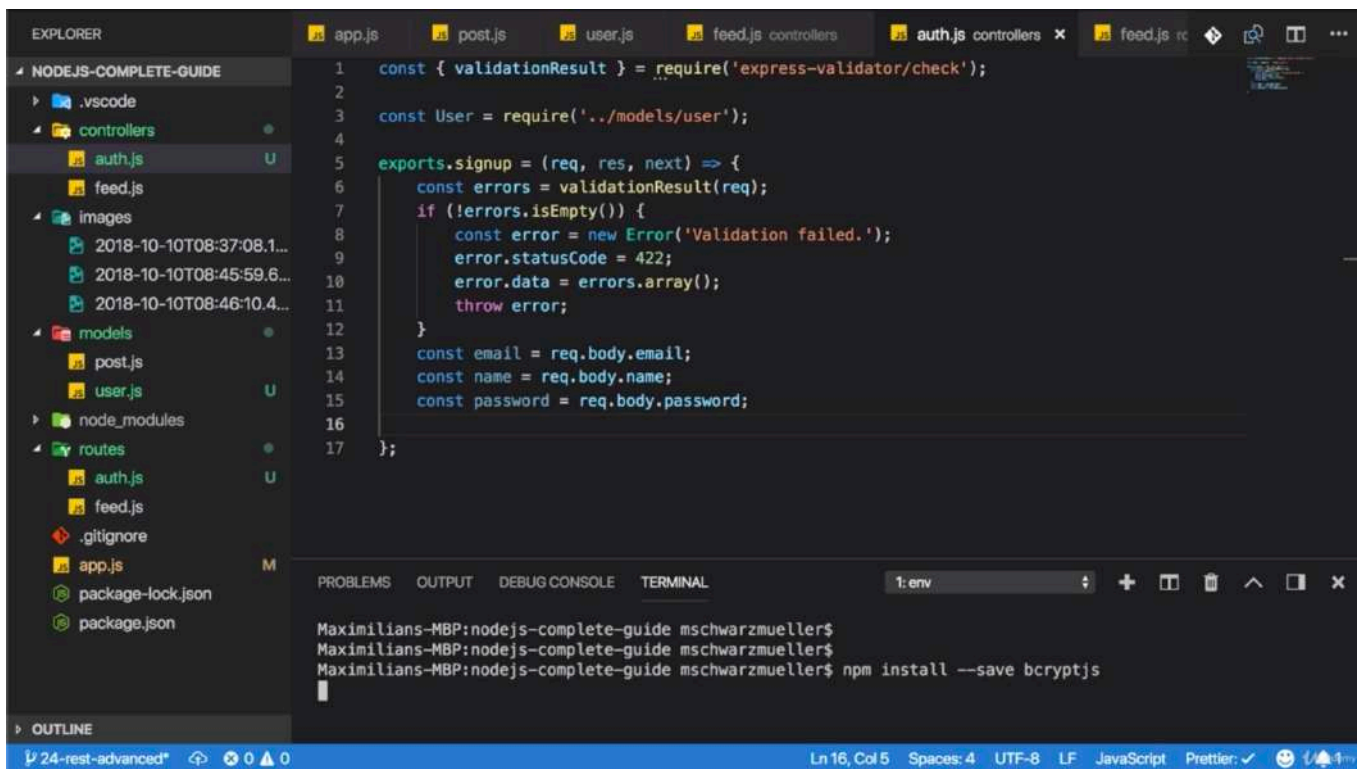
```

25  if(
26      file.mimetype === 'image/png' ||
27      file.mimetype === 'image/jpg' ||
28      file.mimetype === 'image/jpeg'
29  ) {
30      cb(null, true);
31  } else {
32      cb(null, false);
33  }
34 }
35
36 // app.use(bodyParser.urlencoded()); // x-www-form-urlencoded <form>
37 app.use(bodyParser.json()); // application/json
38 app.use(multer({storage: fileStorage, fileFilter: fileFilter}).single('image'))
39 app.use('/images', express.static(path.join(__dirname, 'images')));
40
41 app.use((req, res, next) => {
42     res.setHeader('Access-Control-Allow-Origin', '*');
43     res.setHeader(
44         'Access-Control-Allow-Methods',
45         'OPTIONS, GET, POST, PUT, PATCH, DELETE'
46     );
47     res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
48     next();
49 });
50
51 app.use('/feed', feedRoutes);
52 app.use('/auth', authRoutes);
53
54 app.use((error, req, res, next) => {
55     console.log(error);
56     const status = error.statusCode || 500;
57     const message = error.message;
58     const data = error.data;
59     res.status(status).json({ message: message, data: data });
60 });
61
62 mongoose
63     .connect(
64         'mongodb+srv://maximilian:rldnjs12@cluster0-z3v1k.mongodb.net/message?retryWrites=true'
65     )
66     .then(result => {
67         app.listen(8080);
68     })
69     .catch(err => console.log(err));

```

* Chapter 383: Signing User Up

1. update
 - ./controllers/auth.js(b)
 - ./models/user.js(b)
 - App.js(f)

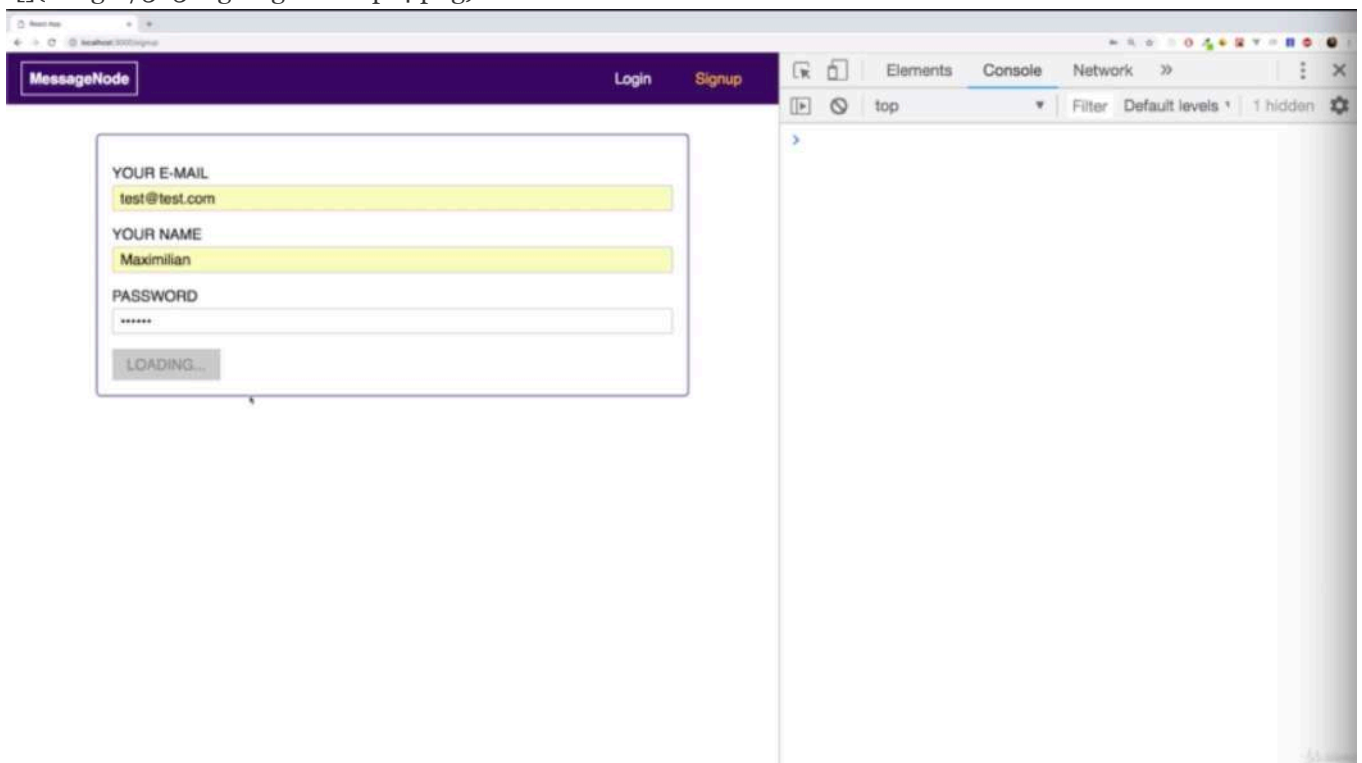


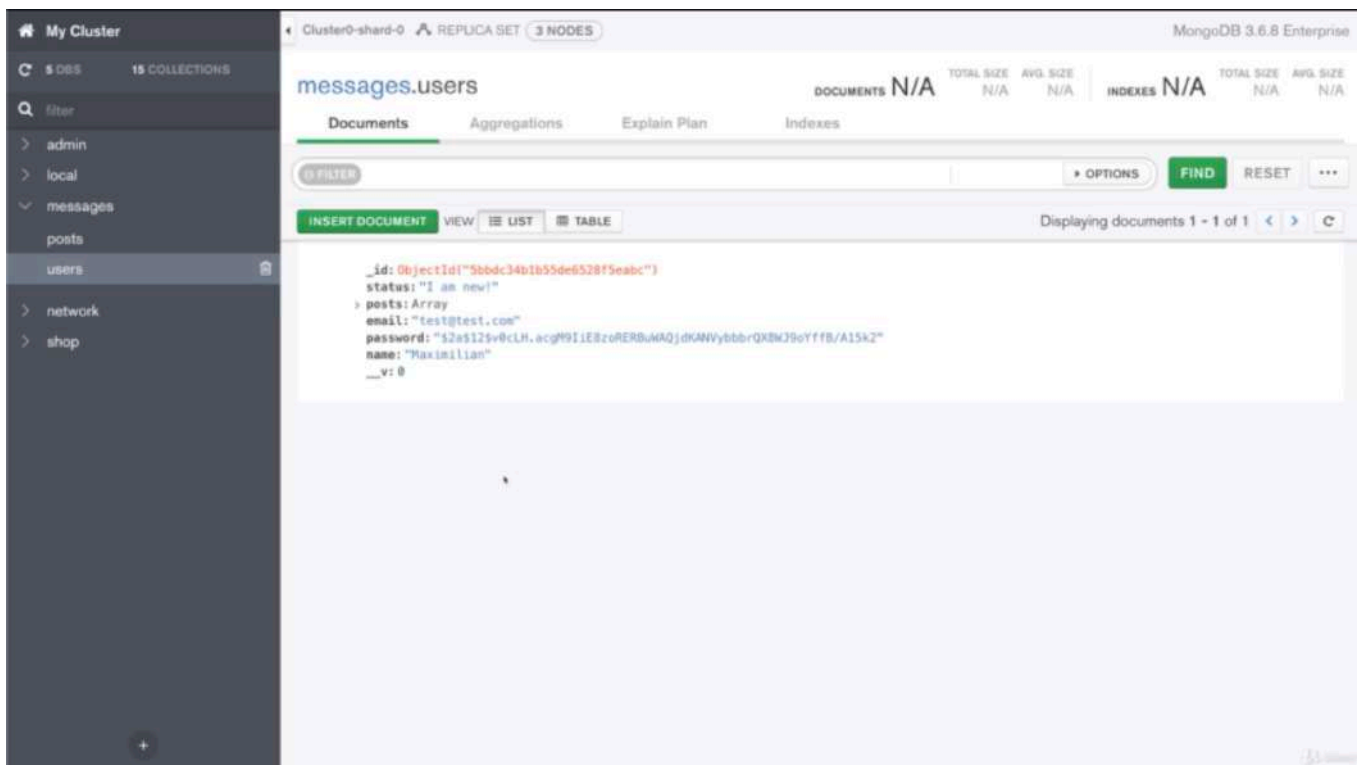
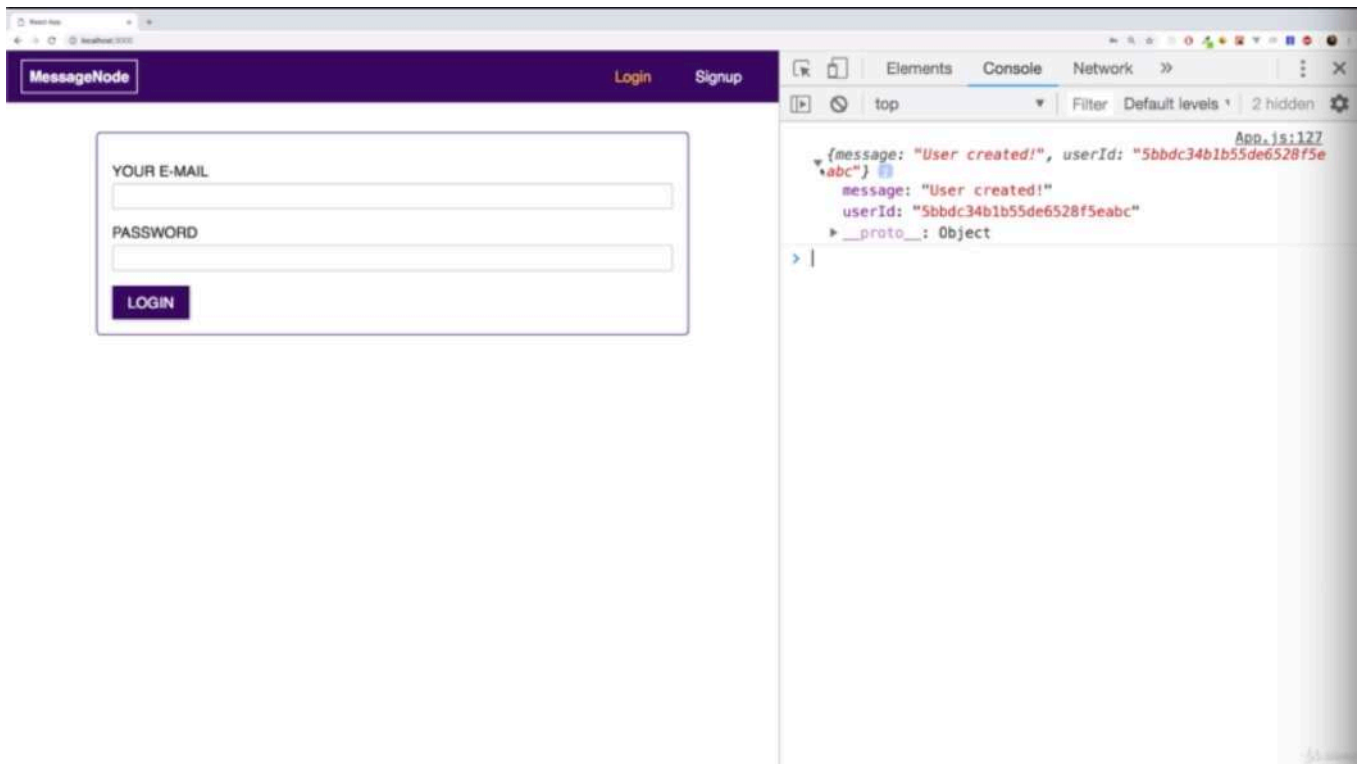
The image shows a VS Code editor with the file explorer on the left displaying a project structure for 'nodejs-complete-guide'. The main editor area shows the 'auth.js' file with the following code:

```
1 const { validationResult } = require('express-validator/check');
2
3 const User = require('../models/user');
4
5 exports.signup = (req, res, next) => {
6   const errors = validationResult(req);
7   if (!errors.isEmpty()) {
8     const error = new Error('Validation failed.');
```

The terminal at the bottom shows the command: `npm install --save bcryptjs` being executed in the project directory.

- i will install a new package called 'bcrypt.js' which allows us to hash a password in a secure way.





```

1  ../controllers/auth.js(b)
2
3  const { validationResult } = require('express-validator/check');
4  const bcrypt = require('bcryptjs');
5
6
7  const User = require('../models/user');
8
9  exports.signup = (req, res, next) => {
10     const errors = validationResult(req);
11     if(!errors.isEmpty()){
12         const error = new Error('Validation failed');
13         error.statusCode = 422;
14         error.data = errors.array();

```



```

15         throw error;
16     }
17     const email = req.body.email;
18     const name = req.body.name;
19     const password = req.body.password;
20     /**password which is plaintext with a salt of 12,
21     * so a strength of 12.
22     */
23     bcrypt.hash(password, 12)
24         .then(hashedPw => {
25         const user = new User({
26             email: email,
27             passowrd: hashedPw,
28             name: name
29         });
30         /**we can add 'return'
31         * so that we can add another '.then()'
32         */
33         return user.save();
34     })
35     .then(result => {
36         res.status(201).json({message: 'User created!', userId: result._id});
37     })
38     .catch(err => {
39         if(!err.statusCode) {
40             err.statusCode = 500;
41         }
42         next(err);
43     })
44 }

```

```

1  //./models/user.js(b)
2
3  const mongoose = require('mongoose');
4  const Schema = mongoose.Schema;
5
6  const userSchema = new Schema({
7      email: {
8          type: String,
9          required: true
10     },
11     password: {
12         type: String,
13         required: true
14     },
15     name: {
16         type: String,
17         required: true
18     },
19     status: {
20         type: String,
21         default: 'I am new!'
22     },
23     posts: [{
24         type: Schema.Types.ObjectId,
25         ref: 'Post'
26     }]

```

```

27 });
28
29 module.exports = mongoose.model('User', userSchema);

1 //App.js(f)
2
3 import React, { Component, Fragment } from 'react';
4 import { Route, Switch, Redirect, withRouter } from 'react-router-dom';
5
6 import Layout from './components/Layout/Layout';
7 import Backdrop from './components/Backdrop/Backdrop';
8 import Toolbar from './components/Toolbar/Toolbar';
9 import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
10 import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
11 import ErrorHandler from './components/ErrorHandler/ErrorHandler';
12 import FeedPage from './pages/Feed/Feed';
13 import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
14 import LoginPage from './pages/Auth/Login';
15 import SignupPage from './pages/Auth/Signup';
16 import './App.css';
17
18 class App extends Component {
19   state = {
20     showBackdrop: false,
21     showMobileNav: false,
22     /**we should change 'true' of 'isAuth' to 'false'
23      * so that we start unauthenticated.
24      */
25     isAuth: false,
26     token: null,
27     userId: null,
28     authLoading: false,
29     error: null
30   };
31
32   componentDidMount() {
33     const token = localStorage.getItem('token');
34     const expiryDate = localStorage.getItem('expiryDate');
35     if (!token || !expiryDate) {
36       return;
37     }
38     if (new Date(expiryDate) <= new Date()) {
39       this.logoutHandler();
40       return;
41     }
42     const userId = localStorage.getItem('userId');
43     const remainingMilliseconds =
44       new Date(expiryDate).getTime() - new Date().getTime();
45     this.setState({ isAuth: true, token: token, userId: userId });
46     this.setAutoLogout(remainingMilliseconds);
47   }
48
49   mobileNavHandler = isOpen => {
50     this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
51   };
52
53   backdropClickHandler = () => {

```

```

54     this.setState({ showBackdrop: false, showMobileNav: false, error: null });
55 };
56
57 logoutHandler = () => {
58     this.setState({ isAuth: false, token: null });
59     localStorage.removeItem('token');
60     localStorage.removeItem('expiryDate');
61     localStorage.removeItem('userId');
62 };
63
64 loginHandler = (event, authData) => {
65     event.preventDefault();
66     this.setState({ authLoading: true });
67     fetch('URL')
68         .then(res => {
69             if (res.status === 422) {
70                 throw new Error('Validation failed.');
```

```

110 headers: {
111   'Content-Type': 'application/json'
112 },
113 /**everything which we extract on the server side.
114  * this is now all passed to our backend.
115  *
116  * '.signupForm' is how the data is stored internally in the React app
117  * and this is how we should extract it
118  */
119 body: JSON.stringify({
120   email: authData.signupForm.email.value,
121   password: authData.signupForm.password,
122   name: authData.signupForm.name
123 })
124 })
125 .then(res => {
126   if (res.status === 422) {
127     throw new Error(
128       "Validation failed. Make sure the email address isn't used yet!"
129     );
130   }
131   if (res.status !== 200 && res.status !== 201) {
132     console.log('Error!');
133     throw new Error('Creating a user failed!');
134   }
135   return res.json();
136 })
137 .then(resData => {
138   console.log(resData);
139   this.setState({ isAuthenticated: false, authLoading: false });
140   this.props.history.replace('/');
141 })
142 .catch(err => {
143   console.log(err);
144   this.setState({
145     isAuthenticated: false,
146     authLoading: false,
147     error: err
148   });
149 });
150 };
151
152 setAutoLogout = milliseconds => {
153   setTimeout(() => {
154     this.logoutHandler();
155   }, milliseconds);
156 };
157
158 errorHandler = () => {
159   this.setState({ error: null });
160 };
161
162 render() {
163   let routes = (
164     <Switch>
165     <Route

```

```

166     path="/"
167     exact
168     render={props => (
169         <LoginPage
170             {...props}
171             onLogin={this.loginHandler}
172             loading={this.state.authLoading}
173         />
174     )}
175 />
176 <Route
177     path="/signup"
178     exact
179     render={props => (
180         <SignupPage
181             {...props}
182             onSignup={this.signupHandler}
183             loading={this.state.authLoading}
184         />
185     )}
186 />
187 <Redirect to="/" />
188 </Switch>
189 );
190 if (this.state.isAuth) {
191     routes = (
192         <Switch>
193             <Route
194                 path="/"
195                 exact
196                 render={props => (
197                     <FeedPage userId={this.state.userId} token={this.state.token} />
198                 )}
199             />
200             <Route
201                 path="/:postId"
202                 render={props => (
203                     <SinglePostPage
204                         {...props}
205                         userId={this.state.userId}
206                         token={this.state.token}
207                     />
208                 )}
209             />
210             <Redirect to="/" />
211         </Switch>
212     );
213 }
214 return (
215     <Fragment>
216         {this.state.showBackdrop && (
217             <Backdrop onClick={this.backdropClickHandler} />
218         )}
219         <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
220         <Layout
221             header={

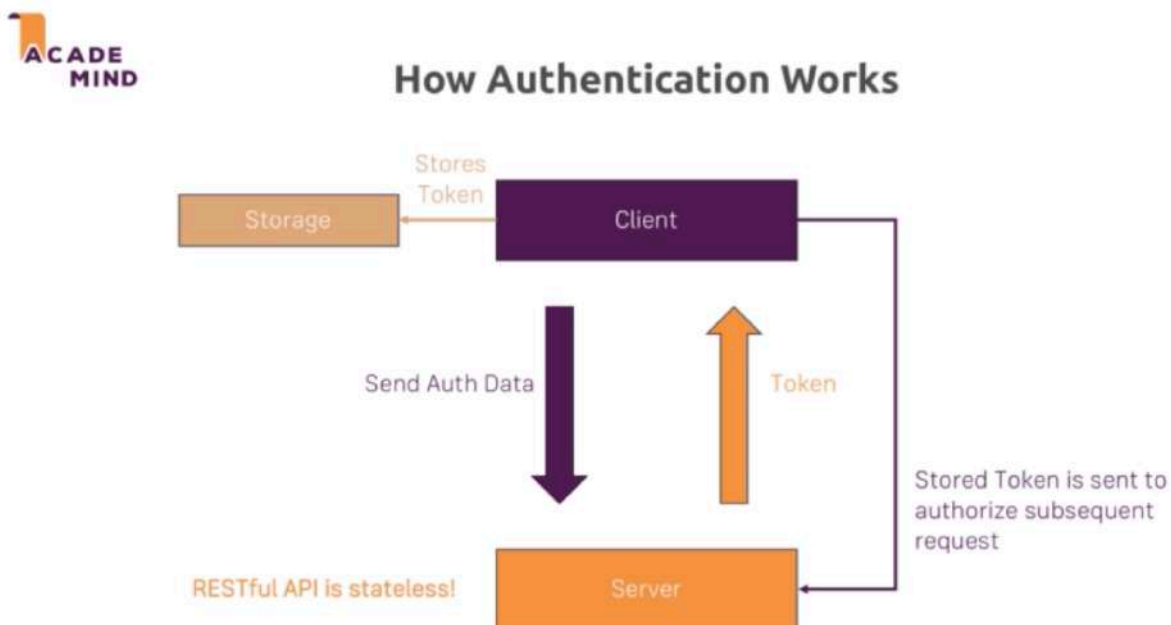
```

```

222 <Toolbar>
223   <MainNavigation
224     onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
225     onLogout={this.logoutHandler}
226     isAuth={this.state.isAuth}
227   />
228 </Toolbar>
229 }
230 mobileNav={
231   <MobileNavigation
232     open={this.state.showMobileNav}
233     mobile
234     onChooseItem={this.mobileNavHandler.bind(this, false)}
235     onLogout={this.logoutHandler}
236     isAuth={this.state.isAuth}
237   />
238 }
239 />
240 {routes}
241 </Fragment>
242 );
243 }
244 }
245
246 export default withRouter(App);
247

```

* Chapter 384: How Does Authentication Work?



- we don't use a session anymore because RESTful APIs are stateless, they don't care about client. you learned about that strict decoupling of server and client and every request should be treated standalone which means every request should have all the data it needs to authenticate itself. with session, the server needs to store data

about the client, the server then stores that a client is authenticated. that is not how RESTful APIs work. the server will not store anything about any client.

- but instead we return so-called 'token' to the client which will be generated on the server and will hold some information which can only be validated by the server which create a token.

- so stored token is attached to every request that targets a resource on the server which requires authentication. and if you change that token on the frontend or you try to create it to fake that you are authenticated, that will be detected because the server used a certain algorithm for generating the token which you can't fake because you don't know the private key by the server.



What's that Token?

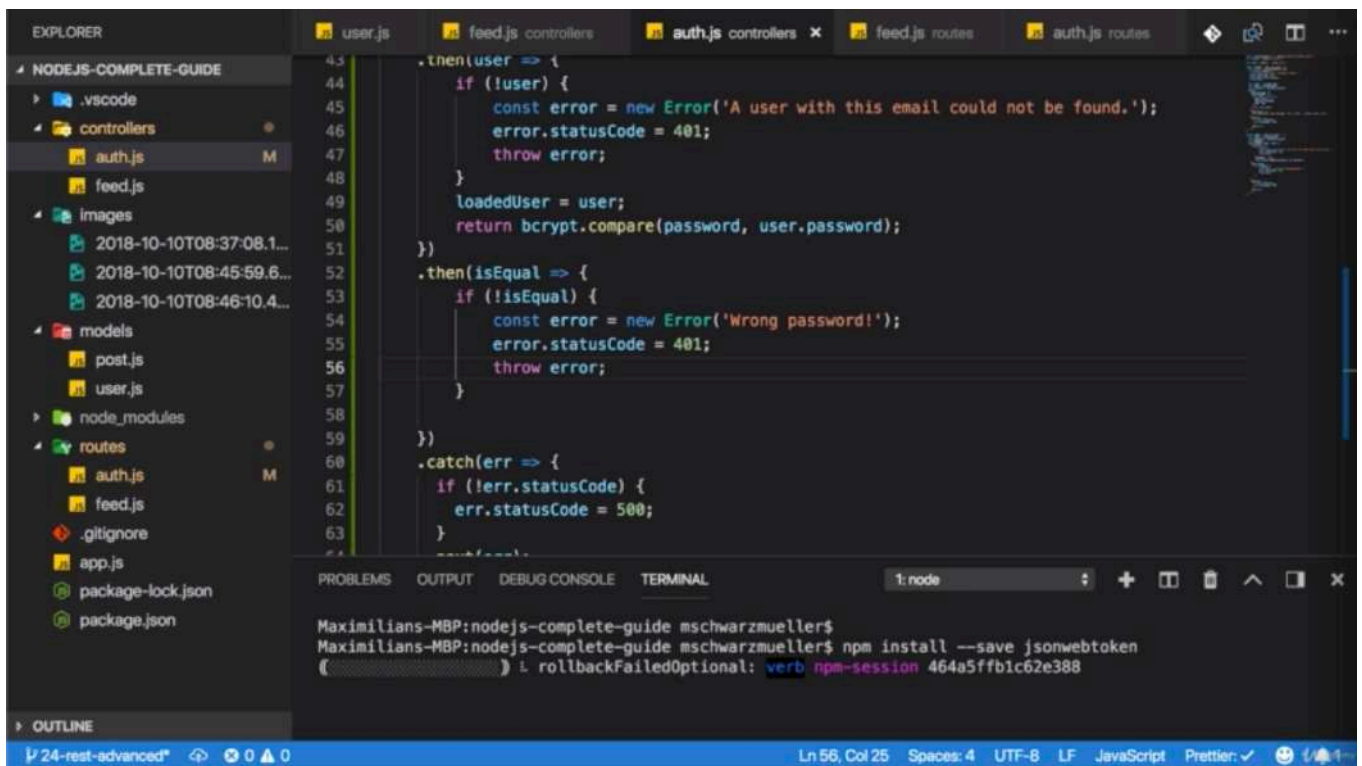


- Token contains JSON Data which is javascript data + signature which is generated on the server with a special private key which is only stored on the server.

- the result is 'JSON Web Token(JWT)' which is returned to the client

* Chapter 386: Logging In & Creating JSON Web Tokens(JWTs)

1. update
 - ./routes/auth.js(b)
 - ./controllers/auth.js(b)
 - App.js(f)



The screenshot shows the VS Code editor interface. The Explorer sidebar on the left displays the project structure, including files like `auth.js`, `feed.js`, and `models`. The main editor area shows the `auth.js` file with the following code:

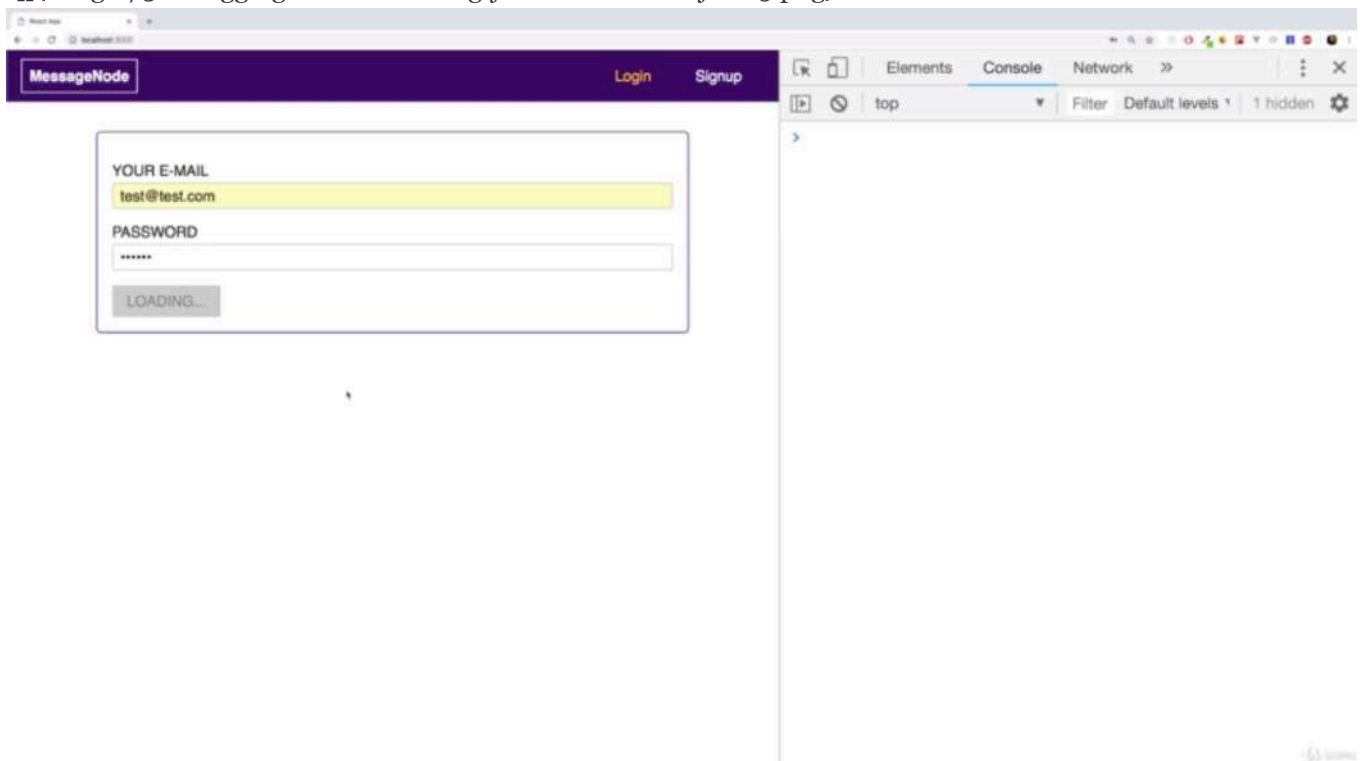
```
43 .then(user => {
44   if (!user) {
45     const error = new Error('A user with this email could not be found.');
```

```
46     error.statusCode = 401;
47     throw error;
48   }
49   loadedUser = user;
50   return bcrypt.compare(password, user.password);
51 })
52 .then(isEqual => {
53   if (!isEqual) {
54     const error = new Error('Wrong password!');
```

```
55     error.statusCode = 401;
56     throw error;
57   }
58 })
59 .catch(err => {
60   if (!err.statusCode) {
61     err.statusCode = 500;
62   }
63 })
```

The terminal at the bottom shows the command `npm install --save jsonwebtoken` being executed, with the output indicating a successful installation of the `jsonwebtoken` package.

- 'npm install --save jsonwebtoken' is the package which allows us to conveniently create such new JSON web tokens.



React App

webpack-dev-server

MessageNode

Feed

Logout

UPDATE

NEW POST

Posted by Maximilian on 10/10/2018

A Duck!!!!

VIEW EDIT DELETE

Posted by Maximilian on 10/10/2018

A cup of coffee

VIEW EDIT DELETE

Next

Elements

Console

Network

top

Filter

Default levels

1 hidden

App.js:83

```
{token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWVpbCI6Ii02M000.XAgM_EM7SuQ1Q8f1EoZ_MU22I2yGHA4l73jcbuBaqM8", userId: "5bbdc34b1b55de6528f5eabc"}
```

React App

webpack-dev-server

MessageNode

Feed

Logout

UPDATE

NEW POST

Posted by Maximilian on 10/10/2018

A Duck!!!!

VIEW EDIT DELETE

Posted by Maximilian on 10/10/2018

A cup of coffee

VIEW EDIT DELETE

Next

Elements

Console

Network

top

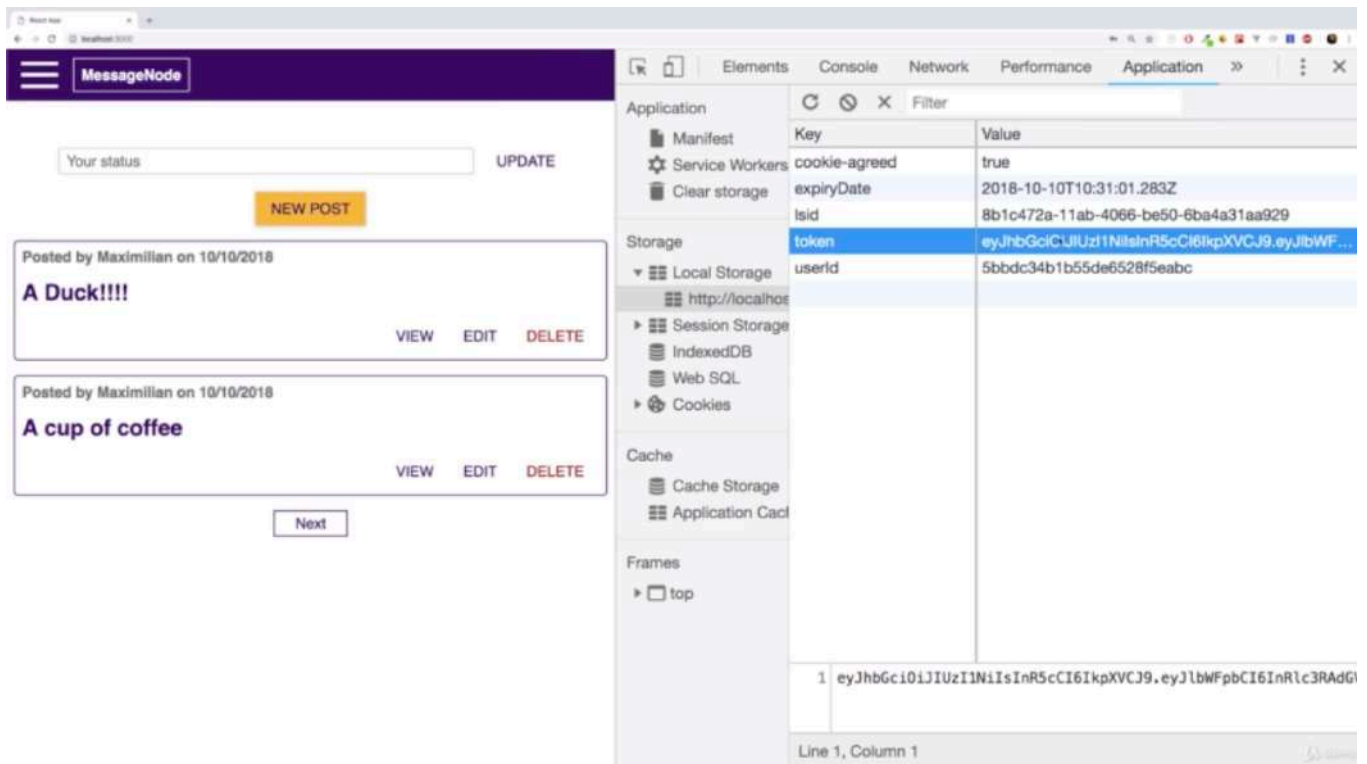
Filter

Default levels

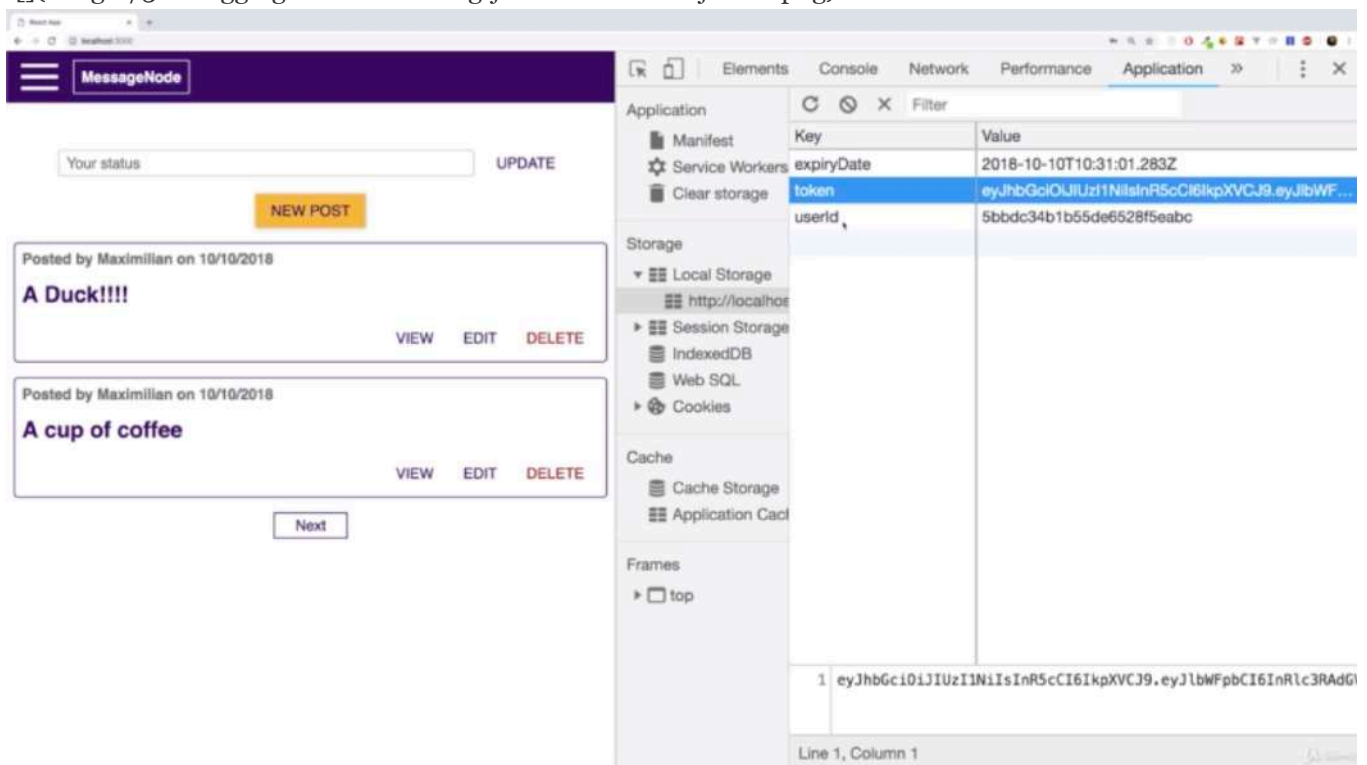
1 hidden

App.js:83

```
{token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWVpbCI6Ii02M000.XAgM_EM7SuQ1Q8f1EoZ_MU22I2yGHA4l73jcbuBaqM8", userId: "5bbdc34b1b55de6528f5eabc"} token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWVpbCI6Ii02M000.XAgM_EM7SuQ1Q8f1EoZ_MU22I2yGHA4l73jcbuBaqM8", userId: "5bbdc34b1b55de6528f5eabc" __proto__: Object
```



- that is the token we generated on the server.



- the other data is data i'm storing here or that was stored way back by other application so we don't need that. that has nothing to do with our app. we have the expiry date which i generated on the client to know when i should remove the token because it expired and this is the token itself.

The top screenshot shows the JWT.io website with a valid token pasted into the 'Encoded' field. The 'Decoded' section shows the token's structure:

- HEADER: ALGORITHM & TOKEN TYPE:**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```
- PAYLOAD: DATA:**

```
{
  "email": "test@test.com",
  "userId": "5bbdc34b1b55de6528f5eabc",
  "iat": 1539163861,
  "exp": 1539167461
}
```
- VERIFY SIGNATURE:**

```
HMACHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  somesupersecretsecret
) = secret base64 encoded
```

The bottom screenshot shows the same JWT.io website, but with a modified token pasted into the 'Encoded' field. The 'Decoded' section shows the token's structure:

- HEADER: ALGORITHM & TOKEN TYPE:**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```
- PAYLOAD: DATA:**

```
{
  "email": "test2@test.com",
  "userId": "5bbdc34b1b55de6528f5eabc",
  "iat": 1539163861,
  "exp": 1539167461
}
```
- VERIFY SIGNATURE:**

```
HMACHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  somesupersecretsecret
) = secret base64 encoded
```

- if i change the email address, the token changes on the left and therefore if i edit this, it will not be validated because now the token is not the same as it was when it was generated on the server with this secret.

The screenshot shows the JWT.io website. On the left, under the 'Encoded' tab, a token is pasted: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnCI6InRlc3QyQHRlc3QyY29tIiwidXNlckkiOiJoInNWJiZGMzNGIxYjU1ZGU2NTI4ZjVlYWJjIiwiaWF0IjoxNTM5MTYzODYxLCJleHAiOjE1MzIxNjc0NjF9.CWVb_tSq-I2IWd1z79Yzcm1whcszt1JnQ3rteXXmfmw`. On the right, under the 'Decoded' tab, the token's structure is shown. The header is `{ "alg": "HS256", "typ": "JWT" }`. The payload is `{ "email": "test2@test.com", "userId": "5bbdc34b1b55de6528f5eabc", "iat": 1539163861, "exp": 1539167461 }`. The signature verification section shows the HMACSHA256 algorithm applied to the base64url encoded header and payload, resulting in the signature `cmVb_tSq-I2IWd1z79Yzcm1whcszt1JnQ3rteXXmfmw`.

- so if i try to generate my own one with a different secret, i will end up with a different token and if i try to edit the data in there, i also do so i can't mess around with the token in the client. the server will detect this.

```

1  //./controllers/auth.js(b)
2
3  const { validationResult } = require('express-validator/check');
4  const bcrypt = require('bcryptjs');
5  const jwt = require('jsonwebtoken');
6
7  const User = require('../models/user');
8
9  exports.signup = (req, res, next) => {
10   const errors = validationResult(req);
11   if (!errors.isEmpty()) {
12     const error = new Error('Validation failed. ');
13     error.statusCode = 422;
14     error.data = errors.array();
15     throw error;
16   }
17   const email = req.body.email;
18   const name = req.body.name;
19   const password = req.body.password;
20   /**password which is plaintext with a salt of 12,
21    * so a strength of 12.
22    */
23   bcrypt
24     .hash(password, 12)
25     .then(hashedPw => {
26       const user = new User({
27         email: email,
28         password: hashedPw,
29         name: name
30       });
31       /**we can add 'return'
32       * so that we can add another '.then()'

```

```

33     */
34     return user.save();
35 })
36 .then(result => {
37     res.status(201).json({ message: 'User created!', userId: result._id });
38 })
39 .catch(err => {
40     if (!err.statusCode) {
41         err.statusCode = 500;
42     }
43     next(err);
44 });
45 };
46
47 exports.login = (req, res, next) => {
48     const email = req.body.email;
49     const password = req.body.password;
50     let loadedUser;
51     User.findOne({ email: email })
52     .then(user => {
53         if (!user) {
54             const error = new Error('A user with this email could not be found.');
```

error.statusCode = 401;

```

56             throw error;
57         }
58         loadedUser = user;
59         /**'password' is what i filled in
60          * 'user.password' is what is stored in the server
61          */
62         return bcrypt.compare(password, user.password);
63     })
64     .then(isEqual => {
65         if (!isEqual) {
66             const error = new Error('Wrong password!');
```

error.statusCode = 401;

```

68             throw error;
69         }
70         /**'jwt.sign()' creates a new signature
71          * and packs that into a new JSON web token.
72          * we can add any data we wanna into the token.
73          */
74         const token = jwt.sign(
75             {
76                 email: loadedUser.email,
77                 userId: loadedUser._id.toString()
78             },
79         /**then you need to pass a 2nd argument
80          * which is that secret,
81          * so that private key which is used for signing
82          * and that is only known to the server
83          * and therefore you can't fake that token on the client.
84          *
85          * you can use 'secret'
86          * but typically you wanna use a longer string
87          * like 'somesupersecretsecret' like that.
88          * this is security mechanism you should add

```

```

89      * because the token is stored in the client.
90      * by the client to whom it belongs
91      * but technically that token could be stolen.
92      * if the user doesn't logout,
93      * another person copies the token from his browser storage
94      * and then he can use it on his own PC forever.
95      */
96      'somesupersecretsecret',
97      { expiresIn: '1h' }
98    );
99    res.status(200).json({ token: token, userId: loadedUser._id.toString() });
100  })
101  .catch(err => {
102    if (!err.statusCode) {
103      err.statusCode = 500;
104    }
105    next(err);
106  });
107 };
108

```

```

1  //./routes/auth.js(b)
2
3  const express = require('express');
4  const { body } = require('express-validator/check');
5
6  const User = require('../models/user');
7  const authController = require('../controllers/auth');
8
9  const router = express.Router();
10
11  router.put(
12    '/signup',
13    [
14      body('email')
15        .isEmail()
16        .withMessage('Please enter a valid email.')
17        .custom((value, { req }) => {
18          return User.findOne({ email: value }).then(userDoc => {
19            if (userDoc) {
20              return Promise.reject('E-Mail address already exists!');
21            }
22          });
23        })
24        .normalizeEmail(),
25      body('password')
26        .trim()
27        .isLength({ min: 5 }),
28      body('name')
29        .trim()
30        .not()
31        .isEmpty()
32    ],
33    authController.signup
34  );
35
36  router.post('/login', authController.login);

```

```

37
38 module.exports = router;
39

1 //App.js(f)
2
3 import React, { Component, Fragment } from 'react';
4 import { Route, Switch, Redirect, withRouter } from 'react-router-dom';
5
6 import Layout from './components/Layout/Layout';
7 import Backdrop from './components/Backdrop/Backdrop';
8 import Toolbar from './components/Toolbar/Toolbar';
9 import MainNavigation from './components/Navigation/MainNavigation/MainNavigation';
10 import MobileNavigation from './components/Navigation/MobileNavigation/MobileNavigation';
11 import ErrorHandler from './components/ErrorHandler/ErrorHandler';
12 import FeedPage from './pages/Feed/Feed';
13 import SinglePostPage from './pages/Feed/SinglePost/SinglePost';
14 import LoginPage from './pages/Auth/Login';
15 import SignupPage from './pages/Auth/Signup';
16 import './App.css';
17
18 class App extends Component {
19   state = {
20     showBackdrop: false,
21     showMobileNav: false,
22     isAuth: false,
23     token: null,
24     userId: null,
25     authLoading: false,
26     error: null
27   };
28
29   componentDidMount() {
30     const token = localStorage.getItem('token');
31     const expiryDate = localStorage.getItem('expiryDate');
32     if (!token || !expiryDate) {
33       return;
34     }
35     if (new Date(expiryDate) <= new Date()) {
36       this.logoutHandler();
37       return;
38     }
39     const userId = localStorage.getItem('userId');
40     const remainingMilliseconds =
41       new Date(expiryDate).getTime() - new Date().getTime();
42     this.setState({ isAuth: true, token: token, userId: userId });
43     this.setAutoLogout(remainingMilliseconds);
44   }
45
46   mobileNavHandler = isOpen => {
47     this.setState({ showMobileNav: isOpen, showBackdrop: isOpen });
48   };
49
50   backdropClickHandler = () => {
51     this.setState({ showBackdrop: false, showMobileNav: false, error: null });
52   };
53

```



```

54 logoutHandler = () => {
55   this.setState({ isAuth: false, token: null });
56   localStorage.removeItem('token');
57   localStorage.removeItem('expiryDate');
58   localStorage.removeItem('userId');
59 };
60
61 loginHandler = (event, authData) => {
62   event.preventDefault();
63   this.setState({ authLoading: true });
64   fetch('http://localhost:8080/auth/login', {
65     method: 'POST',
66     headers: {
67       'Content-Type': 'application/json'
68     },
69     body: JSON.stringify({
70       email: authData.email,
71       password: authData.password
72     })
73   })
74     .then(res => {
75       if (res.status === 422) {
76         throw new Error('Validation failed.');
```

```

77       }
78       if (res.status !== 200 && res.status !== 201) {
79         console.log('Error!');
80         throw new Error('Could not authenticate you!');
81       }
82       return res.json();
83     })
84     .then(resData => {
85       console.log(resData);
86       this.setState({
87         isAuth: true,
88         token: resData.token,
89         authLoading: false,
90         userId: resData.userId
91       });
92       localStorage.setItem('token', resData.token);
93       localStorage.setItem('userId', resData.userId);
94       const remainingMilliseconds = 60 * 60 * 1000;
95       const expiryDate = new Date(
96         new Date().getTime() + remainingMilliseconds
97       );
98       localStorage.setItem('expiryDate', expiryDate.toISOString());
99       this.setAutoLogout(remainingMilliseconds);
100     })
101     .catch(err => {
102       console.log(err);
103       this.setState({
104         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   fetch('http://localhost:8080/auth/signup', {
115     method: 'PUT',
116     headers: {
117       'Content-Type': 'application/json'
118     },
119     body: JSON.stringify({
120       email: authData.signupForm.email.value,
121       password: authData.signupForm.password.value,
122       name: authData.signupForm.name.value
123     })
124   })
125     .then(res => {
126       if (res.status === 422) {
127         throw new Error(
128           "Validation failed. Make sure the email address isn't used yet!"
129         );
130       }
131       if (res.status !== 200 && res.status !== 201) {
132         console.log('Error!');
133         throw new Error('Creating a user failed!');
134       }
135       return res.json();
136     })
137     .then(resData => {
138       console.log(resData);
139       this.setState({ isAuthenticated: false, authLoading: false });
140       this.props.history.replace('/');
141     })
142     .catch(err => {
143       console.log(err);
144       this.setState({
145         isAuthenticated: false,
146         authLoading: false,
147         error: err
148       });
149     });
150 };
151
152 setAutoLogout = milliseconds => {
153   setTimeout(() => {
154     this.logoutHandler();
155   }, milliseconds);
156 };
157
158 errorHandler = () => {
159   this.setState({ error: null });
160 };
161
162 render() {
163   let routes = (
164     <Switch>
165     <Route

```

```

166     path="/"
167     exact
168     render={props => (
169         <LoginPage
170             {...props}
171             onLogin={this.loginHandler}
172             loading={this.state.authLoading}
173         />
174     )}
175 />
176 <Route
177     path="/signup"
178     exact
179     render={props => (
180         <SignupPage
181             {...props}
182             onSignup={this.signupHandler}
183             loading={this.state.authLoading}
184         />
185     )}
186 />
187 <Redirect to="/" />
188 </Switch>
189 );
190 if (this.state.isAuth) {
191     routes = (
192         <Switch>
193             <Route
194                 path="/"
195                 exact
196                 render={props => (
197                     <FeedPage userId={this.state.userId} token={this.state.token} />
198                 )}
199             />
200             <Route
201                 path="/:postId"
202                 render={props => (
203                     <SinglePostPage
204                         {...props}
205                         userId={this.state.userId}
206                         token={this.state.token}
207                     />
208                 )}
209             />
210             <Redirect to="/" />
211         </Switch>
212     );
213 }
214 return (
215     <Fragment>
216         {this.state.showBackdrop && (
217             <Backdrop onClick={this.backdropClickHandler} />
218         )}
219         <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
220         <Layout
221             header={

```

```

222     <Toolbar>
223       <MainNavigation
224         onOpenMobileNav={this.mobileNavHandler.bind(this, true)}
225         onLogout={this.logoutHandler}
226         isAuth={this.state.isAuth}
227       />
228     </Toolbar>
229   }
230   mobileNav={
231     <MobileNavigation
232       open={this.state.showMobileNav}
233       mobile
234       onChooseItem={this.mobileNavHandler.bind(this, false)}
235       onLogout={this.logoutHandler}
236       isAuth={this.state.isAuth}
237     />
238   }
239 />
240 {routes}
241 </Fragment>
242 );
243 }
244 }
245
246 export default withRouter(App);
247

```

* Chapter 387: Using & Validating The Token

1. update
 - ./middleware/is-auth.js(b)
 - ./src/pages/Feed/Feed.js(f)
 - ./routes/feed.js(b)


```
40 loadPosts = direction => {
41   if (direction) {
42     this.setState({ postsLoading: true, posts: [] });
43   }
44   let page = this.state.postPage;
45   if (direction === 'next') {
46     page++;
47     this.setState({ postPage: page });
48   }
49   if (direction === 'previous') {
50     page--;
51     this.setState({ postPage: page });
52   }
53   fetch('http://localhost:8080/feed/posts?page=' + page, {
54     headers: {
55       Authorization: 'Bearer ' + this.props.token
56     }
57   })
58   .then(res => {
59     if (res.status !== 200) {
60       throw new Error('Failed to fetch posts.');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: node

Note that the development build is not optimized.
To create a production build, use `npm run build`.

```
53 fetch('http://localhost:8080/feed/posts?page=' + page)
54   .then(res => {
55     if (res.status !== 200) {
56       throw new Error('Failed to fetch posts.');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Compiling...

- let me show you how this fails. if i remove this header, so this extra configuration on my getPosts request in the frontend.

Feed Node

MessageNode

Feed

Logout

Your status

UPDATE

NEW POST

Application

Manifest

Service Workers

Clear storage

Storage

Local Storage

http://localhost

Session Storage

IndexedDB

Web SQL

Cookies

Cache

Cache Storage

Application Cache

Frames

top

Key

Value

expiryDate

2018-10-10T10:31:01.283Z

token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...

userId

5bbdc34b1b55de6528f5eabc

1

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpX...

Line 1, Column 1

Feed Node

MessageNode

Feed

Logout

Your status

UPDATE

NEW POST

Console

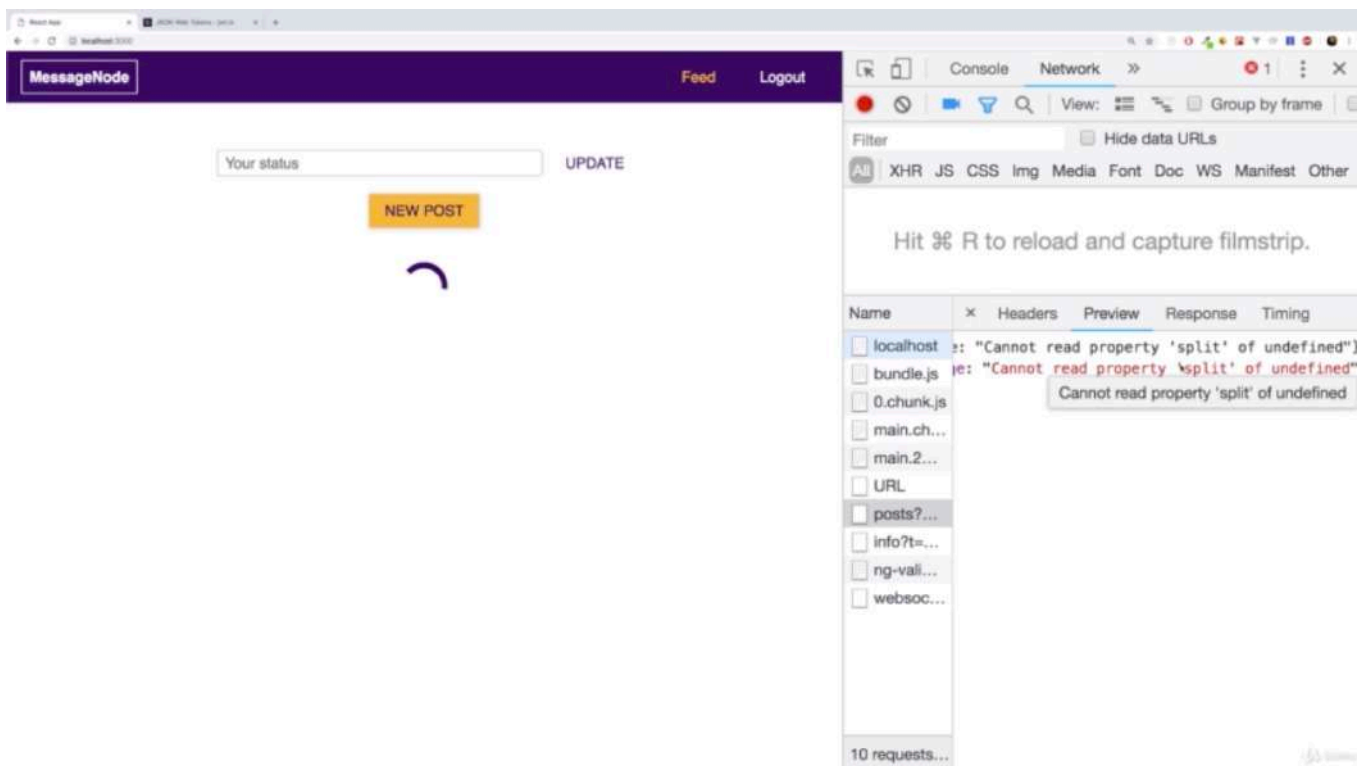
top

Filter

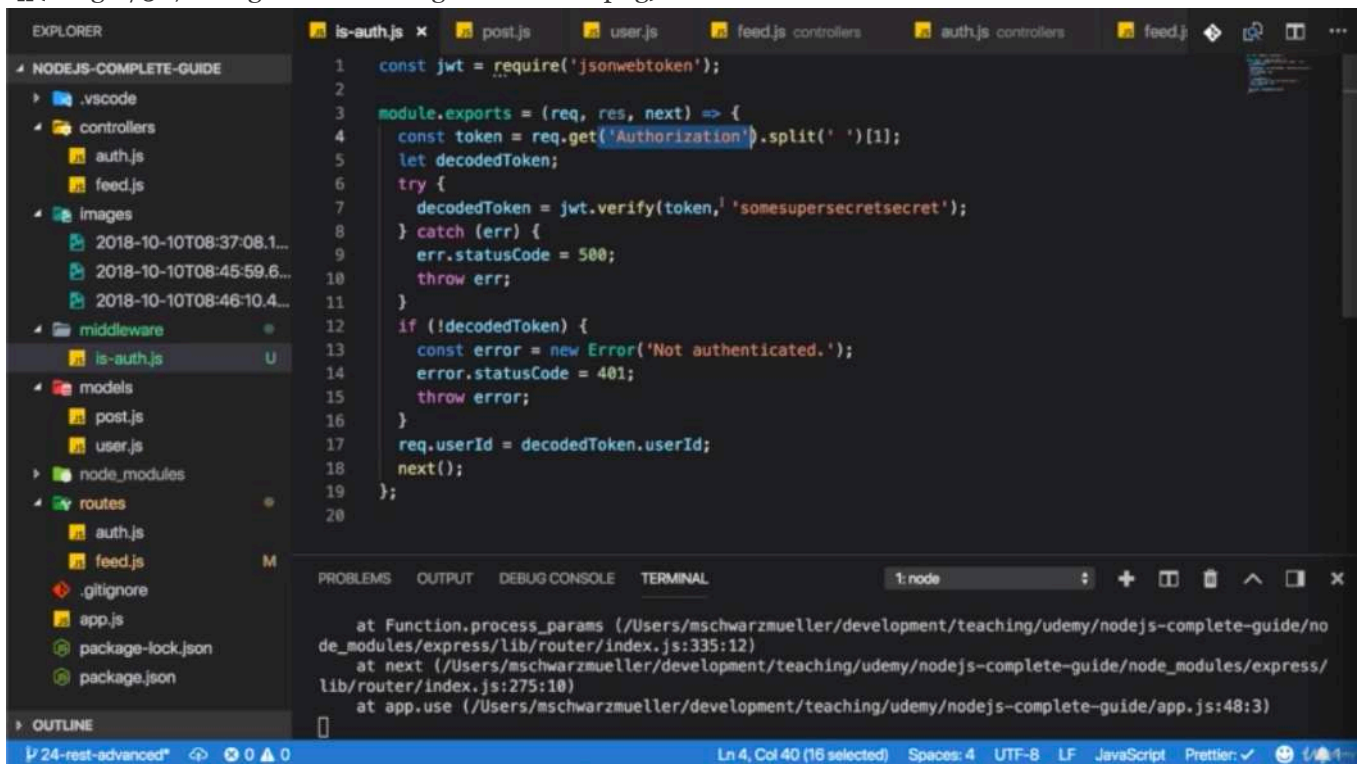
Default levels

> GET http://localhost:8080/feed/posts?age=1 500 (Internal Server Error) Feed.js:53

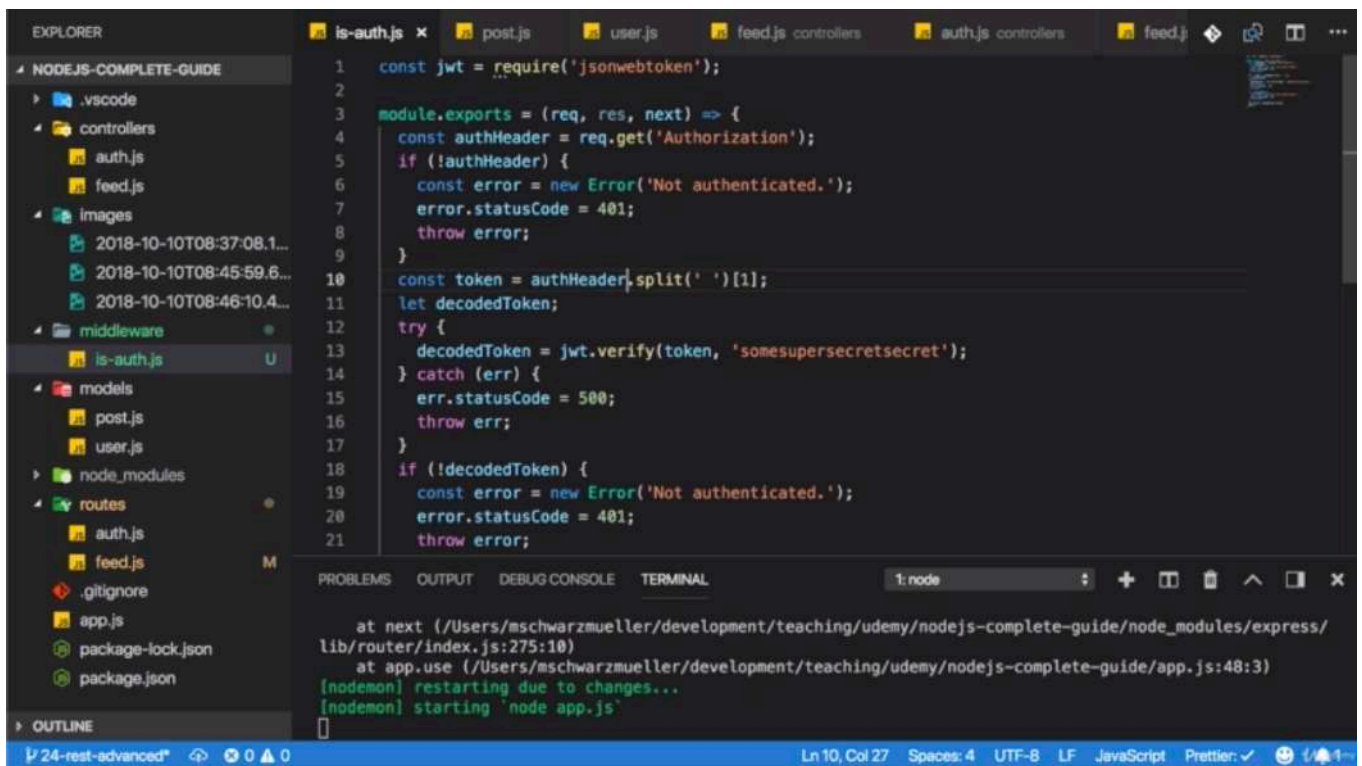
>



- if i remove that and i reload the page, i get failed to fetch posts and in the console log, i also see that here. because it fails to read split of undefined.



- we get a technical error because in my middleware, it can't get anything from that header because the header is not defined.



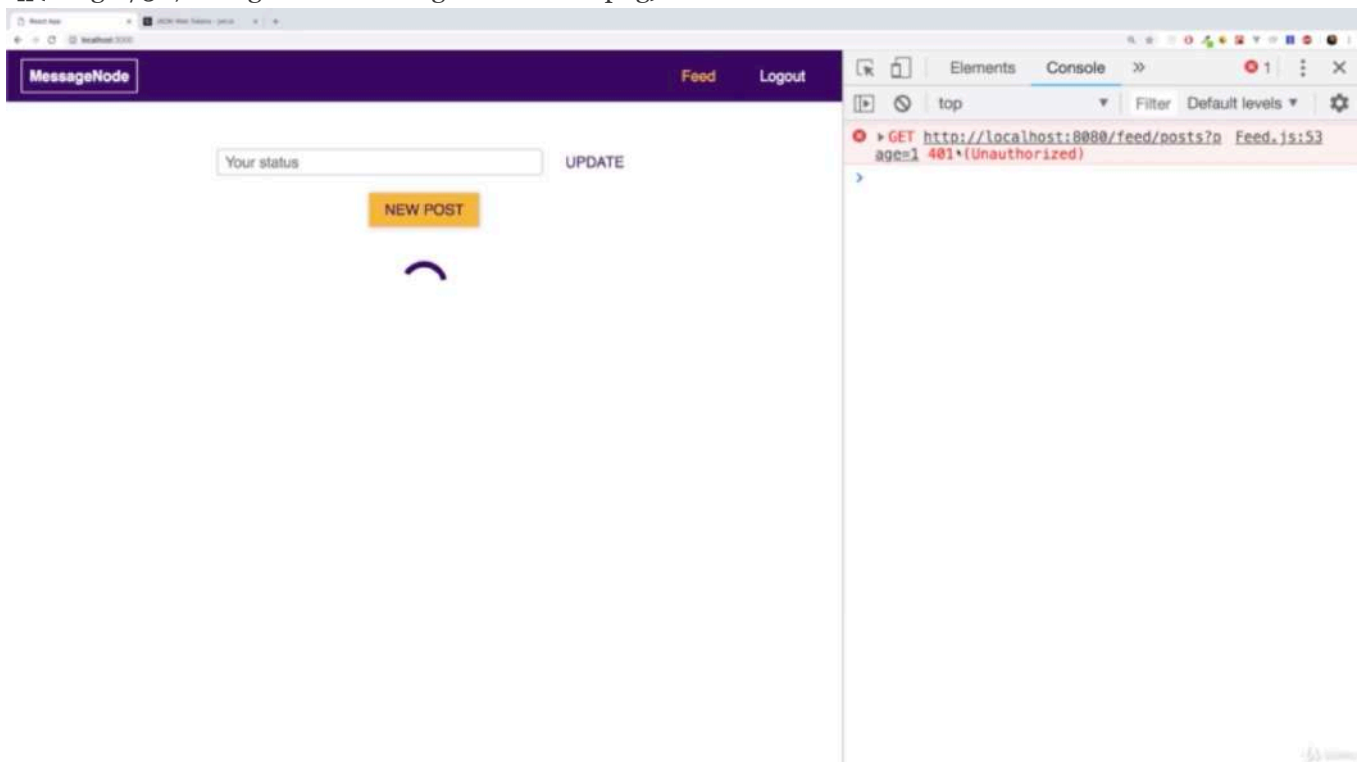
The screenshot shows the VS Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `is-auth.js` file with the following code:

```
1 const jwt = require('jsonwebtoken');
2
3 module.exports = (req, res, next) => {
4   const authHeader = req.get('Authorization');
5   if (!authHeader) {
6     const error = new Error('Not authenticated. ');
7     error.statusCode = 401;
8     throw error;
9   }
10  const token = authHeader.split(' ')[1];
11  let decodedToken;
12  try {
13    decodedToken = jwt.verify(token, 'somesupersecretsecret');
14  } catch (err) {
15    err.statusCode = 500;
16    throw err;
17  }
18  if (!decodedToken) {
19    const error = new Error('Not authenticated. ');
20    error.statusCode = 401;
21    throw error;
22  }
```

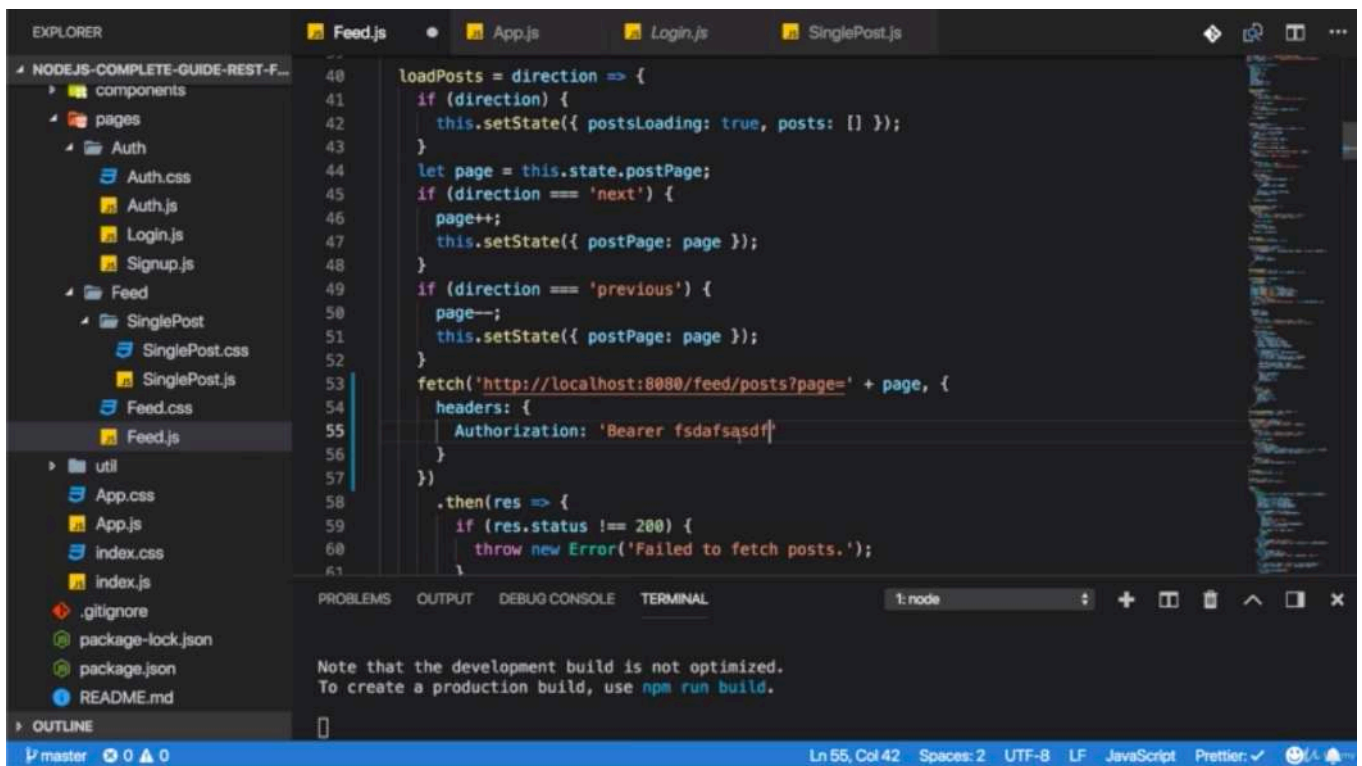
The terminal window at the bottom shows the following output:

```
at next (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/node_modules/express/lib/router/index.js:275:10)
at app.use (/Users/mschwarzmueller/development/teaching/udemy/nodejs-complete-guide/app.js:48:3)
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
```

- therefore we could add an extra check here and first of all get the 'authHeader' would then be 'req.get('Authorization')'



- we still get error but technically we now have a 401 error which is better.



```
40 loadPosts = direction => {
41   if (direction) {
42     this.setState({ postsLoading: true, posts: [] });
43   }
44   let page = this.state.postPage;
45   if (direction === 'next') {
46     page++;
47     this.setState({ postPage: page });
48   }
49   if (direction === 'previous') {
50     page--;
51     this.setState({ postPage: page });
52   }
53   fetch('http://localhost:8080/feed/posts?page=' + page, {
54     headers: {
55       Authorization: 'Bearer fsdafsasdrf'
56     }
57   })
58   .then(res => {
59     if (res.status !== 200) {
60       throw new Error('Failed to fetch posts.');

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL



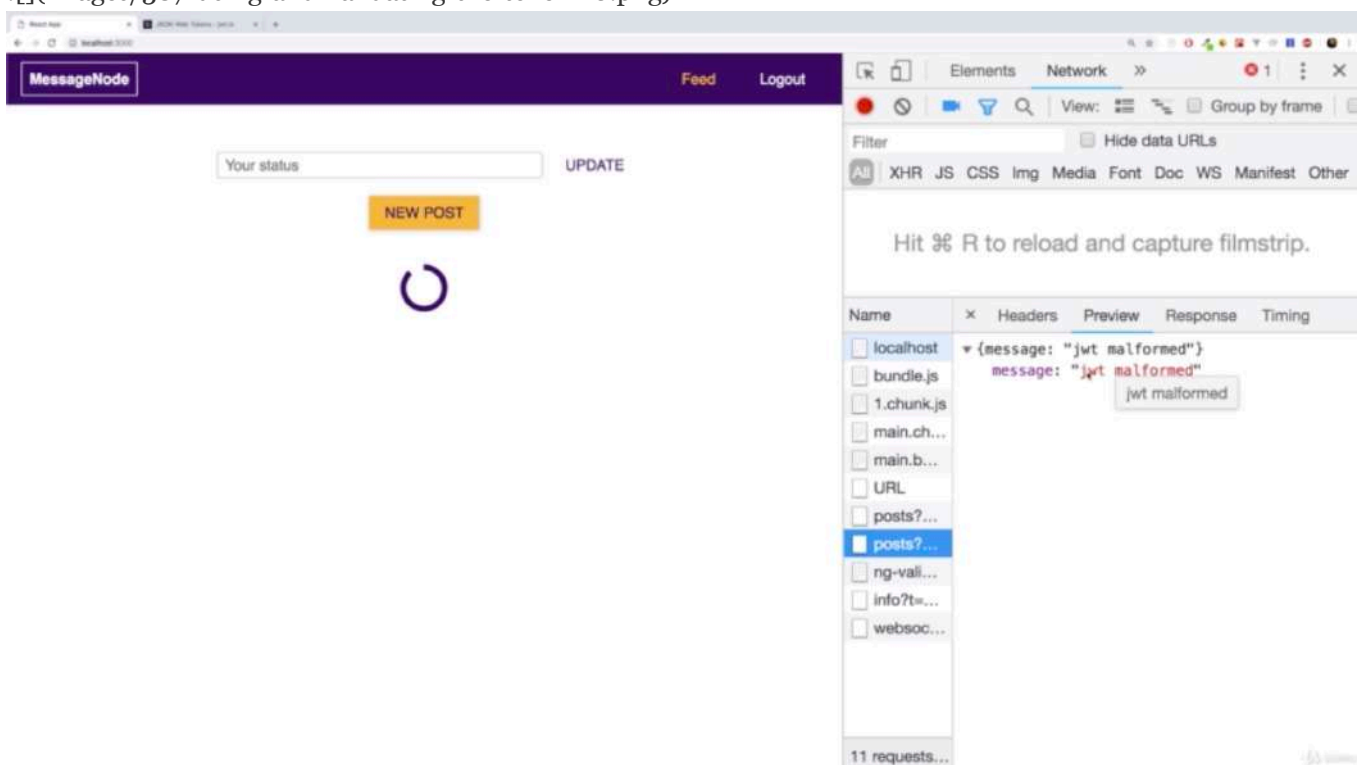
1: node



Note that the development build is not optimized.  
To create a production build, use npm run build.


```

- if i add the some rubbish which is not a valid token,



- now if that reload it still fails and it fails because not the jsonwebtoken package has a problem with the format of our token.

- so now let me revert this and add the valid token

- and if i reload, now our posts load because now we are validating this and we have a valid token indeed.

```
1 //../middleware/auth.js(b)
2
3 const jwt = require('jsonwebtoken');
4
5 module.exports = (req, res, next) => {
6   const authHeader = req.get('Authorization');
7   if(!authHeader) {
8     const error = new Error('Not authenticated.');
```

```

11 }
12 const token = authHeader.split(' ')[1];
13 let decodedToken;
14 try {
15     /**'jwt.verify()' will both decode and verify your token.
16     * 'jwt.decode()' will have decode token
17     * but this will only decode it
18     * and not check if it's valid.
19     * so definitely use 'verify'
20     *
21     * and then you pass in that token
22     * which you extracted from the header
23     * and then the 'secret'
24     * and that has to be the same secret you used for signing the token.
25     * so the same secret you used in your ./controllers/auth.js(b) file,
26     * 'somesupersecretsecret' is the secret which we will use for validating the token
27     */
28     decodedToken = jwt.verify(token, 'somesupersecretsecret');
29 } catch (err) {
30     err.statusCode = 500;
31     throw err;
32 }
33 if (!decodedToken) {
34     const error = new Error('Not authenticated. ');
35     error.statusCode = 401;
36     throw error;
37 }
38 /**if we make it past this if check,
39 * we know that we have a valid token
40 * however and that we were able to decode it,
41 * i will extract some information from the token,
42 * the 'userId' and i will store it in the request
43 * so that i can use it in toehr places
44 * where this request will go, like in my routes
45 * and there i will just access the decodedToken
46 * so i can now access my userId field
47 * which i stored in the token.
48 */
49 req.userId = decodedToken.userId;
50 next();
51 };

```

```

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

```

```

16     isEditing: false,
17     posts: [],
18     totalPosts: 0,
19     editPost: null,
20     status: '',
21     postPage: 1,
22     postsLoading: true,
23     editLoading: false
24 };
25
26 componentDidMount() {
27     fetch('URL')
28         .then(res => {
29             if (res.status !== 200) {
30                 throw new Error('Failed to fetch user status.');
```

31 }
32 return res.json();
33 })
34 .then(resData => {
35 this.setState({ status: resData.status });
36 })
37 .catch(this.catchError);
38
39 this.loadPosts();
40 }
41
42 loadPosts = direction => {
43 if (direction) {
44 this.setState({ postsLoading: true, posts: [] });
45 }
46 let page = this.state.postPage;
47 if (direction === 'next') {
48 page++;
49 this.setState({ postPage: page });
50 }
51 if (direction === 'previous') {
52 page--;
53 this.setState({ postPage: page });
54 }
55 /**using headers keeps URLs beautiful
56 * and header makes a lot of sense for meta information like the token
57 */
58 fetch('http://localhost:8080/feed/posts?page=' + page, {
59 headers: {
60 /**'Authorization' header the official header
61 * use for passing authentication information to the backend
62 * remember that on the backend in the app.js(b) file
63 * where we added our course headers,
64 * i did enable the authorization header
65 * 'res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization')'
66 * you need to have that enabled for this to work.
67 */
68 /**'Bearer ' is just a convention to kind of identify
69 * that the type of token you have
70 * and the bearer token is simply an authentication token,
71 * you typically use bearer for JSON Web Token.

```

72     * but you could work without Bearer
73     * but it's a common convention
74     * and therefore i wanna keep that convention.
75     */
76     Authorization : 'Bearer ' + this.props.token
77   }
78 })
79   .then(res => {
80     if (res.status !== 200) {
81       throw new Error('Failed to fetch posts.');

```

```

128
129 finishEditHandler = postData => {
130   this.setState({
131     editLoading: true
132   });
133   // Set up data (with image!)
134   let url = 'URL';
135   if (this.state.editPost) {
136     url = 'URL';
137   }
138
139   fetch(url)
140     .then(res => {
141       if (res.status !== 200 && res.status !== 201) {
142         throw new Error('Creating or editing a post failed!');
143       }
144       return res.json();
145     })
146     .then(resData => {
147       const post = {
148         _id: resData.post._id,
149         title: resData.post.title,
150         content: resData.post.content,
151         creator: resData.post.creator,
152         createdAt: resData.post.createdAt
153       };
154       this.setState(prevState => {
155         let updatedPosts = [...prevState.posts];
156         if (prevState.editPost) {
157           const postIndex = prevState.posts.findIndex(
158             p => p._id === prevState.editPost._id
159           );
160           updatedPosts[postIndex] = post;
161         } else if (prevState.posts.length < 2) {
162           updatedPosts = prevState.posts.concat(post);
163         }
164         return {
165           posts: updatedPosts,
166           isEditing: false,
167           editPost: null,
168           editLoading: false
169         };
170       });
171     })
172     .catch(err => {
173       console.log(err);
174       this.setState({
175         isEditing: false,
176         editPost: null,
177         editLoading: false,
178         error: err
179       });
180     });
181 };
182
183 statusInputChangeHandler = (input, value) => {

```

```

184     this.setState({ status: value });
185 };
186
187 deletePostHandler = postId => {
188     this.setState({ postsLoading: true });
189     fetch('URL')
190         .then(res => {
191             if (res.status !== 200 && res.status !== 201) {
192                 throw new Error('Deleting a post failed!');
193             }
194             return res.json();
195         })
196         .then(resData => {
197             console.log(resData);
198             this.setState(prevState => {
199                 const updatedPosts = prevState.posts.filter(p => p._id !== postId);
200                 return { posts: updatedPosts, postsLoading: false };
201             });
202         })
203         .catch(err => {
204             console.log(err);
205             this.setState({ postsLoading: false });
206         });
207 };
208
209 errorHandler = () => {
210     this.setState({ error: null });
211 };
212
213 catchError = error => {
214     this.setState({ error: error });
215 };
216
217 render() {
218     return (
219         <Fragment>
220             <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
221             <FeedEdit
222                 editing={this.state.isEditing}
223                 selectedPost={this.state.editPost}
224                 loading={this.state.editLoading}
225                 onCancelEdit={this.cancelEditHandler}
226                 onFinishEdit={this.finishEditHandler}
227             />
228             <section className="feed__status">
229                 <form onSubmit={this.statusUpdateHandler}>
230                     <Input
231                         type="text"
232                         placeholder="Your status"
233                         control="input"
234                         onChange={this.statusInputChangeHandler}
235                         value={this.state.status}
236                     />
237                     <Button mode="flat" type="submit">
238                         Update
239                     </Button>

```

```

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

```

```

1  //./routes/feed.js(b)
2
3  const express = require('express');
4  const { body } = require('express-validator/check');
5
6  const feedController = require('../controllers/feed');
7  const isAuth = require('../middleware/is-auth');
8
9  const router = express.Router();
10

```



```

11 // GET /feed/posts
12 router.get('/posts', isAuth, feedController.getPosts);
13
14 // POST /feed/post
15 router.post(
16   '/post',
17   [
18     body('title')
19       .trim()
20       .isLength({ min: 5 }),
21     body('content')
22       .trim()
23       .isLength({ min: 5 })
24   ],
25   feedController.createPost
26 );
27
28 router.get('/post/:postId', feedController.getPost);
29
30 router.put('/post/:postId', [
31   body('title')
32     .trim()
33     .isLength({ min: 5 }),
34   body('content')
35     .trim()
36     .isLength({ min: 5 })
37 ], feedController.updatePost
38 );
39
40 router.delete('/post/:postId', feedController.deletePost);
41
42 module.exports = router;
43

```

* Chapter 388: Adding Auth Middleware To All Routes

1. update
 - ./routes/feed.js(b)
 - ./src/pages/Feed/Feed.js(f)
 - ./src/pages/Feed/SinglePost/SinglePost.js(f)

```

1 //./routes/feed.js(b)
2
3 const express = require('express');
4 const { body } = require('express-validator/check');
5
6 const feedController = require('../controllers/feed');
7 const isAuth = require('../middleware/is-auth');
8
9 const router = express.Router();
10
11 // GET /feed/posts
12 router.get('/posts', isAuth, feedController.getPosts);
13
14 // POST /feed/post

```

```

15 router.post(
16   '/post',
17   isAuth,
18   [
19     body('title')
20       .trim()
21       .isLength({ min: 5 }),
22     body('content')
23       .trim()
24       .isLength({ min: 5 })
25   ],
26   feedController.createPost
27 );
28
29 router.get('/post/:postId', isAuth, feedController.getPost);
30
31 router.put('/post/:postId',
32   isAuth,
33   [
34     body('title')
35       .trim()
36       .isLength({ min: 5 }),
37     body('content')
38       .trim()
39       .isLength({ min: 5 })
40   ], feedController.updatePost
41 );
42
43 router.delete('/post/:postId', isAuth, feedController.deletePost);
44
45 module.exports = router;
46

```

```

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

```

```

25
26 componentDidMount() {
27   fetch('URL')
28     .then(res => {
29       if (res.status !== 200) {
30         throw new Error('Failed to fetch user status.');

```

```

81 statusUpdateHandler = event => {
82   event.preventDefault();
83   fetch('URL')
84     .then(res => {
85       if (res.status !== 200 && res.status !== 201) {
86         throw new Error("Can't update status!");
87       }
88       return res.json();
89     })
90     .then(resData => {
91       console.log(resData);
92     })
93     .catch(this.catchError);
94   };
95
96 newPostHandler = () => {
97   this.setState({ isEditing: true });
98   };
99
100 startEditPostHandler = postId => {
101   this.setState(prevState => {
102     const loadedPost = { ...prevState.posts.find(p => p._id === postId) };
103
104     return {
105       isEditing: true,
106       editPost: loadedPost
107     };
108   });
109   };
110
111 cancelEditHandler = () => {
112   this.setState({ isEditing: false, editPost: null });
113   };
114
115 finishEditHandler = postData => {
116   this.setState({
117     editLoading: true
118   });
119   const formData = new FormData();
120   formData.append('title', postData.title);
121   formData.append('content', postData.content);
122   formData.append('image', postData.image);
123   let url = 'http://localhost:8080/feed/post';
124   let method = 'POST';
125   if (this.state.editPost) {
126     url = 'http://localhost:8080/feed/post/' + this.state.editPost._id;
127     method = 'PUT';
128   }
129
130   fetch(url, {
131     method: method,
132     body: formData,
133     headers: {
134       Authorization: 'Bearer ' + this.props.token
135     }
136   })

```

```

137 .then(res => {
138   if (res.status !== 200 && res.status !== 201) {
139     throw new Error('Creating or editing a post failed!');
140   }
141   return res.json();
142 })
143 .then(resData => {
144   console.log(resData);
145   const post = {
146     _id: resData.post._id,
147     title: resData.post.title,
148     content: resData.post.content,
149     creator: resData.post.creator,
150     createdAt: resData.post.createdAt
151   };
152   this.setState(prevState => {
153     let updatedPosts = [...prevState.posts];
154     if (prevState.editPost) {
155       const postIndex = prevState.posts.findIndex(
156         p => p._id === prevState.editPost._id
157       );
158       updatedPosts[postIndex] = post;
159     } else if (prevState.posts.length < 2) {
160       updatedPosts = prevState.posts.concat(post);
161     }
162     return {
163       posts: updatedPosts,
164       isEditing: false,
165       editPost: null,
166       editLoading: false
167     };
168   });
169 })
170 .catch(err => {
171   console.log(err);
172   this.setState({
173     isEditing: false,
174     editPost: null,
175     editLoading: false,
176     error: err
177   });
178 });
179 };
180
181 statusInputChangeHandler = (input, value) => {
182   this.setState({ status: value });
183 };
184
185 deletePostHandler = postId => {
186   this.setState({ postsLoading: true });
187   fetch('http://localhost:8080/feed/post/' + postId, {
188     method: 'DELETE',
189     headers: {
190       Authorization: 'Bearer ' + this.props.token
191     }
192   })

```

```

193 .then(res => {
194   if (res.status !== 200 && res.status !== 201) {
195     throw new Error('Deleting a post failed!');
196   }
197   return res.json();
198 })
199 .then(resData => {
200   console.log(resData);
201   this.setState(prevState => {
202     const updatedPosts = prevState.posts.filter(p => p._id !== postId);
203     return { posts: updatedPosts, postsLoading: false };
204   });
205 })
206 .catch(err => {
207   console.log(err);
208   this.setState({ postsLoading: false });
209 });
210 };
211
212 errorHandler = () => {
213   this.setState({ error: null });
214 };
215
216 catchError = error => {
217   this.setState({ error: error });
218 };
219
220 render() {
221   return (
222     <Fragment>
223       <ErrorHandler error={this.state.error} onHandle={this.errorHandler} />
224       <FeedEdit
225         editing={this.state.isEditing}
226         selectedPost={this.state.editPost}
227         loading={this.state.editLoading}
228         onCancelEdit={this.cancelEditHandler}
229         onFinishEdit={this.finishEditHandler}
230       />
231       <section className="feed__status">
232         <form onSubmit={this.statusUpdateHandler}>
233           <Input
234             type="text"
235             placeholder="Your status"
236             control="input"
237             onChange={this.statusInputChangeHandler}
238             value={this.state.status}
239           />
240           <Button mode="flat" type="submit">
241             Update
242           </Button>
243         </form>
244       </section>
245       <section className="feed__control">
246         <Button mode="raised" design="accent" onClick={this.newPostHandler}>
247           New Post
248         </Button>

```

```

249     </section>
250     <section className="feed">
251       {this.state.postsLoading && (
252         <div style={{ textAlign: 'center', marginTop: '2rem' }}>
253           <Loader />
254         </div>
255       )}
256       {this.state.posts.length <= 0 && !this.state.postsLoading ? (
257         <p style={{ textAlign: 'center' }}>No posts found.</p>
258       ) : null}
259       {!this.state.postsLoading && (
260         <Paginator
261           onPrevious={this.loadPosts.bind(this, 'previous')}
262           onNext={this.loadPosts.bind(this, 'next')}
263           lastPage={Math.ceil(this.state.totalPosts / 2)}
264           currentPage={this.state.postPage}
265         >
266           {this.state.posts.map(post => (
267             <Post
268               key={post._id}
269               id={post._id}
270               author={post.creator}
271               date={new Date(post.createdAt).toLocaleDateString('en-US')}
272               title={post.title}
273               image={post.imageUrl}
274               content={post.content}
275               onStartEdit={this.startEditPostHandler.bind(this, post._id)}
276               onDelete={this.deletePostHandler.bind(this, post._id)}
277             />
278           ))}
279         </Paginator>
280       )}
281     </section>
282   </Fragment>
283 );
284 }
285 }
286
287 export default Feed;
288

```

```

1  //./src/pages/Feed/SinglePost/SinglePost.js(f)
2
3  import React, { Component } from 'react';
4
5  import Image from '../../components/Image/Image';
6  import './SinglePost.css';
7
8  class SinglePost extends Component {
9    state = {
10      title: '',
11      author: '',
12      date: '',
13      image: '',
14      content: ''
15    };
16

```

```

17 componentDidMount() {
18   const postId = this.props.match.params.postId;
19   fetch('http://localhost:8080/feed/post/' + postId, {
20     headers: {
21       Authorization: 'Bearer ' + this.props.token
22     }
23   })
24   .then(res => {
25     if (res.status !== 200) {
26       throw new Error('Failed to fetch status');
27     }
28     return res.json();
29   })
30   .then(resData => {
31     this.setState({
32       title: resData.post.title,
33       author: resData.post.creator.name,
34       date: new Date(resData.post.createdAt).toLocaleDateString('en-US'),
35       content: resData.post.content
36     });
37   })
38   .catch(err => {
39     console.log(err);
40   });
41 }
42
43 render() {
44   return (
45     <section className="single-post">
46       <h1>{this.state.title}</h1>
47       <h2>
48         Created by {this.state.author} on {this.state.date}
49       </h2>
50       <div className="single-post__image">
51         <Image contain imageUrl={this.state.image} />
52       </div>
53       <p>{this.state.content}</p>
54     </section>
55   );
56 }
57 }
58
59 export default SinglePost;
60

```

* Chapter 389: Connecting Posts & Users

1. update

- ./models/post.js(b)
- ./models/user.js(b)
- ./controllers/feed.js(b)

The image is a composite of three screenshots illustrating a web application's state and data storage.

Top Left: Web Application Interface

The application is running on a device (likely a smartphone) and shows a 'MessageNode' screen. At the top, there's a 'Feed' button and a 'Logout' link. Below this is a status update section with a text input 'Your status' and an 'UPDATE' button. The main section is titled 'New Post' and contains three input fields: 'TITLE', 'IMAGE', and 'CONTENT'. The 'IMAGE' field has a 'Choose file' button and shows 'duck.jpg' as the selected file. Below the 'IMAGE' field, it says 'Please choose an image.' The 'CONTENT' field is a large text area. At the bottom right of the 'New Post' form are 'CANCEL' and 'LOADING...' buttons.

Top Right: Browser Network Tab

The browser's Network tab is open, showing a list of requests. The first request is 'data:image/...' (a data URI for the duck.jpg image). The table below shows the details of the requests:

| Name | Status | Type | Size | Time | Waterfall |
|-----------------|-----------|-----------|-------------|------|-----------|
| localhost | 200 | document | 1.1 KB | 19 | |
| bundle.js | 200 | script | 6.5 KB | 30 | |
| 0.chunk.js | 200 | script | 389 KB | 21 | |
| main.chunk.js | 200 | script | 17.0 KB | 58 | |
| main.0eb7b9... | 200 | script | 1.8 KB | 74 | |
| URL | 200 | font | 1.1 KB | 5 | |
| posts?page=1 | 200 | font | 389 B | 3 | |
| posts?page=1 | 200 | font | 440 B | 24 | |
| ng-validate.js | 200 | font | (from d...) | 2 | |
| info?t=15391... | 200 | xhr | 368 B | 11 | |
| websocket | 101 | websocket | 0 B | Pe | |
| data:image/... | 200 | image | (from ...) | 1 | |
| post | (pending) | font | 0 B | Pe | |

13 requests | 417 KB transferred | Finish: 6.6 min | DOMContentLoaded...

Bottom: MongoDB Compass Interface

The MongoDB Compass interface shows the 'messages.posts' collection. The left sidebar lists the database 'messages' and its collections: 'posts' and 'users'. The main area displays the 'messages.posts' collection with a single document:

```
{
  "_id": "ObjectId('5bbdccc44e93c4369f160d6e5')",
  "title": "A Duck!",
  "content": "A duck, how lovely!",
  "imageUrl": "images/2018-10-10T09:54:12.241Z-duck.jpg",
  "creator": "ObjectId('5b84c34b1b55de6528f5eabc')",
  "createdAt": "2018-10-10 11:54:12.266",
  "updatedAt": "2018-10-10 11:54:12.266",
  "v": 0
}
```

My Cluster Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.8 Enterprise

messages.users

DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

Documents Aggregations Explain Plan Indexes

0 FILTER

OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

```
{
  "_id": ObjectId("5bbdc34b1b55de6528f5eabc"),
  "status": "I am new!",
  "posts": Array
    0: ObjectId("5bbdccc44e93c4369f1600e5")
  "email": "test@test.com",
  "password": "52e5125v0cLH.acg#91iE8zoRERBuAQjdKAWVybbbrQXBWJ9oYffB/A15k2",
  "name": "Maximilian",
  "__v": 1
}
```

My Cluster Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.8 Enterprise

messages.posts

DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

Documents Aggregations Explain Plan Indexes

0 FILTER

OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 1 - 1 of 1

```
{
  "_id": ObjectId("5bbdccc44e93c4369f1600e5"),
  "title": "A Duck!",
  "content": "A duck, how lovely!",
  "imageUrl": "images/2018-10-10T09:54:12.241Z-duck.jpg",
  "creator": ObjectId("5bbdc34b1b55de6528f5eabc"),
  "createdAt": 2018-10-10 11:54:12.266,
  "updatedAt": 2018-10-10 11:54:12.266,
  "__v": 0
}
```

```
1 //./models/post.js(b)
2
3 const mongoose = require('mongoose');
4 const Schema = mongoose.Schema;
5
6 const postSchema = new Schema(
7   {
8     title: {
9       type: String,
10      required: true
11    },
12    imageUrl: {
13      type: String,
14      required: true
```

```

15     },
16     content: {
17       type: String,
18       required: true
19     },
20     creator: {
21       type: Schema.Types.ObjectId,
22       ref: 'User',
23       required: true
24     }
25   },
26   { timestamps: true }
27 );
28
29 module.exports = mongoose.model('Post', postSchema);

```

```

1  //./models/user.js(b)
2
3  const mongoose = require('mongoose');
4  const Schema = mongoose.Schema;
5
6  const userSchema = new Schema({
7    email: {
8      type: String,
9      required: true
10   },
11   password: {
12     type: String,
13     required: true
14   },
15   name: {
16     type: String,
17     required: true
18   },
19   status: {
20     type: String,
21     default: 'I am new!'
22   },
23   /**we store the creator of a post in every post we create
24    * and on the ./models/user.js(b) file,
25    * we add the post to the list of posts for that user.
26    * that means we now need to adjust our ./controllers/feed.js(b) file
27    * in the place where we create new posts.
28    */
29   posts: [{
30     type: Schema.Types.ObjectId,
31     ref: 'Post'
32   }]
33 });
34
35 module.exports = mongoose.model('User', userSchema);

```

```

1  //./controllers/feed.js(b)
2
3  const fs = require('fs');
4  const path = require('path');
5

```

```

6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9  const User = require('../models/user');
10
11 exports.getPosts = (req, res, next) => {
12   const currentPage = req.query.page || 1;
13   const perPage = 2;
14   let totalItems;
15   Post.find()
16     .countDocuments()
17     .then(count => {
18       totalItems = count;
19       return Post.find()
20         .skip((currentPage - 1) * perPage)
21         .limit(perPage)
22     })
23     .then(posts => {
24       res
25         .status(200)
26         .json({
27           message: 'Fetched posts successfully.',
28           posts: posts,
29           totalItems: totalItems
30         });
31     })
32     .catch(err => {
33       if (!err.statusCode) {
34         err.statusCode = 500;
35       }
36       next(err);
37     });
38 };
39
40 exports.createPost = (req, res, next) => {
41   const errors = validationResult(req);
42   if (!errors.isEmpty()) {
43     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

```

45     throw error;
46   }
47   if (!req.file) {
48     const error = new Error('No image provided.');
```

error.statusCode = 422;

```

50     throw error;
51   }
52
53   const imageUrl = req.file.path;
54   const title = req.body.title;
55   const content = req.body.content;
56   /**remember in my ./middleware/is-auth.js(b) file,
57    * i store that in the request object,
58    *
59    *   'req.userId = decodedToken.userId'
60    *
61    * so we have that userId available here.
```

```

62  * with that information,
63  * i will set creator equal to 'req.userId'
64  * that will be string not an objectId
65  * but mongoose will take care about that and convert it.
66  * so now we create a new post assigned to that user
67  */
68
69  let creator;
70  const post = new Post({
71    title: title,
72    content: content,
73    imageUrl: imageUrl,
74    creator: req.userId
75  });
76  post
77    .save()
78    .then(result => {
79      return User.findById(req.userId);
80    })
81    .then(user => {
82      creator = user;
83      user.posts.push(post);
84      return user.save()
85    })
86    .then(result => {
87      res.status(201).json({
88        message: 'Post created successfully!',
89        post: post,
90        creator: { _id: creator._id, name: creator.name }
91      });
92    })
93    .catch(err => {
94      if (!err.statusCode) {
95        err.statusCode = 500;
96      }
97      next(err);
98    });
99  });
100
101  exports.getPost = (req, res, next) => {
102    const postId = req.params.postId;
103    Post.findById(postId)
104      .then(post => {
105        if (!post) {
106          const error = new Error('Could not find post.');
```

```

118 };
119
120 exports.updatePost = (req, res, next) => {
121   const postId = req.params.postId;
122   const errors = validationResult(req);
123   if (!errors.isEmpty()) {
124     const error = new Error('Validation failed, entered data is incorrect.');
```

125 error.statusCode = 422;
126 throw error;

```

127   }
128   const title = req.body.title;
129   const content = req.body.content;
130   let imageUrl = req.body.image;
131   if (req.file) {
132     imageUrl = req.file.path
133   }
134   if (!imageUrl) {
135     const error = new Error('No file picked.');
```

136 error.statusCode = 422;
137 throw error;

```

138   }
139   Post.findById(postId)
140     .then(post => {
141       if (!post) {
142         const error = new Error('Could not find post.');
```

143 error.statusCode = 404;
144 throw error;

```

145       }
146       if (imageUrl !== post.image) {
147         clearImage(post.imageUrl);
148       }
149       post.title = title;
150       post.imageUrl = imageUrl;
151       post.content = content;
152       return post.save();
153     })
154     .then(result => {
155       res.status(200).json({message: 'Post updated!', post: result});
156     })
157     .catch(err => {
158       if (!err.statusCode) {
159         err.statusCode = 500;
160       }
161       next(err);
162     })
163   };
164
165 exports.deletePost = (req, res, next) => {
166   const postId = req.params.postId;
167   Post.findById(postId)
168     .then(post => {
169       if (!post) {
170         const error = new Error('Could not find post.');
```

171 error.statusCode = 404;
172 throw error;

```

173       }

```

```

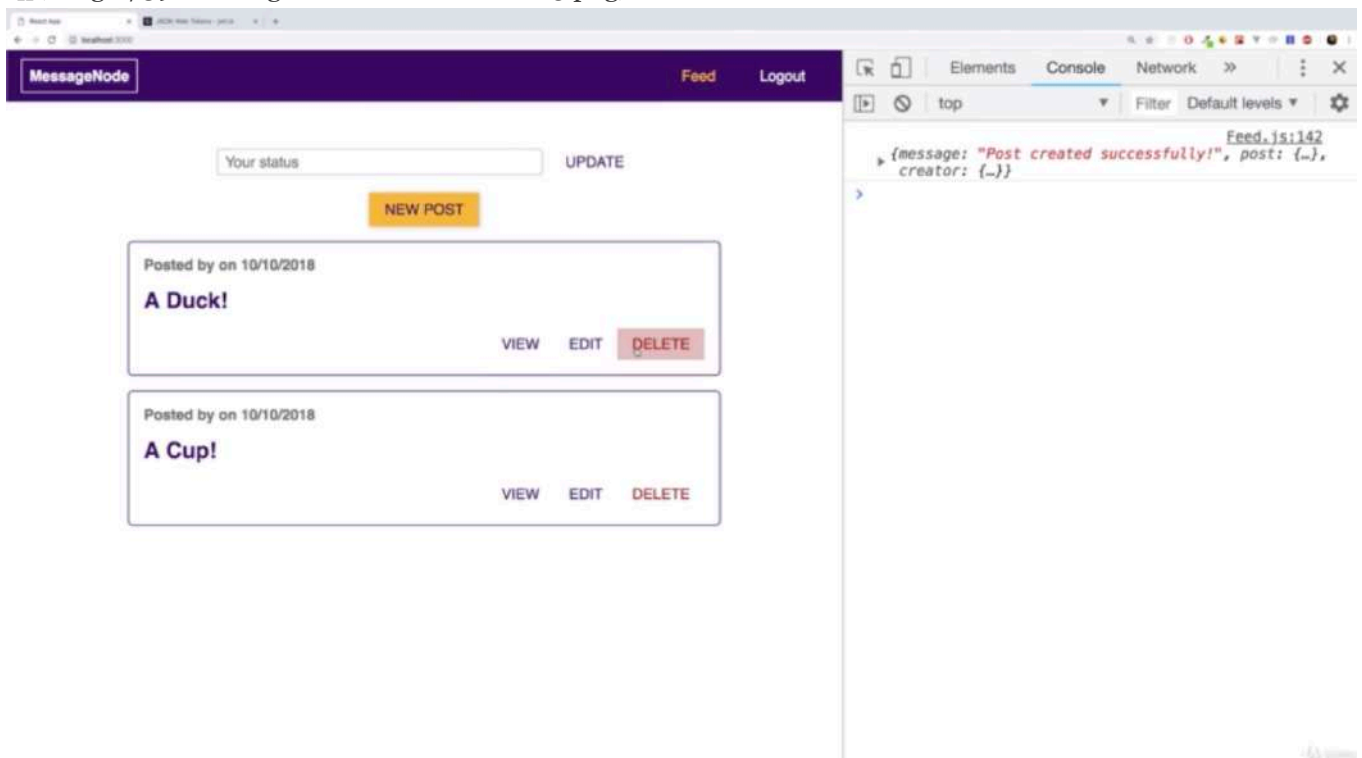
174 //Check logged in user
175 clearImage(post.imageUrl);
176 return Post.findByIdAndRemove(postId);
177 })
178 .then(result => {
179   console.log(result);
180   res.status(200).json({message: 'Deleted post.'});
181 })
182 .catch(err => {
183   if (!err.statusCode) {
184     err.statusCode = 500;
185   }
186   next(err);
187 });
188 }
189
190 const clearImage = filePath => {
191   filePath = path.join(__dirname, '..', filePath)
192   fs.unlink(filePath, err => console.log(err));
193 }
194

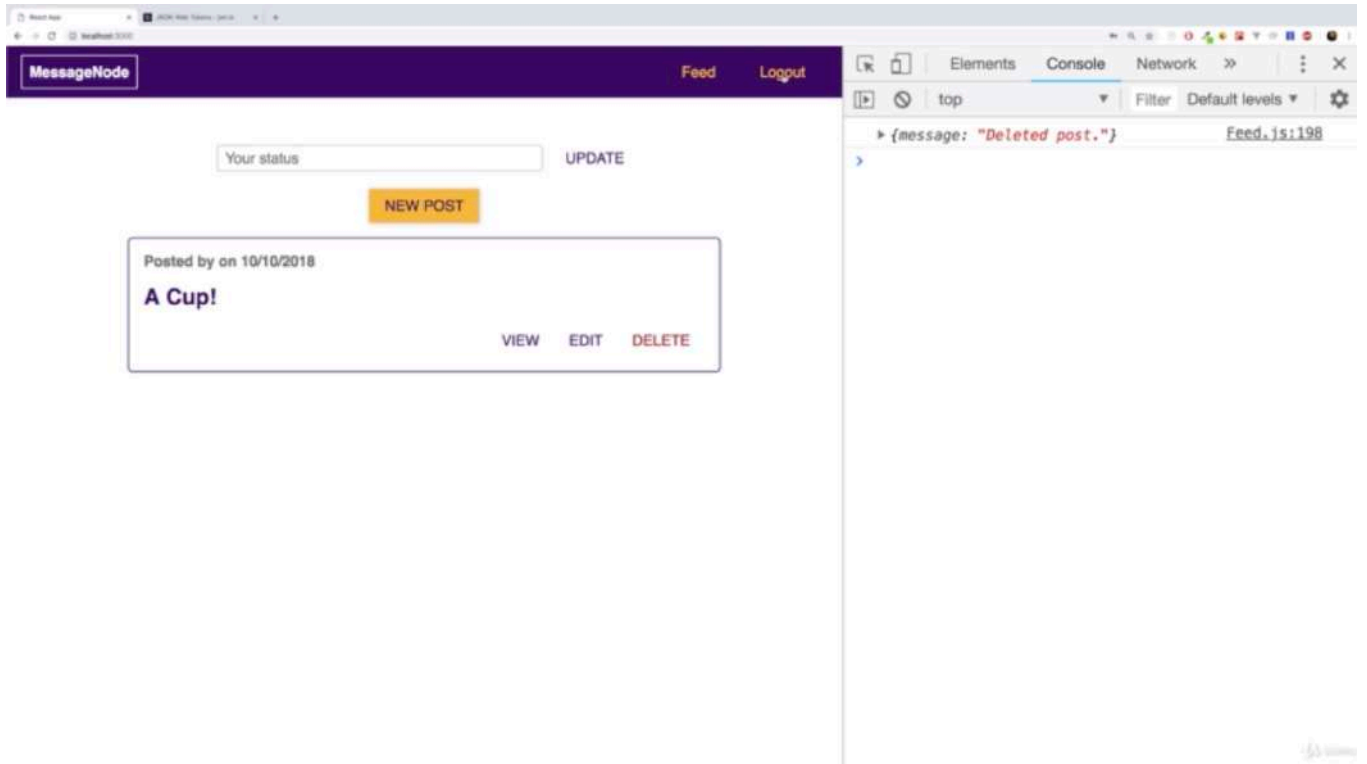
```

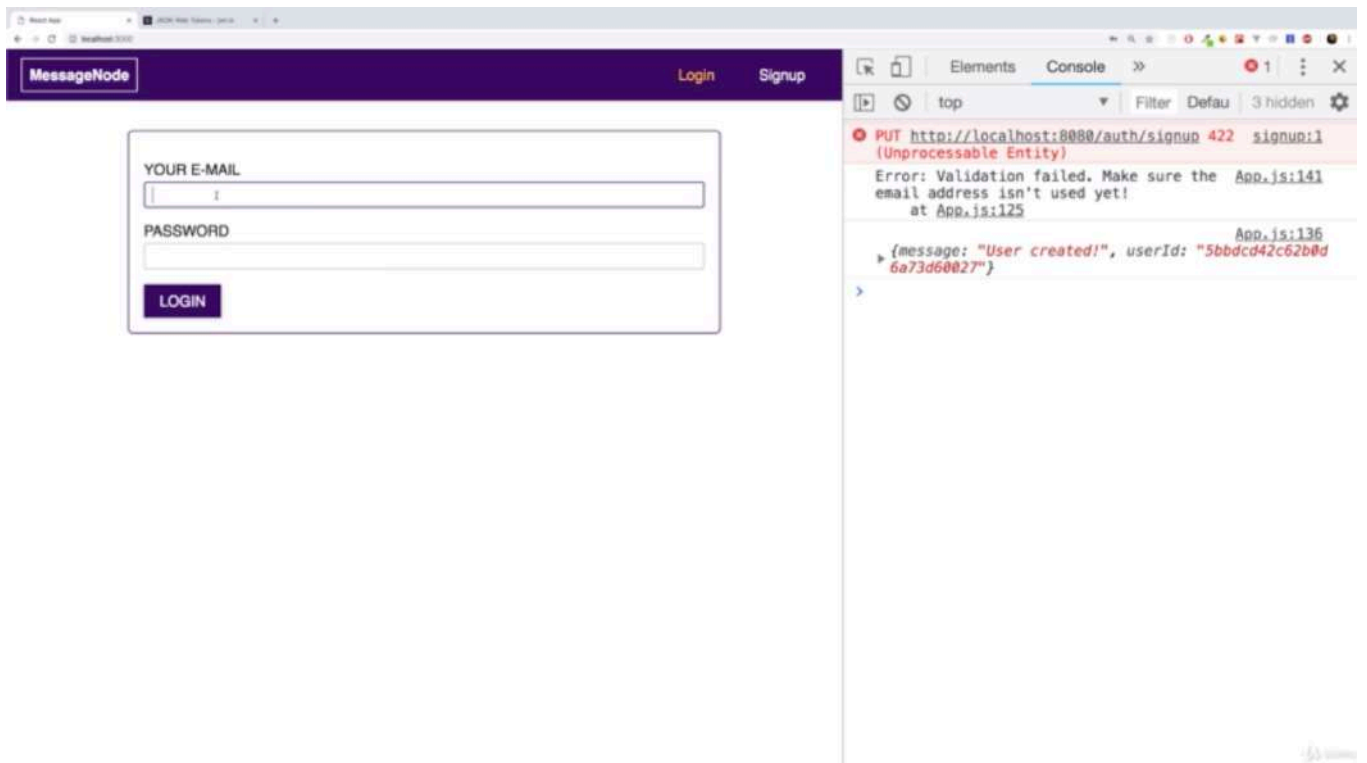
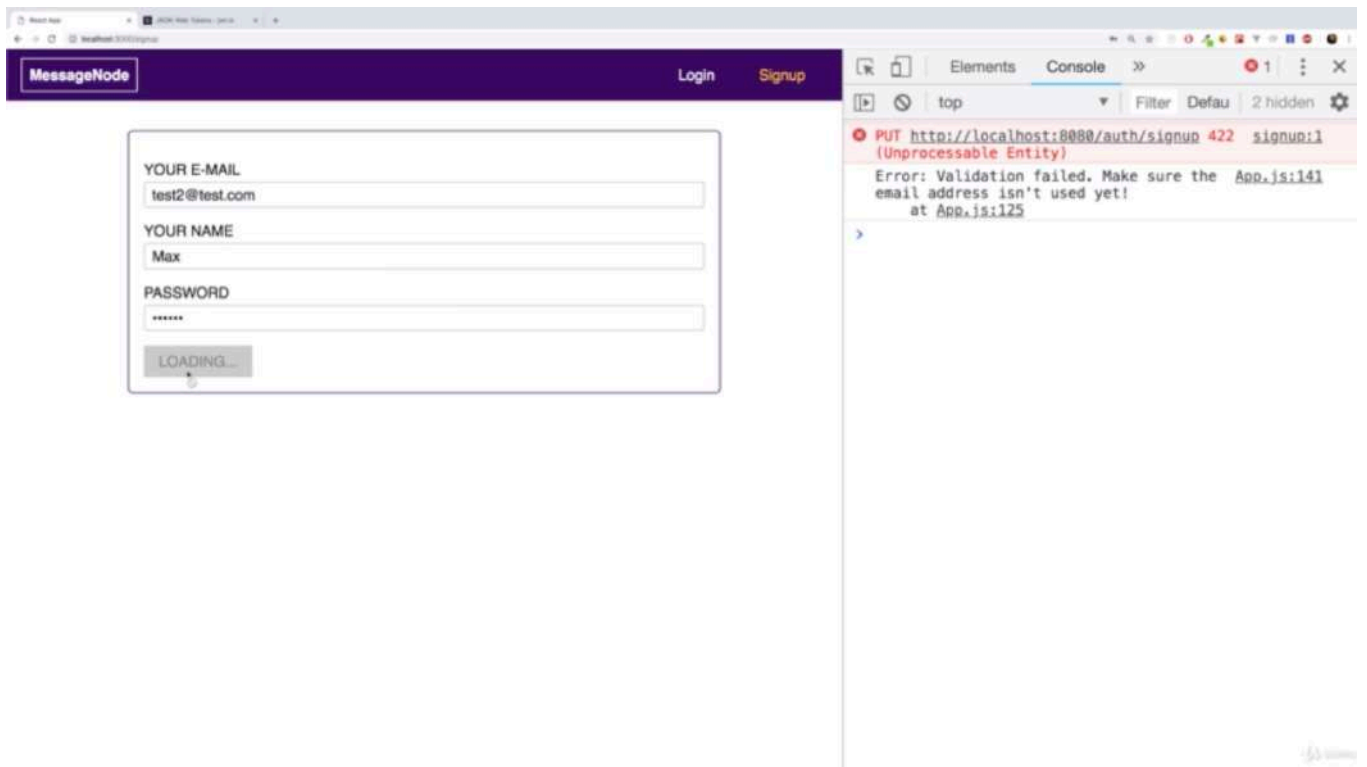
* Chapter 390: Adding Authorization Checks

1. update

- ./controllers/feed.js(b)







- let's use an unused email address and it succeeds.

MessageNode

LoginSignup

YOUR E-MAIL

test2@test.com

PASSWORD

LOGIN

Elements

Console

top

Filter

Defau

3 hidden

PUT http://localhost:8080/auth/signup 422 signup:1 (Unprocessable Entity)
Error: Validation failed. Make sure the email address isn't used yet!
at App.js:125
App.js:136
{message: "User created!", userId: "5bdbc42c62b0d6a73d60027"}

MessageNode

FeedLogout

Your status

UPDATE

NEW POST

Posted by on 10/10/2018

A Cup!

VIEW

EDIT

DELETE

Elements

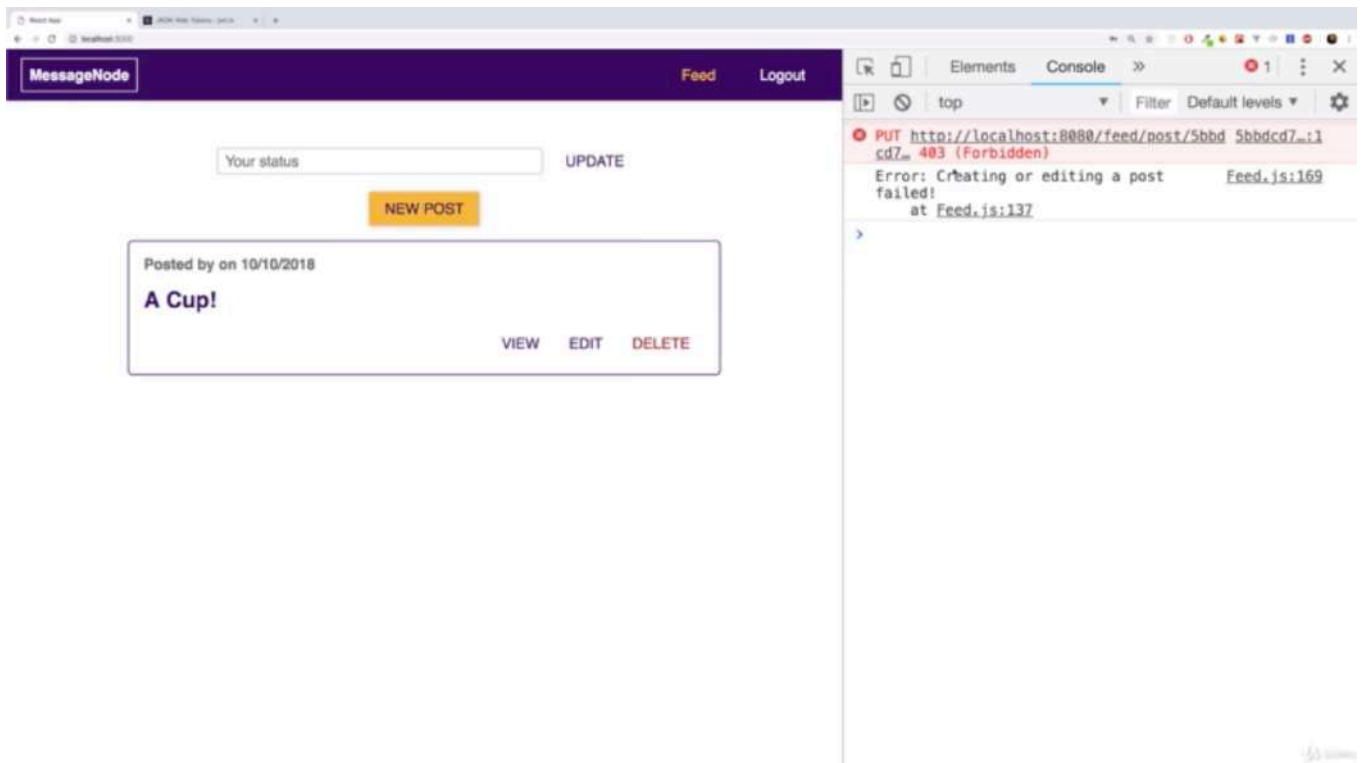
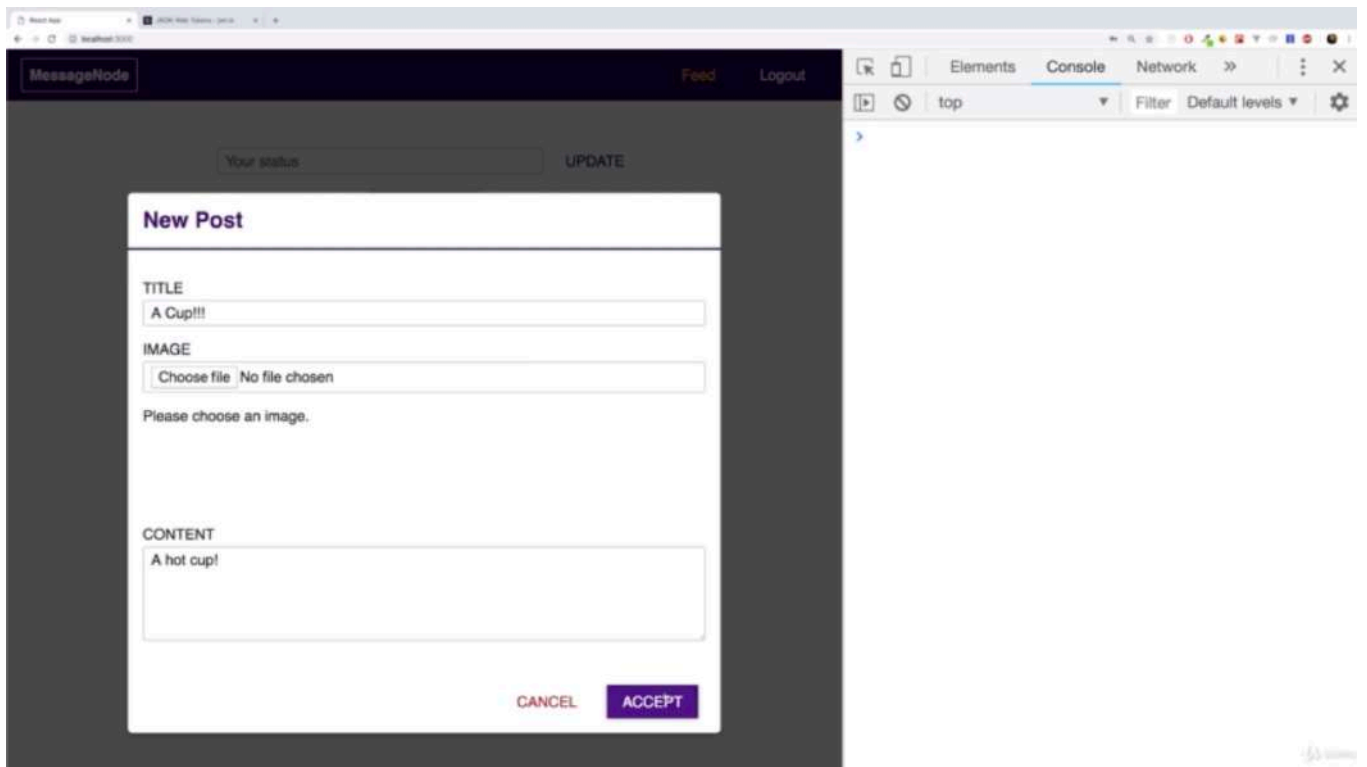
Console

Network

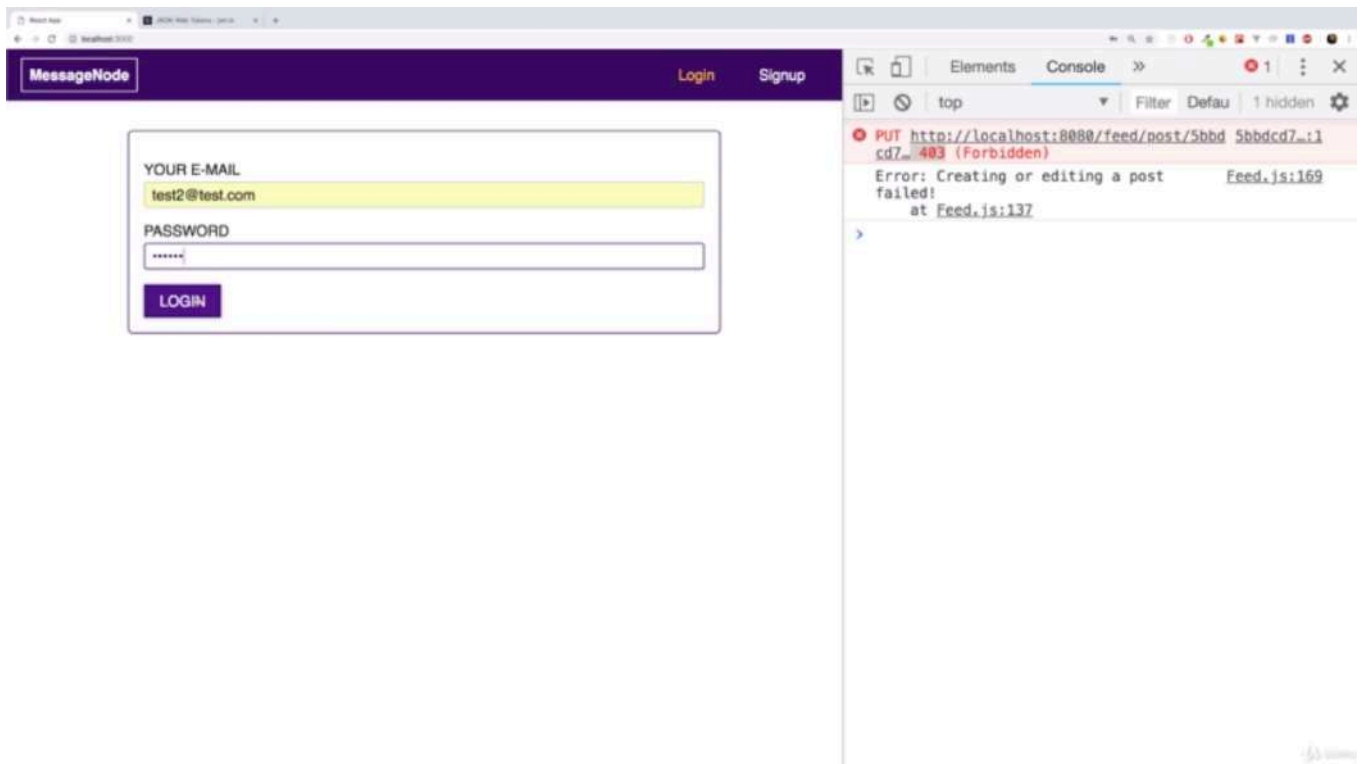
top

Filter

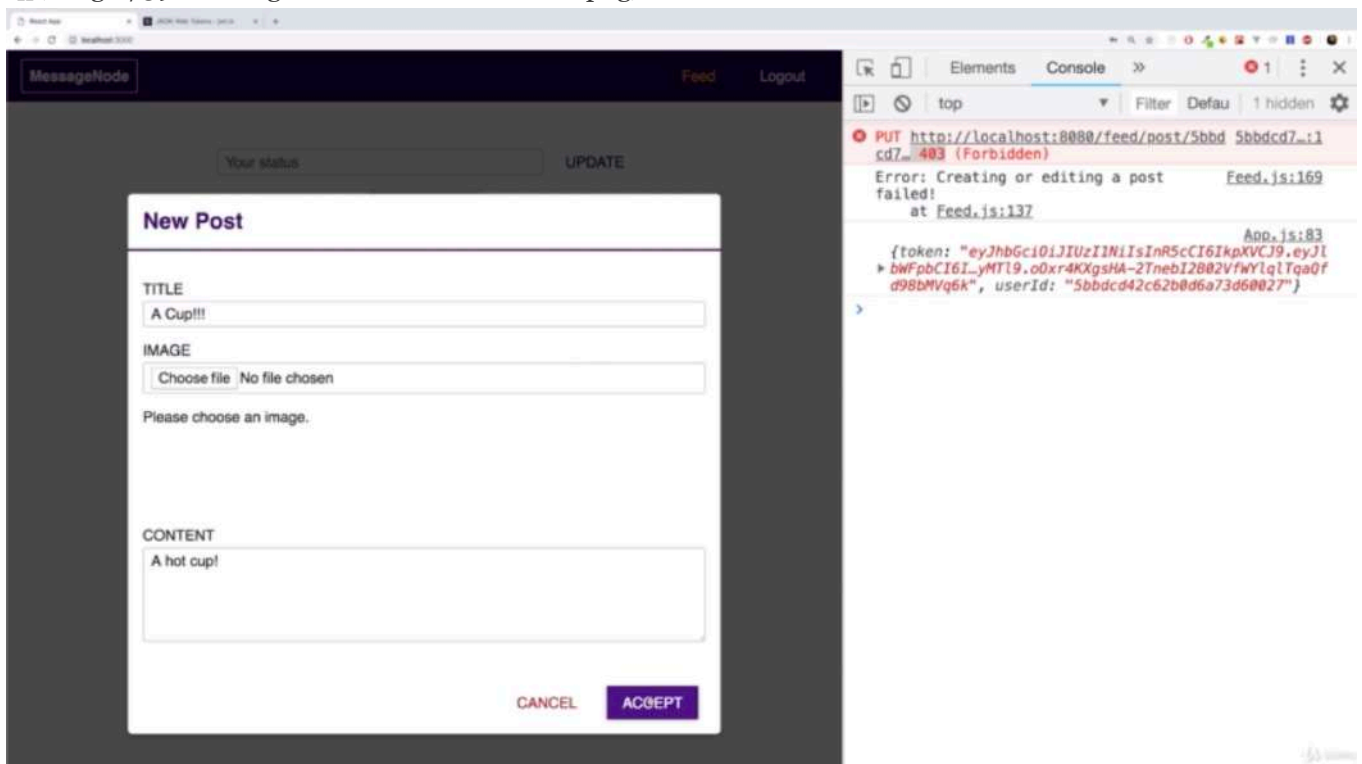
Default levels

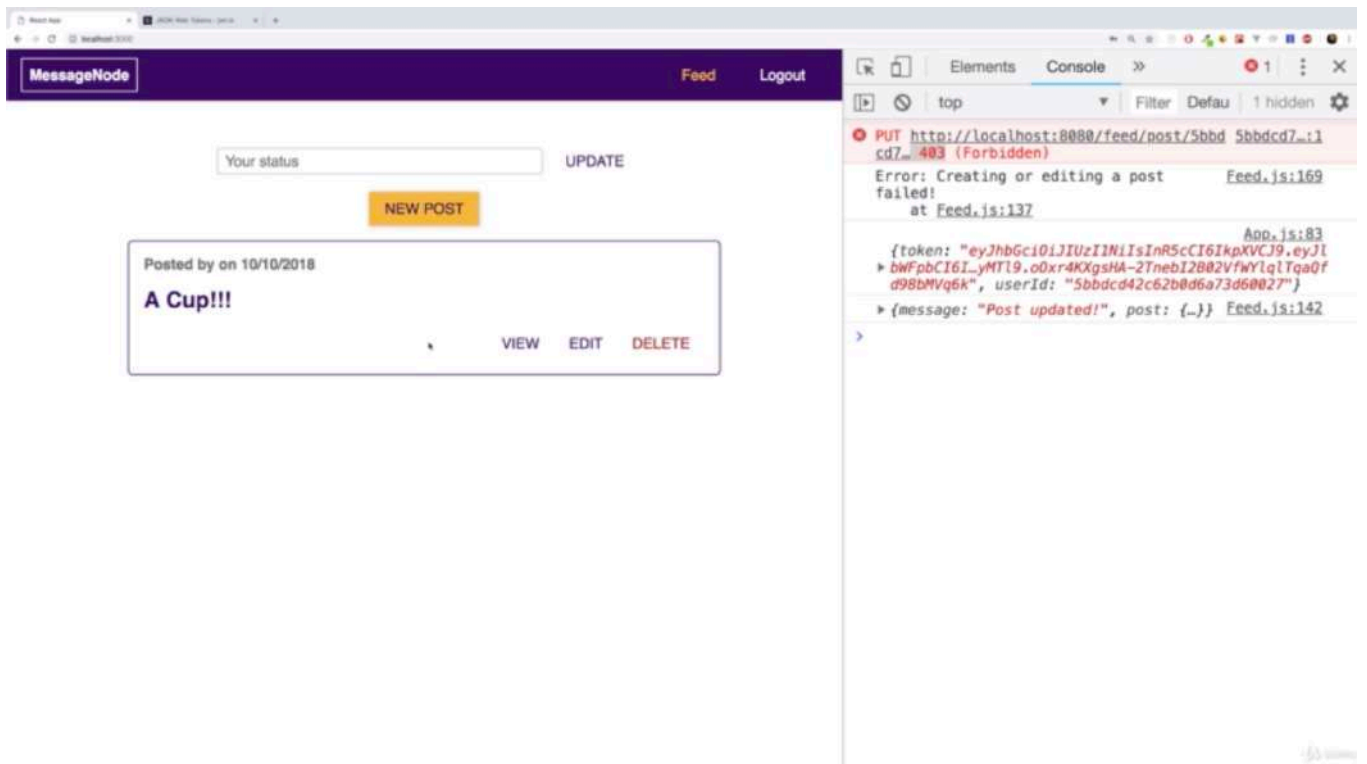


- this failed with 403 error.

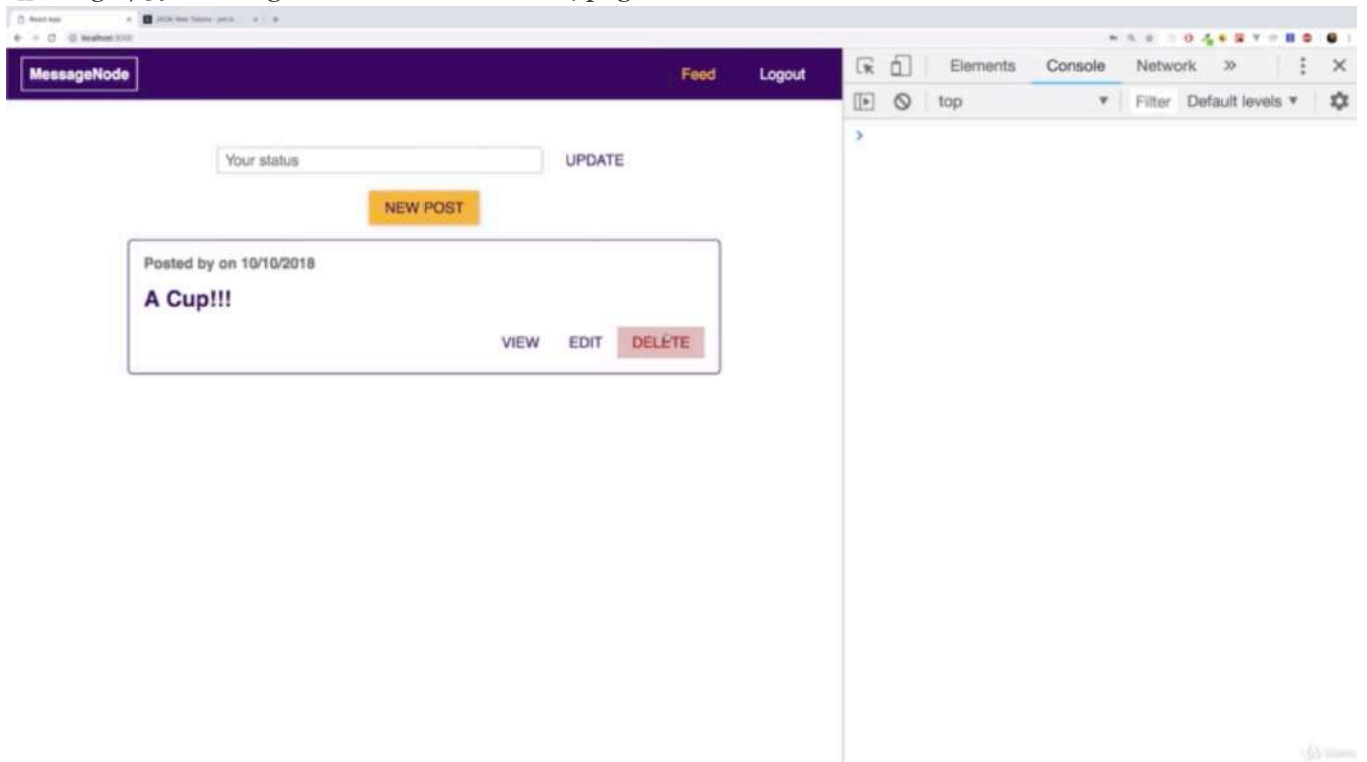


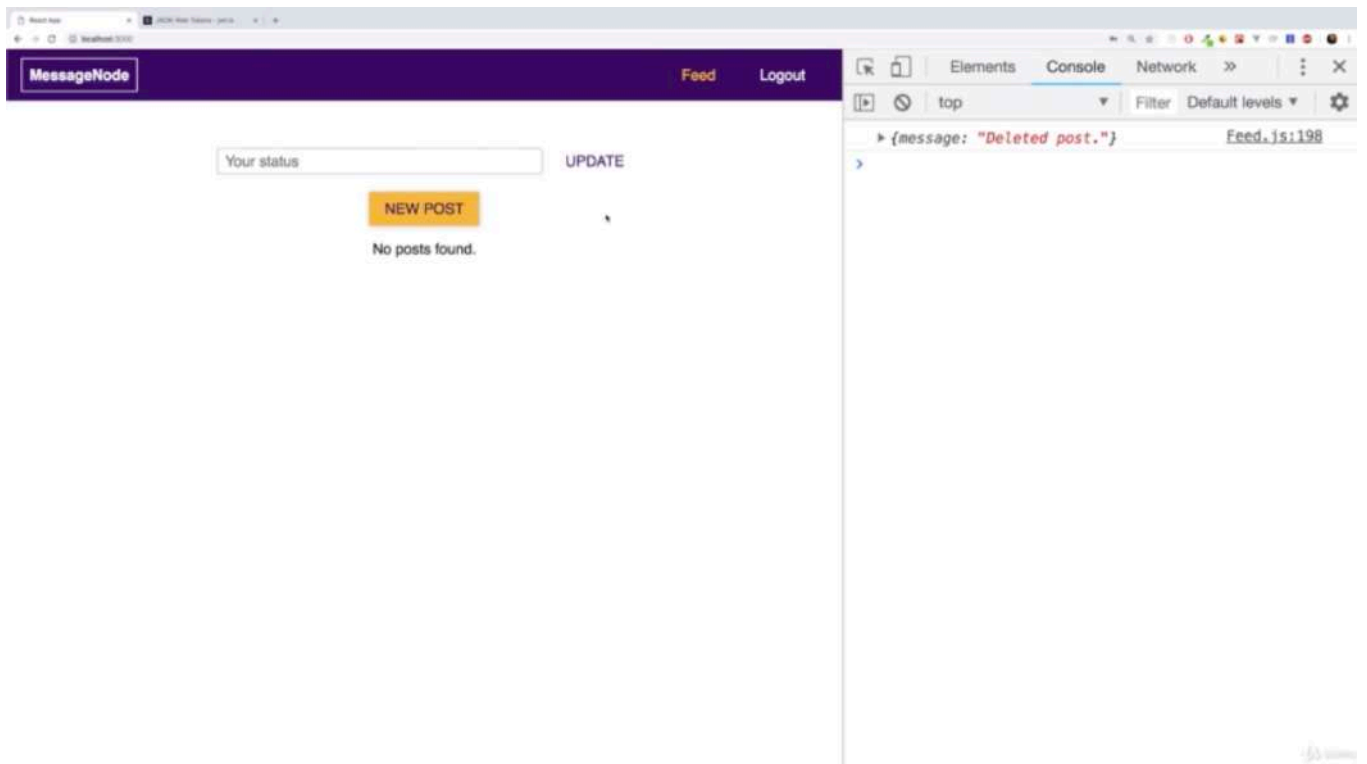
- let's login with the user who did create it and let's now try editing with this user and this now



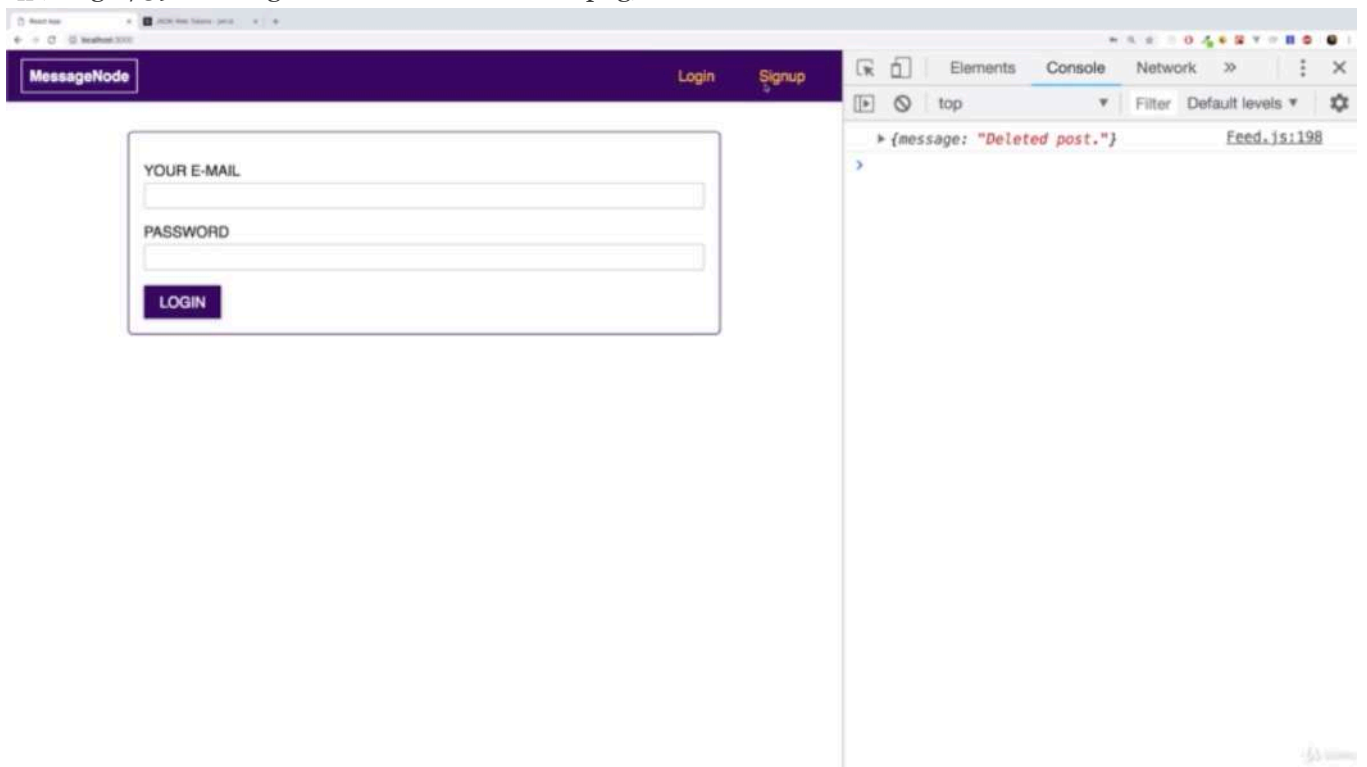


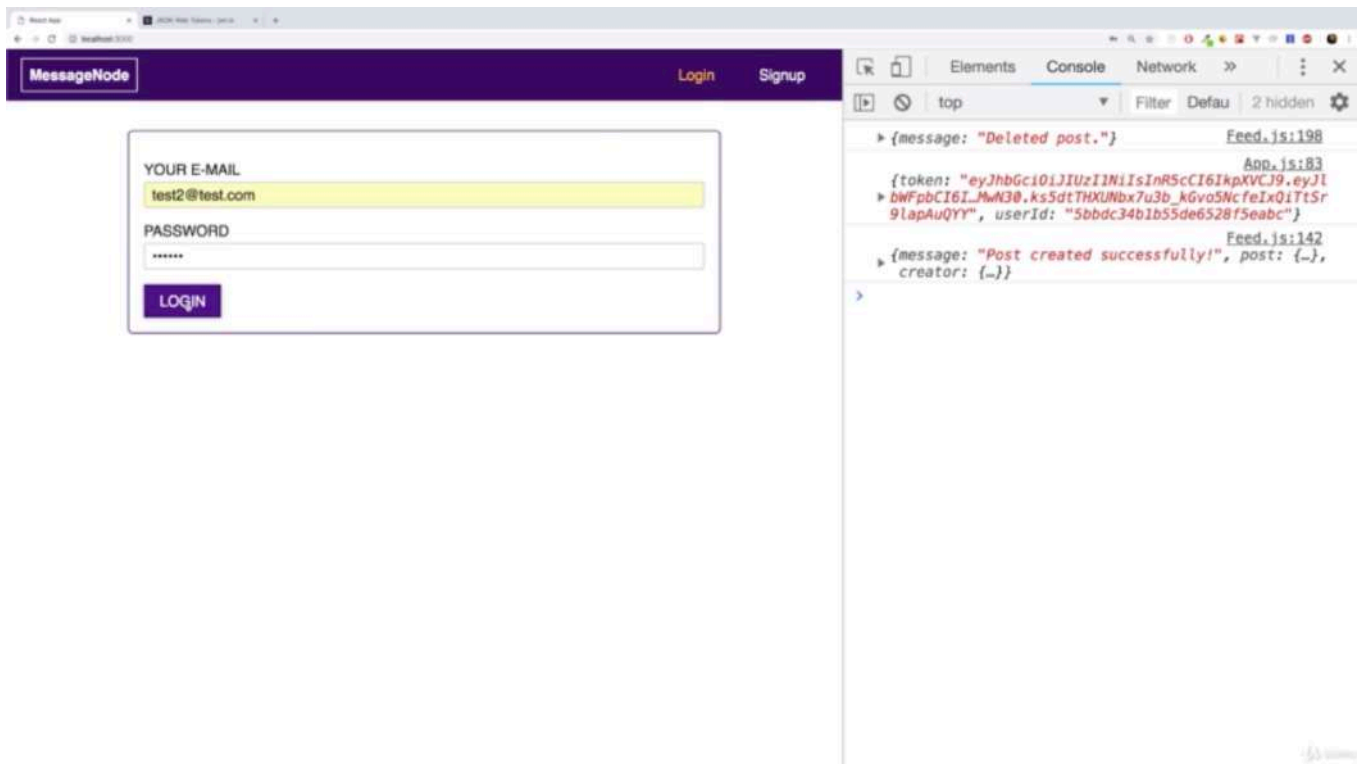
- and try editing and this now succeed.



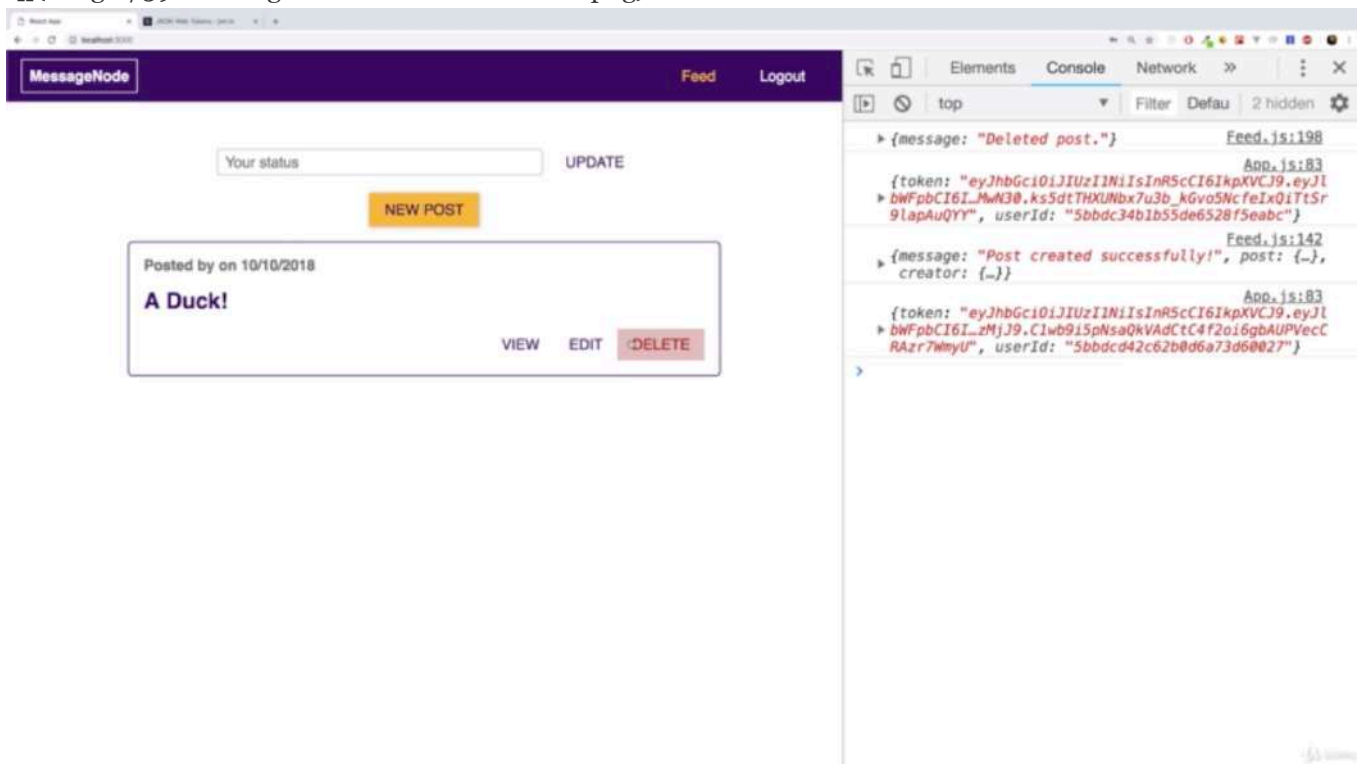


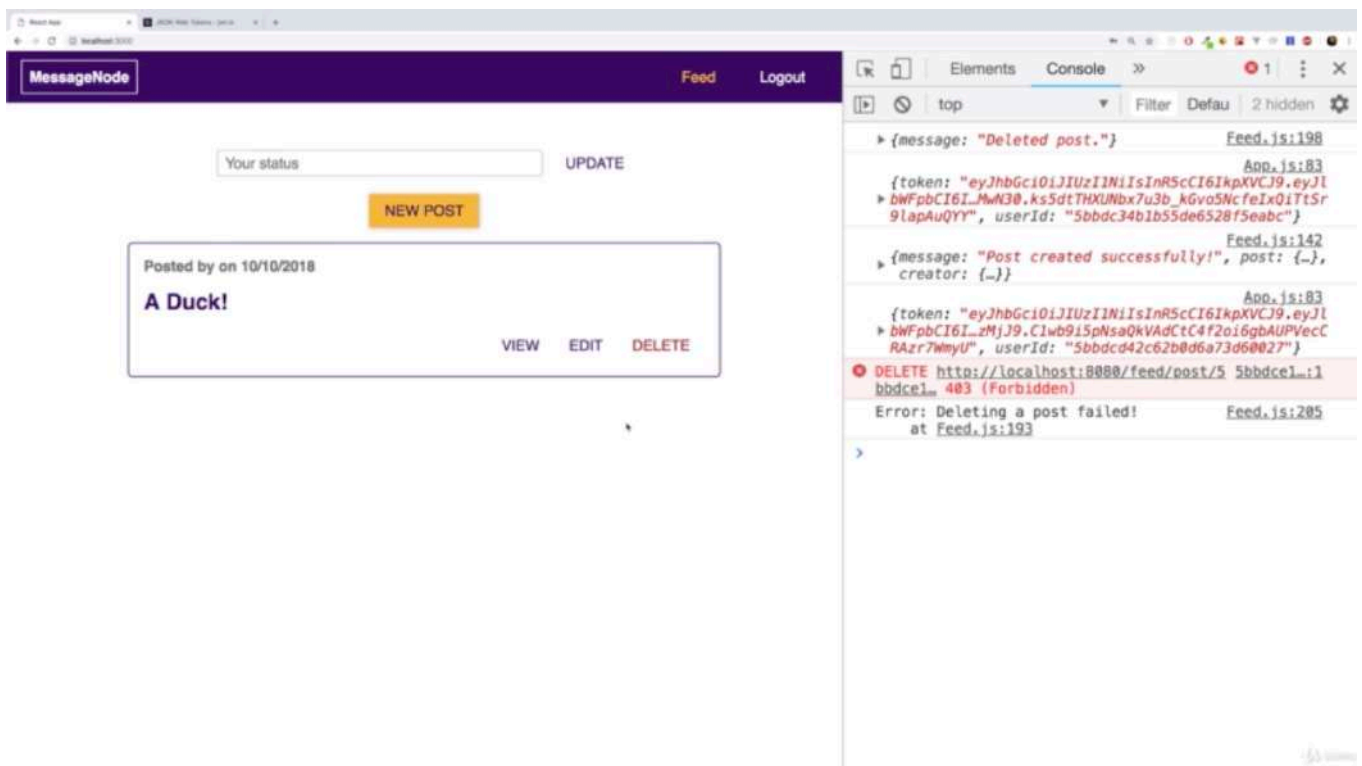
- delete this and it succeeds because this was the right user.





- and logout and login with other user who didn't create post.





- click delete and you can see fails.

```

1  ../controllers/feed.js(b)
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9  const User = require('../models/user');
10
11 exports.getPosts = (req, res, next) => {
12   const currentPage = req.query.page || 1;
13   const perPage = 2;
14   let totalItems;
15   Post.find()
16     .countDocuments()
17     .then(count => {
18       totalItems = count;
19       return Post.find()
20         .skip((currentPage - 1) * perPage)
21         .limit(perPage)
22     })
23     .then(posts => {
24       res
25         .status(200)
26         .json({
27           message: 'Fetched posts successfully.',
28           posts: posts,
29           totalItems: totalItems
30         });
31     })
32     .catch(err => {
33       if (!err.statusCode) {

```

```

34         err.statusCode = 500;
35     }
36     next(err);
37 });
38 };
39
40 exports.createPost = (req, res, next) => {
41     const errors = validationResult(req);
42     if (!errors.isEmpty()) {
43         const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

throw error;

}

if(!req.file){

const error = new Error('No image provided.');

error.statusCode = 422;

throw error;

}

const imageUrl = req.file.path;

const title = req.body.title;

const content = req.body.content;

/**remember in my ./middleware/is-auth.js(b) file,

* i store that in the request object,

*

* 'req.userId = decodedToken.userId'

*

* so we have that userId available here.

* with that information,

* i will set creator equal to 'req.userId'

* that will be string not an objectId

* but mongoose will take care about that and convert it.

* so now we create a new post assigned to that user

*/

let creator;

const post = new Post({

title: title,

content: content,

imageUrl: imageUrl,

creator: req.userId

});

post

.save()

.then(result => {

return User.findById(req.userId);

})

.then(user => {

creator = user;

user.posts.push(post);

return user.save()

})

.then(result => {

res.status(201).json({

message: 'Post created successfully!',

post: post,

```

90         creator: { _id: creator._id, name: creator.name }
91     });
92 })
93 .catch(err => {
94     if (!err.statusCode) {
95         err.statusCode = 500;
96     }
97     next(err);
98 });
99 };
100
101 exports.getPost = (req, res, next) => {
102     const postId = req.params.postId;
103     Post.findById(postId)
104         .then(post => {
105             if (!post) {
106                 const error = new Error('Could not find post.');
```

Could not find post.

```

107                 error.statusCode = 404;
108                 throw error;
109             }
110             res.status(200).json({ message: 'Post fetched.', post: post });
111         })
112         .catch(err => {
113             if (!err.statusCode) {
114                 err.statusCode = 500;
115             }
116             next(err);
117         });
118 };
119
120 exports.updatePost = (req, res, next) => {
121     const postId = req.params.postId;
122     const errors = validationResult(req);
123     if (!errors.isEmpty()) {
124         const error = new Error('Validation failed, entered data is incorrect.');
```

Validation failed, entered data is incorrect.

```

125         error.statusCode = 422;
126         throw error;
127     }
128     const title = req.body.title;
129     const content = req.body.content;
130     let imageUrl = req.body.image;
131     if (req.file) {
132         imageUrl = req.file.path
133     }
134     if (!imageUrl) {
135         const error = new Error('No file picked.');
```

No file picked.

```

136         error.statusCode = 422;
137         throw error;
138     }
139     Post.findById(postId)
140         .then(post => {
141             if (!post) {
142                 const error = new Error('Could not find post.');
```

Could not find post.

```

143                 error.statusCode = 404;
144                 throw error;
145             }

```

```

146     if (post.creator.toString() !== req.userId){
147         const error = new Error('Not authorized!');
148         error.statusCode = 403;
149         throw error;
150     }
151     if(imageUrl !== post.image) {
152         clearImage(post.imageUrl);
153     }
154     post.title = title;
155     post.imageUrl = imageUrl;
156     post.content = content;
157     return post.save();
158 })
159 .then(result => {
160     res.status(200).json({message: 'Post updated!', post: result});
161 })
162 .catch(err => {
163     if (!err.statusCode) {
164         err.statusCode = 500;
165     }
166     next(err);
167 })
168 };
169
170 exports.deletePost = (req, res, next) => {
171     const postId = req.params.postId;
172     Post.findById(postId)
173     .then(post => {
174         if (!post) {
175             const error = new Error('Could not find post. ');
176             error.statusCode = 404;
177             throw error;
178         }
179         if (post.creator.toString() !== req.userId){
180             const error = new Error('Not authorized!');
181             error.statusCode = 403;
182             throw error;
183         }
184         //Check logged in user
185         clearImage(post.imageUrl);
186         return Post.findByIdAndRemove(postId);
187     })
188     .then(result => {
189         console.log(result);
190         res.status(200).json({message: 'Deleted post. '});
191     })
192     .catch(err => {
193         if (!err.statusCode) {
194             err.statusCode = 500;
195         }
196         next(err);
197     });
198 }
199
200 const clearImage = filePath => {
201     filePath = path.join(__dirname, '..', filePath)

```

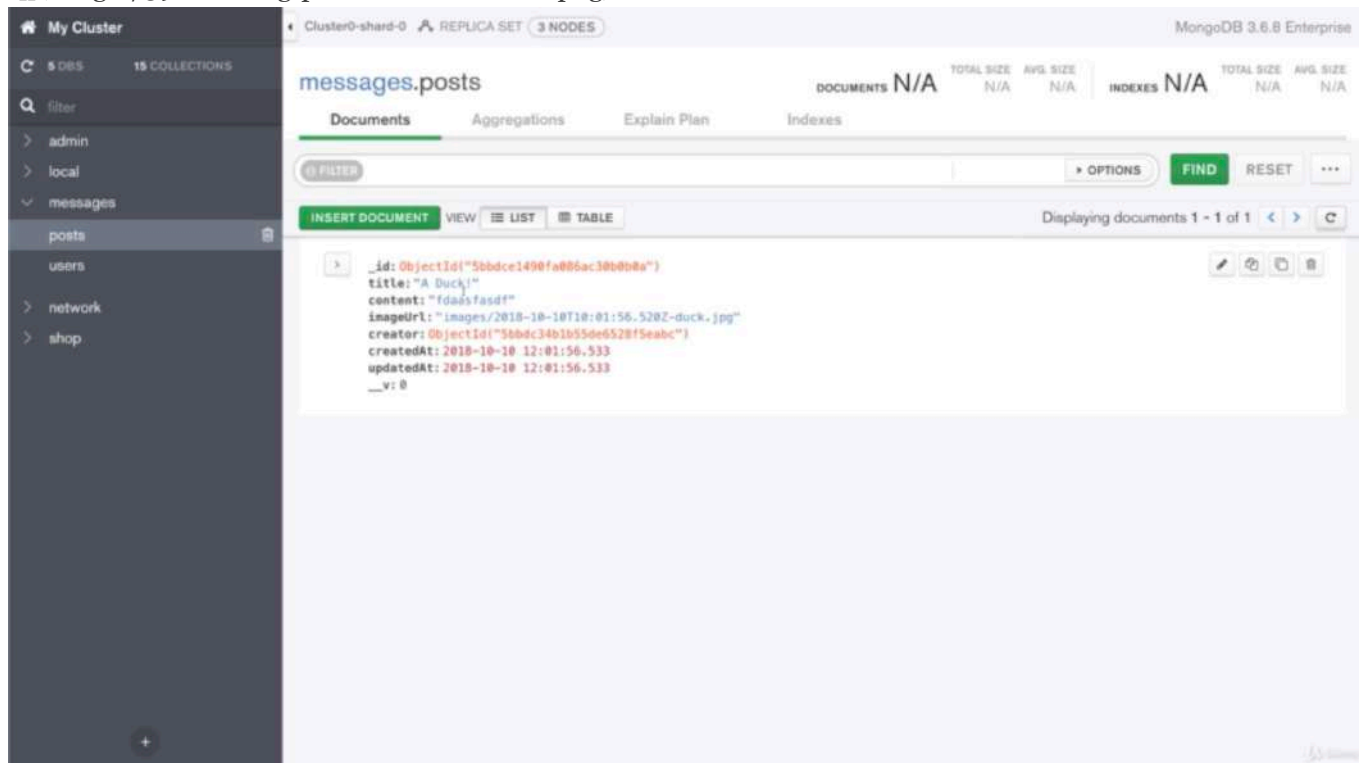
```

202 fs.unlink(filePath, err => console.log(err));
203 }
204

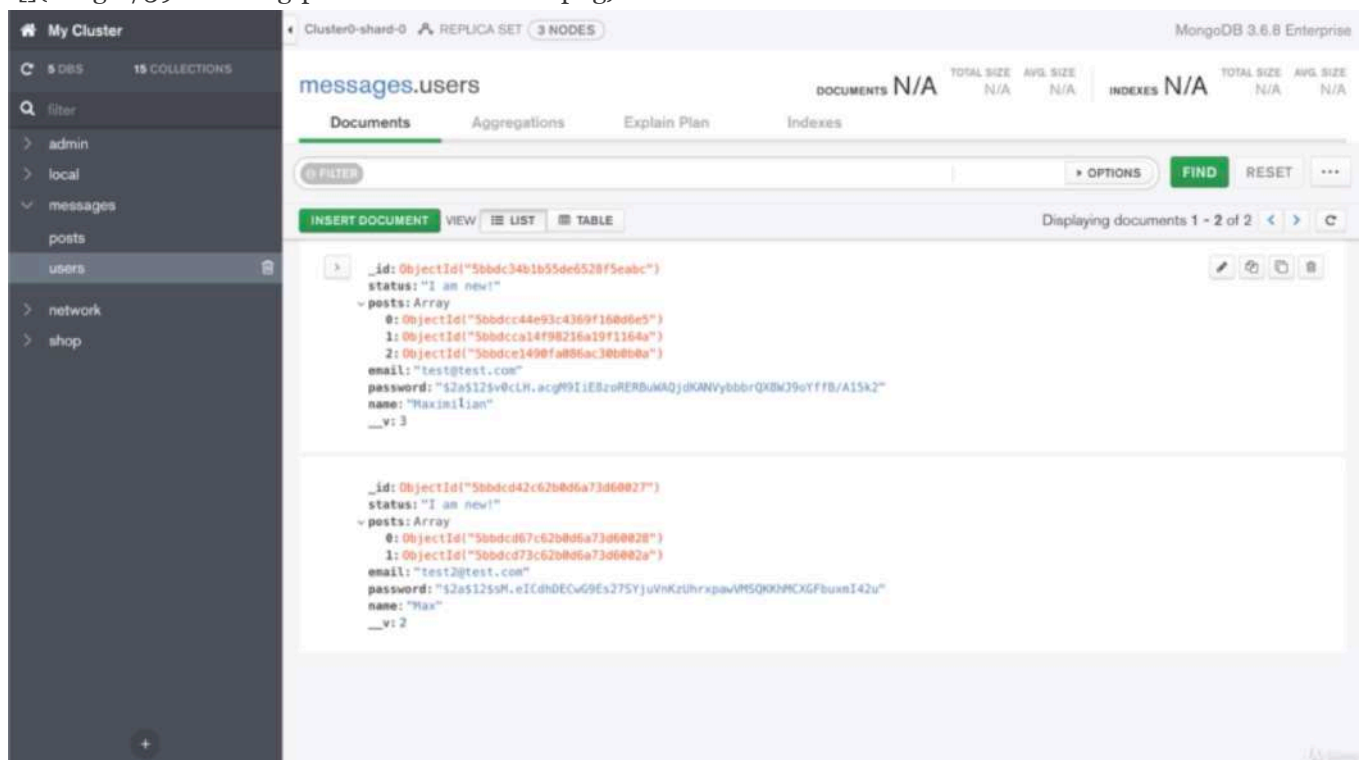
```

* Chapter 391: Clearing Post-User Relations

1. update
- ./controllers/feed.js(b)

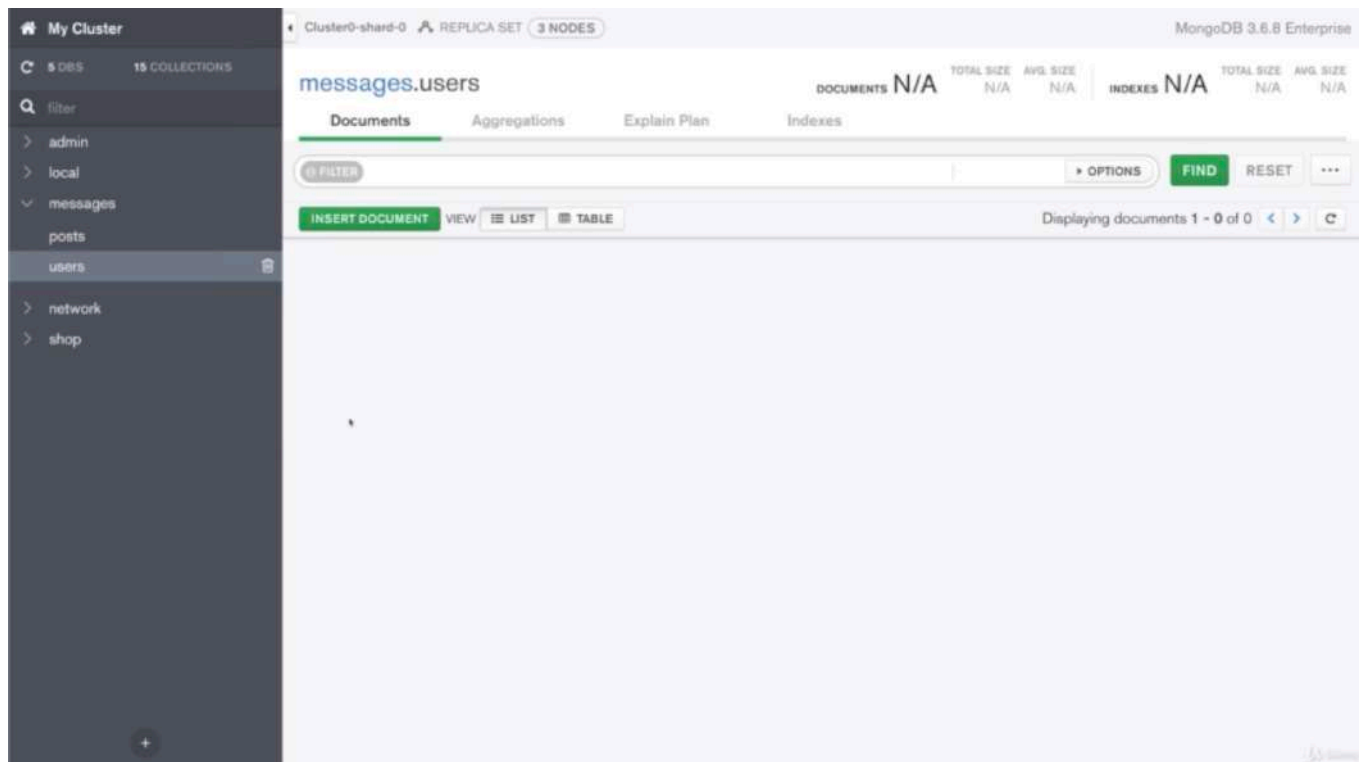
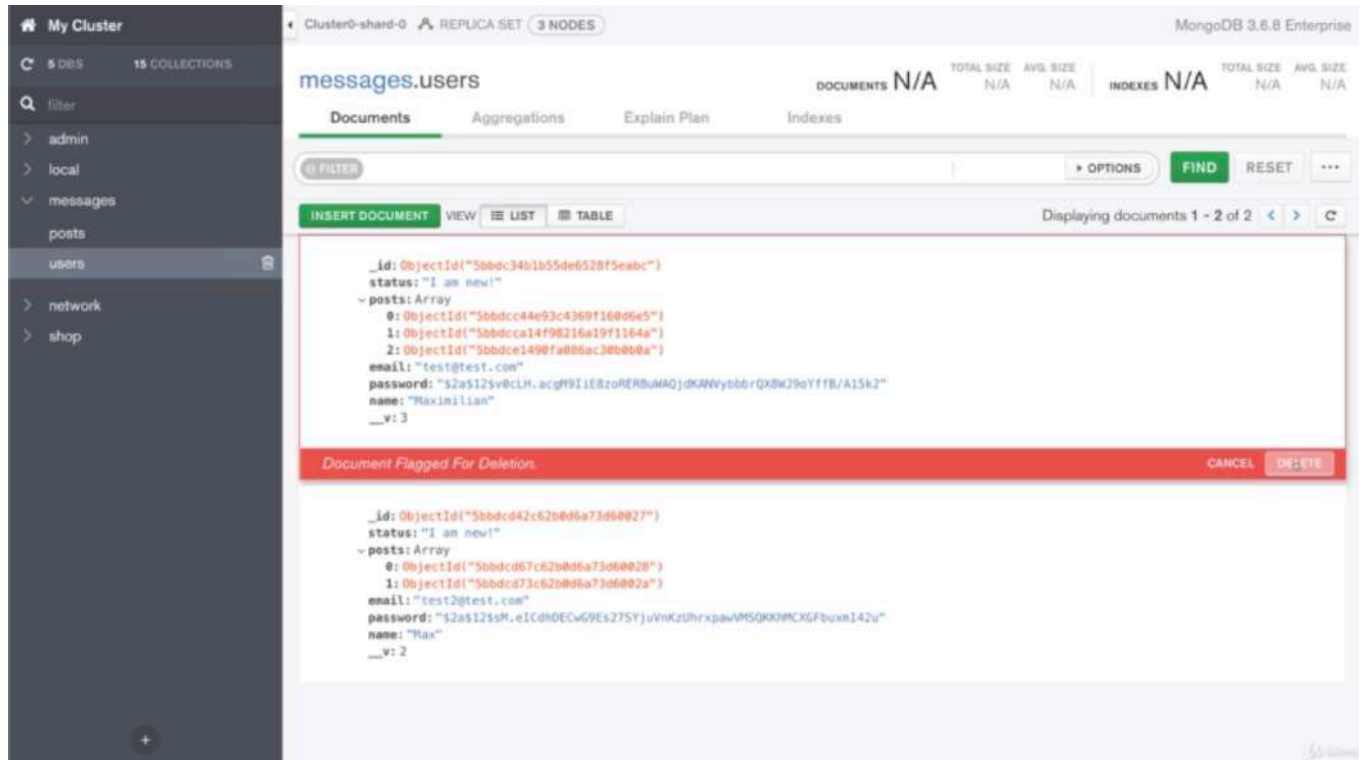


- if we refresh the post collection, we only have one item there,

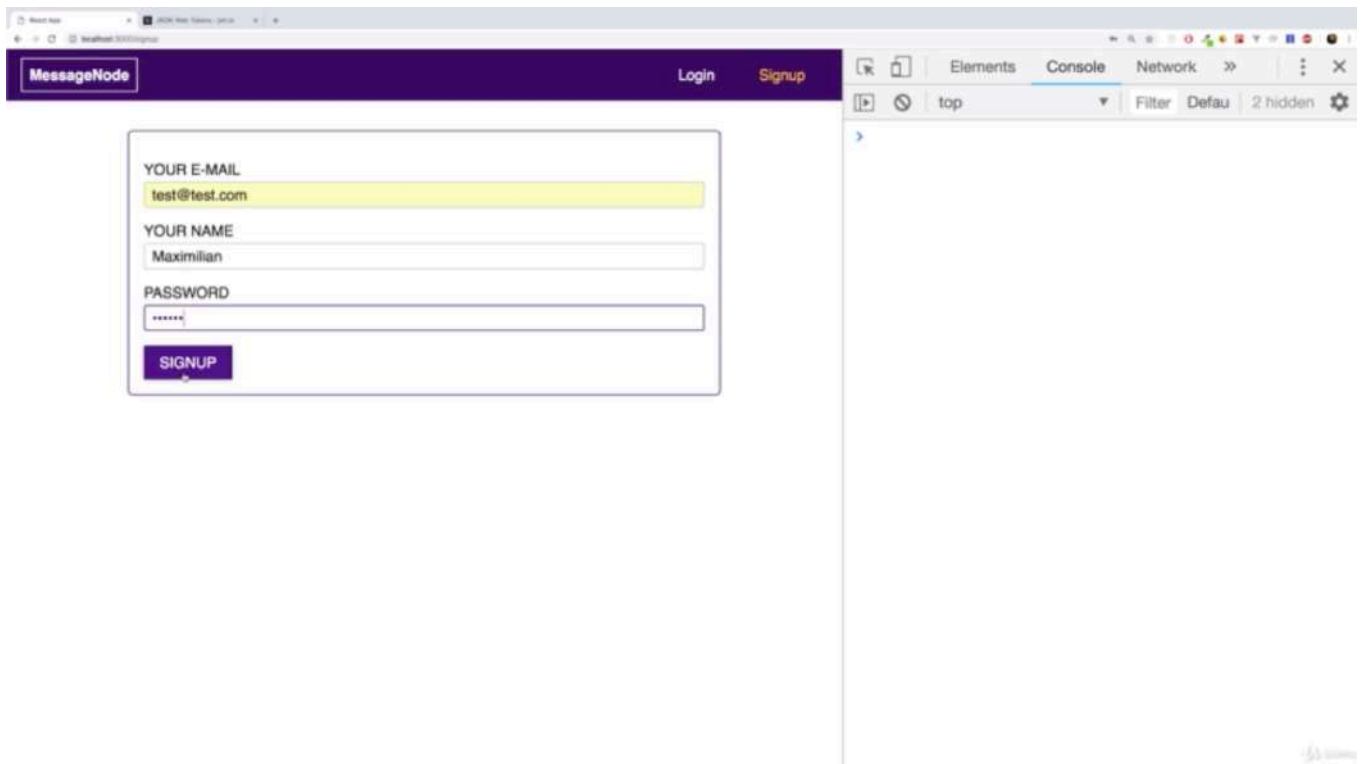


- but if i have a look at my users collection, the first user has a couple of posts. and the 2nd user has so too.

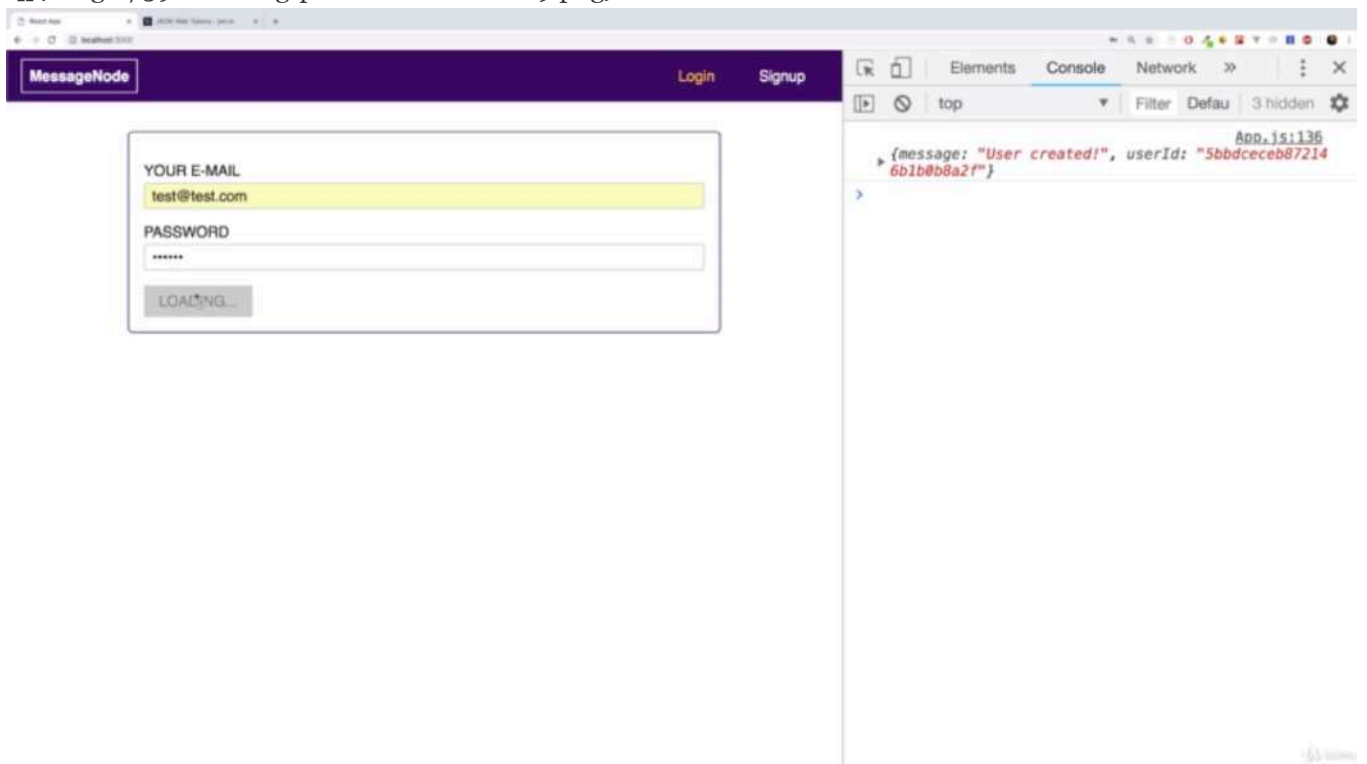
because we have no logic to clear posts when i delete them to clear relations and we should do that.

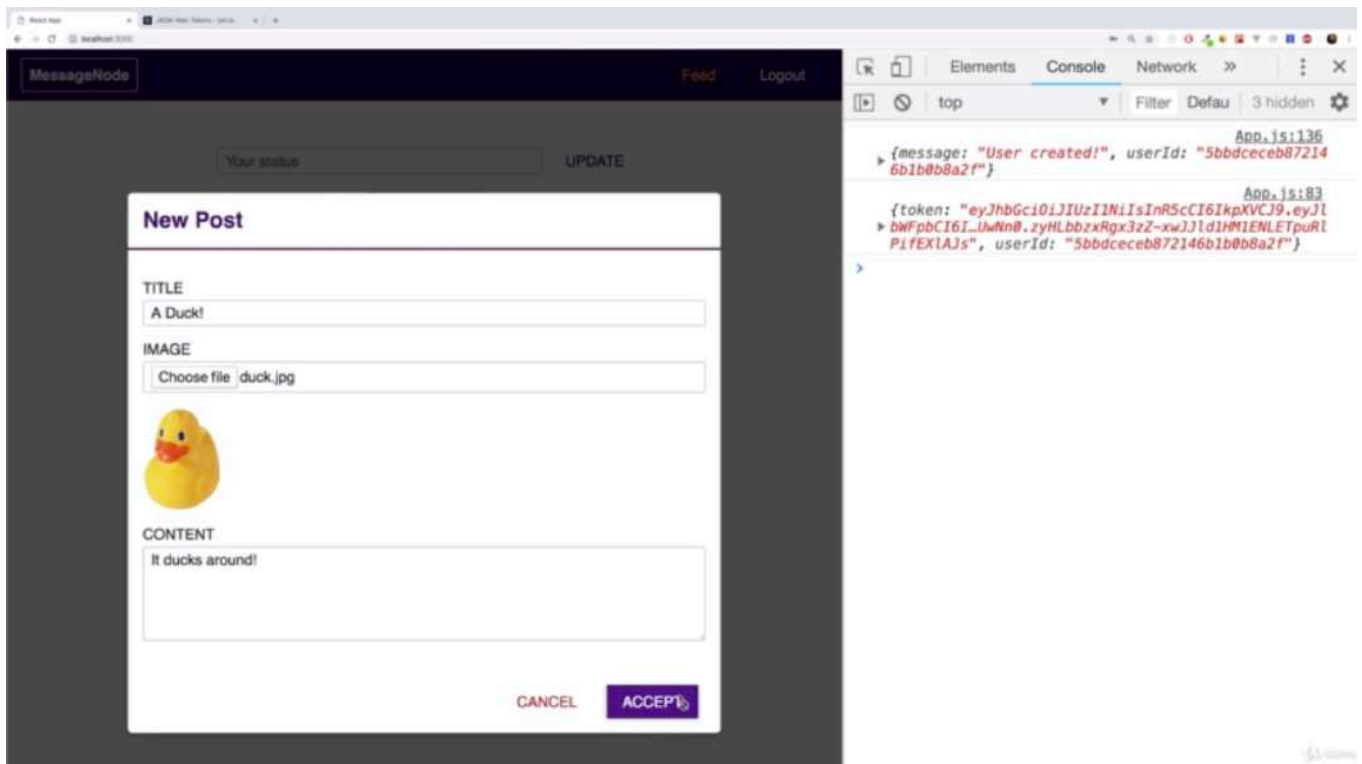


- let me get rid of all users manually by using mongodb compass.

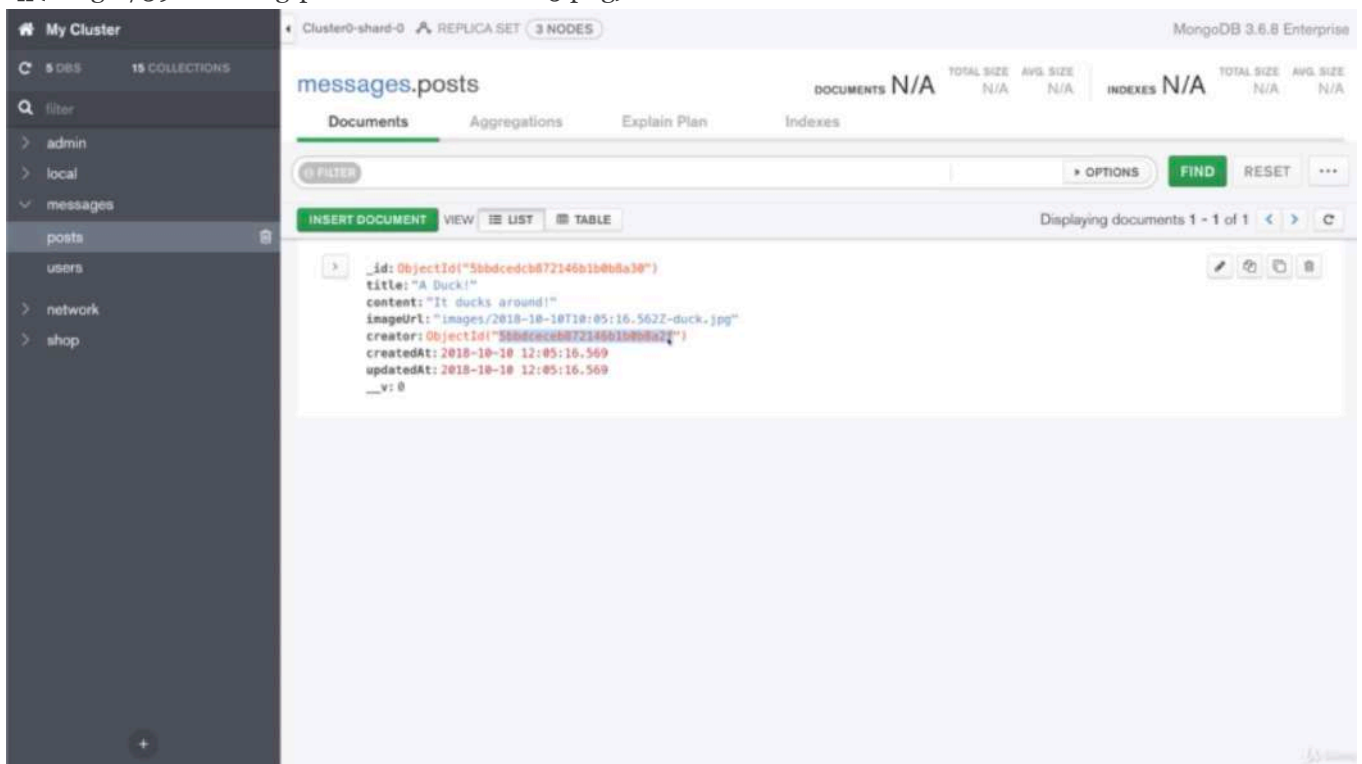


- also get rid of all posts and logout here. let's create a new user because we cleared all users.





- let's login with that user and let's create a new posts.



My Cluster Cluster0-shard-0 REPLICA SET 3 NODES MongoDB 3.6.8 Enterprise

messages.users DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

Documents Aggregations Explain Plan Indexes

Filter OPTIONS FIND RESET ...

INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 1 of 1

```

{
  "_id": ObjectId("5bbdceceb872146b1b0b8a2f"),
  "status": "I am new!",
  "posts": Array
    0: ObjectId("5bbdcedcb872146b1b0b8a2f")
  "email": "test@test.com",
  "password": "s2e5125pz4fNU8mW8773SMtd/Rnx.5M8Q4nJ0Fz15CBjBU2cvJs2eZg5070y",
  "name": "Maximilian",
  "_v": 1
}

```

MessageNode Feed Logout

Your status UPDATE

NEW POST

Posted by on 10/10/2018

A Duck!

VIEW EDIT DELETE

App.js:136 {message: "User created!", userId: "5bbdceceb872146b1b0b8a2f"}

App.js:83 {token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmFpbCI6IjUwNnQyZyHLbGZxRgx3Z2xwJ1d1M1ENLEtpuRlPifEXlAJs", userId: "5bbdceceb872146b1b0b8a2f"}

Feed.js:142 {message: "Post created successfully!", post: {_, creator: {_}}}

Feed Node

Feed Logout

Your status UPDATE

NEW POST

No posts found.

App.js:136 {message: "User created!", userId: "5bbdceceb872146b1b0b8a2f"}

App.js:83 {token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmFpbCI6IjUwNm0uZyHLb2xzRgx3Z2~xwJlIj11M1ENLEtpuRlPlEXlAJs", userId: "5bbdceceb872146b1b0b8a2f"}

Feed.js:142 {message: "Post created successfully!", post: {...}, creator: {...}}

Feed.js:198 {message: "Deleted post."}

My Cluster

Cluster0-shard-0 REPLICA SET 3 NODES

MongoDB 3.6.8 Enterprise

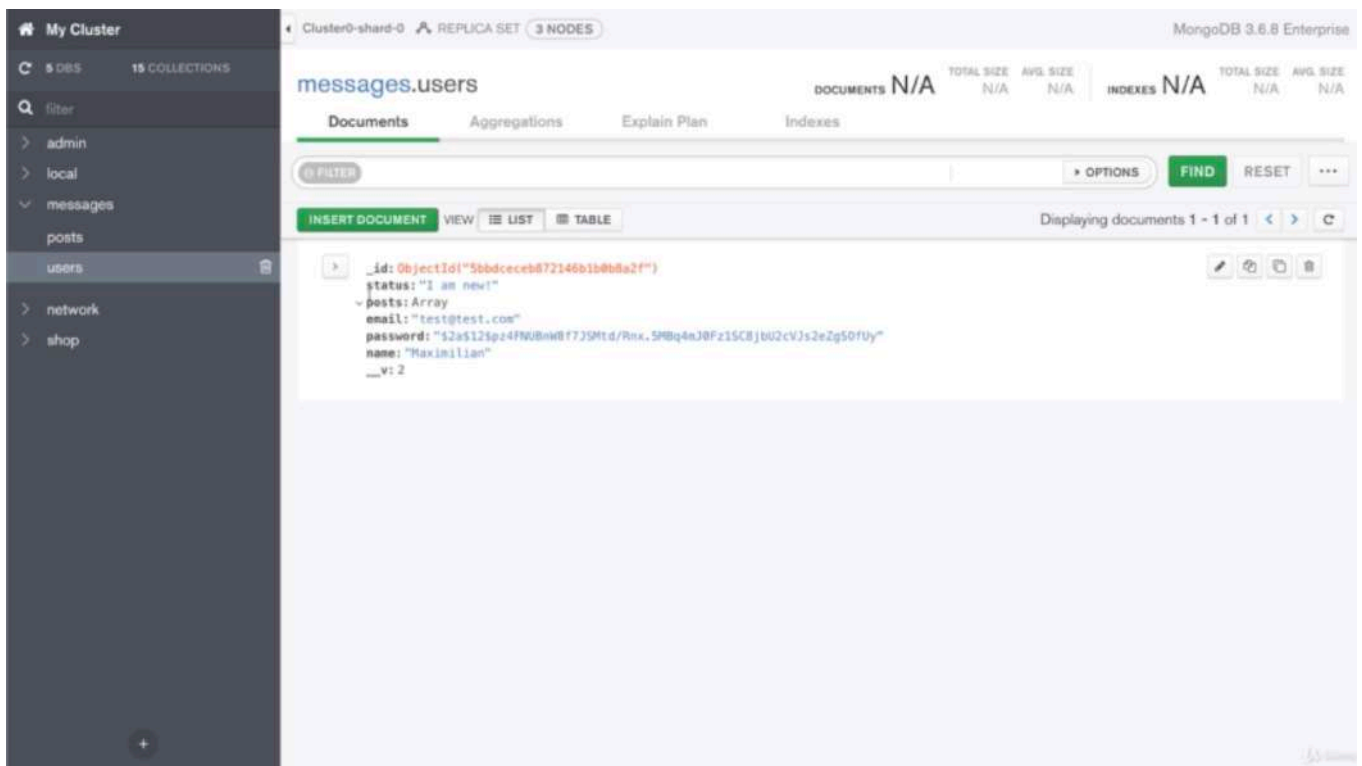
messages.posts

DOCUMENTS N/A TOTAL SIZE N/A AVG. SIZE N/A INDEXES N/A TOTAL SIZE N/A AVG. SIZE N/A

Documents Aggregations Explain Plan Indexes

INSERT DOCUMENT VIEW LIST TABLE

Displaying documents 0 - 0 of 0



- and in users, this array is now also empty.

```

1  //./controllers/feed.js(b)
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9  const User = require('../models/user');
10
11 exports.getPosts = (req, res, next) => {
12   const currentPage = req.query.page || 1;
13   const perPage = 2;
14   let totalItems;
15   Post.find()
16     .countDocuments()
17     .then(count => {
18       totalItems = count;
19       return Post.find()
20         .skip((currentPage - 1) * perPage)
21         .limit(perPage);
22     })
23     .then(posts => {
24       res.status(200).json({
25         message: 'Fetched posts successfully.',
26         posts: posts,
27         totalItems: totalItems
28       });
29     })
30     .catch(err => {
31       if (!err.statusCode) {
32         err.statusCode = 500;
33       }

```

```

34     next(err);
35   });
36 };
37
38 exports.createPost = (req, res, next) => {
39   const errors = validationResult(req);
40   if (!errors.isEmpty()) {
41     const error = new Error('Validation failed, entered data is incorrect.');
```

error.statusCode = 422;

throw error;

}

if (!req.file) {

const error = new Error('No image provided.');

error.statusCode = 422;

throw error;

}

const imageUrl = req.file.path;

const title = req.body.title;

const content = req.body.content;

let creator;

const post = new Post({

title: title,

content: content,

imageUrl: imageUrl,

creator: req.userId

});

post

.save()

.then(result => {

return User.findById(req.userId);

})

.then(user => {

creator = user;

user.posts.push(post);

return user.save();

})

.then(result => {

res.status(201).json({

message: 'Post created successfully!',

post: post,

creator: { _id: creator._id, name: creator.name }

});

})

.catch(err => {

if (!err.statusCode) {

err.statusCode = 500;

}

next(err);

});

};

84

85 exports.getPost = (req, res, next) => {

86 const postId = req.params.postId;

87 Post.findById(postId)

88 .then(post => {

89 if (!post) {

```

90     const error = new Error('Could not find post.');
```

```

91     error.statusCode = 404;
```

```

92     throw error;
```

```

93   }
```

```

94   res.status(200).json({ message: 'Post fetched.', post: post });
95 })
```

```

96 .catch(err => {
97   if (!err.statusCode) {
98     err.statusCode = 500;
99   }
100   next(err);
101 });
102 };
103
104 exports.updatePost = (req, res, next) => {
105   const postId = req.params.postId;
106   const errors = validationResult(req);
107   if (!errors.isEmpty()) {
108     const error = new Error('Validation failed, entered data is incorrect.');
```

```

109     error.statusCode = 422;
```

```

110     throw error;
```

```

111   }
```

```

112   const title = req.body.title;
```

```

113   const content = req.body.content;
```

```

114   let imageUrl = req.body.image;
```

```

115   if (req.file) {
116     imageUrl = req.file.path;
```

```

117   }
```

```

118   if (!imageUrl) {
119     const error = new Error('No file picked.');
```

```

120     error.statusCode = 422;
```

```

121     throw error;
```

```

122   }
```

```

123   Post.findById(postId)
124     .then(post => {
125       if (!post) {
126         const error = new Error('Could not find post.');
```

```

127         error.statusCode = 404;
```

```

128         throw error;
```

```

129       }
```

```

130       if (post.creator.toString() !== req.userId) {
131         const error = new Error('Not authorized!');
```

```

132         error.statusCode = 403;
```

```

133         throw error;
```

```

134       }
```

```

135       if (imageUrl !== post.imageUrl) {
136         clearImage(post.imageUrl);
137       }
```

```

138       post.title = title;
```

```

139       post.imageUrl = imageUrl;
```

```

140       post.content = content;
```

```

141       return post.save();
142     })
143     .then(result => {
144       res.status(200).json({ message: 'Post updated!', post: result });
145     })

```

```

146     .catch(err => {
147         if (!err.statusCode) {
148             err.statusCode = 500;
149         }
150         next(err);
151     });
152 };
153
154 exports.deletePost = (req, res, next) => {
155     const postId = req.params.postId;
156     Post.findById(postId)
157         .then(post => {
158             if (!post) {
159                 const error = new Error('Could not find post. ');
160                 error.statusCode = 404;
161                 throw error;
162             }
163             if (post.creator.toString() !== req.userId) {
164                 const error = new Error('Not authorized! ');
165                 error.statusCode = 403;
166                 throw error;
167             }
168             // Check logged in user
169             clearImage(post.imageUrl);
170             return Post.findByIdAndRemove(postId);
171         })
172         .then(result => {
173             return User.findById(req.userId);
174         })
175         .then(user => {
176             user.posts.pull(postId);
177             return user.save();
178         })
179         .then(result => {
180             res.status(200).json({ message: 'Deleted post.' });
181         })
182         .catch(err => {
183             if (!err.statusCode) {
184                 err.statusCode = 500;
185             }
186             next(err);
187         });
188 };
189
190 const clearImage = filePath => {
191     filePath = path.join(__dirname, '..', filePath);
192     fs.unlink(filePath, err => console.log(err));
193 };
194

```

* Chapter 392: Wrap Up

Module Summary

From "Classic" to REST API

- Most of the server-side code does not change, only request + response data is affected
- More Http methods are available
- The REST API server does not care about the client, requests are handled in isolation => No sessions

Authentication

- Due to no sessions being used, authentication works differently
- Each request needs to be able to send some data that proves that the request is authenticated
- JSON Web Tokens ("JWT") are a common way of storing authentication information on the client and proving authentication status
- JWTs are signed by the server and can only be validated by the server