

# 26. Understanding Async Await In Node.js

## \* Chapter 395: What Is Async Await All About?

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





What?

Asynchronous Requests in a Synchronous Way\*

\*Only by the way it looks, NOT by the way it behaves

JS Slides

- async and await are 2 keywords which are part of the core javascript language and not an exclusive part of the node.js runtime. they are not part of node.js but you can use them in node.js project.
- Async allows you to write asynchronous requests so requests where you have some operation that takes a little while and comes back later in a synchronous way and you see there is an asterisk after synchronous way because in a way it allows you to write asynchronous statements in a way that looks synchronous but it still isn't a synchronous requests.

```
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  /**you could define a function that gets executed
17   * once it's done instead of '.then()' like below
18   *
19   *   Post.find().countDocuments(count => {})
20   *
21   * but we don't use callbacks
22   * because in there, we would use that code
23   *
24   *   Post.find().countDocuments(count => {
25     totalItems = count;
26     return Post.find()
27       .skip((currentPage - 1) * perPage)
28       .limit(perPage);
29   })
30   *
31   * and then we would need a callback in the '.find()' function
32   * and we would nest all these callbacks(count => {})
33   * leading to very unreadable code
34   * that is why you often prefer promises
35   * even though you could do it with callbacks
36   * because you have one then block after each other
37   * and it's very readable.
38   */
39   .countDocuments()
40   .then(count => {
41     totalItems = count;
42     return Post.find()
43       .skip((currentPage - 1) * perPage)
44       .limit(perPage);
45   })
46   .then(posts => {
47     res.status(200).json({
48       message: 'Fetched posts successfully.',
49       posts: posts,
50       totalItems: totalItems
51     });
52   })
53   .catch(err => {
54     if (!err.statusCode) {
55       err.statusCode = 500;
56     }
57     next(err);
58   });
59  /**if there is some code under the async codes,
60   * then code under the async codes will be executed
61   * before async code(.then, .catch).
62   * the reason for that is that
63   * with '.then()' we define code snippets
64   * or we define functions that should run in the future
65   * once it is longer taking a synchronous operation is done
66   * and it's called asynchronous
67   * because it doesn't happen instantly
68   * but it takes a little while

```

```

69  *
70  * callbacks are another way of working with asynchronous code
71  */
72 };
73
74 exports.createPost = (req, res, next) => {
75   const errors = validationResult(req);
76   if (!errors.isEmpty()) {
77     const error = new Error('Validation failed, entered data is incorrect.');
```

```

78     error.statusCode = 422;
79     throw error;
80   }
81   if (!req.file) {
82     const error = new Error('No image provided.');
```

```

83     error.statusCode = 422;
84     throw error;
85   }
86   const imageUrl = req.file.path;
87   const title = req.body.title;
88   const content = req.body.content;
89   let creator;
90   const post = new Post({
91     title: title,
92     content: content,
93     imageUrl: imageUrl,
94     creator: req.userId
95   });
96   post
97     .save()
98     .then(result => {
99       return User.findById(req.userId);
100     })
101     .then(user => {
102       creator = user;
103       user.posts.push(post);
104       return user.save();
105     })
106     .then(result => {
107       res.status(201).json({
108         message: 'Post created successfully!',
109         post: post,
110         creator: { _id: creator._id, name: creator.name }
111       });
112     })
113     .catch(err => {
114       if (!err.statusCode) {
115         err.statusCode = 500;
116       }
117       next(err);
118     });
119 };
120
121 exports.getPost = (req, res, next) => {
122   const postId = req.params.postId;
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 res.status(200).json({ message: 'Post fetched.', post: post });
131 }
132 .catch(err => {
133 if (!err.statusCode) {
134 err.statusCode = 500;
135 }
136 next(err);
137 });
138 };
139
140 exports.updatePost = (req, res, next) => {
141 const postId = req.params.postId;
142 const errors = validationResult(req);
143 if (!errors.isEmpty()) {
144 const error = new Error('Validation failed, entered data is incorrect.');

145 error.statusCode = 422;
146 throw error;
147 }
148 const title = req.body.title;
149 const content = req.body.content;
150 let imageUrl = req.body.image;
151 if (req.file) {
152 imageUrl = req.file.path;
153 }
154 if (!imageUrl) {
155 const error = new Error('No file picked.');

156 error.statusCode = 422;
157 throw error;
158 }
159 Post.findById(postId)
160 .then(post => {
161 if (!post) {
162 const error = new Error('Could not find post.');

163 error.statusCode = 404;
164 throw error;
165 }
166 if (post.creator.toString() !== req.userId) {
167 const error = new Error('Not authorized!');

168 error.statusCode = 403;
169 throw error;
170 }
171 if (imageUrl !== post.imageUrl) {
172 clearImage(post.imageUrl);
173 }
174 post.title = title;
175 post.imageUrl = imageUrl;
176 post.content = content;
177 return post.save();
178 })
179 .then(result => {
180 res.status(200).json({ message: 'Post updated!', post: result });

```

181     })
182     .catch(err => {
183       if (!err.statusCode) {
184         err.statusCode = 500;
185       }
186       next(err);
187     });
188   };
189
190   exports.deletePost = (req, res, next) => {
191     const postId = req.params.postId;
192     Post.findById(postId)
193       .then(post => {
194         if (!post) {
195           const error = new Error('Could not find post. ');
196           error.statusCode = 404;
197           throw error;
198         }
199         if (post.creator.toString() !== req.userId) {
200           const error = new Error('Not authorized!');
201           error.statusCode = 403;
202           throw error;
203         }
204         // Check logged in user
205         clearImage(post.imageUrl);
206         return Post.findByIdAndRemove(postId);
207       })
208       .then(result => {
209         return User.findById(req.userId);
210       })
211       .then(user => {
212         user.posts.pull(postId);
213         return user.save();
214       })
215       .then(result => {
216         res.status(200).json({ message: 'Deleted post.' });
217       })
218       .catch(err => {
219         if (!err.statusCode) {
220           err.statusCode = 500;
221         }
222         next(err);
223       });
224   };
225
226   const clearImage = filePath => {
227     filePath = path.join(__dirname, '..', filePath);
228     fs.unlink(filePath, err => console.log(err));
229   };
230

```

## \* Chapter 396: Transforming “Then Catch” To “Async Await”

1. update

- ./controllers/feed.js(b)

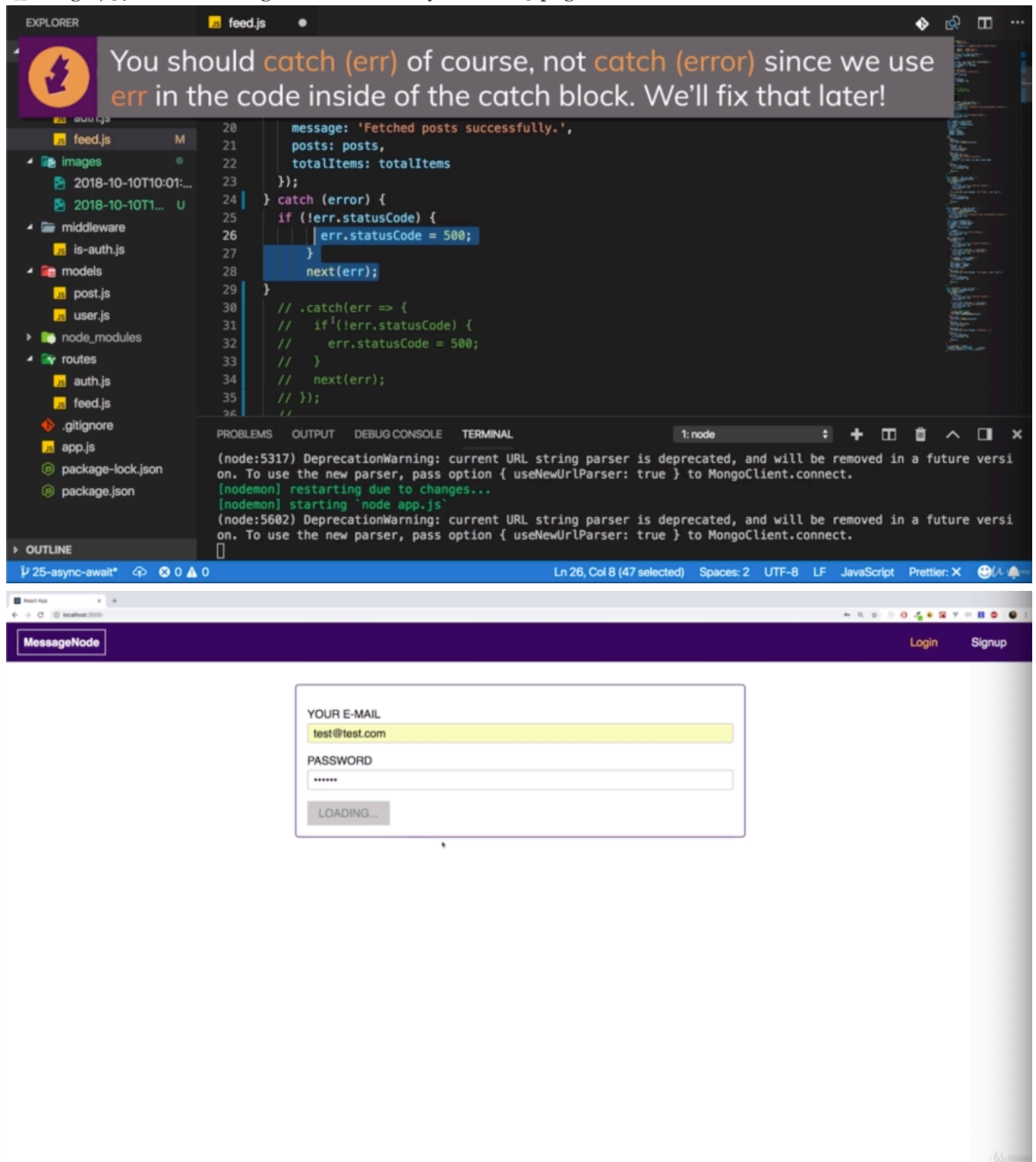


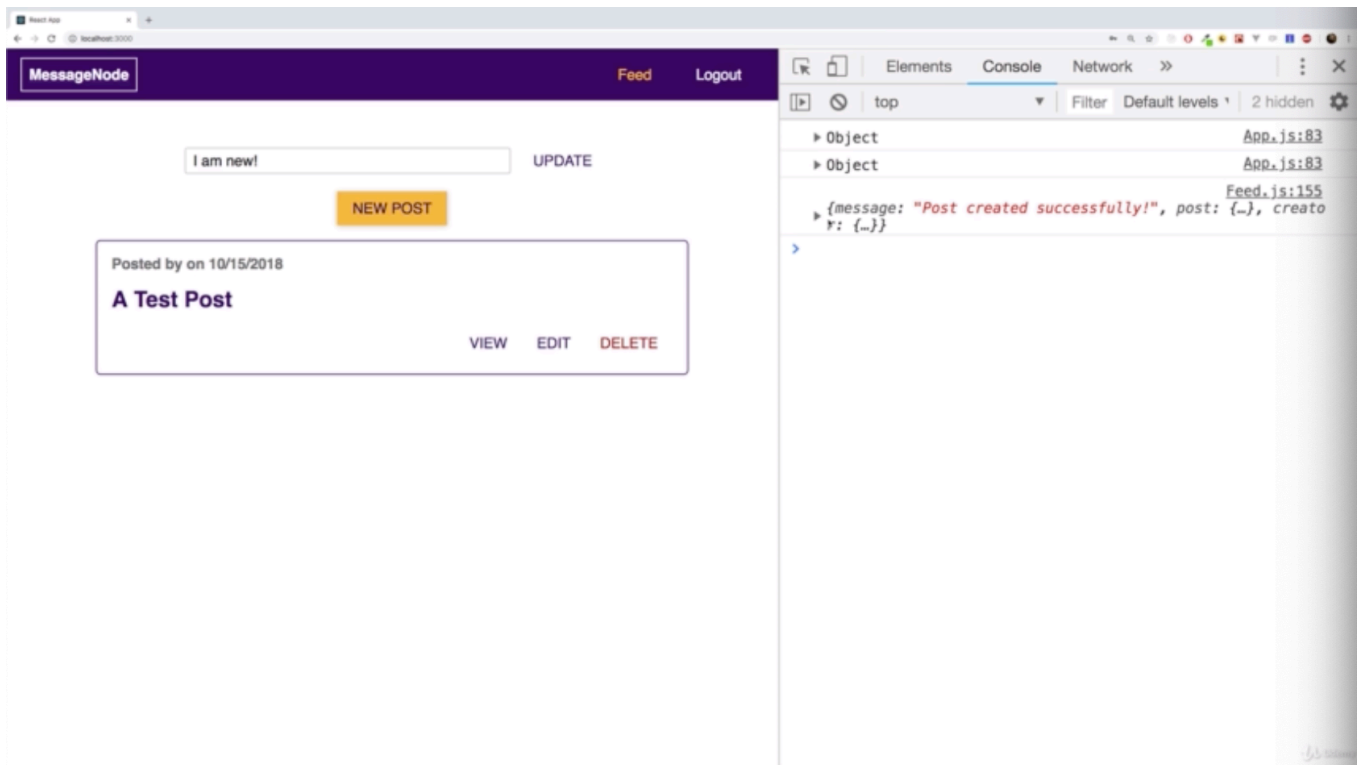
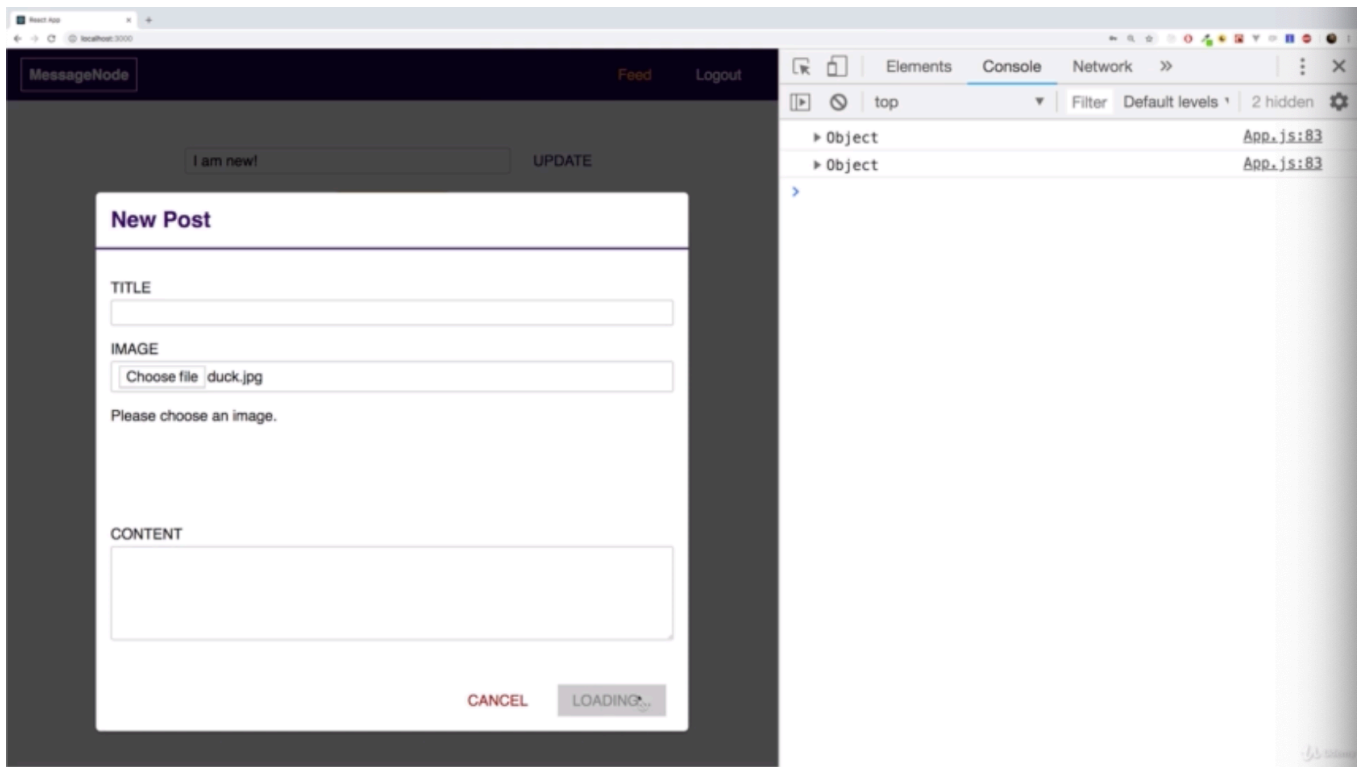


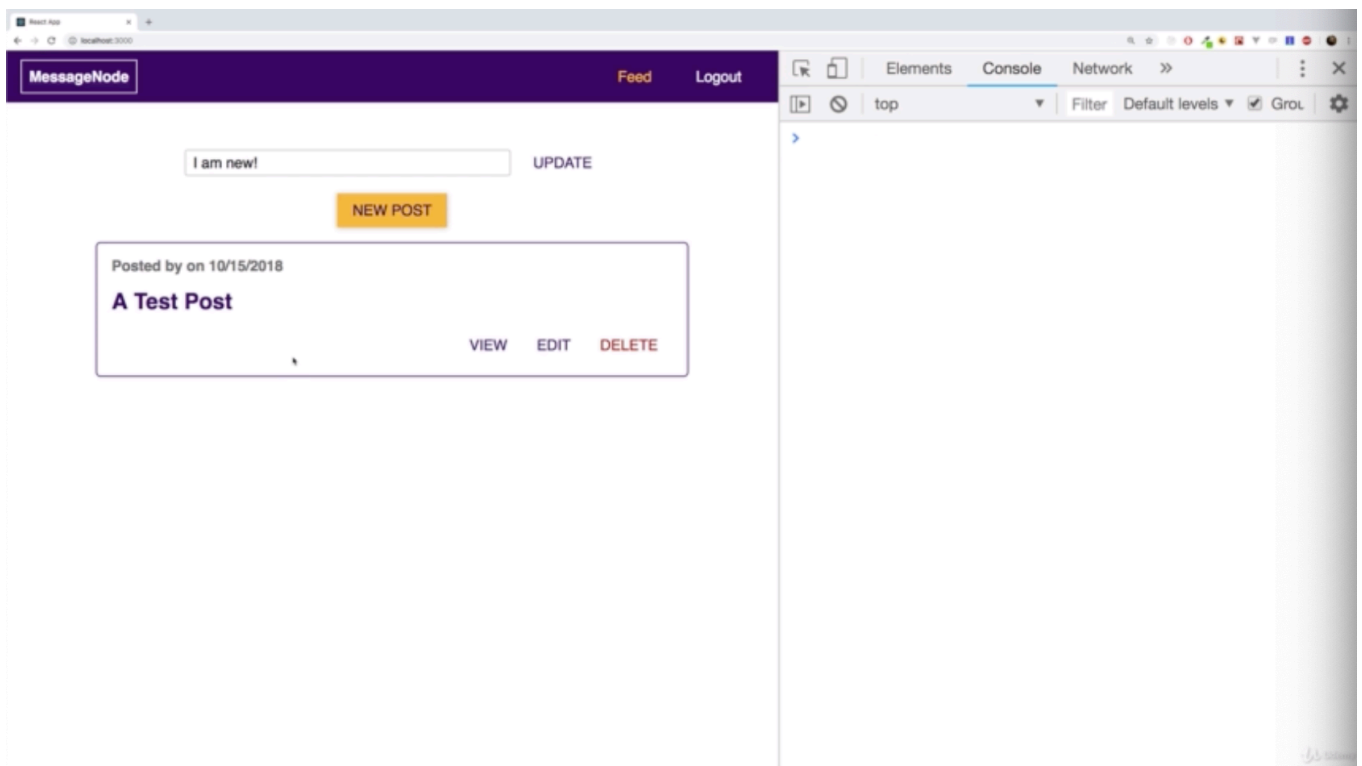












```
1  ../controllers/feed.js(b)
2
3  const fs = require('fs');
4  const path = require('path');
5
6  const { validationResult } = require('express-validator/check');
7
8  const Post = require('../models/post');
9  const User = require('../models/user');
10 /**if you use async and await,
11  * then prepend async in front of arrow function
12  * where you you plan to use await keyword
13  * so where you wanna use these 2 keywords
14  * they always are used together async in front of the function
15  *
16  */
17
18 exports.getPosts = async (req, res, next) => {
19   const currentPage = req.query.page || 1;
20   const perPage = 2;
21   let totalItems;
22   /**since this now runs almost like asynchronous code,
23   * we use 'try and catch'
24   */
25   try {
26     /**you can write this syntax almost
27     * as if it would run synchronously.
28     * you can get your count a new constant or variable you create
29     * by awaiting Post.find().countDocuments()
30     * and then you would get rid of this '.then()' block
31     *
32     * this looks like normal javascript code
33     * but behind the scenes async await it takes your code
34     * and transforms it into the old '.then()'-like structure we used.
35     * so it uses then behind the scenes we just can't see it.
```



```

36  * and we have this more convenient way of writing our asynchronous code.
37  *
38  * await does some behind the scenes transformation of your code
39  * it takes your code and adds then after it
40  * gets the result of that operation
41  * and then stores it in totalItems.
42  * then moves onto the next line execute that inside of that ".then" block
43  * it creates implicitly.
44  *
45  * so the exact same code we had before
46  * this is done by a single await behind the scenes.
47  */
48  const totalItems = await Post.find().countDocuments();
49  const posts = await Post.find()
50    .skip((currentPage - 1) * perPage)
51    .limit(perPage);
52
53  res.status(200).json({
54    message: 'Fetched posts successfully.',
55    posts: posts,
56    totalItems: totalItems
57  });
58  } catch (error) {
59    if (!err.statusCode) {
60      err.statusCode = 500;
61    }
62    next(err);
63  }
64  };
65
66  exports.createPost = (req, res, next) => {
67    const errors = validationResult(req);
68    if (!errors.isEmpty()) {
69      const error = new Error('Validation failed, entered data is incorrect.');
```

```

92     })
93     .then(user => {
94         creator = user;
95         user.posts.push(post);
96         return user.save();
97     })
98     .then(result => {
99         res.status(201).json({
100             message: 'Post created successfully!',
101             post: post,
102             creator: { _id: creator._id, name: creator.name }
103         });
104     })
105     .catch(err => {
106         if (!err.statusCode) {
107             err.statusCode = 500;
108         }
109         next(err);
110     });
111 };
112
113 exports.getPost = (req, res, next) => {
114     const postId = req.params.postId;
115     Post.findById(postId)
116         .then(post => {
117             if (!post) {
118                 const error = new Error('Could not find post. ');
119                 error.statusCode = 404;
120                 throw error;
121             }
122             res.status(200).json({ message: 'Post fetched.', post: post });
123         })
124         .catch(err => {
125             if (!err.statusCode) {
126                 err.statusCode = 500;
127             }
128             next(err);
129         });
130 };
131
132 exports.updatePost = (req, res, next) => {
133     const postId = req.params.postId;
134     const errors = validationResult(req);
135     if (!errors.isEmpty()) {
136         const error = new Error('Validation failed, entered data is incorrect. ');
137         error.statusCode = 422;
138         throw error;
139     }
140     const title = req.body.title;
141     const content = req.body.content;
142     let imageUrl = req.body.image;
143     if (req.file) {
144         imageUrl = req.file.path;
145     }
146     if (!imageUrl) {
147         const error = new Error('No file picked. ');

```

```

148     error.statusCode = 422;
149     throw error;
150 }
151 Post.findById(postId)
152   .then(post => {
153     if (!post) {
154       const error = new Error('Could not find post.');
```

155 error.statusCode = 404;

156 throw error;

157 }

158 if (post.creator.toString() !== req.userId) {

159 const error = new Error('Not authorized!');

160 error.statusCode = 403;

161 throw error;

162 }

163 if (imageUrl !== post.imageUrl) {

164 clearImage(post.imageUrl);

165 }

166 post.title = title;

167 post.imageUrl = imageUrl;

168 post.content = content;

169 return post.save();

170 })

171 .then(result => {

172 res.status(200).json({ message: 'Post updated!', post: result });

173 })

174 .catch(err => {

175 if (!err.statusCode) {

176 err.statusCode = 500;

177 }

178 next(err);

179 });

180 };
181

```

182 exports.deletePost = (req, res, next) => {
183   const postId = req.params.postId;
184   Post.findById(postId)
185     .then(post => {
186       if (!post) {
187         const error = new Error('Could not find post.');
```

188 error.statusCode = 404;

189 throw error;

190 }

191 if (post.creator.toString() !== req.userId) {

192 const error = new Error('Not authorized!');

193 error.statusCode = 403;

194 throw error;

195 }

196 // Check logged in user

197 clearImage(post.imageUrl);

198 return Post.findByIdAndRemove(postId);

199 })

200 .then(result => {

201 return User.findById(req.userId);

202 })

203 .then(user => {

```
204     user.posts.pull(postId);
205     return user.save();
206   })
207   .then(result => {
208     res.status(200).json({ message: 'Deleted post.' });
209   })
210   .catch(err => {
211     if (!err.statusCode) {
212       err.statusCode = 500;
213     }
214     next(err);
215   });
216 };
217
218 const clearImage = filePath => {
219   filePath = path.join(__dirname, '..', filePath);
220   fs.unlink(filePath, err => console.log(err));
221 };
222
```