Handling Errors

Author: Bipul Kumar

2020

# : Handling Errors

Now we are nearing the ending of this course, our app is taking shape. But we are not handling one thing right now, handling errors. Our app does not behave the way we wanted to or even when it does, the user of the app does not. So, we want to have something in place which displays useful message to the user when something goes wrong.

There are all kinds of errors that can happen. Some example can be maybe user just typed in an invalid password or user id combination. So that is not the end of the world for the application. We do not want to break our application because of that.

Lets see how we can handle some errors in the Angular Means Stack App.

1. **Testing different places to handle errors:** Back in our app, lets start with logging in and signing up because things can go wrong because we can simply enter invalid email addresses and password combinations. As of now we are already showing some validation errors to the users but if we are trying to create a user with an email address that is already existing then we get an error on the console and where our app is already broken. That is something we do not want to do we do not want to break our app. We should be able to recover from that and give the user another tries.
   1. Go to the app and try to add another user by signing up with same email ID and observe the console now for errors.
   2. To add that error handling the first simple thing is to ADD a handler in our observable which is responsible to send the erroneous request. So in the case of signing up in the auth.service.ts we will add it in createUser() method where we are handling the response but not handling the error. We can do that by adding the second argument to the subscribe method.
   3. Update CreateUser method in auth.service.ts with below code and understand the highlighted code piece. Here we added the error handler and just logged at this time. Save the file and observe the console in the application now. Here wwe are still loading infinitely but we are able to so the error in the console as of now.

  createUser(email: string, password: string) {

    const authData: AuthData = { email: email, password: password };

    this.http

      .post("http://localhost:3000/api/user/signup", authData)

      .subscribe(response => {

        console.log(response);

      }, error => {

        console.log(error);

      });

  }

* 1. The main issue that we still have that our loader does not stops at all. That’s because in our signup.component.ts we have isloading = true right before we sent requests to create user inside the onsignup method because till now we were not expecting these requests to fail.
  2. Now the simple solution here is to simply return our request observable here on auth.service.ts because we don’t need result in our auth.service.ts anyways but subscribe in sign up component. Lets make changes in auth.service.ts now and update the createUser method there with below code.

createUser(email: string, password: string) {

    const authData: AuthData = { email: email, password: password };

    return this.http

      .post("http://localhost:3000/api/user/signup", authData);

}

* 1. We removed subscribe from this method and will add it to the signup.component.ts’s onsignUp Method. Update the onSignUp method with below code. We are not doing anything with success, so we are passing null for first argument in the subscription and then we are adding error function and are setting isLoading to false inside the method. Now save the files and observe the spinner going away if you are signing up again with the same existing user.

  onSignup(form: NgForm) {

    if (form.invalid) {

      return;

    }

    this.isLoading = true;

    this.authService.createUser(form.value.email, form.value.password)

    .subscribe(null,

      error =>{

        this.isLoading = false;

      });

  }

* 1. Now when we do enter something valid in signup component for registering a new user, we see the app has failed. So, something is still broken in our app. Now the problem here is not that it would not send that request here (it did). The problem is that it does not redirect us. Lets have a look at the auth.service.ts for createuser method, where it can’t redirect us because the logic is not there yet. And now we have moved the subscribe method from service to signup.omponent.ts, we will do the redirection in signup.component.ts
  2. So, error handling is kind of working but not ideally. So we will remove the subscribe method from signup.component.ts again. It was nice seeing one way that we just saw. We will now put it back to auth.service.ts like we had before update the code snippet for signup.component.ts

 onSignup(form: NgForm) {

    if (form.invalid) {

      return;

    }

    this.isLoading = true;

    this.authService.createUser(form.value.email, form.value.password);

  }

* 1. Update the CreateUser() method with below code snippet. Here in Success case we will redirect now to base route now. But in error case, we will do something different as we need to disable that spinner.

 createUser(email: string, password: string) {

    const authData: AuthData = { email: email, password: password };

    return this.http

      .post("http://localhost:3000/api/user/signup", authData)

      .subscribe(() => {

       this.router.navigate["/"];

      });

  }

* 1. We want to disable the loader in login and sign up component now without subscribing to the component because we need to subscribe in our auth.service because of various things we need to do there. Now there is more than one way of working around this.
     1. **Tap Operator –** We could subscribe into components and utilize and operator in the auth service. This operator allows to execute some actions before the observable is done.

Read more about it on link - <https://rxjs-dev.firebaseapp.com/api/operators/tap>. This is perfectly fine but we can also use a different approach.

* + 1. With a different approach we can use the subject we already have like authstatusListener in authservice.ts. We were not using it so far in signup or login components yet as we were not interested so far to know if user is authenticated or not
  1. Lets update the code in signup.component.ts and understand the code.

import { Component, OnInit, OnDestroy } from "@angular/core";

import { NgForm } from "@angular/forms";

import { Subscription } from "rxjs";

import { AuthService } from "../auth.service";

@Component({

  templateUrl: "./signup.component.html",

  styleUrls: ["./signup.component.css"]

})

export class SignupComponent implements OnInit, OnDestroy {

  isLoading = false;

  private authStatusSub: Subscription;

  constructor(public authService: AuthService) {}

  ngOnInit() {

    this.authStatusSub = this.authService.getAuthStatusListener().subscribe(

      authStatus => {

        this.isLoading = false;

      }

    );

  }

  onSignup(form: NgForm) {

    if (form.invalid) {

      return;

    }

    this.isLoading = true;

    this.authService.createUser(form.value.email, form.value.password);

  }

  ngOnDestroy() {

    this.authStatusSub.unsubscribe();

  }

}

* 1. Lets update the createUser() method now in the auth.service.ts. file and understand the code. Now save the files and observe the sign up functionality.

  createUser(email: string, password: string) {

    const authData: AuthData = { email: email, password: password };

    this.http

      .post("http://localhost:3000/api/user/signup", authData)

      .subscribe(() => {

       this.router.navigate(["/"]);

      }, error => {

        this.authStatusListener.next(false);

      });

  }

* 1. Update the same flow in login.component.ts with the same logic as well and Login Component will also do the similar thing.

import { Component, OnInit, OnDestroy } from "@angular/core";

import { NgForm } from "@angular/forms";

import { AuthService } from "../auth.service";

import { Subscription } from "rxjs";

@Component({

  templateUrl: "./login.component.html",

  styleUrls: ["./login.component.css"]

})

export class LoginComponent implements OnInit, OnDestroy {

  isLoading = false;

  private authStatusSub: Subscription;

  constructor(public authService: AuthService) {}

  ngOnInit() {

    this.authStatusSub = this.authService.getAuthStatusListener().subscribe(

      authStatus => {

        this.isLoading = false;

      }

    );

  }

  ngOnDestroy() {

    this.authStatusSub.unsubscribe();

  }

  onLogin(form: NgForm) {

    if (form.invalid) {

      return;

    }

    this.isLoading = true;

    this.authService.login(form.value.email, form.value.password);

  }

}

* 1. Also update the auth.service.ts’s login method with below code

 login(email: string, password: string) {

    const authData: AuthData = { email: email, password: password };

    this.http

      .post<{ token: string; expiresIn: number, userId: string }>(

        "http://localhost:3000/api/user/login",

        authData

      )

      .subscribe(response => {

        const token = response.token;

        this.token = token;

        if (token) {

          const expiresInDuration = response.expiresIn;

          this.setAuthTimer(expiresInDuration);

          this.isAuthenticated = true;

          this.userId = response.userId;

          this.authStatusListener.next(true);

          const now = new Date();

          const expirationDate = new Date(now.getTime() + expiresInDuration \* 1000);

          console.log(expirationDate);

          this.saveAuthData(token, expirationDate, this.userId);

          this.router.navigate(["/"]);

        }

      }, error => {

        this.authStatusListener.next(false);

      });

  }

1. **Testing different places to handle errors:** It will still be nice if we can get an error message out of this. For this we would use angular material component, ofcourse we could build something that can display the error message.
   1. We will use the dialog component from Material. Lets quickly make some code changes. We will take some source code from material. This is a more advanced solution but easy solution can be to just add a div on signup.component.ts that we can bind with some error message using \*ngif structural directive.
   2. Let’s use the dialog component in our solution. For this we can add a global error handler in the application. We can add a new file *“error-interceptor.ts”* file on the app folder (root level)
   3. We can copy the code from auth-interceptor to this error-interceptor file. We can cleanup the code a little bit and will make it like below code.

import {

    HttpInterceptor,

    HttpRequest,

    HttpHandler,

    HttpErrorResponse

  } from "@angular/common/http";

import { catchError } from "rxjs/operators";

import { throwError } from "rxjs";

  export class ErrorInterceptor implements HttpInterceptor {

    constructor() {}

    intercept(req: HttpRequest<any>, next: HttpHandler) {

          return next.handle(req).pipe(

              catchError((error: HttpErrorResponse)  => {

                console.log(error);

                alert(error.error.error.message)

                return throwError(error)

              })

          );

    }

  }

* + 1. Purple highlight – We added the catchError and throwError from rxjs. And we created the intercept similar to auth interceptor which we will add to app.module.ts
    2. Green Highlight – We will handle next handle req. Here we want to handle the response. Handle will also listen to the response there and will return us the response observable. We can hook into it and listen to events. We can use the pipe method provided by rxjs where we can add an operator to the stream. We will add an operator catchError operator to the stream here. As the names suggests, it will allow us to catch an error. Here we will pass a function where we will get error which will be of type HttpErrorResponse.
    3. Blue highlight – We will return an observable inside of catchError even we have have no error. We can use throwError() method which we imported fromrxJS. This will generate a new observable to which we can just pass that error. Now we will just return that observable.
    4. Red Highlight – We can do some error handling error, for the time being we are using alerts and are now reading the message property from error objects. Error object is the response body of the error response.
  1. Lets add this interceptor just similar to auth-interceptor in the app.module’s providers array and add its respective import. Now Save the pages and observe the errors and app alerts when you encounter error in login and signup module.

Import section

import { ErrorInterceptor } from "./error-interceptor";

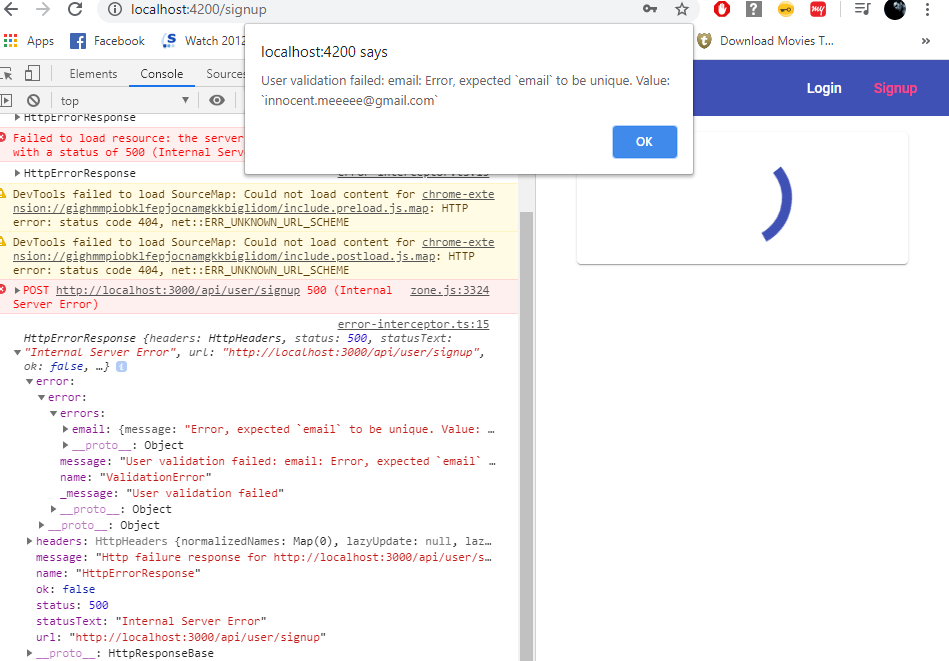
Provider Section.

 providers: [

    { provide: HTTP\_INTERCEPTORS, useClass: AuthInterceptor, multi: true },

    { provide: HTTP\_INTERCEPTORS, useClass: ErrorInterceptor, multi: true }

  ],



1. **Displaying the basic Error Dialog –** Now we have built the error interceptor and displayed the error on boring alert box. Lets now add the material dialog box to see the error.
   1. Update app module to import MatDialogModule for the angular app. You can read more about MatDialogModule in angular material docs

Imports

import { MatDialogModule } from '@angular/material/dialog';

imports array:

imports: [

    BrowserModule,

    AppRoutingModule,

    ReactiveFormsModule,

    FormsModule,

    BrowserAnimationsModule,

    MatInputModule,

    MatCardModule,

    MatButtonModule,

    MatToolbarModule,

    MatExpansionModule,

    MatProgressSpinnerModule,

    MatPaginatorModule,

    MatDialogModule,

    HttpClientModule

  ],

* 1. Now lets create an error component at root folder level with command “ng g c error” in console.
  2. Update Error.Component.html with below code

<h1>An Error Occured </h1>

<p> {{ message }}</p>

* 1. Update Error.component.ts with below code:

import { Component, OnInit } from '@angular/core';

@Component({

  selector: 'app-error',

  templateUrl: './error.component.html',

  styleUrls: ['./error.component.css']

})

export class ErrorComponent implements OnInit {

  constructor() { }

  message: ' An unknown error occured!';

  ngOnInit(): void {

  }

}

* 1. Now we need to do something special. Since we are going to load that component dynamically and not through selector and not through routing, we have to tell angular that it needs to prepared to eventually create this component. And for that we will now update our app module to add a fifth section now which will be called the **entrycomponents** array. We will add errorComponent in this array which will inform angular that this component is going to be used even though angular cant see it.

import { BrowserModule } from "@angular/platform-browser";

import { BrowserAnimationsModule } from "@angular/platform-browser/animations";

import { NgModule } from "@angular/core";

import { ReactiveFormsModule, FormsModule } from "@angular/forms";

import { HttpClientModule, HTTP\_INTERCEPTORS } from "@angular/common/http";

import { MatInputModule } from '@angular/material/input';

import { MatCardModule } from '@angular/material/card';

import { MatButtonModule } from '@angular/material/button';

import { MatToolbarModule } from '@angular/material/toolbar';

import { MatExpansionModule } from '@angular/material/expansion';

import { MatProgressSpinnerModule } from '@angular/material/progress-spinner';

import { MatPaginatorModule } from '@angular/material/paginator';

import { MatDialogModule } from '@angular/material/dialog';

import { AppComponent } from "./app.component";

import { PostCreateComponent } from "./posts/post-create/post-create.component";

import { HeaderComponent } from "./header/header.component";

import { PostListComponent } from "./posts/post-list/post-list.component";

import { AppRoutingModule } from "./app-routing.module";

import { LoginComponent } from "./auth/login/login.component";

import { SignupComponent } from "./auth/signup/signup.component";

import { AuthInterceptor } from "./auth/auth-interceptor";

import { ErrorInterceptor } from "./error-interceptor";

import { ErrorComponent } from './error/error.component';

@NgModule({

  declarations: [

    AppComponent,

    PostCreateComponent,

    HeaderComponent,

    PostListComponent,

    LoginComponent,

    SignupComponent,

    ErrorComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    ReactiveFormsModule,

    FormsModule,

    BrowserAnimationsModule,

    MatInputModule,

    MatCardModule,

    MatButtonModule,

    MatToolbarModule,

    MatExpansionModule,

    MatProgressSpinnerModule,

    MatPaginatorModule,

    MatDialogModule,

    HttpClientModule

  ],

  providers: [

    { provide: HTTP\_INTERCEPTORS, useClass: AuthInterceptor, multi: true },

    { provide: HTTP\_INTERCEPTORS, useClass: ErrorInterceptor, multi: true }

  ],

  bootstrap: [AppComponent],

  entryComponents : [ErrorComponent]

})

export class AppModule {}

* 1. Now lets update the error-interceptor.cs code and understand the code. Here we added the MatDialog service from material and then added the error component. We also added the @Injectable because we are now inserting the MatDialog service into the interceptor. We also added the constructor and added the matdialog service. We are now handling the error where we now call the open method and asking angular to open the ErrorComponent. Save the files and run the application now and try to create the error scenario on login or signup component.

import {

    HttpInterceptor,

    HttpRequest,

    HttpHandler,

    HttpErrorResponse

  } from "@angular/common/http";

import { catchError } from "rxjs/operators";

import { throwError } from "rxjs";

import { Injectable } from "@angular/core";

import { MatDialog } from '@angular/material/dialog';

import { ErrorComponent } from "./error/error.component";

@Injectable()

  export class ErrorInterceptor implements HttpInterceptor {

    constructor( private dialog :  MatDialog) {}

    intercept(req: HttpRequest<any>, next: HttpHandler) {

          return next.handle(req).pipe(

              catchError((error: HttpErrorResponse)  => {

               this.dialog.open(ErrorComponent);

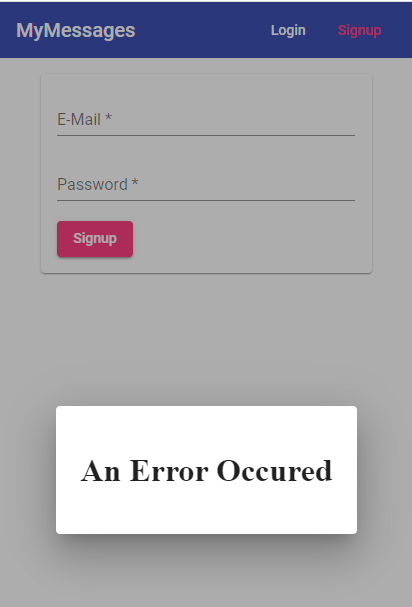
                return throwError(error)

              })

          );

    }

  }



1. **Adding an error Dialog –** We have couple of problems and the fact that we did not see the messages.
   1. Now make some updates in code to see proper messages on the screen. Update the error-interceptor.ts’s intercept method with below code . In below code we are now returning data object where we are passing message property. The errorMessage property has a default value and additionally it also has a check where it can get values from error property

Interceptor method

    intercept(req: HttpRequest<any>, next: HttpHandler) {

          return next.handle(req).pipe(

              catchError((error: HttpErrorResponse)  => {

                let errorMessage = "An Unknown Error Has Occured!";

                if(error.error.message){

                    errorMessage = error.error.message;

                }

               this.dialog.open(ErrorComponent, { data: {message: errorMessage}});

                return throwError(error)

              })

          );

    }

* 1. Now lets update the error.component.ts file with below code. Here we are injecting MAT-DIALOG-DATA into the component which it will get from error interceptor.

import { Component, OnInit, Inject } from '@angular/core';

import { MAT\_DIALOG\_DATA } from '@angular/material/dialog';

@Component({

  selector: 'app-error',

  templateUrl: './error.component.html',

  styleUrls: ['./error.component.css']

})

export class ErrorComponent implements OnInit {

  constructor(@Inject(MAT\_DIALOG\_DATA) public data: {message:string}) { }

  ngOnInit(): void {

  }

}

* 1. Now we can update the error.component.html file with below code for data binding and better styling. Here we have taken code syntax sample from material dialog documentation that we talked about in earlier section in this document. Now save the files and run athe error scenarios on the app and observe the working dialog as shown in screenshot below.

<h1 mat-dialog-title >An Error Occured </h1>

<div mat-dialog-content>

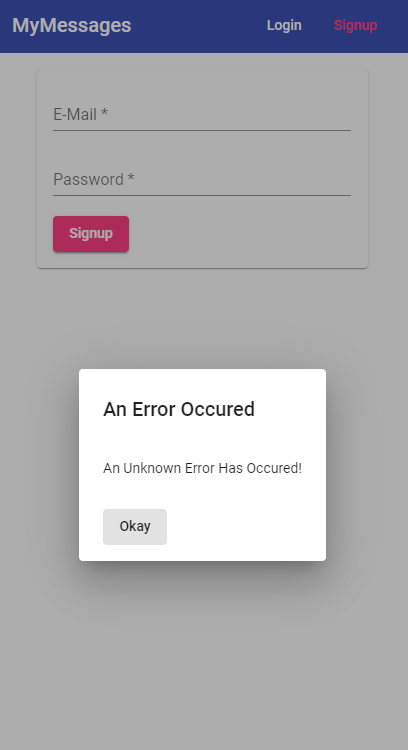
    <p class="mat-body-1"> {{ data.message }}</p>

</div>

<div mat-dialog-actions>

    <button mat-button mat-dialog-close>Okay</button>

</div>



1. **Returning Error Messages on the server –** We have got the error message dialog which shows generic error message so far. Now e can send a proper message from server.
   1. Lets open user.js file in the routes folder of backend nodejs. And update below code

const express = require("express");

const bcrypt = require("bcrypt");

const jwt = require("jsonwebtoken");

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

const router = express.Router();

router.post("/signup", (req, res, next) => {

  bcrypt.hash(req.body.password, 10).then(hash => {

    const user = new User({

      email: req.body.email,

      password: hash

    });

    user

      .save()

      .then(result => {

        res.status(201).json({

          message: "User created!",

          result: result

        });

      })

      .catch(err => {

        res.status(500).json({

          message: "Invalid authentication credentials!"

        });

      });

  });

});

router.post("/login", (req, res, next) => {

  let fetchedUser;

  User.findOne({ email: req.body.email })

    .then(user => {

      if (!user) {

        return res.status(401).json({

          message: "Auth failed"

        });

      }

      fetchedUser = user;

      return bcrypt.compare(req.body.password, user.password);

    })

    .then(result => {

      if (!result) {

        return res.status(401).json({

          message: "Auth failed"

        });

      }

      const token = jwt.sign(

        { email: fetchedUser.email, userId: fetchedUser.\_id },

        "secret\_this\_should\_be\_longer",

        { expiresIn: "1h" }

      );

      res.status(200).json({

        token: token,

        expiresIn: 3600,

        userId: fetchedUser.\_id

      });

    })

    .catch(err => {

      return res.status(401).json({

        message: "Invalid authentication credentials!"

      });

    });

});

module.exports = router;

* 1. Update errors in posts.js as well

const express = require("express");

const multer = require("multer");

const Post = require("../models/post");

const checkAuth = require("../middleware/check-auth");

const router = express.Router();

const MIME\_TYPE\_MAP = {

  "image/png": "png",

  "image/jpeg": "jpg",

  "image/jpg": "jpg"

};

const storage = multer.diskStorage({

  destination: (req, file, cb) => {

    const isValid = MIME\_TYPE\_MAP[file.mimetype];

    let error = new Error("Invalid mime type");

    if (isValid) {

      error = null;

    }

    cb(error, "backend/images");

  },

  filename: (req, file, cb) => {

    const name = file.originalname

      .toLowerCase()

      .split(" ")

      .join("-");

    const ext = MIME\_TYPE\_MAP[file.mimetype];

    cb(null, name + "-" + Date.now() + "." + ext);

  }

});

router.post(

  "",

  checkAuth,

  multer({ storage: storage }).single("image"),

  (req, res, next) => {

    const url = req.protocol + "://" + req.get("host");

    const post = new Post({

      title: req.body.title,

      content: req.body.content,

      imagePath: url + "/images/" + req.file.filename,

      creator: req.userData.userId

    });

    post

      .save()

      .then(createdPost => {

        res.status(201).json({

          message: "Post added successfully",

          post: {

            ...createdPost,

            id: createdPost.\_id

          }

        });

      })

      .catch(error => {

        res.status(500).json({

          message: "Creating a post failed!"

        });

      });

  }

);

router.put(

  "/:id",

  checkAuth,

  multer({ storage: storage }).single("image"),

  (req, res, next) => {

    let imagePath = req.body.imagePath;

    if (req.file) {

      const url = req.protocol + "://" + req.get("host");

      imagePath = url + "/images/" + req.file.filename;

    }

    const post = new Post({

      \_id: req.body.id,

      title: req.body.title,

      content: req.body.content,

      imagePath: imagePath,

      creator: req.userData.userId

    });

    Post.updateOne({ \_id: req.params.id, creator: req.userData.userId }, post)

      .then(result => {

        if (result.nModified > 0) {

          res.status(200).json({ message: "Update successful!" });

        } else {

          res.status(401).json({ message: "Not authorized!" });

        }

      })

      .catch(error => {

        res.status(500).json({

          message: "Couldn't udpate post!"

        });

      });

  }

);

router.get("", (req, res, next) => {

  const pageSize = +req.query.pagesize;

  const currentPage = +req.query.page;

  const postQuery = Post.find();

  let fetchedPosts;

  if (pageSize && currentPage) {

    postQuery.skip(pageSize \* (currentPage - 1)).limit(pageSize);

  }

  postQuery

    .then(documents => {

      fetchedPosts = documents;

      return Post.count();

    })

    .then(count => {

      res.status(200).json({

        message: "Posts fetched successfully!",

        posts: fetchedPosts,

        maxPosts: count

      });

    })

    .catch(error => {

      res.status(500).json({

        message: "Fetching posts failed!"

      });

    });

});

router.get("/:id", (req, res, next) => {

  Post.findById(req.params.id)

    .then(post => {

      if (post) {

        res.status(200).json(post);

      } else {

        res.status(404).json({ message: "Post not found!" });

      }

    })

    .catch(error => {

      res.status(500).json({

        message: "Fetching post failed!"

      });

    });

});

router.delete("/:id", checkAuth, (req, res, next) => {

  Post.deleteOne({ \_id: req.params.id, creator: req.userData.userId })

    .then(result => {

      console.log(result);

      if (result.n > 0) {

        res.status(200).json({ message: "Deletion successful!" });

      } else {

        res.status(401).json({ message: "Not authorized!" });

      }

    })

    .catch(error => {

      res.status(500).json({

        message: "Deleting posts failed!"

      });

    });

});

module.exports = router;

* 1. update the check-auth.js as well with below code. Now all service methods are returning some meaningful error.

const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {

  try {

    const token = req.headers.authorization.split(" ")[1];

    const decodedToken = jwt.verify(token, "secret\_this\_should\_be\_longer");

    req.userData = { email: decodedToken.email, userId: decodedToken.userId };

    next();

  } catch (error) {

    res.status(401).json({ message: "You are not authenticated!" });

  }

};

Save files and see the error messages now. This concludes our module. In the similar way we can handle errors across the applications in a more delicate way.