# Module Introduction

## Introduction



### In the last module we did image upload, now it is time to work with our post data again

### Now I want handle pagination using pagination component from angular material package

## Topics covered

### Adding Pagination component

### Working on pagination backend

### Connecting Angular Paginator to Backend

### Fetching Posts Correctly

### Finishing Touches

# Adding Pagination component to angular project

## THEORY

### We want to give flexibility to user on how many posts(records) he wants to display on the front end

### We will use angular material design package to achieve that

### Angular material design module has many components and paginator is one of them

### NOTE :

### Angular material design is a package which we install at project level ,

### The package dependency is downloaded to node\_modules folder when we do “npm install”.

### But this package has 100s of components and components are logically organized(divided) into modules

### In order to use any component in our project we need to import the corresponding module in our project ( at app.module.ts)

### We will use below paginator component from MatPaginatorModule

### <https://material.angular.io/components/paginator/overview>

### 

### Take a look at below highlighted items

### 

## PRACTICLE:

### Import the MatPaginatorModule at app.module.ts

### 

### Now we have the paginator module imported, we can add <mat-paginator> to post-list.component.html

### 

### The above step is not enough, this component needs some configuration, basically inputs like

#### Length – How many page

#### PageSize - how many posts to display per page

#### PageSizeOptions – User can choose how many posts he wants on the page

### Update the mat paginator component with below 3 property bindings

<mat-paginator [length]="totalPosts" [pageSize]="postsPerPage" [pageSizeOptions]="pageSizeOptions" (page)="onChangedPage($event)"

### Define the corresponding variables in post-list.component.ts file

  totalPosts = 10;

  postsPerPage = 5;

  pageSizeOptions = [1, 2, 5, 10];

### run the app and see the changes

### 

### The next step is to handle user actions when user changes the page (with changing page size from drop down)

### We are talking about 2 component interaction with each other here

### Post List component

### Mat paginator component

### Page list component provided inputs to mat paginator and receives output

### The interaction could be related below diagram

### 

### We have done the part of sending input from post list component to mat paginator

### Let’s do receiving the output from paginator component(child component) into post list component(parent component)

### Add below code to receive the output event and output event data from child component

(page)="onChangedPage($event)"

### 

### Add below event handler in post-list.component.ts file

import { PageEvent } from "@angular/material/paginator";

  onChangedPage(pageData: PageEvent) {

    console.log(pageData);

  }

### Run the App and see the output of PageEvent data

### Click next page and see the results on console , observe all the information sent by event emitter

### 

### Next step is to add some space between post list and paginator

### Add below style in post-create.component.css

mat-paginator {

  margin-top: 1rem;

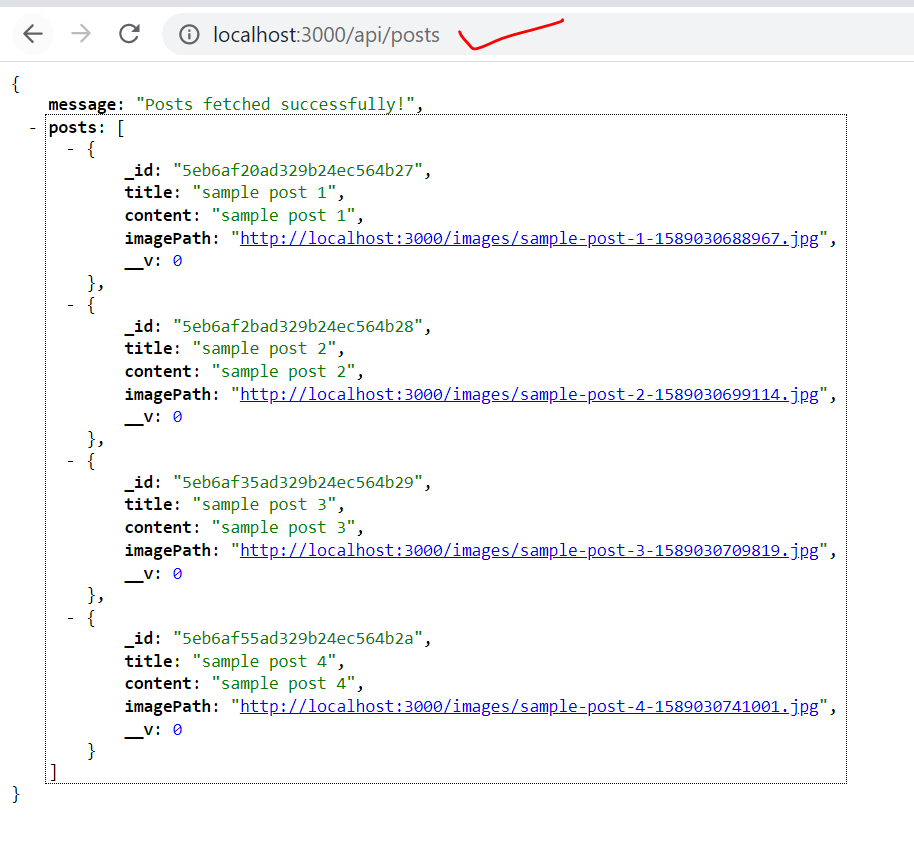
}

### Run the app and see the changes

# Working on pagination backend (server side)

### In this section we will make changes to server side get api to accept parameters to serve data page wise

The get api looks like below , all we need to do is pass the pagination inputs as part of query parameters and accept them in the api.



The uri would like below after adding

<http://localhost:3000/api/posts?pageSize=2&page=3>

We need make below code changes to read the query parameters

Change the get api code as below in file backend/routes/posts.js

router.get("", (req, res, next) => {

  const pageSize = req.query.pagesize;

  const page = req.query.page;

  console.log(pageSize);

  console.log(page);

  Post.find().then((documents) => {

    res.status(200).json({

      message: "Posts fetched successfully!",

      posts: documents,

    });

  });

});

Next step is to change Post.find() mongoose query to use the pageSize and page parameters

router.get("", (req, res, next) => {

  const pageSize = req.query.pagesize;

  const currentPage = req.query.page;

  console.log(pageSize);

  console.log(currentPage);

const postQuery = Post.find();

  // if inputs are valid

  if (pageSize && currentPage) {

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

  }

  postQuery.find().then((documents) => {

    res.status(200).json({

      message: "Posts fetched successfully!",

      posts: documents,

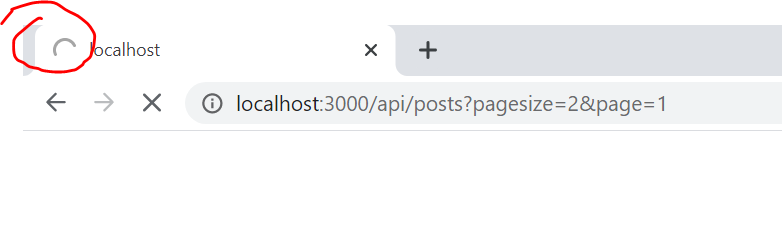
    });

  });

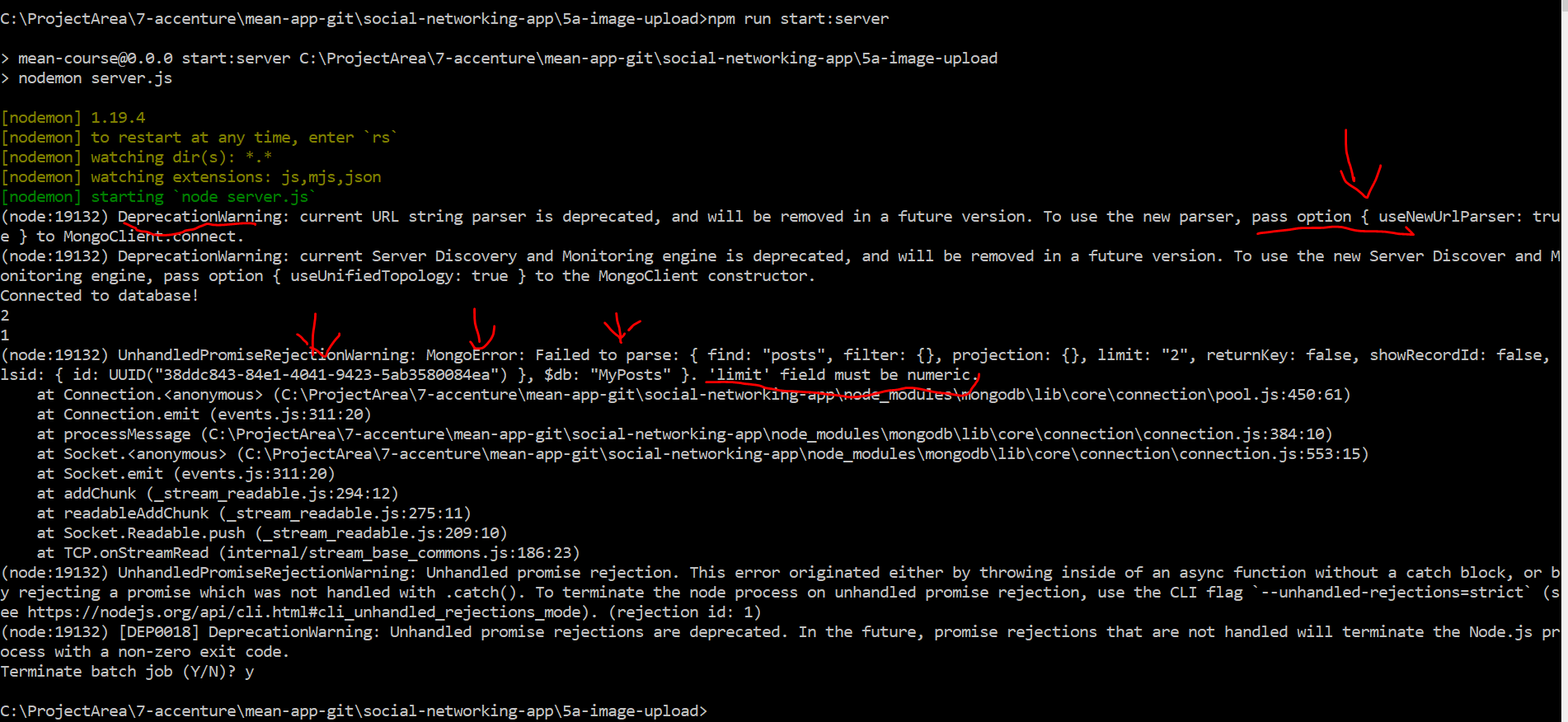
});

Run the api using url

<http://localhost:3000/api/posts?pagesize=2&page=1>



The api will not return the output, we should notice below error on server console



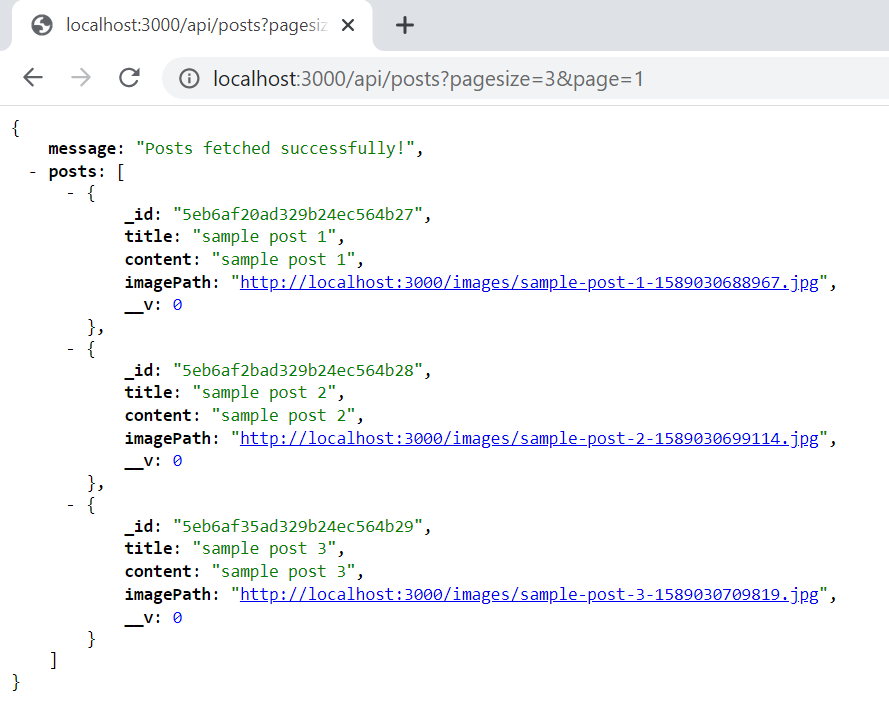
Ignore the deprecation warning and notice the next error , the limit function of find api is expecting numeric.

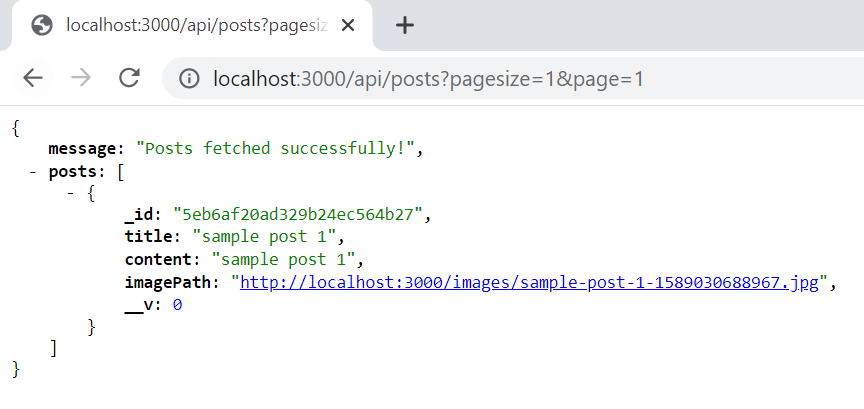
Change the query parameters received to numeric with below code change ( just add + sign)

  const pageSize = +req.query.pagesize;

  const currentPage = +req.query.page;

Run the API the see the changes





# Connecting Angular Paginator to Backend

### In this section we connect angular application to backend server

### Paginator is present in post-list component and post list component calls post service to get data

### Let’s change post service’s get API to send pagination parameters like pagesize and currentPage parameters

### Change the getposts method in post.service.ts as highlighted below

getPosts(postsPerPage: number, currentPage: number) {

    const queryParams = `?pagesize=${postsPerPage}&page=${currentPage}`;

    this.http

      .get<{ message: string; posts: any }>(

        "http://localhost:3000/api/posts" + queryParams

      )

      .pipe(

        map((postData) => {

          return postData.posts.map((post) => {

            return {

              title: post.title,

              content: post.content,

              id: post.\_id,

              imagePath: post.imagePath,

            };

          });

        })

      )

      .subscribe((transformedPosts) => {

        this.posts = transformedPosts;

        this.postsUpdated.next([...this.posts]);

      });

  }

### Make below code changes in post-list component where getposts service call is happening and also onChangedPage event. ( code not provided , type it out looking at below screenshot)

### 

### Run the app and verify

### 

### There are few more issues which we need to fix

#### we can show a spinner after each page change

#### when we have zero posts to show we can hide the paginator

#### in the above screen shot, total number of posts 10 needs to be fetched from database

#### after deleting a post, we need to refresh the screen by re fetching the data from DB

# Fetching Posts Correctly

## Fix the spinner

### we can show a spinner after each page change as is practical cases it could take some time to get data from server.

### We already have implemented spinner in ngOnInit method, we can reuse the same

### Perform the below code change as shown

### 

## Hide paginator

### Hide the paginator by adding a conditional statement as shown below

### 

## Fixing the total count of posts at backend

### In order to fix the total count issue we need to first fetch it from back end

### Lets change the backend get posts api to return not only posts requested but also total number of posts present in the DB

### Example : if there are 100 posts in db and request is for 10 records from 2nd page, the response should include 10 records and also total count of posts 100

### Sample : {posts : [1,2,3,4,5,6,7,8,9], totalPosts: 100}

### DO the below changes to getPosts api in backend

### 

### Code below (changes are highlighted )

router.get("", (req, res, next) => {

  const pageSize = +req.query.pagesize;

  const currentPage = +req.query.page;

  let fetchedPosts;

  console.log(pageSize);

  console.log(currentPage);

  const postQuery = Post.find();

  // if inputs are valid

  if (pageSize && currentPage) {

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

  }

  postQuery

    .find()

    .then((documents) => {

      fetchedPosts = documents;

      return Post.count();

    })

    .then((count) => {

      res.status(200).json({

        message: "Posts fetched successfully!",

        posts: fetchedPosts,

        maxPosts: count,

      });

    });

});

### Run just the backend and see the response returning maxPosts

### 

## Fixing the total count of posts at angular project ( changes to posts.service.js)

### Make the below code changes in post.service.ts as a first step to receive the maxPosts ( totalPostsCount)

### 

### Code to paste for get service function

getPosts(postsPerPage: number, currentPage: number) {

    const queryParams = `?pagesize=${postsPerPage}&page=${currentPage}`;

    this.http

      .get<{ message: string; posts: any; maxPosts: number }>(

        "http://localhost:3000/api/posts" + queryParams

      )

      .pipe(

        map((postData) => {

          return {

            posts: postData.posts.map((post) => {

              return {

                title: post.title,

                content: post.content,

                id: post.\_id,

                imagePath: post.imagePath,

              };

            }),

            maxPosts: postData.maxPosts,

          };

        })

      )

      .subscribe((transformedPostData) => {

        this.posts = transformedPostData.posts;

        this.postsUpdated.next({

          posts: [...this.posts],

          postCount: transformedPostData.maxPosts,

        });

      });

  }

### Explain the above code changes

### The above step still doesn’t fix the problem as response structure of get api is changes with addition of total post count, it will break the code in many places.

### All the other code need to be aligned to this new response starting with

### 

### Change the subject as below to receive both posts and postcount

  private postsUpdated = new Subject<{ posts: Post[]; postCount: number }>();

### update addpost by removing the unwanted code as we are navigating to root refresh the component

### 

### Code

  addPost(title: string, content: string, image: File) {

    const postData = new FormData();

    postData.append("title", title);

    postData.append("content", content);

    postData.append("image", image, title);

    this.http

      .post<{ message: string; post: Post }>(

        "http://localhost:3000/api/posts",

        postData

      )

      .subscribe(() => {

        this.router.navigate(["/"]);

      });

  }

### Remove unwanted to code after subscription for updatePost as well

  updatePost(id: string, title: string, content: string, image: File | string) {

    let postData: Post | FormData;

    if (typeof image === "object") {

      postData = new FormData();

      postData.append("id", id);

      postData.append("title", title);

      postData.append("content", content);

      postData.append("image", image, title);

    } else {

      postData = {

        id: id,

        title: title,

        content: content,

        imagePath: image,

      };

    }

    this.http

      .put("http://localhost:3000/api/posts/" + id, postData)

      .subscribe((response) => {

        this.router.navigate(["/"]);

      });

  }

### In case of delete we need go to server and fetch the data again. The elegent way of doing this to remove subscription here and performing the fetch inside post-list component

### 

### Change the code to

### 

### After above code changes is posts.service.ts we need to make some more changes in post-list.component.ts

## Changes in post-list.component.ts

### Make changes to delete as shown below and explain

### 

### Make below code changes to ngOnInit()

### 

### Run the app and you should be all set