**EXPRESS JS**

**Package Commands to install**

* npm init
* npm i express dotenv
* npm i –D nodemon
* npm i morgan

**Run Files**

* Npm start
* Npm run dev
* Nodemon

**Package. JSON**

{

  "name": "practice",

  "version": "1.0.0",

  "description": "",

  "main": "server.js",

  "scripts": {

    "start": "SET NODE\_ENV=PRODUCTION & node server",

    "dev": "nodemon server"

  },

  "author": "",

  "license": "ISC",

  "devDependencies": {

    "nodemon": "^2.0.4"

  },

  "dependencies": {

    "dotenv": "^8.2.0",

    "express": "^4.17.1",

    "morgan": "^1.10.0"

  }

}

**Server .JS**

const express= require('express');

const dotenv =require('dotenv');

const router=require('./routers/router')

const morgan=require('morgan');

dotenv.config({path:'./config/config.env'});

const app=express();

if(process.env.NODE\_ENV==='DEVELOPMENT')

{

    app.use(morgan('dev'))

}

app.use('/api/page',router);

const port=process.env.PORT || 5000;

app.listen(port,console.log(`server running in ${process.env.NODE\_ENV} mode using port  ${port}`));

**Router. JS**

const express =  require('express');

const {

    getData,

    createData,

    getSingleData,

    updateSingleData,

    deleteSingleData } = require('../controllers/controller');

const router =express.Router();

router.route('/')

.get(getData)

.post(createData);

router.route('/:id')

.get(getSingleData)

.put(updateSingleData)

.delete(deleteSingleData);

module.exports=router;

**Controller. JS**

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

    res.status(200).send(

        {

            success:true,

            msg:'view all data'

        }

    )

}

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

    res.status(200).send(

        {

            success:true,

            msg:'create data'

        }

    )

}

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

    res.status(200).send(

        {

            success:true,

            msg:`view data of id ${req.params.id}`

        }

    )

}

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

    res.status(200).send(

        {

            success:true,

            msg:`update data of id ${req.params.id}`

        }

    )

}

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

    res.status(200).send(

        {

            success:true,

            msg:`delete data of id ${req.params.id}`

        }

    )

}

**Config. env**

NODE\_ENV=DEVELOPMENT

PORT=5000

**MONGO DB**

**Run and Install Setup**

* **mongodb.com**
  + Login
  + Create cluster
  + Database access
  + Network access
* **MongoDB Compass download install**
  + Connect with compass
  + Connect with application

**Package Commands to install**

* npm i mongoose
* npm I colors

**Config .env**

NODE\_ENV = DEVELOPMENT

PORT = 5000

MONGO\_URI = mongodb+srv://shairy:shairy@123@demo.d8fem.mongodb.net/demo?retryWrites=true&w=majority

**db .js**

const mongoose = require('mongoose');

const connectDB=async()=>

{

    const conn= await mongoose.connect(process.env.MONGO\_URI,{

        useNewUrlParser:true,

        useCreateIndex:true,

        useFindAndModify:false,

         useUnifiedTopology: true

    });

    console.log(`MongoDB Connected: ${conn.connection.host}`);

};

module.exports = connectDB;

**Modals- Register.js(Table)**

const mongoose =require('mongoose');

const RegisterSchema = new mongoose.Schema({

    name:{

        type : String,

        required:[true,'Please enter you name'],

        unique:true,

        trim:true,

        maxlength: [50,'name cant be more than 50 charcter']

    },

    email:{

        type:String,

        required:[true,'Please add avalid Email']

         },

    address:{

        type:String,

        required:[true,'Please enter you address'],

         },

    course:{

        type:[String],

        required:true,

        enum:['php',

              'java',

              'dotnet']

    },created:{

        type:Date,

        default:Date.now

    }

});

module.exports=mongoose.model('Register',RegisterSchema);

**Controller.js**

const Register= require('../models/Register')

exports.getData=async(req,res,next)=>{

     try{

        const reg = await Register.find();

        res.status(200).json({

                success:true,

                data:reg

            });

       }catch(err)

       {

           res.status(400).json({

                success:false,

            });

       }

}

exports.createData=async(req,res,next)=>{

    try{

        const reg = await Register.create(req.body);

        res.status(201).json({

                success:true,

                data:reg

            });

       }catch(err)

       {

           res.status(400).json({

                success:false,

                err: err

            });

       }

};

exports.getSingleData=async(req,res,next)=>{

       try{

        const reg = await Register.findById(req.params.id);

        if(!reg)

        {

            return res.status(400).json({ success:false});

        }

        res.status(200).json({

                success:true,

                data:reg

            });

       }catch(err)

       {

            console.log(err);

           res.status(400).json({

                success:false,

            });

       }

}

exports.updateSingleData=async (req,res,next)=>{

    try{

        const reg = await Register.findByIdAndUpdate(req.params.id,req.body,

            {

             new:true,

             runValidators:true

            });

        if(!reg)

        {

            return res.status(400).json({ success:false});

        }

        res.status(200).json({

                success:true,

                data:reg

            });

       }catch(err)

       {

            console.log(err);

           res.status(400).json({

                success:false,

            });

       }

}

exports.deleteSingleData=async(req,res,next)=>{

     try{

        const reg = await Register.findByIdAndDelete(req.params.id);

        if(!reg)

        {

            return res.status(400).json({ success:false});

        }

        res.status(200).json({

                success:true,

                data:{}

            });

       }catch(err)

       {

            console.log(err);

           res.status(400).json({

                success:false,

            });

       }

}

**Server.js**

const express= require('express');

const dotenv =require('dotenv');

const morgan=require('morgan');

const connectDB= require('./config/db');

dotenv.config({path:'./config/config.env'});

connectDB();

const router=require('./routers/router')

const app=express();

app.use(express.json());

if(process.env.NODE\_ENV==='DEVELOPMENT')

{

    app.use(morgan('dev'))

}

app.use('/api/page',router);

const port=process.env.PORT || 5000;

const server=app.listen(

    port,

    console.log(`server running in ${process.env.NODE\_ENV} mode using port  ${port}`));

process.on('unhandledRejection',(err,promise)=>{

    console.log(`Error: ${err.message}`);

    server.close(()=>process.exit(1));

})

**Error Handling**

**Middleware=>async.js**

const asyncHandler = fn=>(req,res,next)=>

Promise.resolve(fn(req,res,next)).catch(next);

module.exports=asyncHandler

**Middleware=>error.js**

const ErrorResponse = require('../util/errorResponse')

const errorHandler = (err,req,res,next)=>

{

    let error = {...err};

    error.message=err.message;

    console.log(err);

    if(err.name==='CastError'){

        const message=`User not found with id of ${err.value}`;

        error= new ErrorResponse(message,404)

    }

    if(err.code===11000)

    {

        const message='duplicate field value';

        error= new ErrorResponse(message,400)

    }

    if(err.name==='ValidationError'){

         const message=Object.values(err.errors).map(val=>val.message);

        error= new ErrorResponse(message,400)

    }

    res.status(error.statusCode || 500).json({

        suceess:false,

        error:error.message || 'Server Error'

    });

};

module.exports = errorHandler;

**util=>errorresponse.js**

class ErrorResponse extends Error{

    constructor(message,statusCode){

        super(message);

        this.statusCode=statusCode;

    }

}

module.exports = ErrorResponse;

**controller.js**

const ErrorResponse= require('../util/errorResponse')

const asyncHandler =require('../middleware/async')

const Register= require('../models/Register')

exports.getData=asyncHandler(async(req,res,next)=>{

        const reg = await Register.find();

        res.status(200).json({success:true,data:reg});

       });

exports.createData=asyncHandler(async(req,res,next)=>{

        const reg = await Register.create(req.body);

        res.status(201).json({success:true,data:reg});

       });

exports.getSingleData=asyncHandler(async(req,res,next)=>{

        const reg = await Register.findById(req.params.id);

        if(!reg)

        {

          return next(new ErrorResponse(`User not found with id of ${req.params.id}`,404));

        }

        res.status(200).json({success:true,data:reg});

       });

exports.updateSingleData=asyncHandler(async (req,res,next)=>{

        const reg = await Register.findByIdAndUpdate(req.params.id,req.body,

            {

             new:true,

             runValidators:true

            });

        if(!reg)

        {

             return next(new ErrorResponse(`User not found with id of ${req.params.id}`,404));

        }

        res.status(200).json({success:true,data:reg });

       });

exports.deleteSingleData=asyncHandler(async(req,res,next)=>{

        const reg = await Register.findByIdAndDelete(req.params.id);

        if(!reg)

        {

           return next(new ErrorResponse(`User not found with id of ${req.params.id}`,404));

        }

        res.status(200).json({success:true,data:{} });

       });

**Server.js**

const express= require('express');

const dotenv =require('dotenv');

const morgan=require('morgan');

const errorHandler= require('./middleware/error');

const connectDB= require('./config/db');

dotenv.config({path:'./config/config.env'});

connectDB();

const router=require('./routers/router')

const app=express();

app.use(express.json());

if(process.env.NODE\_ENV==='DEVELOPMENT')

{

    app.use(morgan('dev'))

}

app.use('/api/page',router);

app.use(errorHandler);

const port=process.env.PORT || 5000;

const server=app.listen(

    port,

    console.log(`server running in ${process.env.NODE\_ENV} mode using port  ${port}`));

process.on('unhandledRejection',(err,promise)=>{

    console.log(`Error: ${err.message}`);

    server.close(()=>process.exit(1));

})

**Seeder.js**

**Package Commands to install**

* node seeder –d
* node seeder -i

const fs =require('fs');

const mongoose = require('mongoose');

const dotenv= require('dotenv');

//load env variable

dotenv.config({path:'./config/config.env'});

//Load Models

const model= require('./model/assignment');

//connect DB

mongoose.connect(process.env.MONGO\_URI,

        {

            useNewUrlParser:true,

            useFindAndModify:false,

            useCreateIndex:true,

            useUnifiedTopology:true

        });

//Read JSON File

const assignments = JSON.parse(

    fs.readFileSync(`${\_\_dirname}/\_data/assignment.json`,'utf-8'))

//Import into DB

const importData = async()=>

{

    try{

        await model.create(assignments);

        console.log('Data Imported...');

        process.exit();

    }

    catch(err)

    {

        console.error(err);

    }

}

//Delete data

const deleteData = async()=>

{

    try{

        await model.deleteMany();

        console.log('Data Destryed..');

        process.exit();

    }

    catch(err)

    {

        console.error(err);

    }

}

if(process.argv[2]=== '-i')

{

    importData();

}

else if(process.argv[2]=== '-d')

{

    deleteData();

}

**Controller.js**

**For Numeric Operation**

exports.getAssignment=asyncHandler(async(req,res,next)=>{

   let query;

   let querystr = JSON.stringify(req.query);

   querystr =querystr.replace(/\b(gt|gte|lt|lte|in)\b/g, match=>`$${match}`);

   console.log(querystr);

 query=assignment.find(JSON.parse(querystr));

        const getAssignment= await query;

        res.status(200).send({success:true,count:getAssignment.length,data:getAssignment})

});

**Select and sort**

exports.getAssignment=asyncHandler(async(req,res,next)=>{

   let query;

   //copy req.query

   const resquery={...req.query};

   //fiels to exclude

   const removefield=['select','sort'];

   //loop over removefield and delete from reqquery

   removefield.forEach(param=>delete resquery[param]);

   console.log(resquery)

   let querystr = JSON.stringify(resquery);

   //create operator for relational value

   querystr =querystr.replace(/\b(gt|gte|lt|lte|in)\b/g, match=>`$${match}`);

   console.log(querystr);

   //Finding resorce

   query=assignment.find(JSON.parse(querystr));

   //select fields

   if(req.query.select){

    const fields=req.query.select.split(',').join(' ');

    query=query.select(fields);

   }

   //sort

if(req.query.sort){

    const sortby=req.query.sort.split(',').join(' ');

    query=query.sort(sortby);

   }

   else{

       query=query.sort('-created')

   }

   //execute query

   const getAssignment= await query;

   res.status(200).send({success:true,count:getAssignment.length,data:getAssignment})

});

**Pagination**

exports.getAssignment=asyncHandler(async(req,res,next)=>{

   let query;

   //copy req.query

   const resquery={...req.query};

   //fiels to exclude

   const removefield=['select','sort','page','limit'];

   //loop over removefield and delete from reqquery

   removefield.forEach(param=>delete resquery[param]);

   console.log(resquery)

   let querystr = JSON.stringify(resquery);

   //create operator for relational value

   querystr =querystr.replace(/\b(gt|gte|lt|lte|in)\b/g, match=>`$${match}`);

   console.log(querystr);

   //Finding resorce

   query=assignment.find(JSON.parse(querystr));

   //select fields

   if(req.query.select){

    const fields=req.query.select.split(',').join(' ');

    query=query.select(fields);

   }

   //sort

if(req.query.sort){

    const sortby=req.query.sort.split(',').join(' ');

    query=query.sort(sortby);

   }

   else{

       query=query.sort('-created')

   }

   //pagination

   const page= parseInt(req.query.page,10)|| 1;

   const limit= parseInt(req.query.limit,10)|| 1;

   const startIndex= (page - 1)\* limit;

   const endIndex= page \* limit;

   const total=await assignment.countDocuments();

   query=query.skip(startIndex).limit(limit);

   //execute query

   const getAssignment= await query;

   //pagination result

   const pagination={};

   if(endIndex<total)

   {

       pagination.next={

           page:page+1,

           limit

       }

   }

   if(startIndex>0)

   {

       pagination.prev={

           page:page-1,

           limit

       }

   }

   res.status(200).send({success:true,count:getAssignment.length,pagination,data:getAssignment})

});

**Model=>course.js**

const mongoose=require('mongoose');

const courseSchema = new mongoose.Schema({

    title:{

        type : String,

        required:[true,'Please enter you course Title'],

    },

    description:{

        type:String,

        required:[true,'Please add a valid Description']

         },

    created:{

        type:Date,

        default:Date.now

    },

    assignment:{

        type:mongoose.Schema.ObjectId,

        ref:'assignment',

        required:true

    }

});

module.exports=mongoose.model('course',courseSchema);

**controller course.js**

const ErrorResponse= require('../util/errorResponse')

const asyncHandler =require('../middleware/async')

const course=require('../model/course');

exports.getCourses=asyncHandler(async(req,res,next)=>{

   let query;

   if(req.params.assignmentId)

   {

        query=course.find({assignment:req.params.assignmentId});

   }

   else{

       query=course.find();

   }

   const courses =await query;

   res.status(200).json({

       success:true,

       count: course.length,

       data:courses

   })

})

**seeder.js**

const fs =require('fs');

const mongoose = require('mongoose');

const dotenv= require('dotenv');

//load env variable

dotenv.config({path:'./config/config.env'});

//Load Models

const assignment= require('./model/assignment');

const course= require('./model/course');

//connect DB

mongoose.connect(process.env.MONGO\_URI,

        {

            useNewUrlParser:true,

            useFindAndModify:false,

            useCreateIndex:true,

            useUnifiedTopology:true

        });

//Read JSON File

const assignments = JSON.parse(

    fs.readFileSync(`${\_\_dirname}/\_data/assignment.json`,'utf-8'))

const courses = JSON.parse(

    fs.readFileSync(`${\_\_dirname}/\_data/course.json`,'utf-8'))

//Import into DB

const importData = async()=>

{

    try{

        await assignment.create(assignments);

        await course.create(courses);

        console.log('Data Imported...');

        process.exit();

    }

    catch(err)

    {

        console.error(err);

    }

}

//Delete data

const deleteData = async()=>

{

    try{

        await assignment.deleteMany();

        await course.deleteMany();

        console.log('Data Destryed..');

        process.exit();

    }

    catch(err)

    {

        console.error(err);

    }

}

if(process.argv[2]=== '-i')

{

    importData();

}

else if(process.argv[2]=== '-d')

{

    deleteData();

}

**router-courses**

const express = require('express');

const

{getCourses

}=require('../controller/courses');

const router = express.Router({mergeParams:true});

router.route('/').get(getCourses);

module.exports=router;

**router.js**

const express = require('express');

const {getAssignment,

    createAssignment,

    getOneAssignment,

    updateAssignment,

    deleteAssignment}=require('../controller/controller');

/// include other resource router

const courseRouter= require('./courses');

const router = express.Router();

// Re route inro another

router.use('/:assignmentId/courses',courseRouter)

router.route('/')

.get(getAssignment)

.post(createAssignment);

router.route('/:id')

.get(getOneAssignment)

.delete(deleteAssignment)

.put(updateAssignment);

module.exports=router;

**server.js**

const express =require('express');

const dotenv =require('dotenv');

const morgan= require('morgan');

const route=require('./router/router');

const courses=require('./router/courses');

const connectDB = require('./config/db')

const errorHandler= require('./middleware/error');

dotenv.config({path:'./config/config.env'});

connectDB();

const app=express();

if(process.env.NODE\_ENV==='DEVELOPMENT')

{

    app.use(morgan('dev'))

}

app.use(express.json())

app.use('/api/assignment',route);

app.use('/api/courses',courses);

app.use(errorHandler);

const port=process.env.PORT || 5000;

const server=app.listen(port,console.log(`server running in

${process.env.NODE\_ENV} mode using port  ${port}`));

process.on('unhandledRejection',(err,promise)=>{

    console.log(`Error : ${err.message}`);

    server.close(()=>process.exit(1));

})

**Populate/Virtual/ Reverse Populate**

**model**

const mongoose=require('mongoose');

const assignmentSchema = new mongoose.Schema({

    assignmentHeading:{

        type : String,

        required:[true,'Please enter you Heading'],

    },

    assignmentDescription:{

        type:String,

        required:[true,'Please add a valid Description']

         },

    course:{

        type:[String],

        required:true,

        enum:['php',

              'java',

              'dotnet']

    },

    averageCost:{

         type:Number,

        required:true,

    }

    ,created:{

        type:Date,

        default:Date.now

    }

},

{

   toObject:{virtuals:true} ,

   toJSON:{virtuals:true}

}

);

//cascade delete

assignmentSchema.pre('remove',async function(next){

   console.log(`course removed from assignment ${this.\_id}`);

    await this .model('course').deleteMany({assignment:this,\_id})

    next();

})

//reverse populate

assignmentSchema.virtual('courses',{

    ref:'course',

    localField:'\_id',

    foreignField:'assignment',

    justOne:false

})

module.exports=mongoose.model('assignment',assignmentSchema);

**controller - courses**

const ErrorResponse= require('../util/errorResponse')

const asyncHandler =require('../middleware/async')

const course=require('../model/course');

exports.getCourses=asyncHandler(async(req,res,next)=>{

   let query;

   if(req.params.assignmentId)

   {

        query=course.find({assignment:req.params.assignmentId});

   }

   else{

       query=course.find().populate({path:'assignment',select:'assignmentHeading'});

   }

   const courses =await query;

   res.status(200).json({

       success:true,

       count: course.length,

       data:courses

   })

})

**Controller-controller**

const ErrorResponse= require('../util/errorResponse')

const asyncHandler =require('../middleware/async')

const assignment=require('../model/assignment');

exports.getAssignment=asyncHandler(async(req,res,next)=>{

   let query;

   //copy req.query

   const resquery={...req.query};

   //fiels to exclude

   const removefield=['select','sort','page','limit'];

   //loop over removefield and delete from reqquery

   removefield.forEach(param=>delete resquery[param]);

   console.log(resquery)

   let querystr = JSON.stringify(resquery);

   //create operator for relational value

   querystr =querystr.replace(/\b(gt|gte|lt|lte|in)\b/g, match=>`$${match}`);

   console.log(querystr);

   //Finding resorce

   query=assignment.find(JSON.parse(querystr)).populate('courses');

   //select fields

   if(req.query.select){

    const fields=req.query.select.split(',').join(' ');

    query=query.select(fields);

   }

   //sort

if(req.query.sort){

    const sortby=req.query.sort.split(',').join(' ');

    query=query.sort(sortby);

   }

   else{

       query=query.sort('-created')

   }

   //pagination

   const page= parseInt(req.query.page,10)|| 1;

   const limit= parseInt(req.query.limit,10)|| 1;

   const startIndex= (page - 1)\* limit;

   const endIndex= page \* limit;

   const total=await assignment.countDocuments();

   query=query.skip(startIndex).limit(limit);

   //execute query

   const getAssignment= await query;

   //pagination result

   const pagination={};

   if(endIndex<total)

   {

       pagination.next={

           page:page+1,

           limit

       }

   }

   if(startIndex>0)

   {

       pagination.prev={

           page:page-1,

           limit

       }

   }

   res.status(200).send({success:true,count:getAssignment.length,pagination,data:getAssignment})

});

exports.createAssignment=asyncHandler(async(req,res,next)=>{

        const createAssignment=await assignment.create(req.body);

        res.status(200).json({success:true,data:createAssignment})

});

exports.getOneAssignment=asyncHandler(async(req,res,next)=>{

    const getOneAssignment= await assignment.findById(req.params.id);

    res.status(200).send({success:true,data:getOneAssignment})

});

exports.updateAssignment=asyncHandler(async(req,res,next)=>{

     const updateAssignment= await assignment.findByIdAndUpdate(req.params.id,req.body);

    res.status(200).send({success:true,data:updateAssignment})

});

exports.deleteAssignment=asyncHandler(async(req,res,next)=>{

     const deleteAssignment=await assignment.findById(req.params.id);

   assignment.remove();

   res.status(200).send({success:true,data:deleteAssignment})

});

**File Upload**

**Package Commands to install**

* npm I express-fileupload

exports.photoUpload =asyncHandler(async(req,res,next)=>{

     const photoUpload= await assignment.findById(req.params.id);

    if(!photoUpload)

    {

        return next(

            new ErrorResponse(`Photo Not found with id ${req.params.id}`,404)

        );

    }

    if(!req.files)

    {

        return next(

            new ErrorResponse(`Please Upload a file`,400)

        );

    }

const file=req.files.file;

if(!file.mimetype.startsWith('image'))

    {

        return next(

            new ErrorResponse(`Please Upload an image file`,400)

        );

    }

    if(!file.size > process.env.MAX\_FILE\_UPLOAD)

    {

        return next(

            new ErrorResponse(`Please Upload a file LESS THAN ${process.env.MAX\_FILE\_UPLOAD}`,400)

        );

    }

    //create custom file name

    file.name=`photo\_${photoUpload.\_id}${path.parse(file.name).ext}`;

    file.mv(`${process.env.FILE\_UPLOAD\_PATH}/${file.name}`,async err=>{

        if(err)

        {

            console.error(err);

            return next(new ErrorResponse(`problrem with fil upload`),500)

        }

        await assignment.findByIdAndUpdate(req.params.id,{photo:file.name});

        res.status(200).send({success:true,data:file.name})

    });

    console.log(file.name);

});

**Config.env**

**FILE\_UPLOAD\_PATH=./public/uploads**

**MAX\_FILE\_UPLOAD=1000000**

**Server,js**

const path= require('path');

const express =require('express');

const dotenv =require('dotenv');

const morgan= require('morgan');

const fileupload = require('express-fileupload');

const route=require('./router/router');

const courses=require('./router/courses');

const connectDB = require('./config/db')

const errorHandler= require('./middleware/error');

dotenv.config({path:'./config/config.env'});

connectDB();

const app=express();

if(process.env.NODE\_ENV==='DEVELOPMENT')

{

    app.use(morgan('dev'))

}

//File upload

app.use(fileupload());

//static folder

app.use(express.static(path.join(\_\_dirname,'public')));

app.use(express.json())

app.use('/api/assignment',route);

app.use('/api/courses',courses);

app.use(errorHandler);

const port=process.env.PORT || 5000;

const server=app.listen(port,console.log(`server running in

${process.env.NODE\_ENV} mode using port  ${port}`));

process.on('unhandledRejection',(err,promise)=>{

    console.log(`Error : ${err.message}`);

    server.close(()=>process.exit(1));

})

**All filter outside for each router**

**Middleware->advance Result**

const advancedResult =(model,populate)=>async(req,res,next)=>{

let query;

   //copy req.query

   const resquery={...req.query};

   //fiels to exclude

   const removefield=['select','sort','page','limit'];

   //loop over removefield and delete from reqquery

   removefield.forEach(param=>delete resquery[param]);

   console.log(resquery)

   let querystr = JSON.stringify(resquery);

   //create operator for relational value

   querystr =querystr.replace(/\b(gt|gte|lt|lte|in)\b/g, match=>`$${match}`);

   console.log(querystr);

   //Finding resorce

   query=model.find(JSON.parse(querystr));

   //select fields

   if(req.query.select){

    const fields=req.query.select.split(',').join(' ');

    query=query.select(fields);

   }

   //sort

if(req.query.sort){

    const sortby=req.query.sort.split(',').join(' ');

    query=query.sort(sortby);

   }

   else{

       query=query.sort('-created')

   }

   //pagination

   const page= parseInt(req.query.page,10)|| 1;

   const limit= parseInt(req.query.limit,10)|| 1;

   const startIndex= (page - 1)\* limit;

   const endIndex= page \* limit;

   const total=await model.countDocuments();

   query=query.skip(startIndex).limit(limit);

   if(populate)

   {

       query=query.populate(populate);

   }

   //execute query

   const results= await query;

   //pagination result

   const pagination={};

   if(endIndex<total)

   {

       pagination.next={

           page:page+1,

           limit

       }

   }

   if(startIndex>0)

   {

       pagination.prev={

           page:page-1,

           limit

       }

   }

   res.advancedResult={

       success:true,

       count:results.length,

       pagination,

       data:results

   }

   next();

}

module.exports=advancedResult;

**router.js**

const express = require('express');

const {getAssignment,

    createAssignment,

    getOneAssignment,

    updateAssignment,

    deleteAssignment,

    photoUpload}=require('../controller/controller');

    const advanceResult= require('../middleware/advancedResult')

    /// include other resource router

const courseRouter= require('./courses');

const assignment = require('../model/assignment');

const router = express.Router();

// Re route inro another

router.use('/:assignmentId/courses',courseRouter)

router.route('/:id/photo').put(photoUpload);

router.route('/')

.get(advanceResult(assignment,'courses'),getAssignment)

.post(createAssignment);

router.route('/:id')

.get(getOneAssignment)

.delete(deleteAssignment)

.put(updateAssignment);

module.exports=router;

**controller.js**

exports.getAssignment=asyncHandler(async(req,res,next)=>{

   res.status(200).json(res.advancedResult)

});

**Router->courses.js**

const express = require('express');

const

{getCourses

}=require('../controller/courses');

const courses= require('../model/course');

const advanceResult= require('../middleware/advancedResult')

const router = express.Router({mergeParams:true});

router.route('/').get(advanceResult(courses,{

    path:'router',

    select:'name description'

}),getCourses);

module.exports=router;

**control courses.js**

const ErrorResponse= require('../util/errorResponse')

const asyncHandler =require('../middleware/async')

const course=require('../model/course');

exports.getCourses=asyncHandler(async(req,res,next)=>{

   if(req.params.assignmentId)

   {

        const courses=course.find({assignment:req.params.assignmentId});

        return res.status(200).json({

            success:true,

            count:course.length,

            data:courses

        });

   }

   else{

       res.status(200).json(res.advancedResult);

   }

})