const mongoose = require('mongoose');

const { doc } = require('prettier');

const slugify = require('slugify');

const validator = require('validator');

const tourSchema = new mongoose.Schema({

name:{

type:String,

required:[true,'A tour must have a name'],

unique:true,

trim:true,

maxLength:[40, 'A tour name must have less or equal than 40 characters'],

minLength:[10, 'A tour name must have more or equal than 10 characters'],

// validate:[validator.isAlpha,'Tour name must only contain characters'],

},

slug:String,

price:{

type:Number,

required:[true, 'A tour must have a price']

},

priceDiscount:{

type:Number,

validate: {

validator:function(val){

// this only points to current doc on NEW document creation

// npm i validator

return val < this.price;

},

message:'Discount price ({VALUE}) should be below regular price'

}

},

difficulty:{

type:String,

required:[true, 'A tour must have a difficulty'],

enum:{ //enumarator

values:['easy','medium','difficult'],

message:'Difficulty is either: easy, medium, difficult'

}

},

duration:{

type:Number,

required:[true, 'A tour must have a duration']

},

maxGroupSize:{

type:Number,

required:[true, 'A tour must have a group size']

},

rating:{

type:Number,

default:4.5

},

ratingsAverage:{

type:Number,

default:4.5,

//min and max validation can also use in dates

min:[1, 'Rating must be above 1.0'],

max:[5, 'Rating must be below 5.0'],

},

ratingsQuantity:{

type:Number,

default:0

},

summary:{

type:String,

trim:true,

required:[true,'A tour must have a summary']

},

description:{

type:String,

trim:true,

required:[true,'A tour must have a description']

},

imageCover:{

type:String,

trim:true,

required:[true,'A tour must have a cover photo']

},

images:[String],

createdAt:{

type:Date,

default:Date.now()

// select:false // if you dont want to select the date on the query

},

startDates:[Date],

secretTour:{

type:Boolean,

default:false

}

}, // end of scheme

{

toJSON:{virtuals:true}, // enable this to make the virtual property active

toJbject:{virtuals:true},

});

// virtual property //virtual property is a virtual field that not in the collection

tourSchema.virtual('durationWeeks').get(function(){

return this.duration / 7;

});

//document middleware: runs Before save() and.create()

tourSchema.pre('save',function(next){

this.slug = slugify(this.name,{lower:true});

next();

});

// tourSchema.pre('save',function(next){

// console.log('Will save Document....');

// next();

// });

// tourSchema.post('save',function(doc, next){

// console.log(doc);

// next();

// });

// QUERY MIDDLEWARE

tourSchema.pre(/^find/, function(next){

// tourSchema.pre('find', function(next){

// let make a clock to measure how long it takes to execute the current query.

// Simply set a property onto this object

// because this query object is really just a regular object.

this.start = Date.now();

this.find({secretTour:{$ne:true}})

next();

});

// this middleware is gonna run after the query is already executed,

// therefore it cannot acccess to the documents that will return,

// because that query has already finished at this point

tourSchema.post(/^find/, function(docs, next){

console.log(`Query took ${Date.now() - this.start} milliseconds`);

next();

});

// AGGREGATION MIDDLEWARE

// Aggregation Middleware allows us to add hooks before or after an aggregation happens

tourSchema.pre('aggregate', function(next){

this.pipeline().unshift({ $match: {secretTour: {$ne:true}} });

console.log(this.pipeline())

next();

});

const TourModel = mongoose.model('Tour', tourSchema);

module.exports = TourModel;

//there are 4 types of middleware in mongoose

//1. document, 2. query, 3. aggregate, 4. model middleware

//use or install slugify package ex: npm i slugify