Mongoose Image

Upload Image

- Now we will learn all about uploading images with the Multer package
- Multer is a very popular middleware to handle multi-part form data, which is a form in coding that's used to upload files from a form.
- Multer is basically a node/express middleware for multi-part form data.
- For more check
- https://github.com/expressjs/multer

Insert photos in our documents…

- In order to be able to store images in our landmarks, we need to alter our landmark model in the previous example

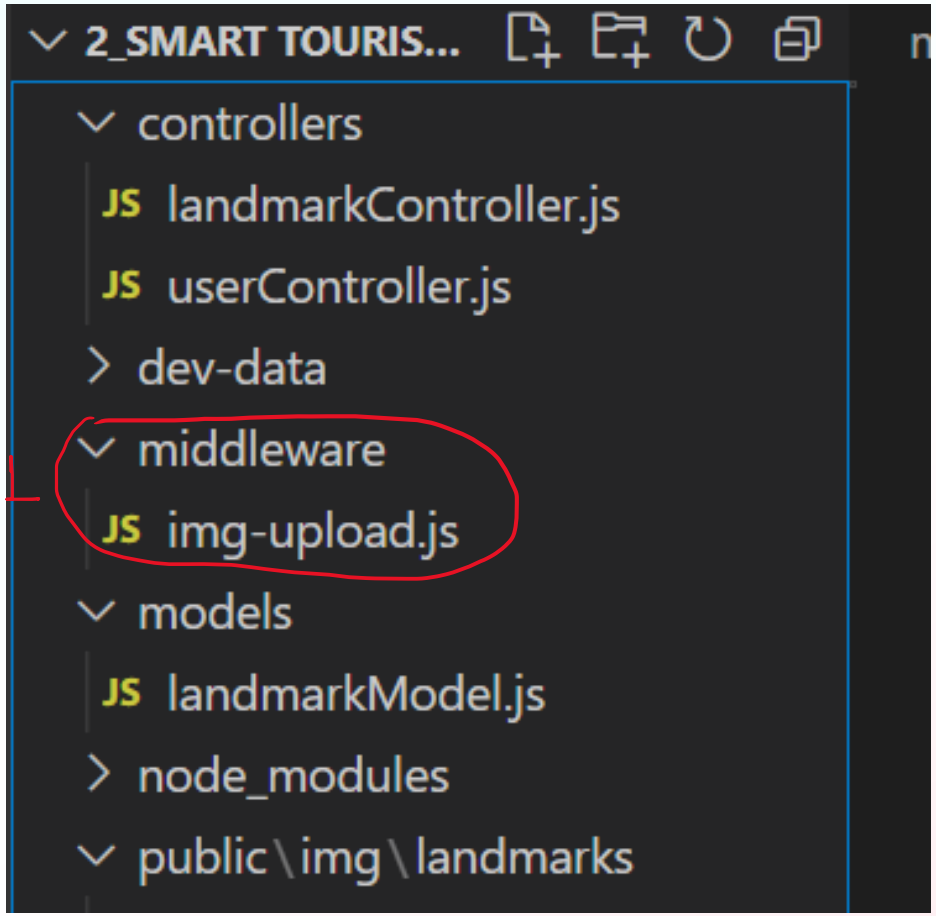
```
const mongoose = require('mongoose');

//schema
const landmarkSchema = new mongoose.Schema({
    type:{
      type: String,
      required:[true, "A landmark must be landmark or musuem"]// s
    },
    name: String,
    description: String,
    ratingsAverage: Number,
    ratingsQuantity: Number,
    skataya:String,
    photo: {
      type: String,
      default: 'default.jpg'
    }
    });
  //model out of  schema
  var Landmark = mongoose.model("Landmark",landmarkSchema);

  module.exports = Landmark;
```

- Install multer

  npm i multer

- We may add a new folder -> middleware folder : there I want to store my own custom middleware

- Multer already is a middleware
  - ➢ but we will configure it to our requirements -> wrap it into our own middleware -> export it -> use it



Object used for file format validation

```
landmarkRoutes.js | landmarkController.js | img-upload.js
middleware > img-upload.js > ...
1  //import multer
2  const multer = require('multer');
3
4  const MIME_TYPE_MAP = {
5    'image/png': 'png',
6    'image/jpeg': 'jpeg',
7    'image/jpg': 'jpg'
8  };
```

```
//Multer accepts an options object
const fileUpload = multer({
  //Limits of the uploaded data
  limits: 500000,

  //most basic is dest or storage property,tells Multer where to upload files
  // if we omit this files will be kept in memory (not stored in disk)
  storage: multer.diskStorage({
    destination: (req, file, cb) => {
      cb(null, `${__dirname}/../public/img/landmarks`);
    },
```

disk storage engine -> gives us full control on storing files to disk

Two options available

- destination
- Filename

They are both **functions** -> they determine where file should be stored.

- destination -> determine within which folder the uploaded files should be stored
- filename -> determine what the file should be named inside the folder.(If no filename is given-> file will be given a random name without file extension)

```
//most basic is dest or storage property,tells Multer where to upload files
// if we omit this files will be kept in memory (not stored in disk)
storage: multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, `${__dirname}/../public/img/landmarks`);
  },

  filename: (req, file, cb) => {
    //originalname  Name of the file on the user's computer
    cb(null,file.originalname);
  }
}),
```

Destination can be considered a **callback function** has access to the current request, currently uploaded file, to a callback function.

To define the destination -> call that callback function (cb)

- first argument is an error if there is one (if not just null)
- second argument is the actual destination.

```
//most basic is dest or storage property,tells Multer where to upload files
// if we omit this files will be kept in memory (not stored in disk)
storage: multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, `${__dirname}/../public/img/landmarks`);
  },

  filename: (req, file, cb) => {
    //originalname  Name of the file on the user's computer
    cb(null,file.originalname);
  }
}),
```

fileFilter: filter in Multer is a callback function

- It has access to request, file, and a callback function.

- in this function the goal is to test if the uploaded file is an image. (in this example)

If it is so->we pass true into the callback function,

Else false along with an error.

```
//Multer accepts an options object
const fileUpload = multer({
  //Limits of the uploaded data in bytes
  limits: 500000,

  //most basic is dest or storage property,tells Multer where to upload files
  // if we omit this files will be kept in memory (not stored in disk)
  storage: multer.diskStorage({
    destination: (req, file, cb) => {
      cb(null, `${__dirname}/../public/img/landmarks`);
    },

    filename: (req, file, cb) => {
      //originalname  Name of the file on the user's computer
      cb(null,file.originalname);
    }
  }),
  fileFilter: (req, file, cb) => {
    //file.mimetype :Mime type of the file
    //The in operator returns true if a property exists in an object else  false.
    const isValid = file.mimetype in MIME_TYPE_MAP;
  //conditional operator that assigns a value to a variable
  //based on some condition. variablename = (condition) ? value1:value2
    let error = isValid ? null : new Error('Invalid mime type!');
    cb(error, isValid);
  }
});
```

landmarkRoutes.js | landmarkController.js | img-upload.js | userController.js

middleware > img-upload.js > ...

```
1  //import multer
2  const multer = require('multer');
3
4  const MIME_TYPE_MAP = {
5    'image/png': 'png',
6    'image/jpeg': 'jpeg',
7    'image/jpg': 'jpg'
8  };
9
10
11 //Multer accepts an options object
12 const fileUpload = multer({
13   //Limits of the uploaded data in bytes
14   limits: 500000,
15
16   //most basic is dest or storage property,tells Multer where to upload files
17   // if we omit this files will be kept in memory (not stored in disk)
18   storage: multer.diskStorage({
19     destination: (req, file, cb) => {
20       cb(null, `${__dirname}/../public/img/landmarks`);
21     },
22
23     filename: (req, file, cb) => {
24       //originalname  Name of the file on the user's computer
25       cb(null,file.originalname);
26     }
27   }),
28   fileFilter: (req, file, cb) => {
29     //file.mimetype :Mime type of the file
30     //The in operator returns true if a property exists in an object else  false.
31     const isValid = file.mimetype in MIME_TYPE_MAP;
32   //conditional operator that assigns a value to a variable
33   //based on some condition. variablename = (condition) ? value1:value2
34     let error = isValid ? null : new Error('Invalid mime type!');
35     cb(error, isValid);
36   }
37 });
38 //export middleware to use it wherever we wish
39 module.exports = fileUpload.single('photo');
40
```

routes > JS landmarkRoutes.js > ...

```
1  //import express
2  const express= require('express');
3  //import multer configured middleware
4  const fileUpload = require('../middleware/img-upload');
5  //import landmarkController module
6  const landmarkController=require(`${__dirname}/../controllers/landmarkController`);
7  //lets create a new router-> it is a middleware
8  const routes = express.Router();
9
10 //landmark Routes
11 routes.route('/')
12 .get(landmarkController.getAllLandmarks)
13 .post(landmarkController.createLandmark)
14
15 routes.route('/:id')
16 .get(landmarkController.getLandmarkById)
17 .patch( fileUpload,landmarkController.updateLandmarkById)
18 .delete(landmarkController.deleteLandmarkById)
19
20 routes.route('/getOne/:type')
21 .get(landmarkController.getLandmarkOneById)
22
23 //routes.route('/:name')
24 //.get(landmarkController.getLandmarkByName)
25
26
27 //routes.route('/:key')
28 //.get(landmarkController.getLandmarkBykey)
29
30 //lets export our module!
31 module.exports= routes;
32
```

① ②

ndmarkRoutes.js ● | JS landmarkController.js ● | JS img-upload.js | JS userController.js

trollers > JS landmarkController.js > ...

```
exports.getLandmarkById = async (req, res) => {···
};
exports.getLandmarkByName = async (req, res) => {···
};
exports.getLandmarkBykey = async (req, res) => {···
};

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

    var mybody=req.body;
    const updates = {mybody};

    if (req.file) {
      const photo = req.file.filename;
      updates.photo = photo;
    }
    // req.body contains data we want to change
    const landmarks = await Landmark.findByIdAndUpdate(req.params.id, updates, {
      new: true, // we want this method to  return new updated document to the client
    });

    res.status(200).json({
      status: 'success',
      data: {
        landmarks
      }
    });
  } catch (err) {
    res.status(404).json({
      status: 'fail',
      message: err
    });
  }
};
```

create object
Check file existence
store in object
use object to update doc


That's all Folks!