		MVC	125	
		router.route	126	
		Image Upload	127	

Date	Title	Page No.
	Map (Mapbox)	135

MVC

→ Implement Design Pattern for Listings

→ Model, View, Controller

controller = callback functions of routes:
e.g. async (req, res) => {...}

In controllers/listings.js

```
- const Listing = require(".. / models / listing");

module.exports.index = async (req, res) =>{

    const allListings = await Listing.find({});
    res.render(" listings/index.ejs", {allListings});
};
```

In routes/listing.js

```
- const listingController = require("../controllers/listings.js");

router.get("/", wrapAsync(listingController.index));
```

router.route

→ Returns an instance of a single route which you can then use to handle HTTP verbs with optional middleware. Use router.route(path) to avoid duplicate route naming and thus typing errors.

```
- router.route("/users/:user_id")
    .get( function (req, res, next) {...} )
    .put( function (req, res, next) {...} )
    .post( function (req, res, next) {...} )
    .delete( function (req, res, next) {...} )
```

Image Upload

→ <form> tag sends url-encoded data by default to the backend.

→ To send file type data to the backend, we will add "enctype" attribute to the <form> tag & in "type" attribute, value is "file" in <input> tag.

```
- <form enctype = "multipart / form-data" >
      <input name = "listing[image]" type = "file" >
  </ form >
```

→ But still in the output of "req.body", we will receive empty object.

→ To parse "multipart" form data, we will use "multer" npm package.

* multer npm package

→ multer is a node.js middleware for handling "multipart/form-data", which is primarily used for uploading files.

```
- const multer = require("multer");
  const upload = multer( { dest: "uploads/" } );
        ↑initialize multer      ↑destination folder to store files
                                 we will change it later when we will
                                 use cloud service

  router.post("/", upload.single("listing[image]"), (req, res)=>{
      res.send(req.file);
  });
```

output: (JSON format)

```
fieldname: "listing [ image]"
originalname: "image_1.jpeg"
encoding: "7bit"
mimetype: "image/jpeg"
destination: "uploads/"
filename: "d92f..."
path: "uploads/d92f..."
size: 9865
```

* Cloud Setup & .env file

→ To store image, we will use free cloud services of "cloudinary". Create an account for developers.

→ To access our account, we will copy cloud name, API key, API secret and paste in .env file.

→ .env : environment variables/credentials
→ In production level application, we do not store any secrets in code but we will store in .env file.

* ~~don~~ dotenv npm package

→ To access variables of .env file

→ We will write following code at the very top of the app.js file. Then, we can access "process.env" object at any of the js files (e.g. in routes/listing.js, ...)

In .env

```
- SECRET="helloworld"
  NODE-ENV="development"
  CLOUD_NAME=dxcoulov
  CLOUD_API-KEY=56813295620
  CLOUD_API_SECRET=PjoWMyproF6
```

In app.js

```
- if (process.env.NODE-ENV != "production") {
      require("dotenv").config();
      console.log(process.env.SECRET); ← helloworld
  }
```

* cloudinary & multer-storage-cloudinary npm package

→ To store files on cloudinary, we will use "cloudinary" & "multer-storage-cloudinary" npm package to access our cloudinary account.

In cloudConfig.js

```
- const cloudinary = require("cloudinary").v2;
  const { CloudinaryStorage }= require("multer-storage-cloudinary");

  cloudinary.config ( {
      cloud_name: process.env.CLOUD-NAME,
      api_key: process.env.CLOUD_API-KEY,
      api_secret: process.env.CLOUD_API-SECRET
  });

  const storage = new CloudinaryStorage ({
      cloudinary: cloudinary,
      params: {
          folder: "wanderlust_DEV",
          allowedformats: [ "png", "jpg", "jpeg"]
      }
  });

  module.exports = { cloudinary , storage };
```

In routes/listing.js

```
- const multer = require("multer");
  const { storage }= require("../cloudConfig.js");
  const upload = multer ({ storage });
```

middleware as per page 128

* Save link in MongoDB

→ Modify image property in listingSchema

```
- image: {
      url: String,
      filename: String
  }
```

In controllers/listings.js

```
- module.exports.createListing = async (req, res, next)=>{
      let url = req.file.path;
      let filename = req.file.filename;

      const newListing = new Listing (req.body.listing);
      newListing.owner = req.user._id;
      newListing.image = {url, filename};

      await newListing.save();
      req.flash ("success", "New Listing Created");
      res.redirect ("/listings");
  };
```

* Update & Preview Image

→ Same as before, we write code for cloudinary, <form> tag in edit form, middleware in put route, etc.

In controllers/listings.js > updateListing

```
- let listing = await Listing.findByIdAndUpdate(id, {...req.body.listing});

if ( typeof req.file !== "undefined") {

    let url = req.file.path;
    let filename = req.file.filename;
    listing.image = { url, filename };
    await listing.save();
}
```

→ To preview image, add <div> to edit form and lower the quality of the image in url itself (image transformation is builtin function of cloudinary)

```
- <div> <img src=" <%= newUrl %> " > </div>
```

In controllers/listings.js > ~~edit~~ renderEditform

```
- ~~let listing image Url~~
- let newUrl = listing. image.url;
newUrl = newUrl. replace("/upload", "/upload/h_300,w_300/");
res. render (" listings/edit.ejs", { listing, newUrl });
```

Map

* Mapbox ~~npm~~ ~~package~~

→ To show map, we will use mapbox library. visit website & create account & copy MAP_TOKEN

docs.mapbox.com/mapbox-gl-js/example/simple-map/

→ There is a code sample written which we will copy-paste in our code at appropriate locations.

→ link & script tag for api of mapbox copy-paste in boilerplate.ejs just above </head>

```
- <link href="https://api.mapbox.com/mapbox-gl-js/v3.1.0/mapbox-gl.css"
        rel = "stylesheet">
  <script src = "https://api.mapbox.com/mapbox-gl-js/v3.1.0/mapbox-gl.js">
  </script>
```

→ create div to show map. In show.ejs

```
- <div class = "col-8 offset-3 mb-3">
     <h3> Where You'll be </h3>
     <div id = "map"> </div>
  </div>
```

→ Give some style to the div for map. In public/css/style.css

```
- #map {
      height: 400px;
      width: 80vh;
  }
```

→ We will copy code of <script> tag from website to a static js file. In public/js/map.js

```
- mapboxgl.accessToken = mapToken;

  const map = new mapboxgl.Map({

        container: "map",  //container ID (from div of map)
        style: "mapbox://styles/mapbox/streets-v12",   ← theme can change from website
        center: [77.209 , 28.613],   ← starting position [lng, lat]
                                                          °E    °N
        zoom: 9  //starting zoom
  });
```

→ In above code, we need MAP-TOKEN, but public static js file cannot access .env variables, so we first make new variable "mapToken" in ejs file & then access it it "map.js" In show.ejs

- At the Top:

```
<script>
    const mapToken = "<%= process.env.MAP_TOKEN %>";
</script>
```

- At the BOTTOM:

```
<script   src = " /js / map.js" > </script>
```

* Geocoding & @mapbox/mapbox-sdk npm package

→ Geocoding is the process of converting addresses into geographic coordinates (like latitude and longitude), which you can use to place markers on a map or position on map.

→ Forward geocoding query type allows you to look up a single location by name and returns its geographic coordinates

→ We will use a JS SDK for working with Mapbox APIs. visit "github.com/mapbox/mapbox-sdk-js"

→ npm install @mapbox/mapbox-sdk

In controllers/listings.js

```
- const mbxGeocoding = require("@mapbox/mapbox-sdk/services/geocoding");
  const mapToken = process.env. MAP_TOKEN;
  const geocodingClient = mbxGeocoding ( {accessToken: mapToken} );

  ..... createListing .....
    let response = await geocodingClient. forwardGeocode ({
            query: "Paris, France",
            limit: 1    ← 5 if not written (default)
    }).send ();
    console.log (response);
```

console output

```
:
body: {
        type: "FeatureCollection",
        query: [ "new", "delhi", "india" ],
        features: [ [Object] ],
        attribution: "NOTICE: © 2023..."
},
:
```

— console.log (response.body.features);

console output:

```
[
  {
     id: "place.31664235",
     type: "Feature",
     place_type: [ "place" ],
     relevance: 1,
     properties: { mapbox-id: "dxJu...", wikidata: "Q987" },
     text: "New Delhi",
     place_name: "New Delhi", "Delhi", "India",
     bbox: [ 76.942051, 28.404263, 77.347105, 28.883588 ],
     center: [ 77.209006, 28.613895 ],
     geometry: { type: "Point", coordinates: [Array] },
     context: [ [Object], [Object], [Object] ]
  }
]
```

- console.log(response.body.features[0].geometry);

console output:

```
{ type: "Point", coordinates: [77.209006, 28.613895] }
```

⇒ now, update query value (page no. 137)

```
-     query: req.body.listing.location,
      limit: 1
```

* Storing Coordinates

→ GeoJSON format =
{ type: "Point", coordinates: [77.209006, 28.613895] }

→ Mapbox gives us GeoJSON format & we can store the same format in MongoDB

→ add geometry property in listingSchema

```
- geometry: {
      type: {
            type: String,  // Don't do { location: { type: String } }
            enum: ["Point"],
            required: true     },
   coordinates: { type: [Number], required: true }
}
```

In controllers/listings.js > createListing

— newListing.geometry = response.body.feature[0].geometry;

* Map Marker

→ To show location on Map from coordinates

In public/js/map.js

```
— const marker = new mapboxgl.Marker()
                 .setLngLat([12.5544, 55.7065])
                 .addTo(map);
```

→ In above, setLngLat method coordinates, we want to access location of listing. But we cannot access in public. so, we will variable from ejs file.

In views/listings/show.ejs

— At the Top
```
<script>
    const mapToken = "<%= process.env.MAP_TOKEN %>";
    const coordinates = <%- JSON.stringify(listing.geometry.coordinates) %>;
</script>
```

→ Now, you can change in map.js
- center: coordinates,

- ..... setLnglat (coordinates) .....

* Marker Popup

→ When we click on marker, it will show popup.
→ In this popup, you can write html code

In show.ejs :
- const listing =<%-JSON.stringify(listing) %>;

In map.js :
- center: listing.geometry.coordinates,

```
- const marker = new mapboxgl.Marker( { color: "red" })
      .setLnglat(listing.geometry.coordinates)
      .setPopup( new mapboxgl.Popup({offset: 25}). setHTML(
       `<h4> ${ listing.location } </h4>
        <p> Exact Location provided after booking </p>`
      .addTo (map);
```