### **Title: Uploading of images when adding a NEW Place to MongoDB Atlas collection(Assignment-10)**

**Part One:**

Introduction:

In this project, we have integrated and created all placed API and fetched data form MongoDB Atlas.

Step: 1

The createPlace function is an asynchronous handler for creating a new place entry in the database. Here's a concise summary:

Uses validationResult to check for errors in the request.

Responds with status 422 and an error message if validation fails.

Extract Data:

Extracts title, description, coordinates, and creator from the request body.

Logs the request body for debugging.

Create Place Object:

Creates a new Place object with the extracted data.

Parses coordinates and sets the image path from req.file.path.

Save Place:

Attempts to save the Place object to the database.

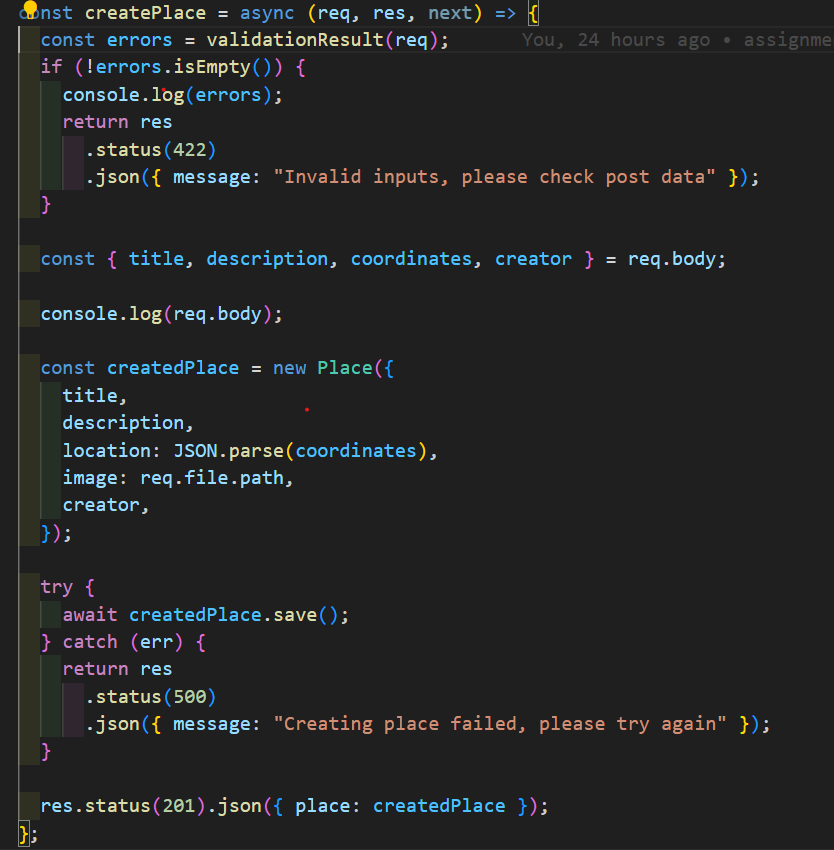
If saving fails, responds with status 500 and an error message.

Response:

If successful, responds with status 201 and the created place object.

This function ensures valid data handling and appropriate client feedback based on the operation's success or failure.

**Screenshot of createPlace Function:**



**Postman Screeenshot:**

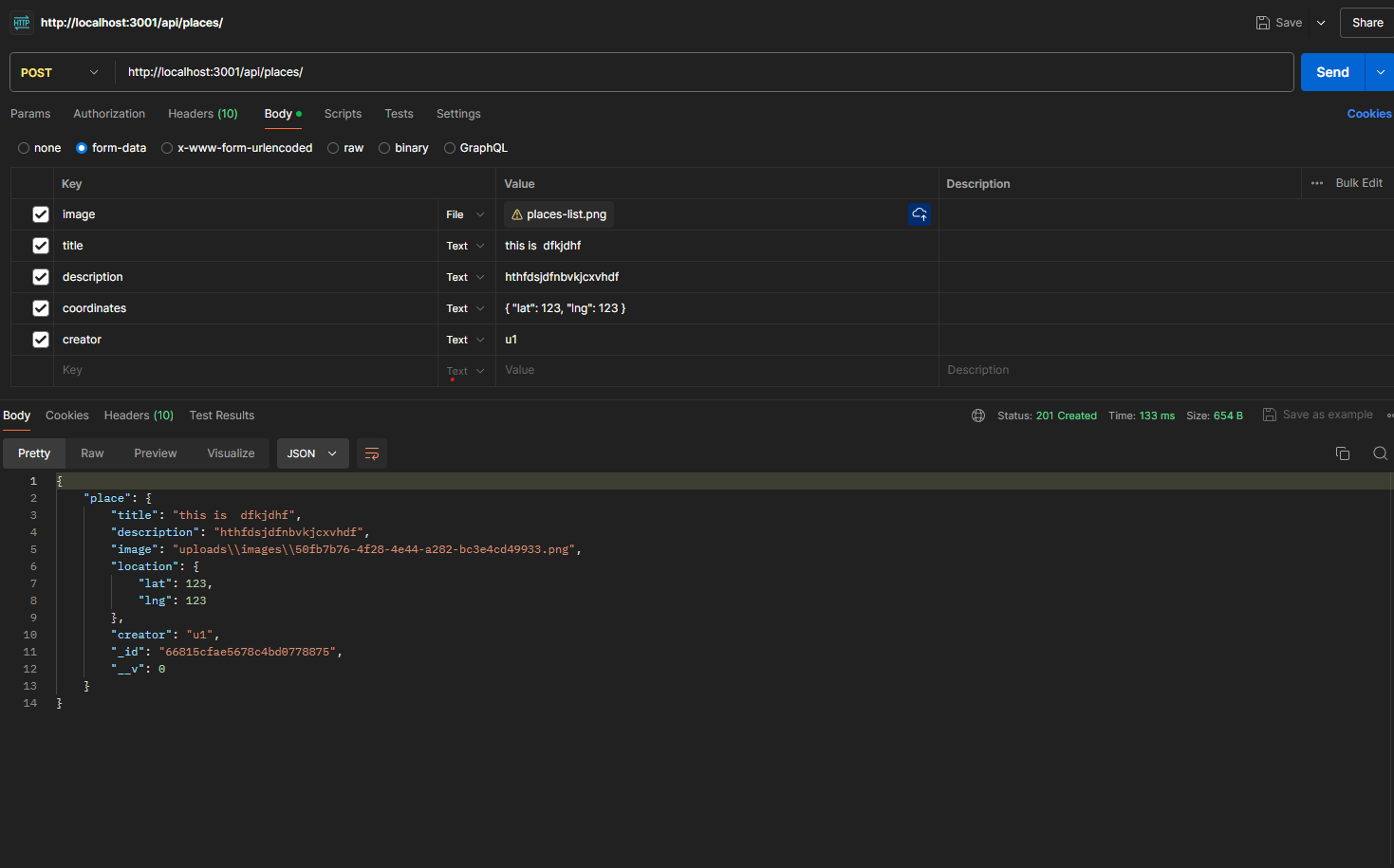
This screenshot displays a typical Postman interface used for testing API endpoints. Key elements include:

Request Headers:

Headers section where key-value pairs are added, such as Content-Type to specify the media type (e.g., application/json).

Request Body:

The body of the request containing the data to be sent to the server, formatted in JSON. Fields might include title, description, coordinates, creator, etc.



B: Place Modal:

Schema Definition:

placeSchema defines the structure of a MongoDB document for places.

It includes fields like title, description, image, location (with subfields lat and lng), and creator.

Each field specifies its data type (String for title, description, image, and creator; Number for lat and lng) and whether it's required (required: true).

Model Creation:

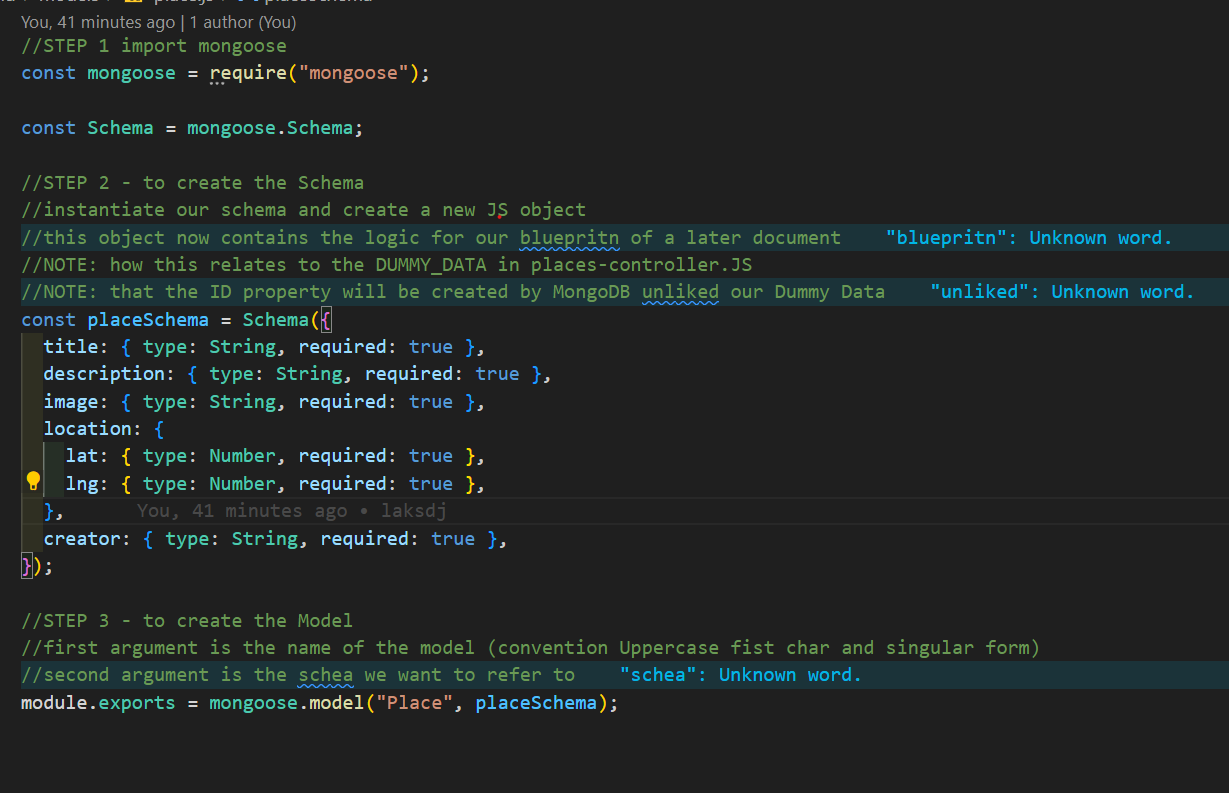
mongoose.model("Place", placeSchema) creates a Mongoose model named "Place" based on placeSchema.

This model allows interaction with MongoDB collection named "places" (plural of "Place"), providing methods like save, find, findOne, update, and delete for CRUD operations.

Usage:

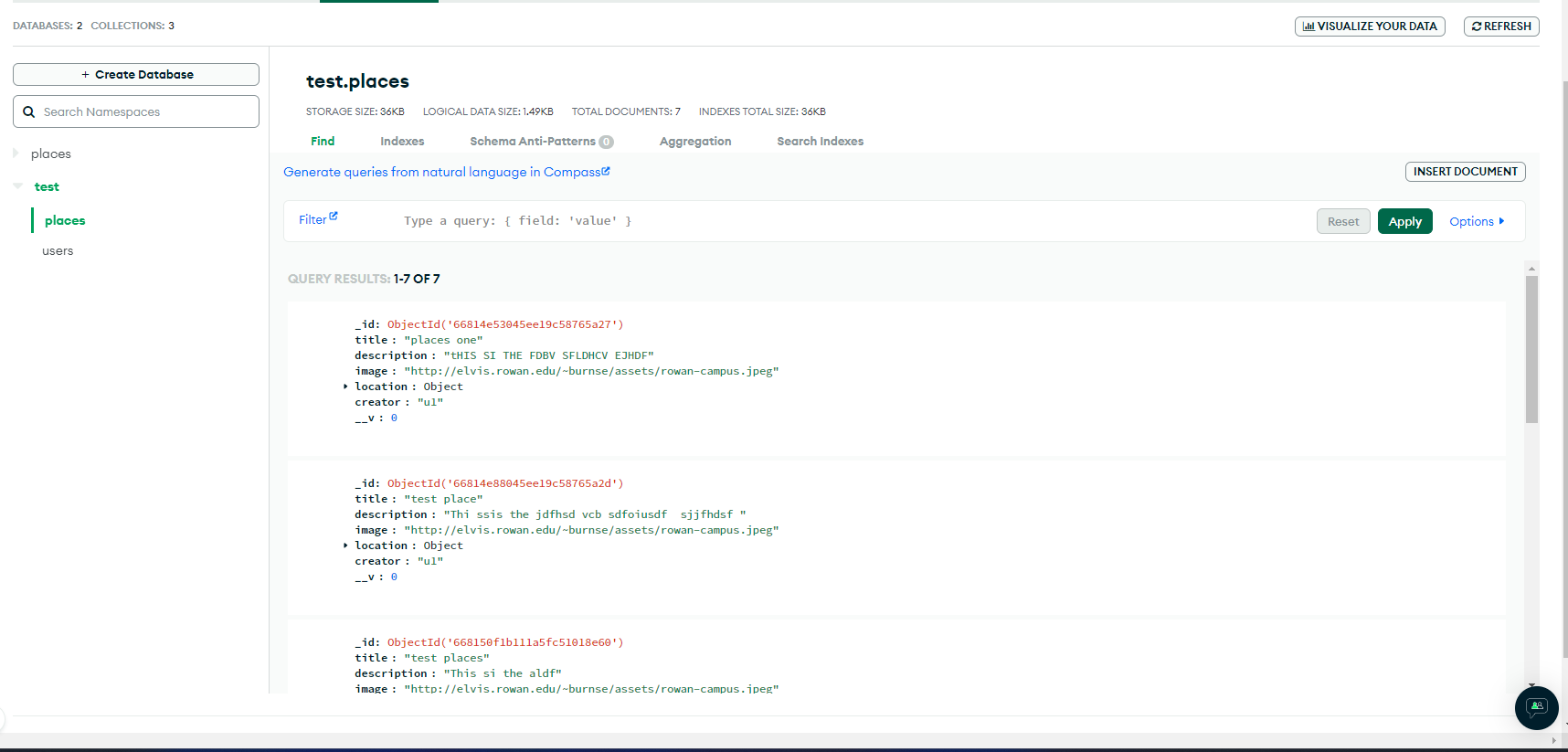
In your application, you can import this model (const Place = require("./models/place")) to interact with MongoDB. For example, to create a new place

**Place Modal Screenshot:**



**MongoDB Atlas Screenshot:**

This document appears to represent a place or location entry in a MongoDB collection, possibly related to a specific user ("u1"). The \_id field uniquely identifies this document within its collection, and the other fields contain descriptive information about the place, including its title, description, image URL, and possibly geographic location details stored as an object.



**Part Two: Exploring Multer for File Uploads**

Introduction:

Multer is a popular middleware for handling file uploads in Node.js. It simplifies the process of receiving uploaded files from clients and storing them on the server. This document explores how to integrate and utilize Multer for various file upload scenarios in a Node.js application.

Multer Overview:

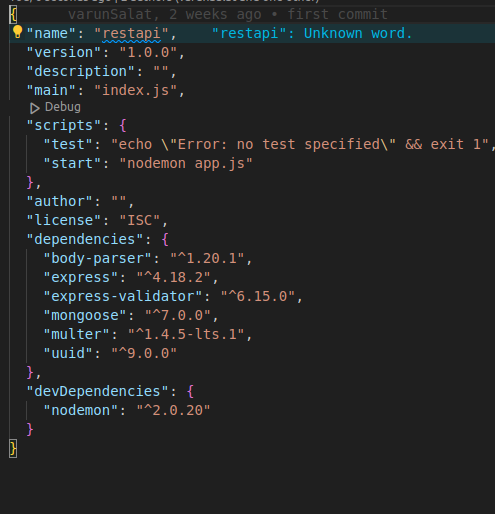
Multer is a middleware for Express and Node.js that facilitates handling multipart/form-data, which is primarily used for uploading files. It provides easy integration with Express and supports various configurations for file handling, such as file size limits, file renaming, and storage destination.

**A: Setting Up Multer in Node.js/Express**

Installation:

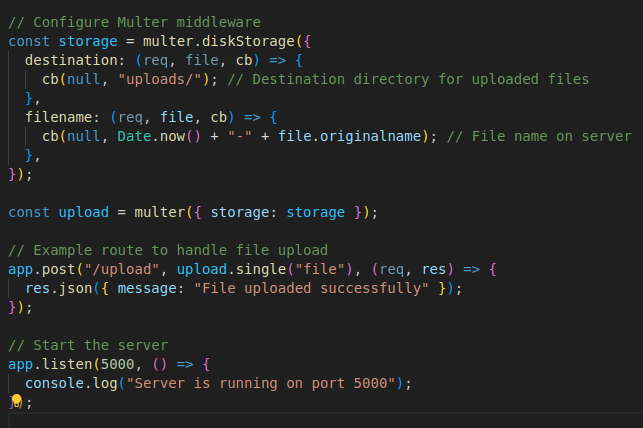
To start using Multer, you need to install it via npm: npm install multer

Screenshot of package.json where multer is installed and can be seen in dependencies:



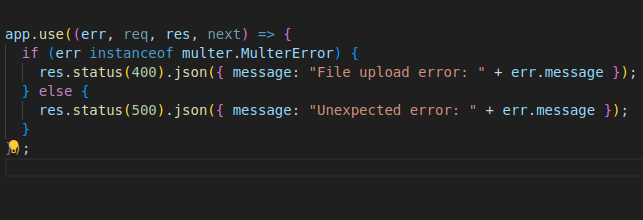
**B: Integration with Express**

Integrate Multer into Express application by requiring it and configuring it to handle file uploads:



**C: Handling File Upload Errors**

Multer provides error handling mechanisms to manage file upload failures, such as exceeding file size limits or unexpected file types. You can define error-handling middleware to manage these scenarios:



**Conclusion:**

Multer simplifies the process of handling file uploads in Node.js applications, offering flexibility and robust features for managing multipart/form-data. By integrating Multer into your Node.js/Express project, you can efficiently handle file uploads, set file upload configurations, and manage file upload errors.