Architecture Design and Software Structure Report

REST API with Express, Mongodb and Mongoose

1. Introduction

In this project I will integrate the REST API that I developed with the Express framework earlier together with the Mongoose schemas and models and the MongoDB database to provide the complete end-to-end implementation of the server to support REST CRUD API.

2. Design and Implementation

2.1 The REST API Specification

We'll be building a RESTful CRUD (Create, Retrieve, Update, Delete) API with Node.js, Express and MongoDB. We'll use Mongoose for interacting with the MongoDB instance.

Implementing the get/promotions API:

```
const express = require('express');
const bodyParser = require('body-parser');
const promotionsRouter = express.Router();
const mongoose = require('mongoose');

const Promotions = require('../models/promotions');


promotionsRouter.use(bodyParser.json());
promotionsRouter.route('/')
.get((req,res,next) => {
    Promotions.find({})
    .then((promotion) => {
        res.statusCode = 200 ;
        res.setHeader('Content-Type', 'application/json');
        res.json(promotion);
    }, (err) => next(err))
    .catch((err) => next(err));
})
```

Implementing the post and put/promotions APIs:

```
.post((req,res,next) => {
    Promotions.create(req.body)
    .then((promotion) => {
        console.log('Promotion created', promotion);
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(promotion);
    }, (err) => next(err))
    .catch((err) => next(err));

})
.put((req,res,next) => {
    res.statusCode = 403 ;
    res.end('PUT operation not supported on /promotions');
})
```

Implementing delete/promotions API:

```
.delete((req,res,next) => {
    Promotions.remove({})
    .then((resp) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'application/json');
        res.json(resp);
    }, (err) => next(err))
    .catch((err) => next(err));
    });
```

Implementing get/promotions/:promotionId API:

```
  promotionsRouter.route('/:promotionId')
.get((req,res,next) => {
 Promotions.findById(req.params.promotionId)
 .then((promotion) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'application/json');
  res.json(promotion);
 },(err) => next(err))
 .catch((err) => next(err));
})
```

Implementing post,put/promotions/:promotionId API:

```
.post((req,res,next) => {
  res.statusCode = 403;
  res.end('POST operation not supported on the /promotions' + req.params.promotionId);
})
.put((req,res,next) => {
Promotions.findByIdAndUpdate(req.params.promotionId, {
  $set: req.body
}, {new: true })
.then((promotion) => {
  res.statusCode = 200;
  res.setHeader('Content-Type','application/json');
  res.json(promotion);
},(err) => next(err))
.catch((err) => next(err));
})
```

Implementing delete/promotions/:promotionId API:

```
.delete((req,res,next) => {
   Promotions.findByIdAndRemove(req.params.promotionId)
   .then((resp)=> {
     res.statusCode = 200;
     res.setHeader('Content-Type', 'application/json');
     res.json(resp);
   }, (err) => next(err))
   .catch((err) => next(err));
 });
```

2.2 Database Schemas, Design and Structure

Here we will continue with the model of promotions and we will see how it does look like the Database Schema, Design and the Structure of this Schema.

we will access to the file promotions.js where we implemented our Database Schema there:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
require('mongoose-currency').loadType(mongoose);
const Currency = mongoose.Types.Currency;

const commentSchema = new Schema({
    rating: {
        type: Number,
        min: 1,
        max: 5,
        required: true
    },
    comment: {
        type: String,
        required: true
    },
    author: {
        type: String,
        required: true
    }
},{ timestamps: true
});
```

as we see here we implemented the commentSchema which we will import it inside the promotionSchema as a field there. Now here we will see how we implemented the promotionSchema:

```
const promotionSchema = new Schema({
    name: {
        type: String,
        required: true,
        unique: true
    },
    description: {
        type: String,
        required: true
    },
    image: {
        type: String,
        required: true
    },
    label: {
        type: String,
        default: ''
    },
    price: {
        type: Currency,
        required: true,
        min: 0
    },
    featured: {
        type: Boolean,
        default: false
    },
    comments:[ commentSchema ]
},

{
        timestamps: true

});

var Promotions = mongoose.model('Promotion', promotionSchema);

module.exports = Promotions;
```

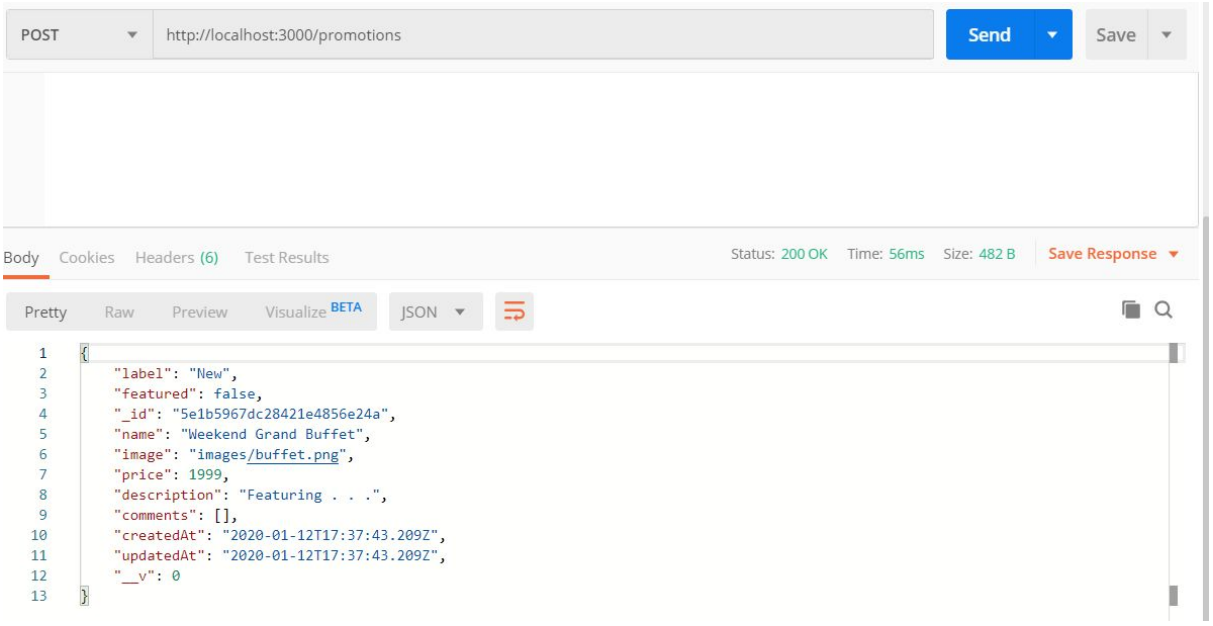
as we can see here, It's clear that the promotionSchema has the JSON format and there is a specific field for the commentSchema which is under the field name of comments.

2.3 Communication

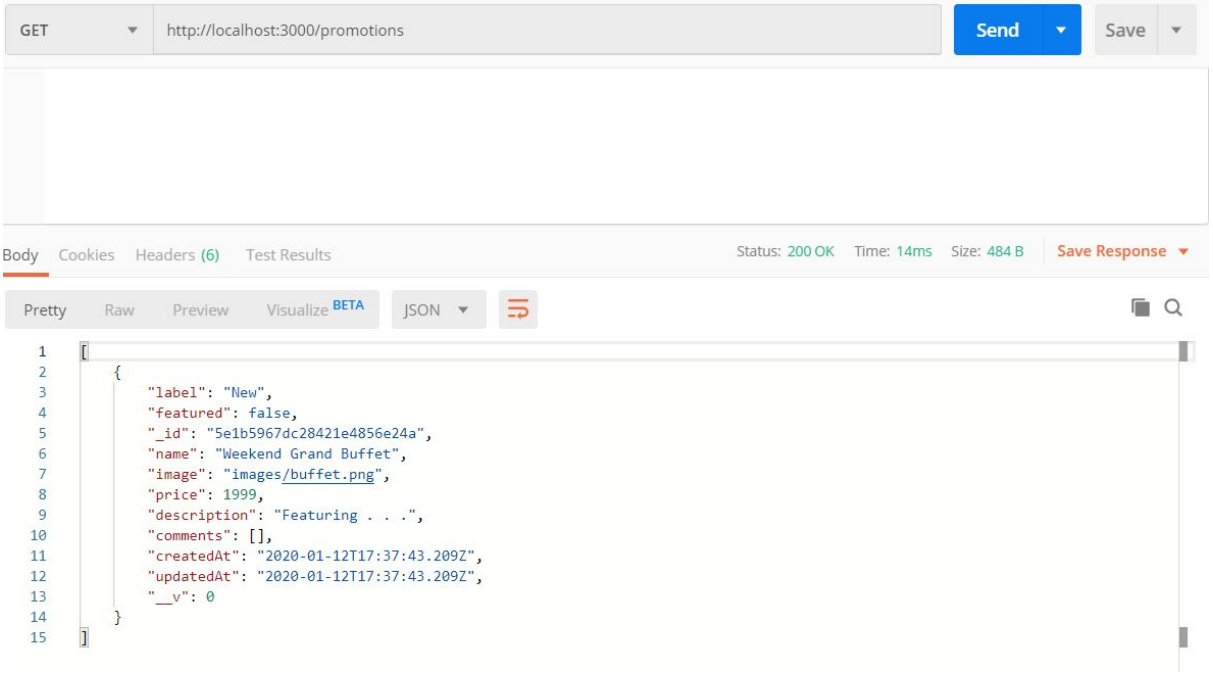
Here we will give the structure of the messages to be communicated between the front-end and the back-end using Postman:Testing our APIs using Postman:

we will take the Promotion model as an example.

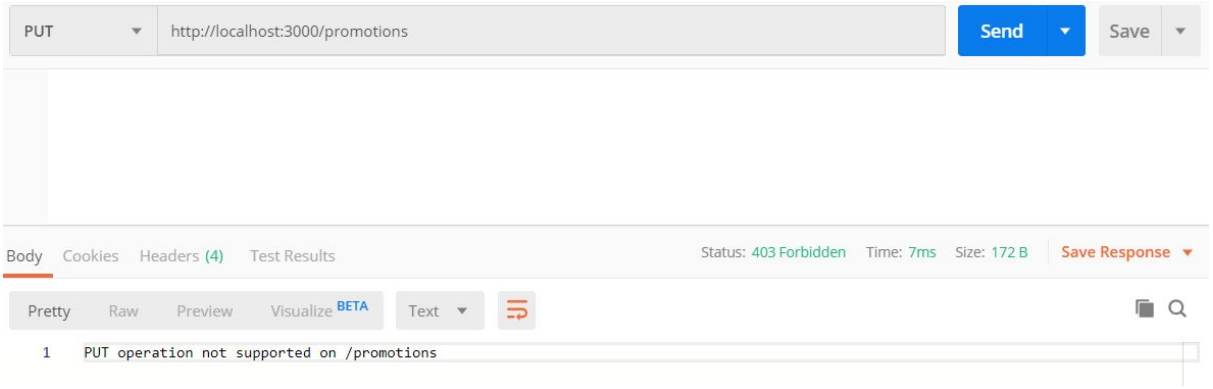
creating a new Promotion using post/promotions API:



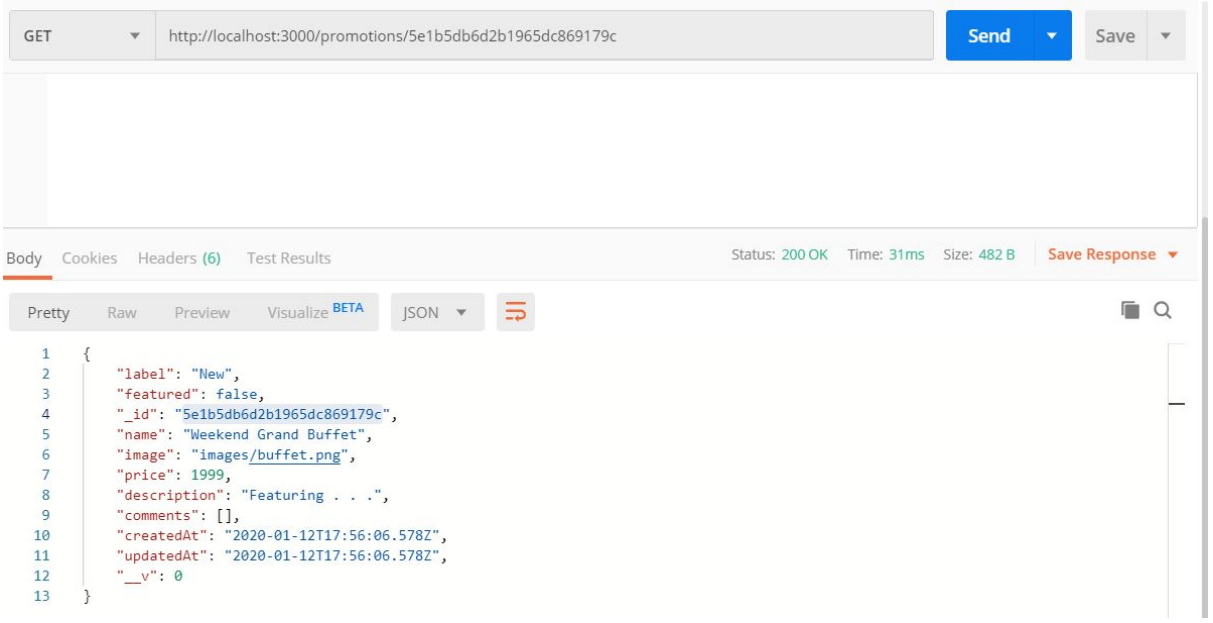
Retrieving all promotions using Get/promotions API:



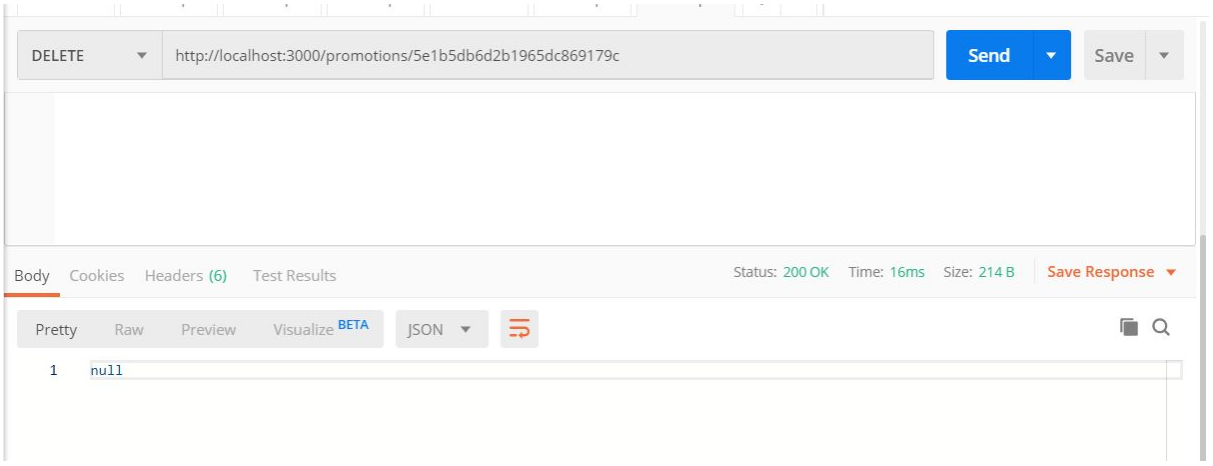
Update the promotion using Put/promotions API :



Retrieving a single promotion using get/promotion/:promotionId:



deleting a promotion using delete/promotions/:promotionId Api:



3. Conclusions

In this assignment, We learned how to build REST APIs in node.js using Express framework and MongoDB.

4. References

https://www.callicoder.com/node-js-express-mongodb-restful-crud-api-tutorial/

https://fr.wikipedia.org/wiki/Representational_state_transfer