REST API

API:

It stands for application programming interface

Application: for example youtube

Interface: like endpoint, if you want to search something, you search via some endpoint or interface

Programming: In code, used programming to extract data of application (youtube) via interface(endpoint)

**Thus, programming (P) is used to dealt with application (A) via interface (endpoint) (I), collectively becomes API**

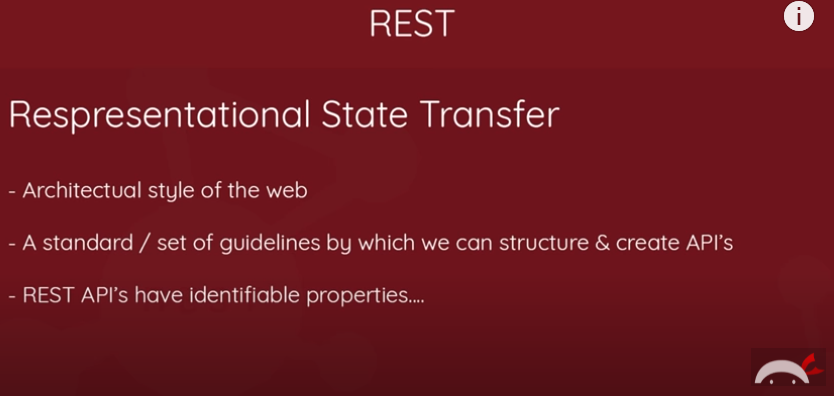


REST:

Respresentational state transfer

Architectual styple of the web

A standard/ set of guidelines by which we can structure and create api

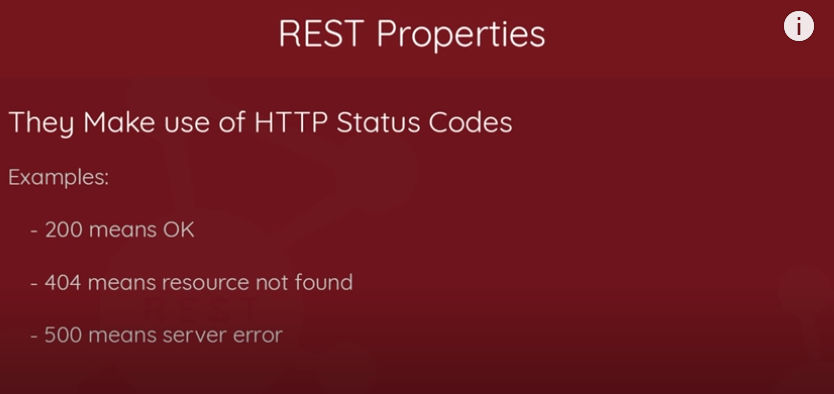


PROPERTIES:

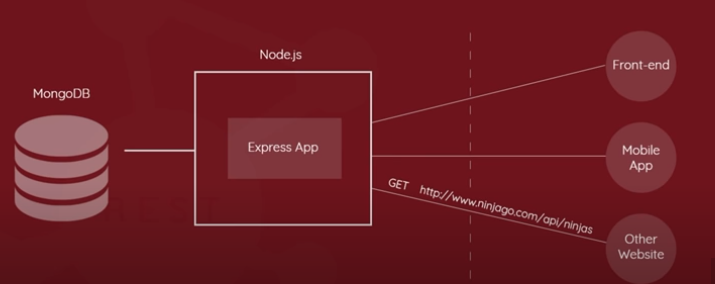
1. Use of Resource Based Url



1. Use of HTTP methods

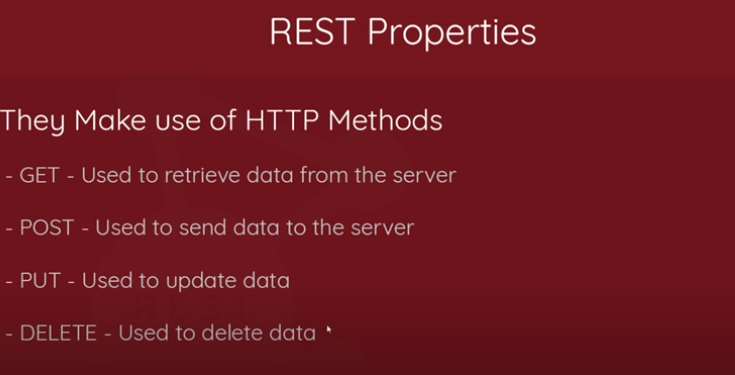


ARCHITECTURE:



HTTP Method:

What request we or client gonna make it

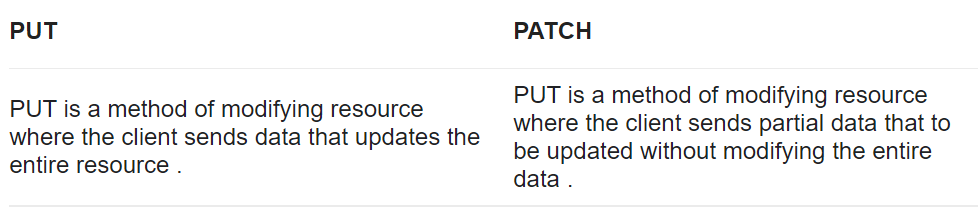


ROUTES:



Update and delete requires id of particular data

PUT and PATCH both used to update the data, then what’s the difference?



START:

Create server.js file

Npm init (to get package.json)

DEPEDENCIES TO INSTALL

Npm I express

Npm I mongoose

Npm I morgan

Npm I nodemon (update package.json)

UPDATE SERVER.JS

const express = require('express')

const morgan = require('morgan')

const mongoose = require('mongoose')

const app = express();

// const productRoutes = require('./api/routes/products')

// const orderRoutes = require('./api/routes/orders')

//mongodb connection

// mongoose.connect('mongodb+srv://admin:Dj21mrMwyo89gAHr@cluster0.dmxhm.mongodb.net/myFirstDatabase?retryWrites=true&w=majority',

// {

//     useNewUrlParser:true,

//     useUnifiedTopology:true

// });

// mongoose.Promise = global.Promise

//middleware

//morgan shows log in terminal or console

app.use(morgan('dev'))

//body parser which is now replaced by express is used to parse body of your choice

app.use(express.urlencoded({extended: false}))//parse simple bodies of url encoded data

app.use(express.json());//this is to extract json data and make it readable

//Insurance to prevent cors error

app.use((req, res, next)=>{

    res.header("Access-Control-Allow-Origin", "\*");

    res.header("Access-Control-Allow-Headers", 'Origin, X-Requested-With, Content-Type, Accept, Authorization');

    if (req.method === 'OPTIONS'){

        res.header("Access-Control-Allow-Methods", "PUT, POST, PATCH, DELETE, GET");

        return res.status(200).json({});

    }

    next();

})

//middleware to handle routes

//middleware to handle routes

app.get('/', (req,res, next)=>{

    res.status(200).json({name: "muneeb"})

});

// app.use('/orders', orderRoutes);

//middleware for error handling

//if error not caught by above routes, then below works

//This one is for 404 error, and shows error Not Found

app.use((req, res, next)=>{

    const error = new Error("Not Found");

    error.status = 404;

    next(error);

})

//This one is for all kind of errors, or 500 error

app.use((error, req, res, next)=>{

    res.status(error.status || 500);

    res.json({

        error:{

            message: error.message

        }

    })

})

//SERVER

const port = process.env.PORT || 3000

app.listen(port, ()=>{

    console.log(`listening to ${port}`)

})

Now run [localhost:3000](http://localhost:3000/) and we get {name: “Muneeb” } as in middle ware I am getting res.status(200).json({name:”Muneeb”})

Later I am going to update it with middle ware with app.use and routes in separate file

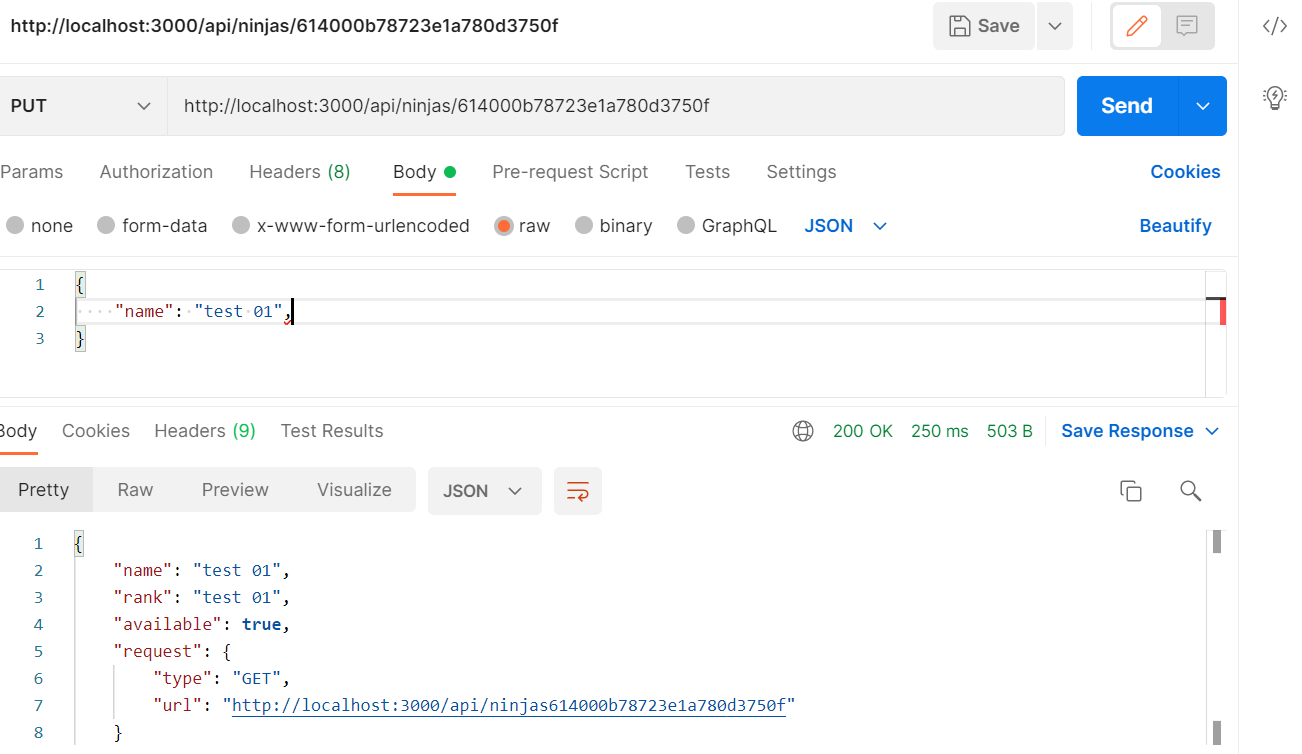
NEXT STEP IS OF ROUTE

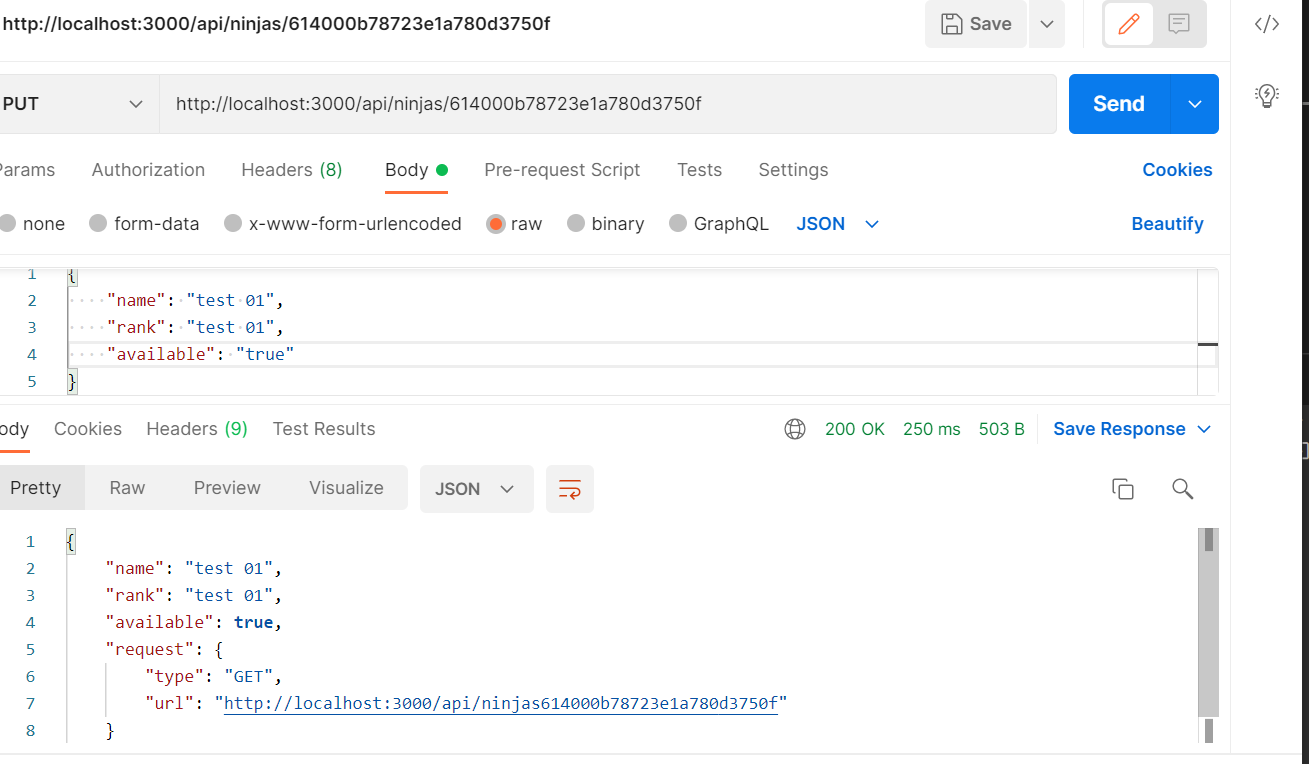
SCHEMA

|  |
| --- |
| const mongoose = require('mongoose') |
|  |  |
|  |  |
|  | const ninjaSchema = mongoose.Schema({ |
|  | name: { |
|  | type: String, |
|  | required: [true, 'Name field is required'] |
|  | }, |
|  | rank: { |
|  | type: String |
|  | }, |
|  | available: { |
|  | type:Boolean, |
|  | default: false |
|  | } |
|  | }); |
|  | module.exports = mongoose.model('Ninja', ninjaSchema) |

PUT:

Postman screenshot:





GeoJson library used for location of geographical location

We have to get nearby locations, for that we required geographical location, and update our schema with it

const mongoose = require('mongoose')

const GeoSchema = new Schema({

    type: {

        type: String,

        default: "Point"

    },

    coordinates = {

        type: [Number],

        index: '2dsphere'

    }

})

const ninjaSchema = mongoose.Schema({

    name: {

        type: String,

        required: [true, 'Name field is required']

    },

    rank: {

        type: String

    },

    available: {

        type:Boolean,

        default: false

    },

    geometry: GeoSchema

});

module.exports = mongoose.model('Ninja', ninjaSchema)