# REST API Primeros pasos

// iniciar proyecto node.js

*npm init -y* // Se crea el package.json

// instalar typescript local si no se instaló global

*npm i typescript --save-dev* //Local

*npm i typescript -g* //Global

// Iniciar proyecto typescript

https://medium.com/@bhagyamangale/tsc-init-4665ec9d7b09

*tsc --init* //Si global. Se crea el tsconfig.json

*npx tsc --init* //Si local. Se crea el tsconfig.json

*git init* // Para crear el repositorio local

// Cambiamos la configuración del *tsconfig.json*

*"target": "es6",*

*"outDir": "./build",*

// Instalación de express, mongoose y morgan

<https://dev.to/mtee/getting-started-with-morgan-3d1m>

// Morgan is a middleware function for logging information

// about the http request/response in a server application.

// Un middleware es un bloque de código que se ejecuta entre

// la petición que hace el usuario (request) hasta que la petición llega al servidor.

*npm i express mongoose morgan*

// nodemon is a tool that helps develop node.js based

// applications by automatically restarting the node

// application when file changes in the directory are detected.

// Instalamos los tipos de datos y módulos de *desarrollo si instalamos typescript así no hay ue instalarlo como está arriba*

*npm install @types/node @types/mongoose @types/express @types/morgan nodemon* ***typescript*** *-D*

// Configuramos el *.gitignore* con:

**// Atención no ignorar build si lo vamos a subir a heroku**

*node\_modules*

// Creamos la carpeta src con *server.ts* //Archivo typescript

Con el contenido que presentamos¨:

import express from 'express'

import morgan from 'morgan'

class Server {

    private app: express.Application

    constructor(){

        this.app = express()

        this.config()

    }

    config(){

        this.app.set('port', process.env.PORT || 3000)

        this.app.use(morgan('dev'))  // Para que muestre las url invocadas

    }

    routes(){

    }

    start(){

        this.app.listen(this.app.get('port'),

        () => {

            console.log(`Server on port: ${this.app.get('port')}`)

        })

    }

}

const server = new Server()

server.start()

// Cambiamos el *package.json* con:

  "scripts": {

    "ts": "tsc -w",

    "dev": "nodemon ./build/server.js",

    "start": "node ./build/server.js"

  },

// Para compilar

*npm run ts* // que como tiene el -w incorporado se compilará si cambiamos el código

// Para ejecutar en desarrollo

*npm run dev* // que como tiene el nodemon se reiniciará el servidor si cambiamos

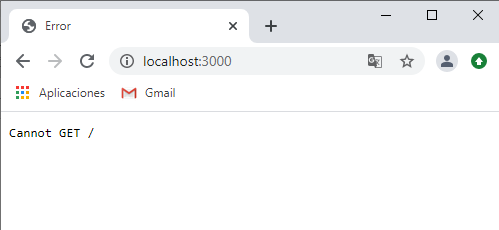
// además como tenemos

// Para ejecutar en producción

*npm start*

// Ya podemos invocar con *localhost:3000*

Aunque como no tenemos rutas la salida en que no puede responder a la ruta /



*PS C:\Users\Adolfo3\Documents\ACurso2021\ASGBD\ProyectosTS\restapitriangulo000****> npm run dev***

*> restapitriangulo000@1.0.0 dev C:\Users\Adolfo3\Documents\ACurso2021\ASGBD\ProyectosTS\restapitriangulo000*

*> nodemon ./build/server.js*

*[nodemon] 2.0.6*

*[nodemon] to restart at any time, enter `rs`*

*[nodemon] watching path(s): \*.\**

*[nodemon] watching extensions: js,mjs,json*

*[nodemon] starting `node ./build/server.js`*

***Server on port: 3000***

***[nodemon] restarting due to changes...***

*[nodemon] starting `node ./build/server.js`*

*Server on port: 3000*

***GET / 404 1.984 ms – 139***

Git:

git init

git add .

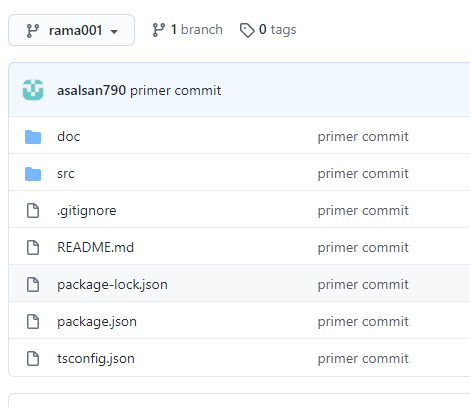
git commit -m "primer commit"

git branch -M rama001   // Escogemos el nombre de la rama

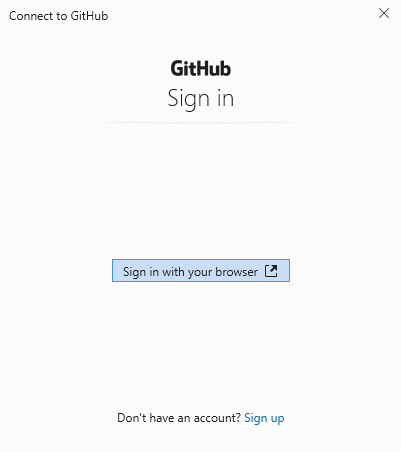
git remote add origin https://github.com/\*\*\*\*\*\*\*\*/restapitriangulo000.git

git push -u origin rama001  // hacemos push de nuestra rama.

Usamos un nombre de rama, *rama001*, distinto del *main* que viene en la documentación



La última versión de git pide identificarse con el navegador:



En Windows 10 se puede elegir el navegador por defecto para que se abra el que deseemos:

1. Haz clic en el menú Inicio. ...
2. Haz clic en Configuración .
3. Abre las aplicaciones predeterminadas: ...
4. Haz clic en tu **navegador** actual (normalmente es **Microsoft** Edge) en la sección "Explorador web", situada en la parte inferior.

Creando nuevas ramas

//Para ver a donde apunta cada rama

git log --oneline --decorate

// Crear una nueva rama

git branch rama002

// Cambiar de rama

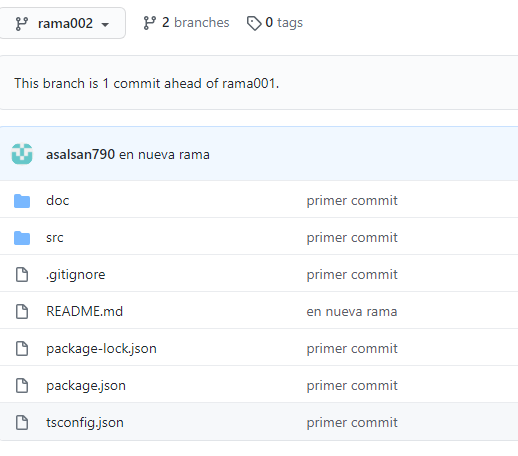
git checkout rama002

git add .

git commit -m "en nueva rama"

git push -u origin rama002

Resultado:



Para subir a heroku:

Después de crear a app en heroku desplegarla desde el GitHub donde la tenemos subida:

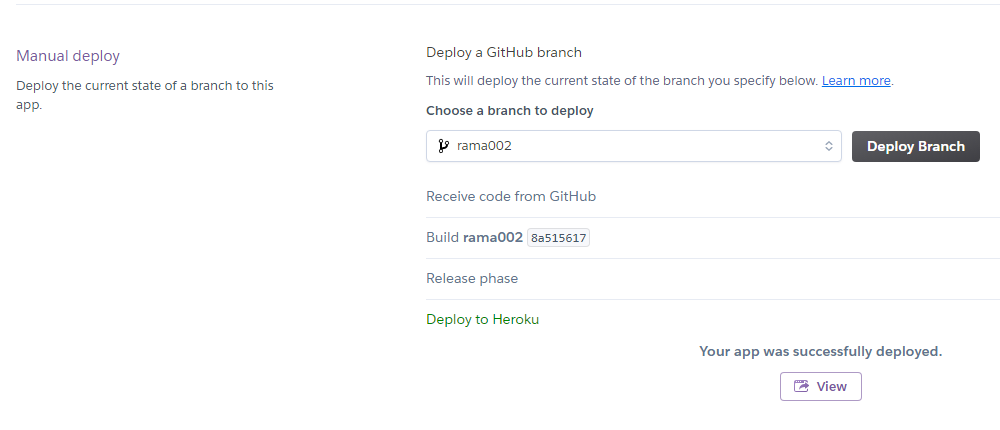
Tendrá que estar compilada de ts a js si estamos ejecutando en desarrollo

tsc –w

siempre lo estará

Luego:

* Que esté compilada
* Subida a GitHub
* Desplegada en heroku



Previamente hay que conectarse:



**Seguimos con la rama002:**

Y creamos la carpeta routes dentro de src

Y el archivo pruebaRoutes.ts

import {Request, Response, Router } from 'express'

class PruebaRoutes {

    private \_router: Router

    constructor() {

        this.\_router = Router()

    }

    get router(){

        return this.\_router

    }

    // Definimos las funciones asociadas a las rutas

    private getPrueba = (req: Request, res: Response) => {

        res.send('Estoy en la /p de la app (con o sin prefijo). Utilizando una función')

    }

    // Aplicamos a la variable de tipo Router métodos get con rutas y las funciones que realizan

    // https://expressjs.com/es/4x/api.html#router.METHOD

    // Para más tarde usarlas en el servidor

    misRutas(){

        this.\_router.get('/',

            (req: Request, res: Response) =>

                    res.send('Estoy en la raiz (con o sin prefijo) de la app. Sin función')

        )

        this.\_router.get('/p', this.getPrueba)

    }

}

// Creamos el objeto

const obj = new PruebaRoutes()

// ejecutamos la asociación rutas > funciones

obj.misRutas()

// Exportamos el parámetro de tipo Router con las rutas asignadas

// Para su uso en el servidor

export const pruebaRoutes = obj.router

Y modificamos el archivo server.ts:

import express from 'express'

import morgan from 'morgan'

import { pruebaRoutes } from './routes/pruebaRoutes'

class Server {

    private app: express.Application

    constructor(){

        this.app = express()

        this.config()

        this.routes()

    }

    private config(){

        this.app.set('port', process.env.PORT || 3000)

        this.app.use(morgan('dev'))  // Para que muestre las url invocadas

    }

    // Usamos la variable tipo Router definida en la clase

    // https://expressjs.com/es/4x/api.html#app.use

    /\*En concreto usamos la sintáxis de la página anterior, aunque sin next() en nuestro caso:

        var router = express.Router();

        router.get('/', function (req, res, next) {

                next();

        });

        app.use(router);\*/

    private routes(){

        this.app.use(pruebaRoutes)

        // Si queremos que todas las rutas tengan un prefijo y poder llamarlas con él:

        this.app.use('/prefijo', pruebaRoutes)

    }

    start(){

        this.app.listen(this.app.get('port'),

        () => {

            console.log(`Server on port: ${this.app.get('port')}`)

        })

    }

}

const server = new Server()

server.start()

**Seguimos con la rama003:**

// Crear una nueva rama para acceso a BD

git branch rama003

// Cambiar de rama

git checkout rama003

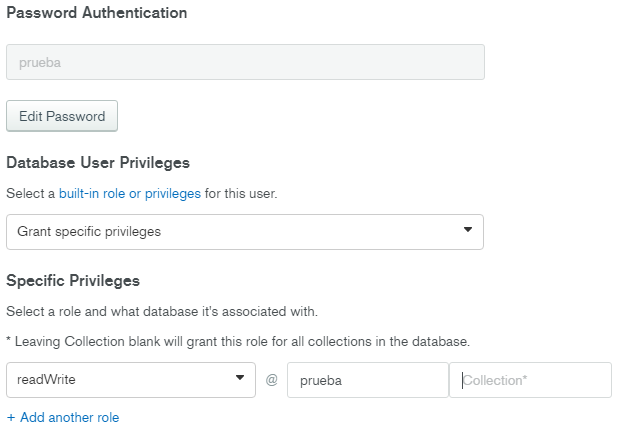
git add .

git commit -m "en nueva rama"

git push -u origin rama003

Conexión con mongoDB

Para atlas creamos un usuario ‘prueba’ con privilegios solo en una bd concreta, en nuestro caso también “prueba”:



Creamos el archivo database.ts:

import mongoose from 'mongoose';

class DataBase {

    private \_cadenaConexion: string = 'mongodb://localhost/test'

    constructor(){

    }

    set cadenaConexion(\_cadenaConexion: string){

        this.\_cadenaConexion = \_cadenaConexion

    }

    conectarBD = async () => {

        const promise = new Promise<string>( async (resolve, reject) => {

            await mongoose.connect(this.\_cadenaConexion, {

                useNewUrlParser: true,

                useUnifiedTopology: true,

                useCreateIndex: true,

                useFindAndModify: true  (comporbar true o false)

            })

            .then( () => resolve(`Conectado a ${this.\_cadenaConexion}`) )

            .catch( (error) => reject(`Error conectando a ${this.\_cadenaConexion}: ${error}`) )

        })

        return promise

    }

    desconectarBD = async () => {

        const promise = new Promise<string>( async (resolve, reject) => {

            await mongoose.disconnect()

            .then( () => resolve(`Desconectado de ${this.\_cadenaConexion}`) )

            .catch( (error) => reject(`Error desconectando de ${this.\_cadenaConexion}: ${error}`) )

        })

        return promise

    }

}

export const db = new DataBase()

Y Actualizamos el archivo server.ts con pruebas de conexión:

import express from 'express'

import morgan from 'morgan'

import { pruebaRoutes } from './routes/pruebaRoutes'

import { db } from './database/database'

class Server {

    private app: express.Application

    constructor(){

        this.app = express()

        this.config()

        this.routes()

    }

    private async config(){

        this.app.set('port', process.env.PORT || 3000)

        this.app.use(morgan('dev'))  // Para que muestre las url invocadas

        const bdLocal = 'test'

        const bdAltas = 'mibd'

        const conexionLocal = `mongodb://localhost/${bdLocal}`

        const conexionAtlas =

            `mongodb+srv://usuario:password@cluster0.viyli.mongodb.net/${bdAltas}?retryWrites=true&w=majority`

        // mongodb+srv://<username>:<password>@cluster0.viyli.mongodb.net/<dbname>?retryWrites=true&w=majority

        db.cadenaConexion = conexionAtlas

        await db.conectarBD()

        .then((mensaje) => {

            console.log(mensaje)

        })

        .catch((mensaje) => {

            console.log(mensaje)

        })

        await db.desconectarBD()

        .then((mensaje) => {

            console.log(mensaje)

        })

        .catch((mensaje) => {

            console.log(mensaje)

        })

    }

    private routes(){

        this.app.use(pruebaRoutes)

        this.app.use('/prefijo', pruebaRoutes)

    }

    start(){

        this.app.listen(this.app.get('port'),

        () => {

            console.log(`Server on port: ${this.app.get('port')}`)

        })

    }

}

const server = new Server()

server.start()

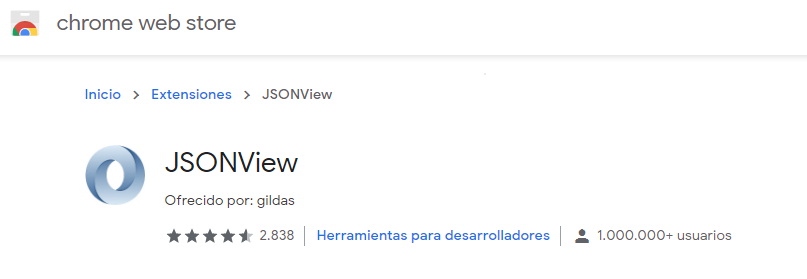
Subimos a Heroku la aplicación rama003 con acceso a Atlas:

Si está ejecutándose en heroku los mensajes de consola los veré con

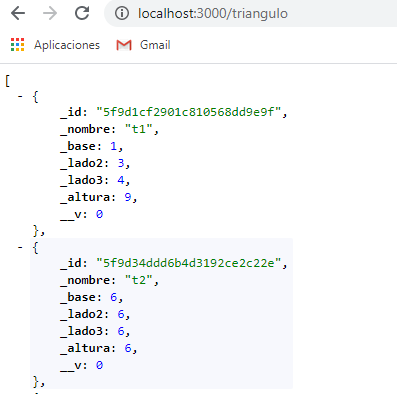
heroku logs -a <APP> --tail

Para subir a heroku poner el usuario y password de atlas, subir a github y luego desplegar, cambiar el usuario y pass y volver a subir a github

He instalado en Chrome:



Así:



He añadido trianguloRoutes.ts:

import {Request, Response, Router } from 'express'

import { Triangulos } from '../model/triangulo'

class TrianguloRoutes {

    private \_router: Router

    constructor() {

        this.\_router = Router()

    }

    get router(){

        return this.\_router

    }

    private getTriangulos = async (req: Request, res: Response) => {

        console.log('hola')

        const query = await Triangulos.find()

        res.json(query)

    }

    misRutas(){

        console.log('en mis rutas')

        this.\_router.get('/', this.getTriangulos)

    }

}

const obj = new TrianguloRoutes()

obj.misRutas()

export const trianguloRoutes = obj.router

el model/triangulo.ts:

import {Schema, model } from 'mongoose'

export class Triangulo{

    private "\_nombre": string

    private \_base: number

    private \_lado2: number

    private \_lado3: number

    private "\_altura": number

    constructor(\_nombre: string, \_base : number, \_lado2 : number, \_lado3 : number

        ){

        this.\_nombre = \_nombre

        this.\_base = \_base

        this.\_lado2 = \_lado2

        this.\_lado3 = \_lado3

    //    this.\_altura = altura

    }

    get nombre(){

        return this.\_nombre

    }

    get base(){

        return this.\_base

    }

    get lado2(){

        return this.\_lado2

    }

    get lado3(){

        return this.\_lado3

    }

    get altura(){

        return this.\_altura

    }

    set altura(\_altura: number){

        /\*

            Si la altura no es la permitida

            levantamos una excepción con throw y su mensaje

            En otro caso asignamos la altura al triángulo

        \*/

        if (\_altura <= 0){

            throw "Altura incorrecta, debe ser > 0"

        }

        this.\_altura = \_altura

    }

    /\*

    Si el método no puede hacer su trabajo levanta una excepción con throw

    y se interrumpe su ejecución en ese punto

    \*/

    perimetro(){

        let perimetro: number

        perimetro = this.\_base+this.\_lado2+this.\_lado3

        if (perimetro == 0){

            throw "Triángulo no creado"

        }

        return perimetro

    }

    area(){

        if (isNaN(this.\_altura)){

            throw "Altura no asignada"

        }

        return (this.\_base\*this.\_altura)/2

    }

}

// Definimos el type

export type tTriangulo = {

    \_nombre: string,

    \_base: number,

    \_lado2: number,

    \_lado3: number,

    \_altura: number

}

// Definimos el Schema

const trianguloSchema = new Schema({

    \_nombre: {

        type: String,

        unique: true  // useCreateIndex: true en la conexión para que se cree el índice único

    },

    \_base: {

        type: Number,

        max: 200

    },

    \_lado2: Number,

    \_lado3: Number,

    \_altura: {

        type: Number,

        min: 5

    }

})

// La colección de la BD: vehiculos (Plural siempre)

export const Triangulos = model('triangulos', trianguloSchema)

Y modificado el server.ts:

import express from 'express'

import morgan from 'morgan'

import { pruebaRoutes } from './routes/pruebaRoutes'

import { trianguloRoutes } from './routes/trianguloRoutes'

import { db } from './database/database'

class Server {

    private app: express.Application

    constructor(){

        this.app = express()

        this.config()

        this.routes()

    }

    private async config(){

        this.app.set('port', process.env.PORT || 3000)

        this.app.use(morgan('dev'))  // Para que muestre las url invocadas

        const bdLocal = 'geometria'

        const bdAltas = 'prueba'

        const userAtlas = '\*\*\*\*\*\*' // Comentar después de desplegar a heroku

        const passAtlas = '\*\*\*\*\*\*\*' // Comentar después de desplegar a heroku

        const conexionLocal = `mongodb://localhost/${bdLocal}`

        const conexionAtlas =

            `mongodb+srv://${userAtlas}:${passAtlas}@cluster0.viyli.mongodb.net/${bdAltas}?retryWrites=true&w=majority`

        // mongodb+srv://<username>:<password>@cluster0.viyli.mongodb.net/<dbname>?retryWrites=true&w=majority

        db.cadenaConexion = conexionLocal

        await db.conectarBD()

        .then((mensaje) => {

            console.log(mensaje)

        })

        .catch((mensaje) => {

            console.log(mensaje)

        })

        /\*

        await db.desconectarBD()

        .then((mensaje) => {

            console.log(mensaje)

        })

        .catch((mensaje) => {

            console.log(mensaje)

        })

\*/

    }

    private routes(){

        this.app.use('/triangulo', trianguloRoutes)

        this.app.use('/prefijo', pruebaRoutes)

    }

    start(){

        this.app.listen(this.app.get('port'),

        () => {

            console.log(`Server on port: ${this.app.get('port')}`)

        })

    }

}

const server = new Server()

server.start()

Hemos añadido identificacionRoutes.ts:

import {Request, Response, Router } from 'express'

import { db } from '../database/database'

class IdentificacionRoutes {

    private \_router: Router

    constructor() {

        this.\_router = Router()

    }

    get router(){

        return this.\_router

    }

    private getId = async (req: Request, res: Response) =>{

        const { password } = req.params

        const { user } = req.params

        setBD(false, user, password) // true BD Local; false BD Atlas

        await db.conectarBD()

        .then( async (mensaje) => {

            console.log(mensaje)

            res.send(mensaje)

        })

        .catch((mensaje) => {

            res.send(mensaje)

            console.log(mensaje)

        })

        db.desconectarBD()

    }

    misRutas(){

        this.\_router.get('/:user&:password', this.getId)

    }

}

const setBD = async (local: boolean, userAtlas: string, passAtlas: string) => {

    const bdLocal = 'geometria'

    const conexionLocal = `mongodb://locadlhost/${bdLocal}`

    if (local) {

        db.cadenaConexion = conexionLocal

    }else{

        const bdAtlas = '\*\*\*\*\*\*\*'

        const conexionAtlas =

        `mongodb+srv://${userAtlas}:${passAtlas}@cluster0.viyli.mongodb.net/${bdAtlas}?retryWrites=true&w=majority`

        db.cadenaConexion = conexionAtlas

    }

}

const obj = new IdentificacionRoutes()

obj.misRutas()

export const identificacionRoutes = obj.router

Para identificarse:

localhost:3000/id/<usuario>&<password>

Para listado de triángulos:

localhost:3000/triangulo

Para trabajar con la BD local escribir *true*

Aunque en desarrollo se puede trabajar también contra Atlas

setBD(false, user, password) // true BD Local; false BD Atlas

Atención hemos desarrollado una version de CRUD de Triángulo que se conecta con Atlas para trabajar con los datos que luego veremos con el REST API se llama ***triangulocrud002***.

En app.ts hay que escribir await setBD(false) // true BD local; false BD Atlas

**Rama 04**

Veremos ahora la creación de un nuevo triángulo en la BD. Lo haremos con el método get y también con el post, viendo las ventajas de este último, que será el que utilicemos normalmente. Para probarlo necesitaremos una utilidad como “postman”.

Nuevo archivo trianguloRoutes.ts:

import {Request, Response, Router } from 'express'

import { Triangulos } from '../model/triangulo'

import { db } from '../database/database'

class TrianguloRoutes {

    private \_router: Router

    constructor() {

        this.\_router = Router()

    }

    get router(){

        return this.\_router

    }

    private getTriangulos = async (req: Request, res: Response) => {

        await db.conectarBD()

        .then( async (mensaje) => {

            console.log(mensaje)

            const query = await Triangulos.find()

            res.json(query)

        })

        .catch((mensaje) => {

            res.send(mensaje)

            console.log(mensaje)

        })

        db.desconectarBD()

    }

    private nuevoTrianguloPost = async (req: Request, res: Response) => {

        console.log('Parametros: ' + req.body)

        // Observar la diferencia entre req.body (para POST)

        // y req.params (para GET con los parámetros en la URL

        const { nombre, base, altura, lado1, lado2 } = req.body

        console.log(nombre)

        const dSchema = {

            \_nombre: nombre,

            \_base: parseInt(base),

            \_lado2: parseInt(lado1),

            \_lado3: parseInt(lado2),

            \_altura: parseInt(altura)

        }

        console.log(dSchema)

        const oSchema = new Triangulos(dSchema)

        await db.conectarBD()

        await oSchema.save()

        .then( (doc) => {

            console.log('Salvado Correctamente: '+ doc)

            res.json(doc)

        })

        .catch( (err: any) => {

            console.log('Error: '+ err)

            res.send('Error: '+ err)

        })

        // concatenando con cadena muestra sólo el mensaje

        await db.desconectarBD()

    }

    private nuevoTrianguloGet = async (req: Request, res: Response) => {

        const { nombre, base, altura, lado1, lado2 } = req.params

        console.log(req.params)

        await db.conectarBD()

        const dSchema = {

            \_nombre: nombre,

            \_base: parseInt(base),

            \_lado2: parseInt(lado1),

            \_lado3: parseInt(lado2),

            \_altura: parseInt(altura)

        }

        const oSchema = new Triangulos(dSchema)

        await oSchema.save()

        .then( (doc) => {

            console.log('Salvado Correctamente: '+ doc)

            res.json(doc)

        })

        .catch( (err: any) => {

            console.log('Error: '+ err)

            res.send('Error: '+ err)

        })

        // concatenando con cadena muestra sólo el mensaje

        await db.desconectarBD()

    }

    misRutas(){

        this.\_router.get('/', this.getTriangulos)

        this.\_router.get('/nuevoG/:nombre&:base&:altura&:lado1&:lado2', this.nuevoTrianguloGet)

        this.\_router.post('/nuevoP', this.nuevoTrianguloPost)

    }

}

const obj = new TrianguloRoutes()

obj.misRutas()

export const trianguloRoutes = obj.router

Observar los dos métodos nuevoTrianguloGet y nuevoTrianguloPost insisto en que el que se utilizará en los proyectos será el Post.

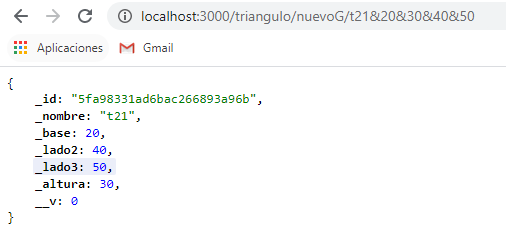
En el server.ts hemos añadido:

        this.app.use(express.json()) // para que nuestro servidor entienda

        // los formatos json desde clientes

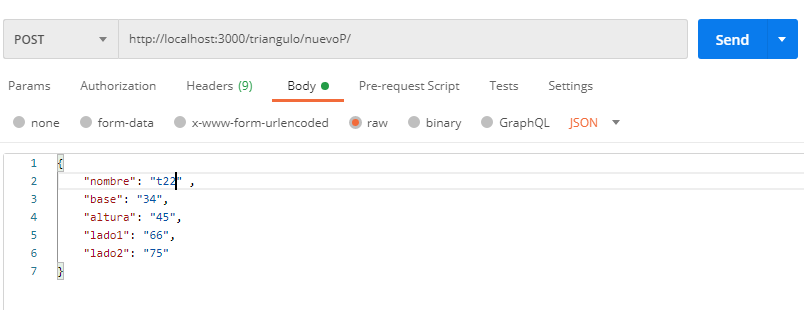
Para crear un nuevo triángulo con get tendremos que escribir la url:

<http://localhost:3000/triangulo/nuevoG/t21&20&30&40&50>



Para crear un nuevo triángulo con post tenemos que usar postman y enviar un body con los parámetros en forma de documento json y la url:

<http://localhost:3000/triangulo/nuevoP>



Una vez enviado (send), tal como tenemos programado

        await oSchema.save()

        .then( (doc) => {

            console.log('Salvado Correctamente: '+ doc)

            res.json(doc)

        })

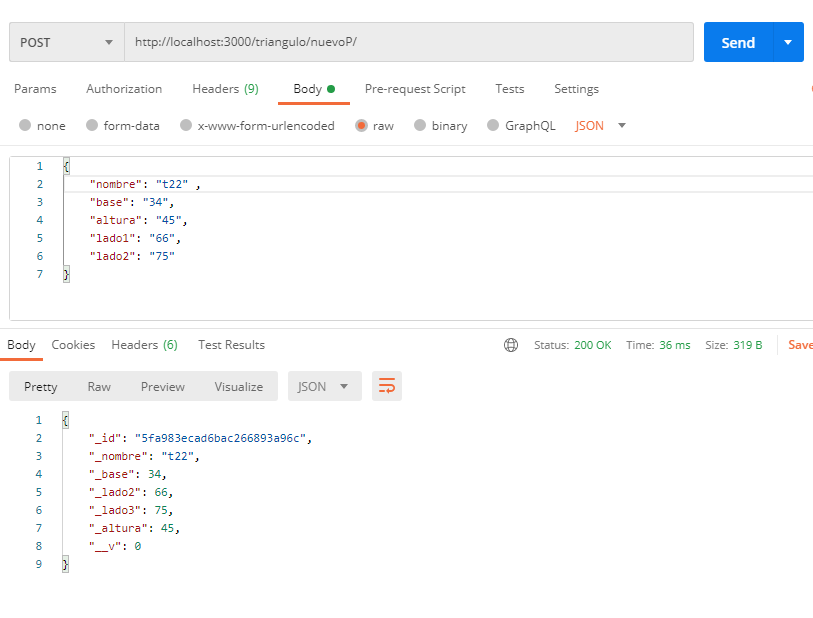
        .catch( (err: any) => {

            console.log('Error: '+ err)

            res.send('Error: '+ err)

        })

Nos devuelve el documento salvado:



Si hubiera algún error de validación nos devolvería el error:

