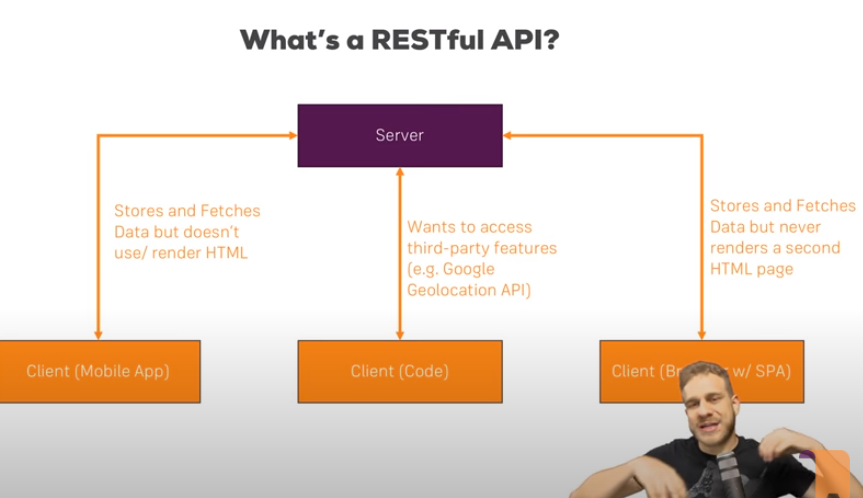
REST API

It stands for representational state transfer

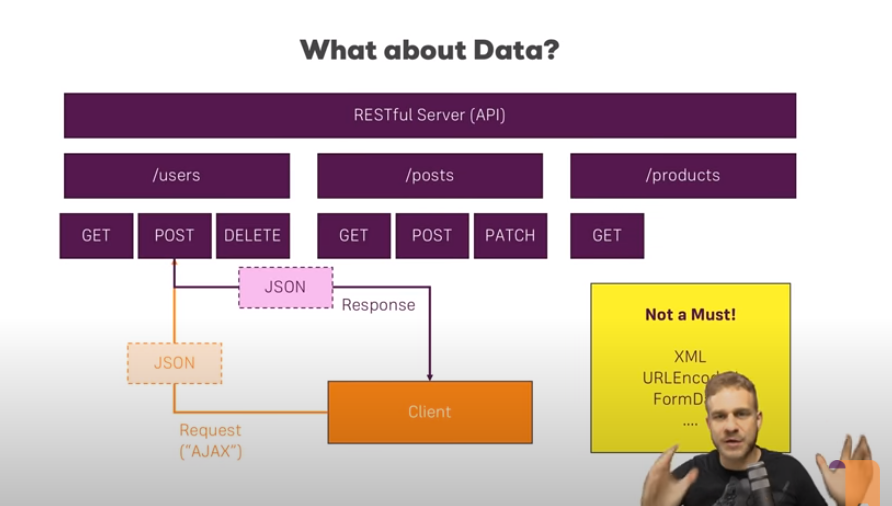
Used to exchange data between client and server, like sending and fetching data to and from server. Data mostly exchange in JSON format, but not mandatory

Does not render html pages

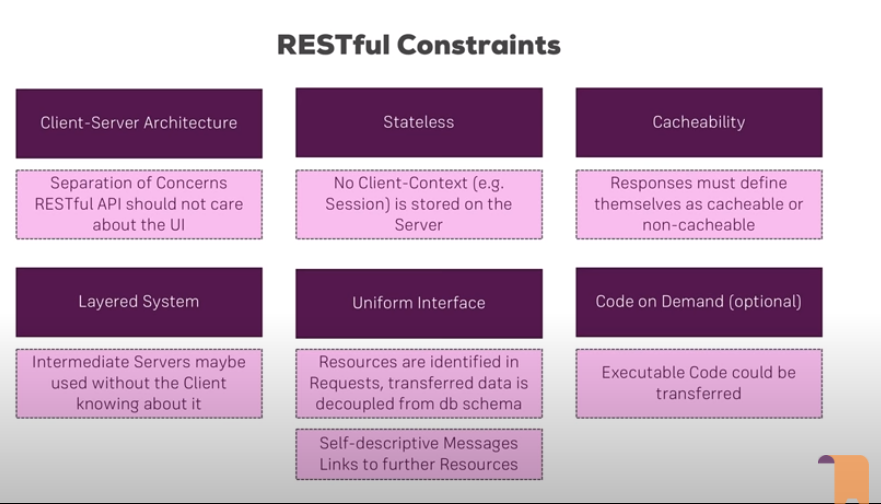
REST just defines a set of rules that says what it is to be a REST api, and a RESTfull service follows those rules.

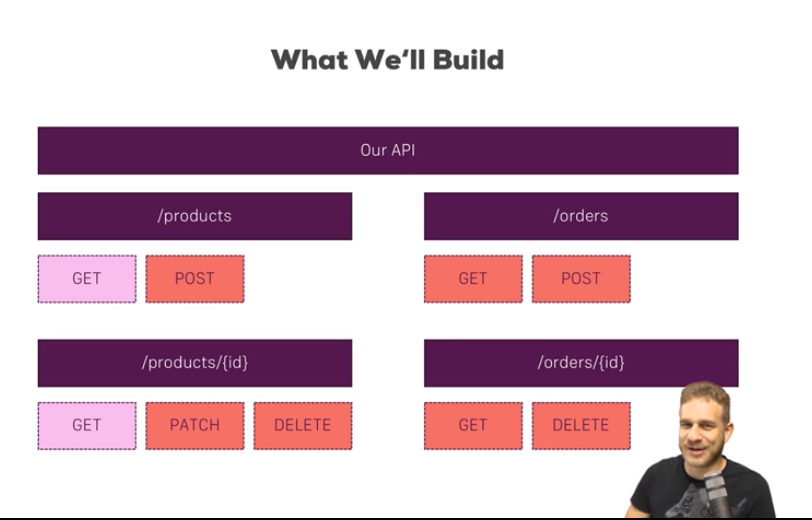


It is a stateless backend, does not care of individual client connected to it.



RESTful Constraints:





Dependencies to install:

Npm I –save-dev morgan (to show logs on terminal or console, not necessary)

Npm I multer (for files, images, like body parser)

Npm init

Npm I express

Npm I –save-dev nodemon

(to make app automatically save, and don’t need to restart it everytime)

After adding it, update package.json file

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "start": "nodemon server.js"

  },

And now app will be start like npm start

SIMPLE GET AND POST METHOD:

Server.js

const http = require('http');

const app = require('./app')

const port = process.env.PORT || 3000

const server = http.createServer(app);

server.listen(port);

app.js

const express = require('express')

const app = express();

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

//middleware

app.use('/products', productRoutes);

module.exports = app;

products.js

const express = require('express')

const router = express.Router(); //to create routes

router.get('/', (req, res, next)=>{

    res.status(200).json({

        message: "Handling GET requests to /products"

    })

})

router.post('/', (req, res, next)=>{

    const product = {

        name: req.body.name,

        price: req.body.price

    }

    res.status(201).json({

        message: "Handling POST requests to /products",

        createdProduct: product

    })

})

//for particular product in products

router.get('/:productId', (req, res, next)=>{

    const id = req.params.productId;

    if(id === 'special'){

        res.status(200).json({

            message: 'You discover the special ID',

            id: id

        })

    }

    else {

        res.status(200).json({

            message: 'You passed an ID'

        })

    }

})

router.patch('/:productId', (req, res, next)=>{

    res.status(200).json({

        message: 'Updated product!'

    })

})

router.delete('/:productId', (req, res, next)=>{

    res.status(200).json({

        message: 'Deleted product'

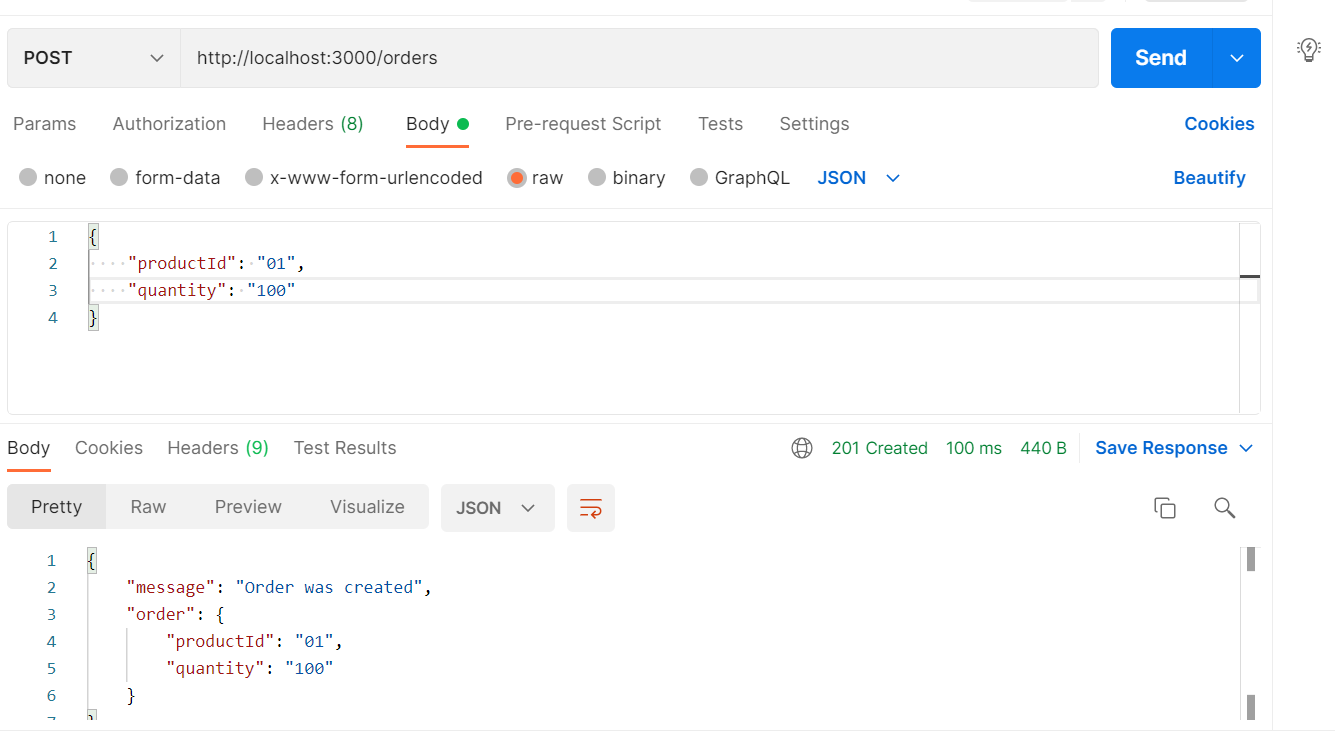
    })

})

module.exports = router;

Till here, we have covered get, post, delete and update route(patch). This is how it works

We can provide name and price in body section of postman and selecting raw and json



Now adding another parent route which is for the orders, creating file orders.js which contains all the routes and which is called in app.js in app.use

Order.js

const express = require('express')

const router = express.Router(); //to create routes

router.get('/', (req, res, next)=>{

    res.status(200).json({

        message: "Orders were fetched"

    })

})

router.post('/', (req, res, next)=>{

    const order = {

        productId: req.body.productId,

        quantity: req.body.quantity

    }

    res.status(201).json({

        message: "Order was created",

        order: order

    })

})

//for particular order in orders

router.get('/:orderId', (req, res, next)=>{

    res.status(200).json({

            message: 'Order details',

            orderId: req.params.orderId

        })

})

router.delete('/:orderId', (req, res, next)=>{

    res.status(200).json({

        message: 'Order deleted',

        orderId: req.params.orderIds

    })

})

module.exports = router;

Same, we can pass json object in postman like we did for products

Calling in app.js

const express = require('express')

const app = express();

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

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

//middleware

app.use('/products', productRoutes);

app.use('/orders', orderRoutes);

module.exports = app;

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Till above, basic routing completed

UPDATED APP.js with error handling (Use this for now):

const express = require('express')

const morgan = require('morgan')

const app = express();

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

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

//morgan shows log in terminal or console

app.use(morgan('dev'))

//middleware

app.use('/products', productRoutes);

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

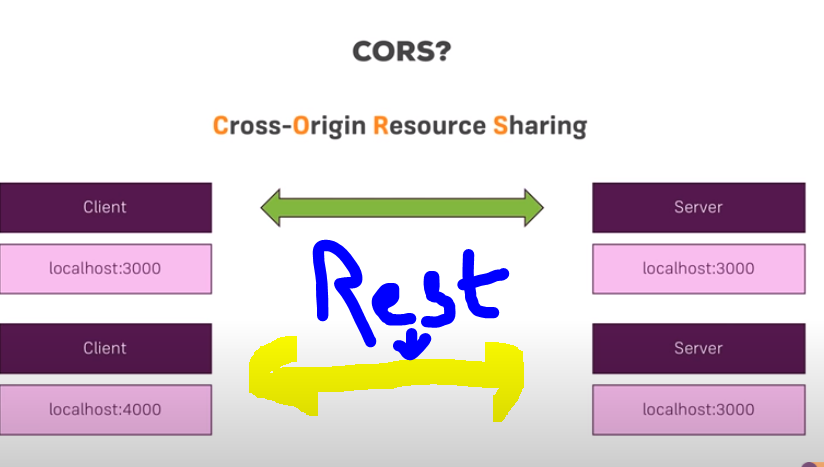
        }

    })

})

module.exports = app;

CORS:

In rest, we have client and server both on different url, so client says that why you are going to different server, to avoid that security issues, we provide headers to avoid those cors errors and asked client to access different server url as well

NOW UPDATED APP.JS with CORS:

Below is the app.js almost have everything, but with db

const express = require('express')

const morgan = require('morgan')

const app = express();

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

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

//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

app.use('/products', productRoutes);

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

        }

    })

})

module.exports = app;

USE OF ATLAS MONGODB:

First update app.js by adding connections

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

app.use('/products', productRoutes);

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

        }

    })

})

module.exports = app;

Now to store and play with db, first create schema for products.js with name product.js

const mongoose = require('mongoose');

const productSchema = mongoose.Schema({

    \_id : mongoose.Schema.Types.ObjectId,

    name: String,

    price: Number

})

module.exports = mongoose.model('Product', productSchema);

Now calling this schema in api products.js and update all apis with this Product

const express = require('express')

const router = express.Router(); //to create routes

const mongoose = require('mongoose')

const Product = require('../Schema/product');

router.get('/', (req, res, next)=>{

    Product.find()

           .exec()

           .then(docs =>{

               console.log(docs);

            //    if (docs.length >= 0){

                res.status(200).json(docs);

        //        }else{

        //            res.status(404).json({

        //                message: "No entries found"

        //            })

        //        }

           })

           .catch(err =>{

               console.log(err);

               res.status(500).json({

                   error:err

               })

           })

})

router.post('/', (req, res, next)=>{

    // const product = {

    //     name: req.body.name,

    //     price: req.body.price

    // }

    const product = new Product({

        \_id: new mongoose.Types.ObjectId(),

        name: req.body.name,

        price: req.body.price

    });

    product.save()

           .then(result=>{

                console.log(result);

                res.status(201).json({

                    message: "Handling POST requests to /products",

                    createdProduct: result

                })

            })

            .catch(err=> {

                console.log(err)

                res.status(500).json({

                    error: err

                })

            });

})

//for particular product in products

router.get('/:productId', (req, res, next)=>{

    const id = req.params.productId;

    Product.findById(id)

           .exec()

           .then(doc=>{

               console.log("From dataabase" + doc)

               if(doc){

                res.status(200).json(doc)

               } else{

                   res.status(404).json({message: 'No valid entry found for ID'})

               }

           })

           .catch(err=>{

               console.log(err);

               res.status(500).json({error: err})

            })

})

router.patch('/:productId', (req, res, next)=>{

    const id = req.params.productId;

    const updateOps = {};

    for (const ops of req.body){

        updateOps[ops.propName] = ops.value; //{name: req.body.newName, price: req.body.newPrice}

    }

    Product.updateOne({\_id:id}, { $set: updateOps})

            .exec()

            .then(result => {

                console.log(result);

                res.status(200).json(result)

            })

            .catch(err => {

                console.log(err);

                res.status(500).json({

                    error: err

                })

            })

    res.status(200).json({

        message: 'Updated product!'

    })

})

router.delete('/:productId', (req, res, next)=>{

    const id = req.params.productId

    Product.remove({\_id: id})

            .exec()

            .then(result => {

                res.status(200).json(result);

            })

            .catch(err => {

                res.status(500).json({

                    error: err

                })

            });

})

module.exports = router;

Updating products.js responses, logic is remain of above one, but we have updated responses and validation

For validation, updated Product.js schema to require true

const mongoose = require('mongoose');

const productSchema = mongoose.Schema({

    \_id : mongoose.Schema.Types.ObjectId,

    name: { type:String, required: true },

    price: { type: Number, required: true }

})

module.exports = mongoose.model('Product', productSchema);

For responses, update products.js file responses

const express = require('express')

const router = express.Router(); //to create routes

const mongoose = require('mongoose')

const Product = require('../Schema/product');

router.get('/', (req, res, next)=>{

    Product.find()

            .select('name price \_id') // to get only these fields

           .exec()

           .then(docs =>{

               const response = {

                   count: docs.length,

                   products: docs.map(doc=>{

                       return {

                           name : doc.name,

                           price : doc.price,

                           \_id : doc.\_id,

                           request: {

                               type: 'GET',

                               url : 'http://localhost:3000/products/' + doc.\_id

                           }

                       }

                   })

               }

            //    if (docs.length >= 0){

                res.status(200).json(response);

        //        }else{

        //            res.status(404).json({

        //                message: "No entries found"

        //            })

        //        }

           })

           .catch(err =>{

               console.log(err);

               res.status(500).json({

                   error:err

               })

           })

})

router.post('/', (req, res, next)=>{

    // const product = {

    //     name: req.body.name,

    //     price: req.body.price

    // }

    const product = new Product({

        \_id: new mongoose.Types.ObjectId(),

        name: req.body.name,

        price: req.body.price

    });

    product.save()

           .then(result=>{

                console.log(result);

                res.status(201).json({

                    message: "Created product successfully",

                    createdProduct: {

                        name: result.name,

                        price: result.price,

                        \_id: result.\_id,

                        request: {

                            type: 'POST',

                            url: 'http://localhost:3000/products/' + result.\_id

                        }

                    }

                })

            })

            .catch(err=> {

                console.log(err)

                res.status(500).json({

                    error: err

                })

            });

})

//for particular product in products

router.get('/:productId', (req, res, next)=>{

    const id = req.params.productId;

    Product.findById(id)

           .select( 'name price \_id')

           .exec()

           .then(doc=>{

               console.log("From dataabase" + doc)

               if(doc){

                const response = {

                    name: doc.name,

                    price: doc.price,

                    \_id: doc.\_id,

                    request:{

                        type: 'GET',

                        url: 'http://localhost:3000/products/' + doc.\_id

                    }

                }

                res.status(200).json(response)

               } else{

                   res.status(404).json({message: 'No valid entry found for ID'})

               }

           })

           .catch(err=>{

               console.log(err);

               res.status(500).json({error: err})

            })

})

router.patch('/:productId', (req, res, next)=>{

    const id = req.params.productId;

    const updateOps = {};

    for (const ops of req.body){

        updateOps[ops.propName] = ops.value; //{name: req.body.newName, price: req.body.newPrice}

    }

    Product.updateOne({\_id:id}, { $set: updateOps})

            .exec()

            .then(result => {

                res.status(200).json({

                    message: "Product Updated",

                    request: {

                        type: 'GET',

                        url: 'http://localhost:3000/products/' + id

                    }

                })

            })

            .catch(err => {

                console.log(err);

                res.status(500).json({

                    error: err

                })

            })

})

router.delete('/:productId', (req, res, next)=>{

    const id = req.params.productId

    Product.remove({\_id: id})

            .exec()

            .then(result => {

                res.status(200).json({

                    message: 'Product deleted',

                    request:{

                        type: 'POST',

                        url: 'http://localhost:3000/products',

                        body: { name: "String", price: 'Number'}

                    }

                });

            })

            .catch(err => {

                res.status(500).json({

                    error: err

                })

            });

})

module.exports = router;

**Updating Orders.js with db and order.js schema with validation**

**First order.js schema**

const mongoose = require('mongoose');

const orderSchema = mongoose.Schema({

    \_id: mongoose.Schema.Types.ObjectId,

    product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },//order of particular product, try to create relation between order and product

    quantity: { type: Number, default: 1 }

});

module.exports = mongoose.model('Order', orderSchema);

**Second orders.js api updation:**

const express = require('express')

const router = express.Router(); //to create routes

const mongoose = require('mongoose');

const Order = require('../Schema/order');

const Product = require('../Schema/product');

router.get('/', (req, res, next)=>{

    Order.find()

        .select("product quantity \_id")

        .exec()

        .then(docs => {

            const response = {

                count : docs.length,

                order: docs.map(doc => {

                    return {

                        \_id : doc.\_id,

                        product: doc.product,

                        quantity: doc.quantity,

                        request : {

                            type: 'GET',

                            url : 'http://localhost:3000/orders/' + doc.\_id

                        }

                    }

                })

            }

            res.status(200).json(response)

        })

        .catch(err => {

            res.status(500).json({

                error: err

            })

        })

})

router.post('/', (req, res, next)=>{

    // const order = {

    //     productId: req.body.productId,

    //     quantity: req.body.quantity

    // }

    Product.findById(req.body.productId)

            .then(product=>{

                if (!product){

                    return res.status(404).json({

                        message: "Product not found"

                    })

                }

                const order = new Order({

                    \_id: mongoose.Types.ObjectId(),

                    quantity: req.body.quantity,

                    product: req.body.productId

                });

                return order.save()

            }).then(result => {

                        const response = {

                            message: 'Order Stored!',

                            createdOrder: {

                                \_id: result.\_id,

                                product: result.product,

                                quantity: result.quantity

                            },

                            request: {

                            type: 'GET',

                            url: 'http://localhost:3000/orders/' + result.\_id

                            }

                        }

                        res.status(201).json(response);

                    })

                    .catch(err=>{

                        console.log(err);

                        res.status(500).json({

                            error: err

                        })

                    })

})

//for particular order in orders

router.get('/:orderId', (req, res, next)=>{

    Order.findById(req.params.orderId)

         .select('-\_\_v')//-ve sign to remove this \_\_v field

         .exec()

         .then(order =>{

            if (!order){

                res.status(404).json({

                    message: "Order not found"

                })

            }

             res.status(200).json({

                 order: order,

                 request: {

                     type: 'GET',

                     url: 'http://localhost:3000/orders'

                 }

             })

         })

         .catch(err => {

             res.status(500).json({

                 error:err

             })

         })

})

router.delete('/:orderId', (req, res, next)=>{

    Order.remove({\_id: req.params.orderId})

        .exec()

        .then(order => {

            res.status(200).json({

                message: "Order Deleted",

                request: {

                    type: 'POST',

                    url: 'http://localhost:3000/orders',

                    body: { productId : 'ID', quantity: "Number"},

                }

            })

        })

        .catch(err => {

            res.status(500).json({

                error:err

         });

    })

});

module.exports = router;

**Update orders.js route with populate method in get route, as we know our orders has relation with product, till now we are getting only orders but not getting product related to that orders, in order to get order and product together, we just add**

**.populate(‘product’) //product as used in ref, because it has relation with order, we can also populate other relations as well if exis. This update code will be**

const express = require('express')

const router = express.Router(); //to create routes

const mongoose = require('mongoose');

const Order = require('../Schema/order');

const Product = require('../Schema/product');

router.get('/', (req, res, next)=>{

    Order.find()

        .select("product quantity \_id")

        .populate('product', 'name')//name of ref property and selecting required fields , incase of relation, otherwise not used

        .exec()

        .then(docs => {

            const response = {

                count : docs.length,

                order: docs.map(doc => {

                    return {

                        \_id : doc.\_id,

                        product: doc.product,

                        quantity: doc.quantity,

                        request : {

                            type: 'GET',

                            url : 'http://localhost:3000/orders/' + doc.\_id

                        }

                    }

                })

            }

            res.status(200).json(response)

        })

        .catch(err => {

            res.status(500).json({

                error: err

            })

        })

})

router.post('/', (req, res, next)=>{

    // const order = {

    //     productId: req.body.productId,

    //     quantity: req.body.quantity

    // }

    Product.findById(req.body.productId)

            .then(product=>{

                if (!product){

                    return res.status(404).json({

                        message: "Product not found"

                    })

                }

                const order = new Order({

                    \_id: mongoose.Types.ObjectId(),

                    quantity: req.body.quantity,

                    product: req.body.productId

                });

                return order.save()

            }).then(result => {

                        const response = {

                            message: 'Order Stored!',

                            createdOrder: {

                                \_id: result.\_id,

                                product: result.product,

                                quantity: result.quantity

                            },

                            request: {

                            type: 'GET',

                            url: 'http://localhost:3000/orders/' + result.\_id

                            }

                        }

                        res.status(201).json(response);

                    })

                    .catch(err=>{

                        console.log(err);

                        res.status(500).json({

                            error: err

                        })

                    })

})

//for particular order in orders

router.get('/:orderId', (req, res, next)=>{

    Order.findById(req.params.orderId)

         .select('-\_\_v')//-ve sign to remove this \_\_v field

         .populate('product')//name of ref property and selecting all fields , incase of relation, otherwise not used

         .exec()

         .then(order =>{

            if (!order){

                res.status(404).json({

                    message: "Order not found"

                })

            }

             res.status(200).json({

                 order: order,

                 request: {

                     type: 'GET',

                     url: 'http://localhost:3000/orders'

                 }

             })

         })

         .catch(err => {

             res.status(500).json({

                 error:err

             })

         })

})

router.delete('/:orderId', (req, res, next)=>{

    Order.remove({\_id: req.params.orderId})

        .exec()

        .then(order => {

            res.status(200).json({

                message: "Order Deleted",

                request: {

                    type: 'POST',

                    url: 'http://localhost:3000/orders',

                    body: { productId : 'ID', quantity: "Number"},

                }

            })

        })

        .catch(err => {

            res.status(500).json({

                error:err

         });

    })

});

module.exports = router;

**Uploading file or an image:**

**First add one field under schema for image, so that we can get this when we want**

const mongoose = require('mongoose');

const productSchema = mongoose.Schema({

    \_id : mongoose.Schema.Types.ObjectId,

    name: { type:String, required: true },

    price: { type: Number, required: true },

    productImage: { type: String, required: true }

})

module.exports = mongoose.model('Product', productSchema);

**We add some upload filters, minor modification in post and get related to product image in products.js route, and rest is same, code is below**

const express = require('express')

const router = express.Router(); //to create routes

const mongoose = require('mongoose')

const multer = require('multer')

//Below work is to enhance image experience with good filename and readable, limits, filters

const storage = multer.diskStorage({

    destination: function(req, file, cb) {

      cb(null, './uploads');

    },

    filename: function(req, file, cb) {

        cb(null, Date.now() + file.originalname);

    }

  });

const fileFilter = (req, file, cb) => {

    //reject or accept file on some conditions

    if (file.mimetype === 'image/jpeg' || file.mimetype === 'image/png' || file.mimetype === 'image/jpg'){

    //accepting or storing file

    cb(null, true);

    }else {

    //rejecting

    cb(null,false);

    }

}

//main code combining all above to upload image

const upload = multer({

    storage: storage,

    limits: {

    fileSize: 1024 \* 2024 \* 5 //bytes, 5 MB

    },

    fileFilter: fileFilter

});//folder where multer uploads all files

//This upload will be used to upload images in post route with upload.single

//starting of routes

const Product = require('../Schema/product');

router.get('/', (req, res, next)=>{

    Product.find()

            // .select('name price productImage \_id') // to get only these fields

           .exec()

           .then(docs =>{

               const response = {

                   count: docs.length,

                   products: docs.map(doc=>{

                       return {

                           name : doc.name,

                           price : doc.price,

                           productImage: doc.productImage,

                           \_id : doc.\_id,

                           request: {

                               type: 'GET',

                               url : 'http://localhost:3000/products/' + doc.\_id

                           }

                       }

                   })

               }

            //    if (docs.length >= 0){

                res.status(200).json(response);

        //        }else{

        //            res.status(404).json({

        //                message: "No entries found"

        //            })

        //        }

           })

           .catch(err =>{

               console.log(err);

               res.status(500).json({

                   error:err

               })

           })

})

//updating with upload.single for image or file

router.post('/', upload.single('productImage'), (req, res, next)=>{

    // const product = {

    //     name: req.body.name,

    //     price: req.body.price

    // }

    console.log(req.file)//available for upload.single

    const product = new Product({

        \_id: new mongoose.Types.ObjectId(),

        name: req.body.name,

        price: req.body.price,

        productImage: req.file.path

    });

    product.save()

           .then(result=>{

                console.log(result);

                res.status(201).json({

                    message: "Created product successfully",

                    createdProduct: {

                        name: result.name,

                        price: result.price,

                        \_id: result.\_id,

                        productImage: result.productImage,

                        request: {

                            type: 'POST',

                            url: 'http://localhost:3000/products/' + result.\_id

                        }

                    }

                })

            })

            .catch(err=> {

                console.log(err)

                res.status(500).json({

                    error: err

                })

            });

})

//for particular product in products

router.get('/:productId', (req, res, next)=>{

    const id = req.params.productId;

    Product.findById(id)

           .select( 'name price \_id productImage')

           .exec()

           .then(doc=>{

               console.log("From dataabase" + doc)

               if(doc){

                const response = {

                    name: doc.name,

                    price: doc.price,

                    \_id: doc.\_id,

                    request:{

                        type: 'GET',

                        url: 'http://localhost:3000/products/' + doc.\_id

                    }

                }

                res.status(200).json(response)

               } else{

                   res.status(404).json({message: 'No valid entry found for ID'})

               }

           })

           .catch(err=>{

               console.log(err);

               res.status(500).json({error: err})

            })

})

router.patch('/:productId', (req, res, next)=>{

    const id = req.params.productId;

    const updateOps = {};

    for (const ops of req.body){

        updateOps[ops.propName] = ops.value; //{name: req.body.newName, price: req.body.newPrice}

    }

    Product.updateOne({\_id:id}, { $set: updateOps})

            .exec()

            .then(result => {

                res.status(200).json({

                    message: "Product Updated",

                    request: {

                        type: 'GET',

                        url: 'http://localhost:3000/products/' + id

                    }

                })

            })

            .catch(err => {

                console.log(err);

                res.status(500).json({

                    error: err

                })

            })

})

router.delete('/:productId', (req, res, next)=>{

    const id = req.params.productId

    Product.remove({\_id: id})

            .exec()

            .then(result => {

                res.status(200).json({

                    message: 'Product deleted',

                    request:{

                        type: 'POST',

                        url: 'http://localhost:3000/products',

                        body: { name: "String", price: 'Number'}

                    }

                });

            })

            .catch(err => {

                res.status(500).json({

                    error: err

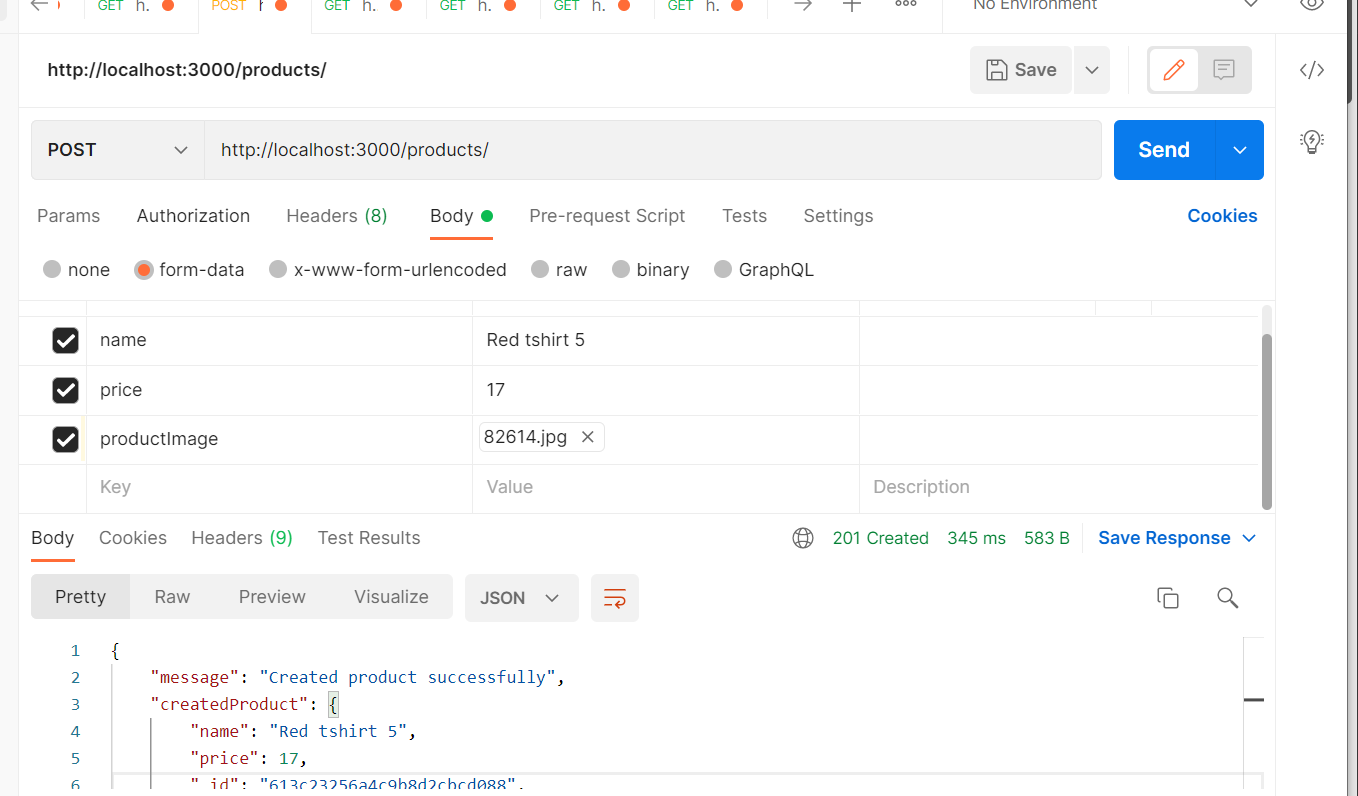
                })

            });

})

module.exports = router;

**In postman, we can do it like**



**Here in productImage, we select type file in one of the key options**

**As we have provided destination to multer uploads/ in storage, it has also created folder and store image in it**

**Now till here, our images are uploading and we are getting it, everything works perfect, but what if we want to use any image or open image in browser or made it publicly available?**

**Update App.js**

**For that we created another middleware.**

//image middleware

app.use('/uploads/', express.static('uploads'))

**so updated file will be;**

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'))

//image middleware

app.use('/uploads/', express.static('uploads'))

//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

app.use('/products', productRoutes);

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

        }

    })

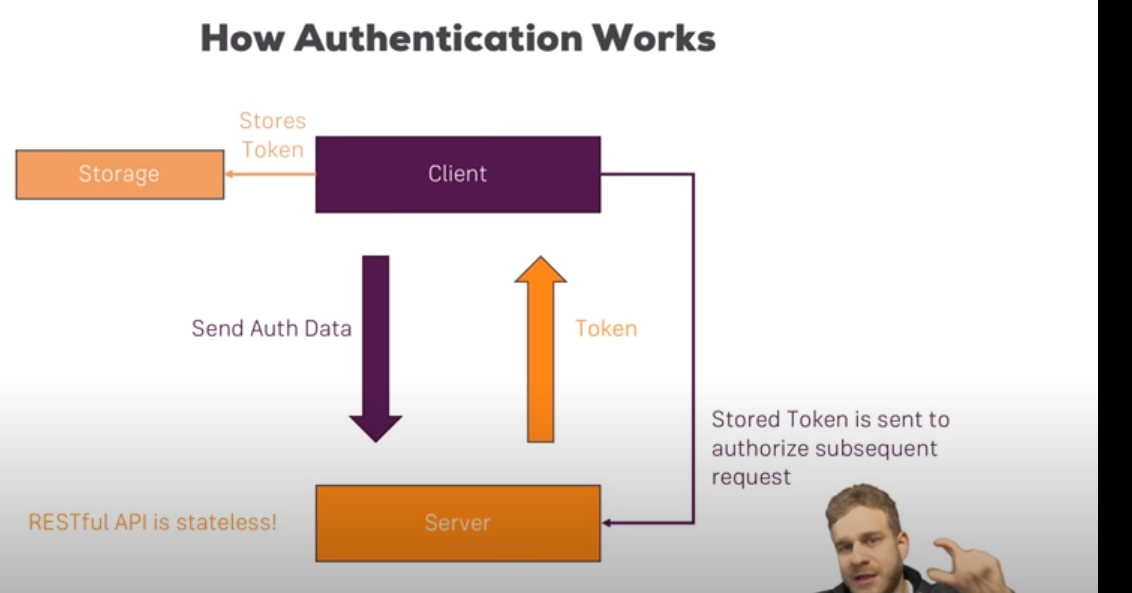
})

module.exports = app;

**Link to access image will be like:**

[1631339831578wp2048438-elon-musk-wallpapers.jpg (1920×1080)](http://localhost:3000/uploads/1631339831578wp2048438-elon-musk-wallpapers.jpg)

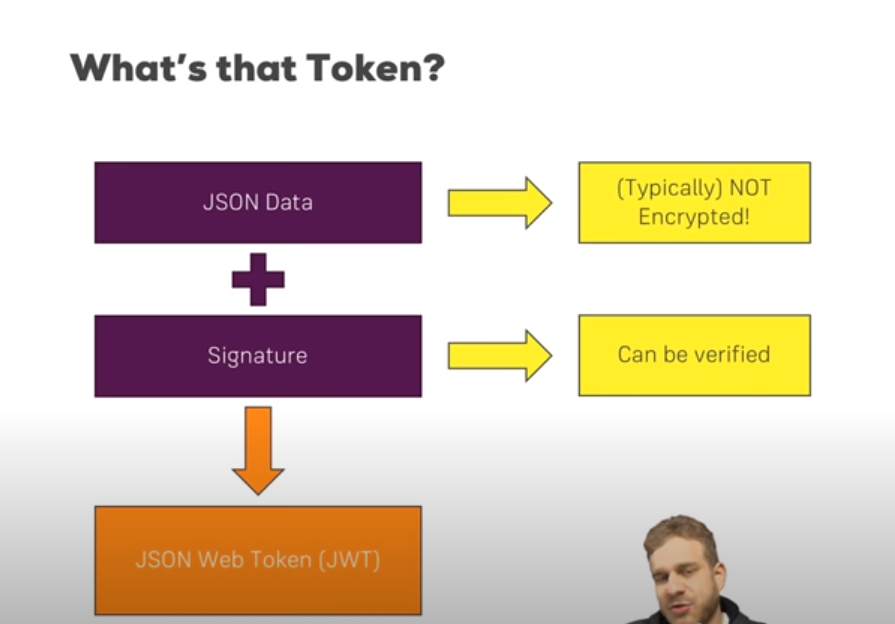
**AUTHENTICATION AND USER SIGNUP:**



**Auth data is basically name email of client etc**

**Authentication is done through token verification on server.**

**It contains private public keys, and only server knows the private keys so it is not possible to manipulate token or do something fishy**



**Dependencies**

**Npm I bcrypt (to hash password to not easily accessible)**

**Npm I jsonwebtoken (to generate token on login)**

**Preparing Schema for user:**

const mongoose = require('mongoose');

const userSchema = mongoose.Schema({

    \_id: mongoose.Schema.Types.ObjectId,

    email : {

         type: String,

         required: true,

         unique: true,

         match:  /[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?/

        },

        //unique do performance optimization but not validation

        //match is used to make email type valid, like @ etc

    password: {type: String, required: true}

});

module.exports = mongoose.model('User', userSchema);

**Now users.js routes to add and delete user:**

const express = require('express')

const router = express.Router(); //to create routes

const mongoose = require('mongoose');

const bcrypt = require('bcrypt');

const User = require('../Schema/user')

//creating new user

router.post('/signup', (req, res, next) => {

    // password: req.body.password //not good, as easily accessible, we have to encrypt or hash password it using package bycrypt

    //below 10 is salt value, which add 10 random strings to avoid password hacking

    //first we check if email already exists or not, if not (else), then create user

    User.find({email: req.body.email})

        .exec()

        .then(user => {

            if (user.length >= 1){

                return res.status(409).json({

                    message: "E-mail already exists"

                })

            }else{

                bcrypt.hash(req.body.password, 10, (err, hash)=>{

                    if (err){

                        return res.status(500).json({

                            error: err

                        })

                    }  else {

                        const user = new User({

                            \_id: new mongoose.Types.ObjectId(),

                            email: req.body.email,

                            password: hash

                        })

                        user

                        .save()

                        .then(result => {

                            console.log(result)

                            res.status(201).json({

                                message: "User created"

                            })

                        })

                        .catch(err=>{

                            console.log(err)

                            res.status(500).json({

                                error: err

                            })

                        })

                    }

                });

            }

        })

})

//delete user route

router.delete('/:userId', (req, res, next)=>{

    User.remove({\_id: req.params.userId})

        .exec()

        .then(result => {

            res.status(200).json({

                message: "User Deleted"

            })

        })

        .catch(err=>{

            console.log(err);

            res.status(500).json({

                error: err

            })

        })

})

module.exports = router;

**To access user api, add middleware in app.js**

**Updated app.js will;**

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')

const userRoutes = require('./api/routes/users');

//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'))

//image middleware

app.use('/uploads/', express.static('uploads'))

//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

app.use('/products', productRoutes);

app.use('/orders', orderRoutes);

app.use('/user', userRoutes);

//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

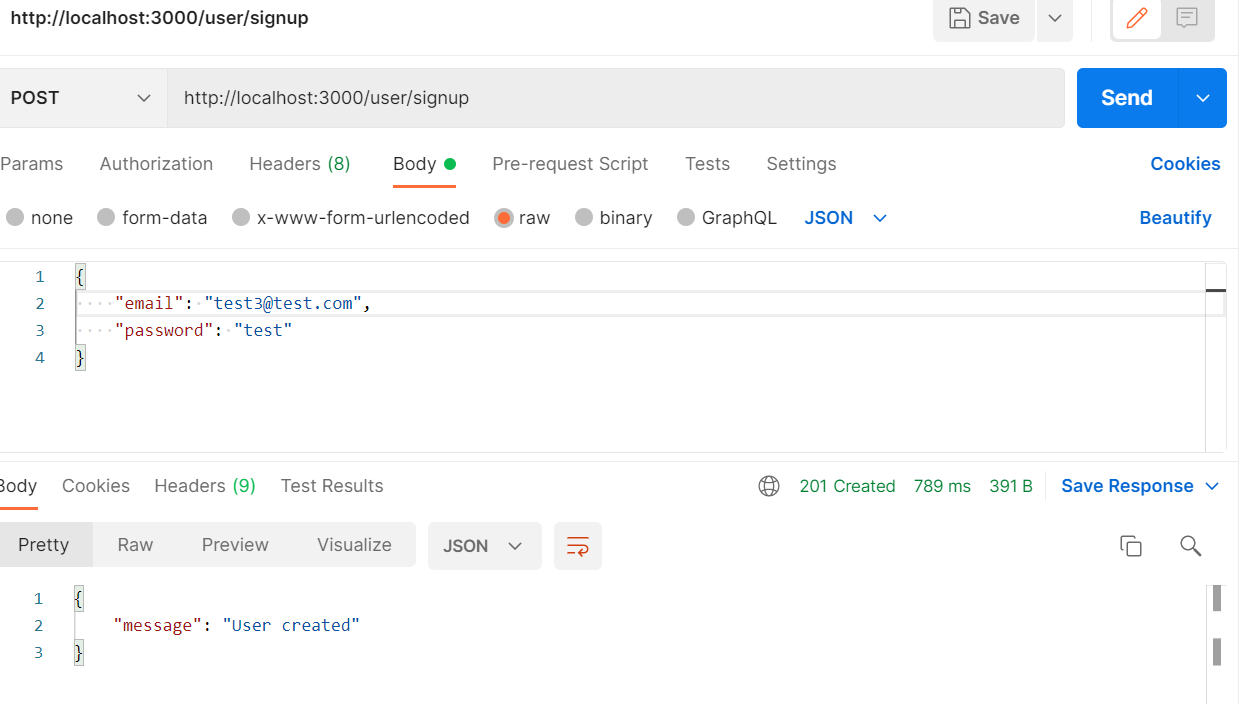
        }

    })

})

module.exports = app;

**Postman signup way:**



**ADDING Login functionality:**

const express = require('express')

const router = express.Router(); //to create routes

const mongoose = require('mongoose');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const User = require('../Schema/user')

//creating new user

router.post('/signup', (req, res, next) => {

    // password: req.body.password //not good, as easily accessible, we have to encrypt or hash password it using package bycrypt

    //below 10 is salt value, which add 10 random strings to avoid password hacking

    //first we check if email already exists or not, if not (else), then create user

    User.find({email: req.body.email})

        .exec()

        .then(user => {

            if (user.length >= 1){

                return res.status(409).json({

                    message: "E-mail already exists"

                })

            }else{

                bcrypt.hash(req.body.password, 10, (err, hash)=>{

                    if (err){

                        return res.status(500).json({

                            error: err

                        })

                    }  else {

                        const user = new User({

                            \_id: new mongoose.Types.ObjectId(),

                            email: req.body.email,

                            password: hash

                        })

                        user

                        .save()

                        .then(result => {

                            console.log(result)

                            res.status(201).json({

                                message: "User created"

                            })

                        })

                        .catch(err=>{

                            console.log(err)

                            res.status(500).json({

                                error: err

                            })

                        })

                    }

                });

            }

        })

})

//login

router.post('/login', (req, res, next)=>{

    User.find({email: req.body.email})

        .exec()

        .then(user => {

            if (user.length <1 ){

                return res.status(401).json({

                    messsage: "Auth failed"

                })

            }

            bcrypt.compare(req.body.password, user[0].password, (err, result)=>{

                   if (err){

                       return res.status(401).json({

                            message: 'Auth failed'

                       })

                   };

                   if (result){

                    //token to login

                       const token = jwt.sign({

                           email: user[0].email,

                           userId: user[0].\_id

                        },

                        process.env.JWT\_KEY,

                        {

                            expiresIn: "1h"

                        }

                    );

                       return res.status(200).json({

                           message: 'Auth successful',

                           token: token

                       })

                   }

                   res.status(401).json({

                        message: 'Auth failed'

                   })

               })

            }

        )

        .catch(err=>{

            console.log(err);

            res.status(500).json({

                error: err

            })

        })

});

//delete user route

router.delete('/:userId', (req, res, next)=>{

    User.remove({\_id: req.params.userId})

        .exec()

        .then(result => {

            res.status(200).json({

                message: "User Deleted"

            })

        })

        .catch(err=>{

            console.log(err);

            res.status(500).json({

                error: err

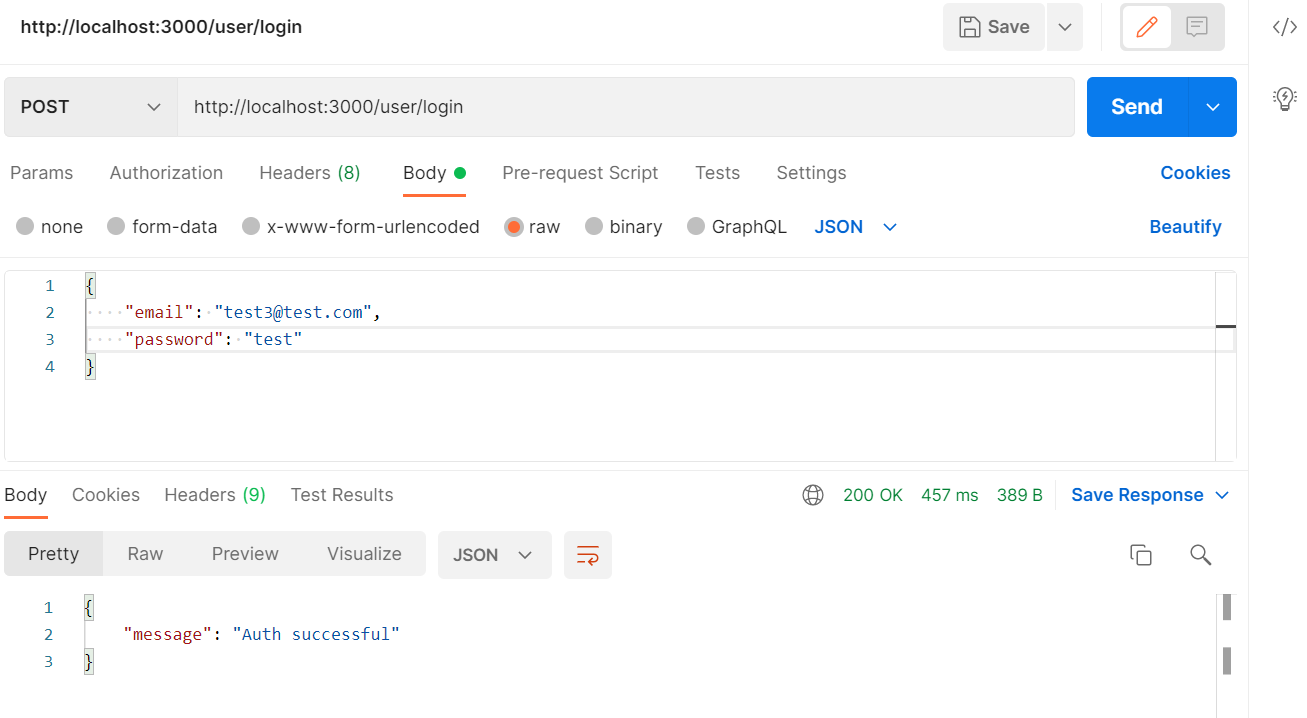
            })

        })

})

module.exports = router;

**Postman screen:**



**Now we have added the login route, but we need to use token so that each person who has login should have token for verification**

**Create auth folder and check-auth.js file**

const jwt = require('jsonwebtoken')

module.exports = (req, res, next) => {

    try {

        const token = req.headers.authorization.split(" ")[1];

        const decoded = jwt.verify(token, process.env.JWT\_KEY);

        req.useData = decoded

        next(); //if successful

    }catch (error){

        return res.status(401).json({

            message: 'Auth failed'

        })

    }

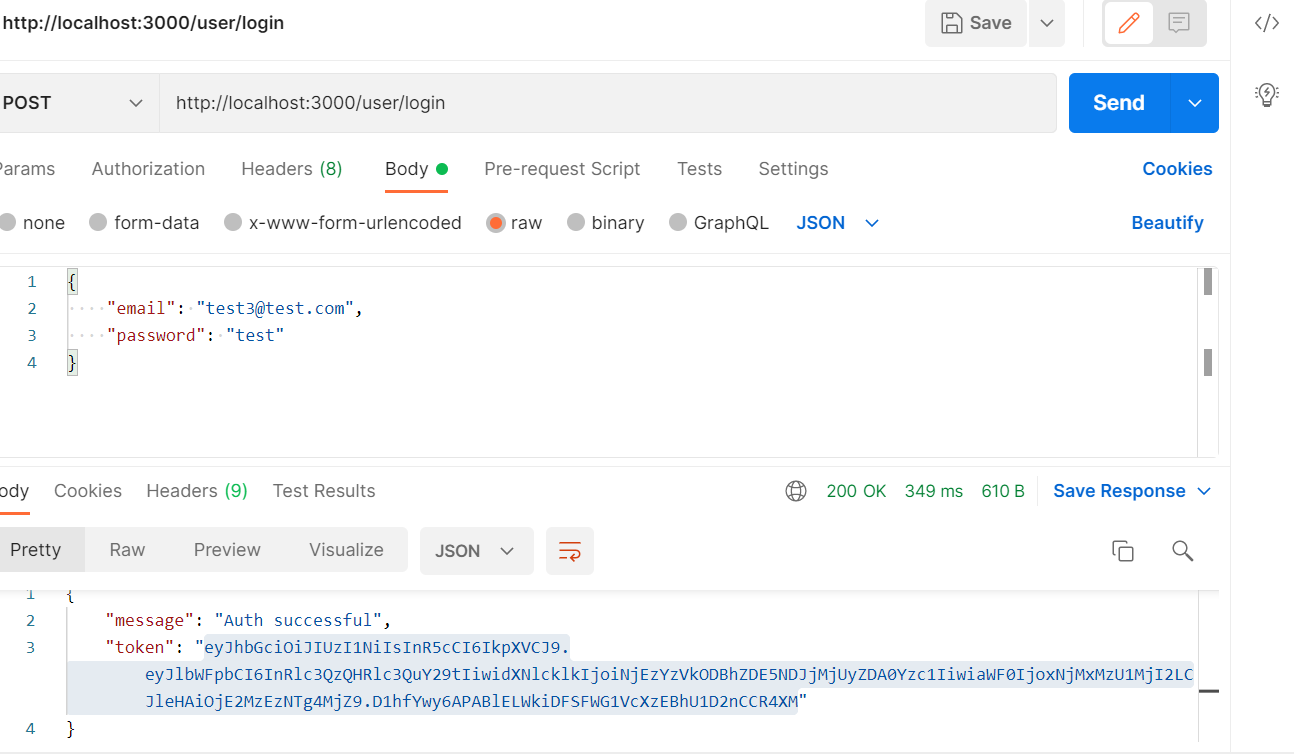
}

**It will use token from headers which we will place, we get token after login**

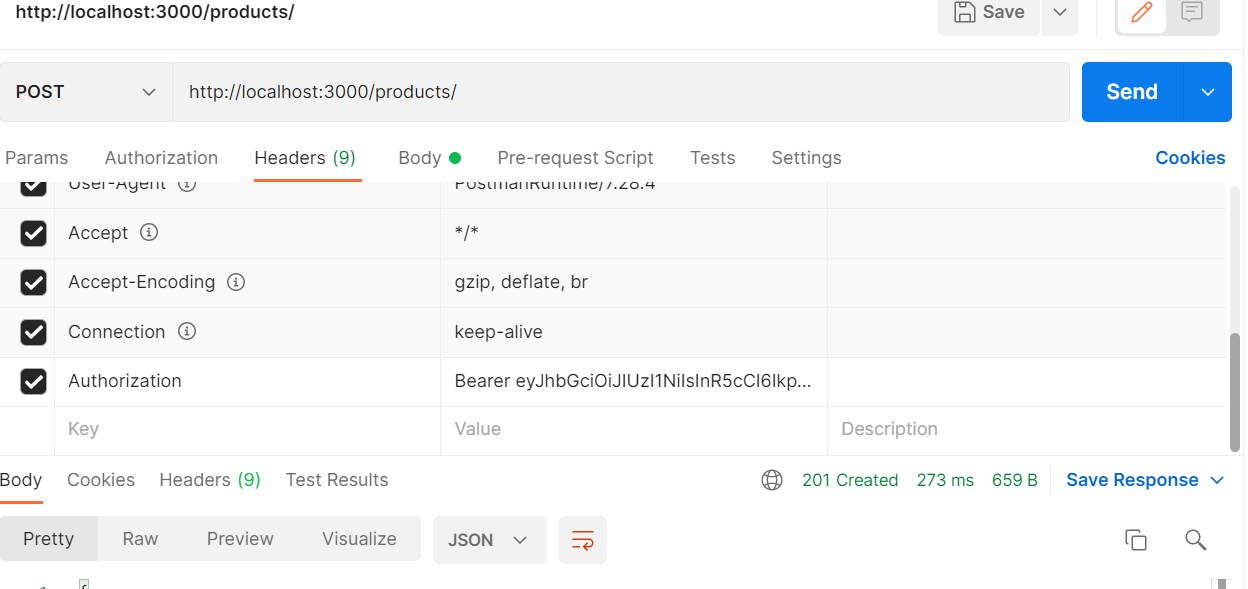
**Steps:**

**First run signup route**

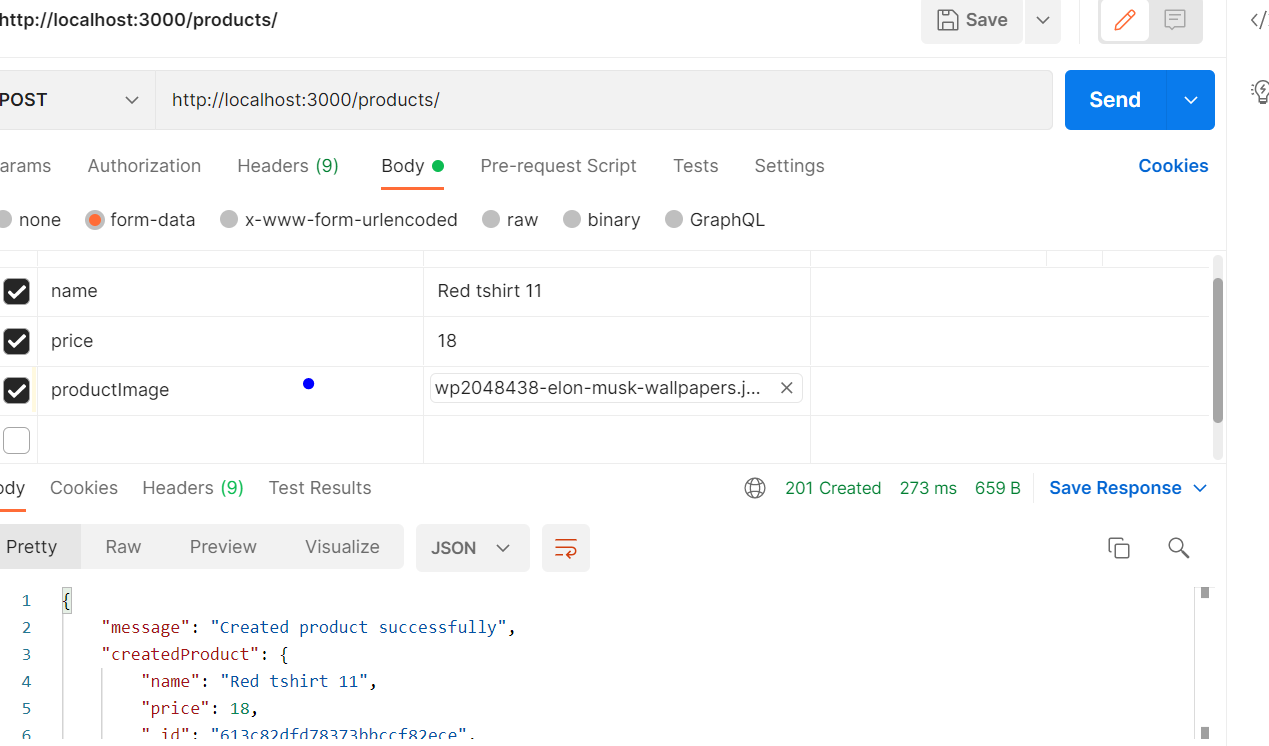
**Second then run login route and took token from their**



**Then in add checkAuth in routes and after that as we are getting token from headers, put token in header of post request**



**Now then run post route**



**Then add checkAuth in all orders as login user can only create delete or get orders**

**Similarly for testing, first put Authorization token in headers of getting order and then get the order**

