https://stackoverflow.com/questions/42921727/how-to-check-req-body-empty-or-not-in-node-express

app.get('/homeworks', async (req, res) => {

    let homeworks = Homework.findAll();

    if(req.query.pass){

        if(req.query.pass="true"){

            homeworks=homeworks.filter(x=>x.grade>=5)

        }

    }

    res.status(200).json(homeworks)

})

app.get('/homeworks/:id', async (req, res) => {

    let homework = await Homework.findByPk(req.params.id)

    if(!homework){

        res.status(404)

    }

    if(req.headers["accept"]=="text/plain"){

        res.status(200).json(homework.content)

    }

    res.status(200).json(homework)

})

function removeOrderItem(orderInfo, position) {

  if (!Array.isArray(orderInfo.items)) {

    throw new Error("Items should be an array");

  }

  const format = orderInfo.items.every(

    (x) => x.hasOwnProperty("price") && x.hasOwnProperty("quantity")

  );

  if (!format) {

    throw new Error("Malformed item");

  }

  if(position>orderInfo.items.length){

    throw new Error('Invalid position')

  }

  orderInfo.total=orderInfo.total-orderInfo.items[position].price\*orderInfo.items[position].quantity

  orderInfo.items.splice(position,1)

  let total=0

//   for (const element of orderInfo.items) {

//     total+=element.price\*element.quantity

//   }

//   orderInfo.total=total

  return orderInfo

}

const app = {

  removeOrderItem,

};

module.exports = app;

pp.get('/ships', async (req, res) => {

   let array = await Ship.findAll();

    if(req.query.pageNo){

        let defaultPageSize=5;

        if(req.query.pageSize){

            defaultPageSize=parseInt(req.query.pageSize)

        }

        if(typeof defaultPageSize =="number" && typeof parseInt(req.query.pageNo)=="number"){

            array=array.splice((parseInt(req.query.pageNo)-1)\*defaultPageSize,defaultPageSize)

        }

    }

    res.status(200).json(array)

})

class Duck{

    constructor(name){

        this.name=name;

        if(typeof name !="string" && !(name instanceof String)){

            throw new Error('name must be string or String')

        }

    }

    swim(){

        return `${this.name} is swimming`

    }

}

class RubberDuck extends Duck{

    float(){

        return `${this.name} floats`

    }

    swim(){

        return `${this.name} can't swim, only float`

    }

}

const app = {

    Duck,

    RubberDuck

}

module.exports = app

/\*

 - funcția distance primește ca parametrii două array-uri

 - fiecare element poate apărea cel mult o dată într-un array; orice apariții suplimentare sunt ignorate

 - distanța dintre cele 2 array-uri este numărul de elemente diferite dintre ele

 - dacă parametrii nu sunt array-uri se va arunca o excepție ("InvalidType")

\*/

/\*

 - the distance function receives as parameters two arrays

 - each element can appear in each array at most once; any duplicates are ignored

 - the distance between the 2 arrays is the number of different elements between them

 - if the parameters are not arrays an exception is thrown ("InvalidType")

\*/

function distance(first, second){

    if(!(first instanceof Array) || !(second instanceof Array)){

        throw new Error('InvalidType');

    }

    else{

        var duplicates = 0;

        var f = new Set(first);

        var s = new Set(second);

        first = [...f];

        second=[...s];

        for(var i=0;i<first.length;i++){

            for(var j=0; j<second.length;j++){

                if(first[i]===second[j]){

                    duplicates++;

                }

            }

        }

        var dist=(first.length+second.length)-(2\*duplicates);

        if(first.length==0 && second.length==0){

            return 0;

        }

        return dist;

    }

}

module.exports.distance = distance

/\*

Definiți un tip obiectual (Widget)

Funcția decorate adaugă la Widget o metodă numită enhance, care crește mărimea unui widget cu "n"

Dacă parametrul trimis nu este un număr, se aruncă o excepție ("InvalidType")

Metoda funcționează și asupra Widget-urilor deja declarate

\*/

/\*

Define a Widget object type is defined

The decorate function adds to Widget a method called enhance which increases the size of a widget with n

If the parameter is not a number an exception is thrown ("InvalidType")

The method also works on already declared Widgets

\*/

class Widget {

constructor(name,size){

       this.name=name;

       this.size=size;

    }

    // TODO: completați implementarea

    // TODO: finish the implementation

    getDescription(){

        return `a ${this.name} of size ${this.size}`

    }

}

function decorate(){

    Widget.prototype.enhance=function(n){

        if(typeof n !="number")

        {

            throw new Error("InvalidType");

        }

        this.size+=n;

    }

}

module.exports.decorate = decorate

module.exports.Widget = Widget

/\*

Exista un tip obiectual definit (Bird)

Să se definească tipul Penguin.

Un pinguin este un tip copil pentru Bird și are în plus metoda swim(distance)

Un pinguin nu poate fi creat fără un nume de tip string

Un pinguin nu poate să zboare și va spune asta dacă i se cere

Dacă se apelează makeNest, un pinguin va apela metoda părintelui său

Vedeți testele pentru formatul exact al mesajelor

\*/

/\*

There is an object type called Bird

Define the Penguin type

A penguin is a child type for Bird and has an additional method called swim(distance)

A penguin cannot be created without a name of type string

A penguin cannot fly and will say that if asked

A penguin can make a nest via its parent's method

See the tests for the accurate format of messages

\*/

class Bird {

    constructor(name){

        this.name = name

    }

    fly(distance){

        return `${this.name} flies ${this.distance}`

    }

    makeNest(){

        return `${this.name} makes a nest`

    }

}

class Penguin  extends Bird {

    constructor(name){

        if (!name || typeof name !== 'string'){

            throw new Error('CreationError')

        }

    super(name);

    }

    fly(distance){

        return `${this.name} is a penguin and cannot fly`

    }

    swim(distance){

        return `${this.name} swims ${distance}`

    }

}

// TODO: implementați tipul Pinguin

// TODO: implement the Penguin type

module.exports.Bird = Bird

module.exports.Penguin = Penguin

 /\*

 - funcția capitalize primește ca parametrii un string și un array

 - dicționarul conține o serie de termeni

 - in textul ințial cuvintele sunt separate de spațiu

 - fiecare termen din dicționar trebuie să apară capitalizat în rezultat

 - rezultatul este un string nou, fără modificarea celui inițial

 - dacă textul nu este un string sau dicționarul nu este un array de string-uri se va arunca o excepție (mesajul TypeError)

\*/

/\*

 - the capitalize function receives as parameters a string and an array

 - the dictionary (the array) contains a series of words

 - in the initial text the words are separated by space

 - each dictionary term has to appear capitalized in the result

 - the result is a new string without modifying the initial one

 - if the text is not string or the dictionary not an array of strings an exception is thrown (message is TypeError)

\*/

function capitalize(text, dictionary){

  if(typeof text!=="string" || !Array.isArray(dictionary))

  {

    throw new Error("TypeError");

  }

  if (dictionary.filter((e) => typeof e !== 'string').length){

        throw new Error('TypeError')

    }

  let result = text

    let items = result.split(' ')

    for (let i = 0; i < items.length; i++){

        if (dictionary.indexOf(items[i]) !== -1){

            items[i] = items[i][0].toUpperCase() + items[i].slice(1, items[i].length)

        }

    }

    return items.join(' ')

}

module.exports.capitalize = capitalize

/\*

 - funcția translate primește ca parametrii un string și un obiect

 - funcția aruncă excepții dacă tipurile nu sunt respectate (mesajul "InvalidType")

 - obiectul dicționar are în cheie valoarea inițială și în valoare traducerea ei

 - valorile din dicționar sunt string-uri

 - funcția înlocuiește fiecare cheie din dicționar găsită în textul inițial cu valoarea tradusă

\*/

/\*

 - the translate function receives as parameters a string and an object

 - the function throws exceptions if the types are not the required ones (message is "InvalidType")

 - the dictionary object has in its keys the inital values and in its values the translation of the key

 - the values in the dictionary are strings

 - the function replaces each dictinary key found in the initial text with the value in the dictionary corresponding to the key

\*/

function translate(text, dictionary){

    // TODO: implementați funcția

    // TODO: implement the function

    if( typeof text!=="string")

    {

        throw new Error("InvalidType");

    }

    if(typeof dictionary!=="object" || !dictionary)

    {

        throw new Error('InvalidType');

    }

    for (let prop in dictionary){

        if (typeof dictionary[prop] !== 'string'){

            throw new Error('TypeError')

        }

    }

    let result = text.split(' ')

    for (let prop in dictionary){

        let position = result.indexOf(prop)

        if (position !== -1){

            result[position] = dictionary[prop]

        }

    }

    return result.join(' ')

}

module.exports.translate = translate

function textProcessor(input, tokens){

    if(typeof input=="string")

    {

        if(input.length>=6)

        {

            tokens.forEach((e)=>

            {

                if( typeof e.tokenName=="string" && typeof e.tokenValue=="string" )

                {

                    var t= "${"+ e.tokenName + "}";

                    input= input.replace(t, e.tokenValue);

                }

                else

                {

                    throw new Error("Invalid array format");

                }

            });

            return input;

        }

        else

        {

        throw new Error("Input should have at least 6 characters");

        }

    }

    else

    {

        throw new Error("Input should be a string");

    }

}

const app = {

    textProcessor: textProcessor

};

module.exports = app;

function applyBonus(employees, bonus){

     return new Promise((resolve, reject) => {

        if(typeof bonus=="number") {

                 var maxsalary=0;

            employees.forEach((e)=>

            {

                if(typeof e.name!="string" && typeof e.salary!="number" )

                {  var eroare1='Invalid array format';

                      reject(new  Error(eroare1));

                }

                else

                {

                    if( e.salary > maxsalary)

                    {

                        maxsalary=e.salary;

                    }

                }

            });

        if(bonus< maxsalary\*0.1)

        {

            reject("Bonus too small");

        }

        resolve(employees.map(e=>{

                    return{name: e.name,salary:e.salary+bonus};

                }));

        }

        else

        { var  eroare="Invalid bonus";

             reject(new Error(eroare));

        }

     })

}

let app = {

    applyBonus: applyBonus,

}

module.exports = app;

function applyDiscount(vehicles, discount){

      return new Promise((resolve, reject) => {

          if(typeof discount=="number")

          {var minprice=9999999;

             vehicles.forEach((e)=>

             { if(typeof e.make!="string" && typeof e.price!="number")

                {

                var eroare2="Invalid array format";

                reject(new Error(eroare2));

                 }

                 else

                 {

                     if(e.price<minprice)

                     {

                         minprice=e.price;

                     }

                 }

             });

             if(discount> minprice\*0.5)

             {

                 reject("Discount too big");

             }

              else {resolve(vehicles.map(e=>{

                    return{make: e.make,price:e.price-discount};

                })); }

          }

          else

          {

              var eroare1='Invalid discount';

              reject(new Error(eroare1));

          }

      });

}

const app = {

    applyDiscount: applyDiscount

};

module.exports = app;

function addTokens(input, tokens){

     if( typeof input!=="string"){

         throw new Error("Invalid input");

     }

    else

    {

        if(input.length <6)

        {

            throw new Error("Input should have at least 6 characters");

        }

    else {

        tokens.forEach((e)=>

         {

         if( typeof e.tokenName !=="string")

         {

             throw new Error("Invalid array format");

         }

         else

         {

         }

        })

        var inputnou;

        if(input.includes("..."))

        { tokens.forEach((e)=>{

            inputnou=input.replace("...", "${" + e.tokenName +"}");});

            return inputnou;

        }

        else

        {return input;

        }

    }

    }

}

const app = {

    addTokens: addTokens

}

module.exports = app;

function applyBlackFriday(products, discount){

    return new Promise((resolve, reject) => {

        if(typeof discount=="number") {

            products.forEach((e)=>

            {

                if(typeof e.name!="string" && typeof e.price!="number" )

                {  var eroare1='Invalid array format';

                      reject(new  Error(eroare1));

                }

            });

        if(discount<=10 && discount>0)

        {      resolve(products.map(e=>{

                    return{name: e.name,price: e.price- e.price\*discount/100};

                }));

        }

        else

        {

            var  eroare2="Discount not applicable";

             reject(new Error(eroare));

        }

        }

        else

        { var  eroare="Invalid discount";

             reject(new Error(eroare));

        }

     })

}

const app = {

    applyBlackFriday: applyBlackFriday

};

module.exports = app;

function processString(input){

    if(input.length==0){

        return 100;

    }else{

        let sir=input.split(' ');

        let sum=0;

        for(var i=0;i<sir.length;i++){

            if(!isNaN(sir[i])){

                if(parseInt(sir[i])%2==0){

                    sum+=parseInt(sir[i]);

                }

            }else{

                console.log(typeof parseInt(sir[i]));

                throw new Error("Item is not a number");

            }

        }

        return 100-sum;

    }

}

const app = {

    processString: processString

}

module.exports = app

function getAverageGrade(gradeItems){

     var sum=0;

    for(let i =0 ;i < gradeItems.length ;i++)

    {

        if(typeof gradeItems[i].grade !=="number"  ||  gradeItems[i].grade=='A' || typeof gradeItems[i].grade ==="undefined" || gradeItems[i].grade <0 )

        {

            throw new Error("Invalid grade");

        }

        else

        {

             sum+=gradeItems[i].grade;

        }

    }

    if(gradeItems.length===0)

    {

        return 0;

    }

    else{

    return  sum/gradeItems.length;

    }

}

const app = {

    getAverageGrade: getAverageGrade

}

module.exports = app

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>A simple app</title>

</head>

<body>

    A simple app

    <table id=main></table>

    <input type="text" id="filter" />

    <button id=reload></button>

</body>

<script>

    window.onload = async () => {

        let btn = document.getElementById('reload')

        let filter = document.getElementById('filter')

        let tbl = document.getElementById('main')

        const preluareDate =()=>{

            const getData = new XMLHttpRequest();

            getData.open('GET', 'http://localhost:8080/cars')

            getData.responseType = 'json'

            getData.onload =()=>{

                const myData = getData.response

                console.log(filter.value)

                if(filter.value.length == 0){

                    for(let i=0;i<myData.length;i++){

                        let linie = document.createElement('tr')

                        for(let element in myData[i]){

                            let coloana = document.createElement('td')

                            coloana.innerText = myData[i][element]

                            linie.append(coloana)

                        }

                        tbl.append(linie)

                    }

                }

                else if(filter.value == 'red'){

                    for(let i=0;i<myData.length;i++){

                        if(myData[i]['color'] == 'red'){

                            let linie = document.createElement('tr')

                            for(let element in myData[i]){

                                let coloana = document.createElement('td')

                                coloana.innerText = myData[i][element]

                                linie.append(coloana)

                            }

                            tbl.append(linie)

                        }

                    }

                }

            }

            getData.send()

        }

        btn.addEventListener('click', preluareDate)

    }

</script>

</html>

<!DOCTYPE html>

<html>

    <body>

        <div id="main"></div>

        <script>

            fetch("http://18.188.244.145:8080/data")

            .then(response=> response.json())

            .then(data=> {

               const json= JSON.stringify(data);

               const div= document.querySelector('#main');

               div.innerHTML=json;

            });

        </script>

    </body>

</html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>A simple app</title>

</head>

<body>

    A simple app

    <input type="text" placeholder="name" id="name" />

    <button id=del></button>

    <table id=main></table>

</body>

<script>

    const buttonDelete = document.getElementById('del')

    const myTable = document.getElementById('main')

    const myText = document.getElementById('name')

    let simpleData;

    function loadData(){

        const myData = new XMLHttpRequest();

        myData.open('GET', 'http://localhost:8080/cars')

        myData.responseType = 'json'

        myData.onload =()=>{

             simpleData = myData.response;

            for(let i=0; i< simpleData.length;i++){

                let linie = document.createElement('tr')

                for(let element in simpleData[i]){

                    let coloana = document.createElement('td')

                    coloana.innerText = simpleData[i][element]

                    linie.append(coloana)

                }

                myTable.append(linie)

            }

        }

        myData.send();

    }

    loadData();

     function deleteData(){

        myTable.innerHTML='';

        console.log(myText.value)

        for(let i=0; i< simpleData.length;i++){

            if(simpleData[i]['name'] == myText.value){

            }else{

                let linie = document.createElement('tr')

                    for(let element in simpleData[i]){

                        let coloana = document.createElement('td')

                        coloana.innerText = simpleData[i][element]

                        linie.append(coloana)

                    }

                    myTable.append(linie)

                }

            }

    }

    buttonDelete.addEventListener('click', deleteData)

</script>

</html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>A simple app</title>

    <script>

            let load = async () => {

                try{

                    let response = await fetch(`/profile.json`)

                }

                catch(err){

                    console.warn(err)

                }

            }

            let convert = () => {

            }

    </script>

</head>

<body>

    <div id="content"></div>

    <input type="button" value="convert" id="convert" onclick="convert()" />

</body>

</html>

function textProcessor(input, tokens){

   if(typeof input!=='string'){

    throw new Error("Input should be a string")

   }

   if(input.length<6){

    throw new Error('Input should have at least 6 characters')

   }

   for (const element of tokens) {

    if(typeof element.tokenName !=="string" || typeof element.tokenValue !=="string"){

        throw new Error('Invalid array format')

    }

   }

        if(!input.includes('${')){

            return input

        }

        for(const element of tokens){

            input=input.replace("${"+element.tokenName+"}",element.tokenValue)

        }

        return input

}

const app = {

    textProcessor: textProcessor

};

module.exports = app;

app.post("/products", (req, res, next) => {

  try {

    if (req.body.constructor === Object && Object.keys(req.body).length === 0) {

      //if(JSON.stringify(req.body)=='{}'){

      //res.status(500).json({ message: "Body is missing" });

      throw "Body is missing"

    }

    if (

      !req.body.hasOwnProperty("name") ||

      !req.body.hasOwnProperty("category") ||

      !req.body.hasOwnProperty("price")

    ) {

      //res.status(500).json({ message: "Invalid body format" });

      throw "Invalid body format"

    }

    if (req.body.price < 0) {

      //res.status(500).json({ message: "Price should be a positive number" });

      throw "Price should be a positive number"

    }

    for (const element of app.locals.products) {

      if (req.body.name == element.name) {

        //res.status(500).json({ message: "Product already exists" });

        throw "Product already exists"

      }

    }

    app.locals.products.push(req.body);

    console.log(req.body);

    res.status(201).json({ message: "Created" });

  } catch (error) {

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

  }