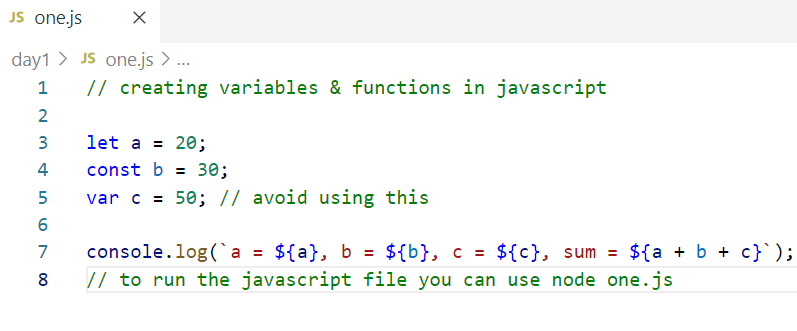
Node.js Training

Node.js: It is a runtime environment used to run Javascript programs at the backend so that you can access various backend resources likes OS, Files, Databases

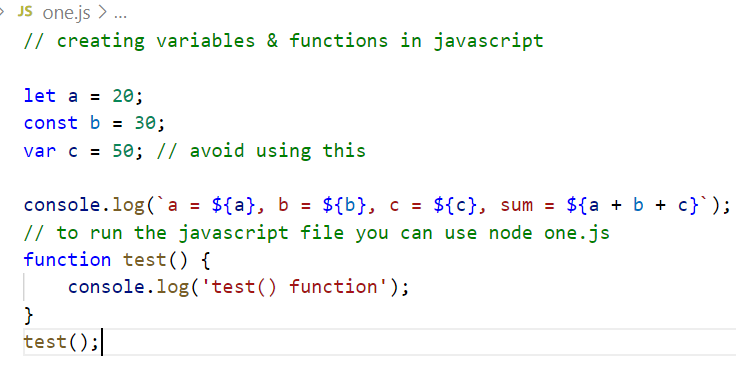
REPL Terminal: It is a node.js terminal which helps you to quickly test your javascript code for simple calculations

REPL stands for Read Evaluate Print and Loop

Understanding variables



Understanding functions



Hoisting:

Javascript loads all the function definitions and var variables at the top of the script hence you can access them before declaring them, however the initializations wouldn’t hoisted

i.e.,

console.log(x)

var x = 30;

The above line only declares x, but doesn’t initialize ‘x’;

which is why you get undefined,

Note: let & const wouldn’t be hoisted you will get error if you access a variable before declaration.

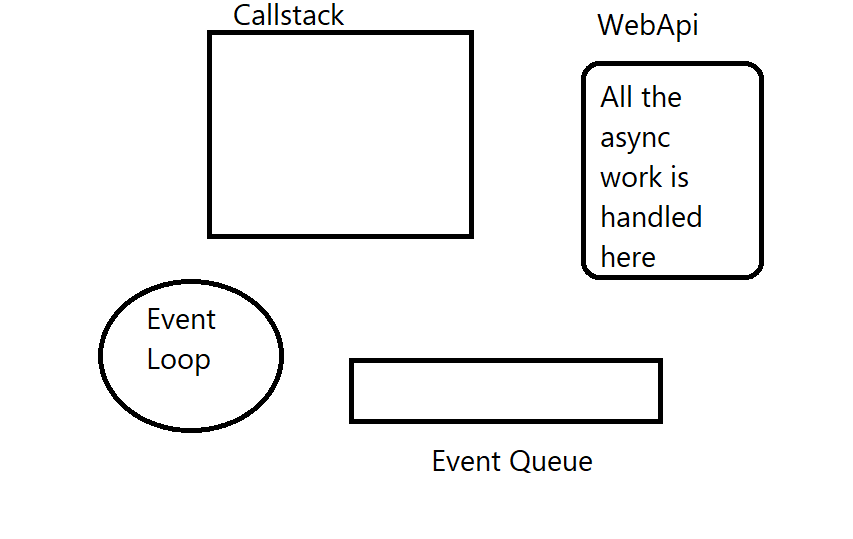
You can create javascript functions as a value but you can’t access it before if its assigned as a value.

Callback functions:

These functions are initialized first but called later based on some events like response from the server or events from some element or some actions or if timer is over, it is used mainly to perform asynchronous tasks.

Event loop mechanism

Node.js is a single threaded model, it runs scripts with only one thread using the callstack.



Once the asynchronous work is over the task will be added to the event queue and the task present in the event queue is loaded by event loop to the callstack for execution only if the call stack is empty, till that time the task will be there in the event queue.

Arrow functions: These are alternate way of writing callbacks

Note: Callbacks can also be synchronous, ex: the callbacks of forEach, map, filter and so on.

Writing arrow functions instead of callback

|  |  |
| --- | --- |
| Callback | Arrow function |
| function() {   return 20; } | () => 20; |
| function(x, y) {  return (x + y);  } | (x, y) => x + y |
| function(x, y) {  console.log(x + y);  return (x+ y);  } | (x, y) => {   console.log(x + y);  return (x + y); } |
| function() {  console.log(“hello”); } | () => console.log(“hello”) |
| function(x) {  return x + 20; } | x => x + 20 |

Rest & Spread operator

Rest operator accepts 0 or more arguments

function foo(x, y, …z) { }

Here …z is a rest parameter which can accept 0 or more arguments

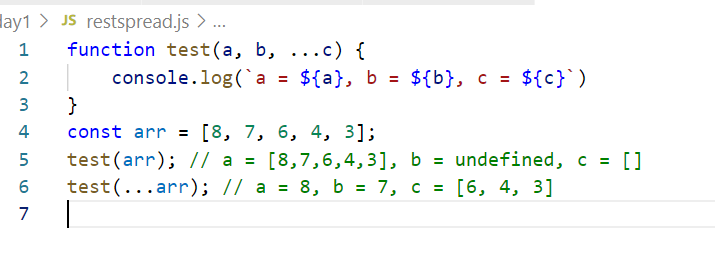
Note: You can have maximum one rest parameter & it must be the last parameter

Spread operator spreads value to multiple parameters, like each value present in the array can be assigned to separate parameters

function foo(x, y, z) { }

arr = [5, 1, 2];

foo(…arr); 5 is passed to x, 1 to y & 2 to z.



Using Rest parameter create a function that can accept a key & an array, the function must iterate the array & find the key is present or not

ex:

key = 2;

arr = [7, 8, 2, 3, 5]

search(key, …arr) // should print 2 is present

key = 9;

arr = [7,8,2,3,5]

search(key, …arr) // should print 9 is not present

== & === operator

== checks only value but not the type

=== checks both value & type

Javascript objects

let emp = { name : “Raj”, salary: 30000 }

Nested objects

let emp = { name : “Raj”, salary: 30000, address : {state : “KA”, city : “BLR”} }

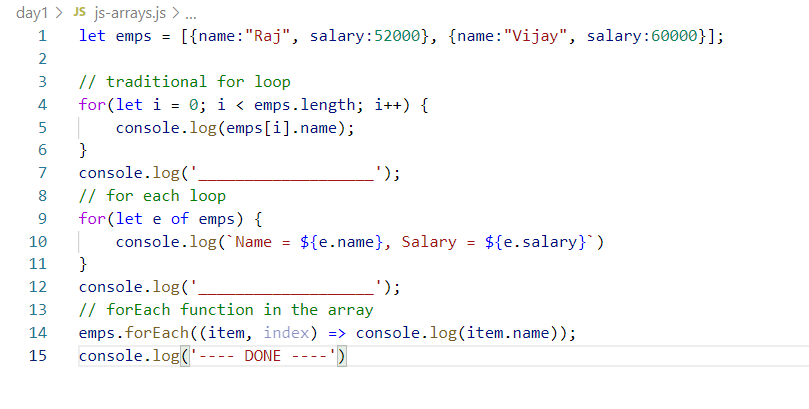
emp.address.state to access state

Javascript object array

let emps = [ {name: “Raj”, salary: 35000}, {name: ”Vijay”, salary: 40000 } ]

emps[0].name to access name of the 1st element

emps[1].name to access name of the 2nd element



Array methods: filter, map, sort, reduce, forEach, splice

forEach: It is used to iterate the elements

map: It is used to iterate the elements & generate a new array from the iterated elements, it is mainly to convert from one type to another type

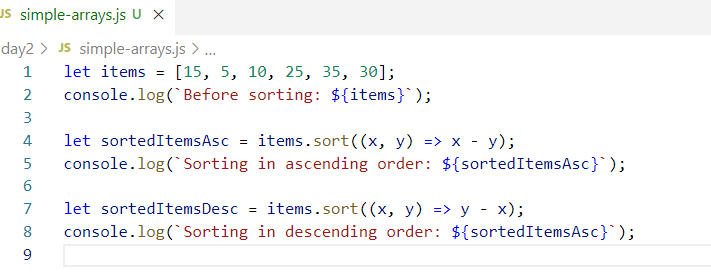
filter: It is used to iterate the elements and generate a new array that matches to specific condition

sort: it used to iterate the elements & sort based on certain conditions like sort based on price, sort based on ram size

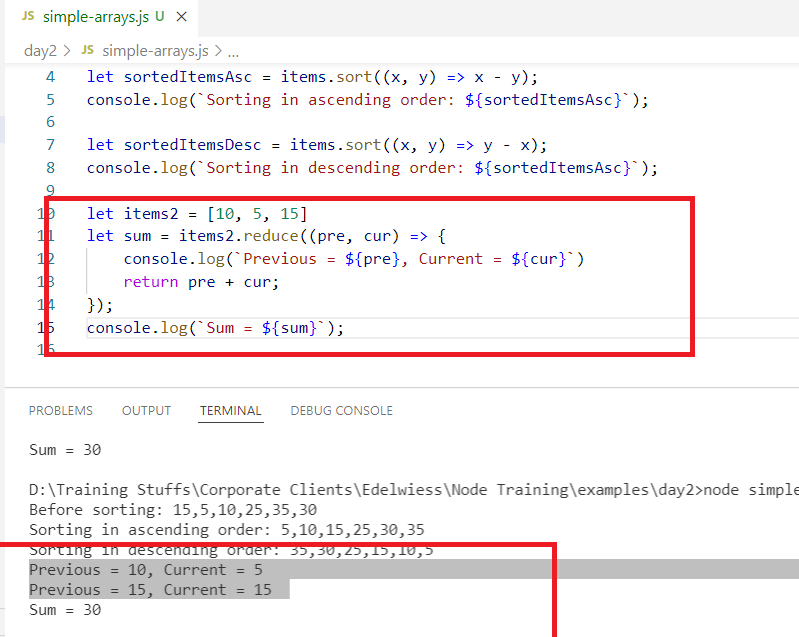
reduce: It used to iterate the elements and reduce it to a single result

splice: it used to remove the elements from the array.

Sort: It takes a callback with 2 parameters which will be compared to sort the data and store them based on the value you get while comparing



reduce: It iterates over the elements in the array & reduces into a single value



Activity:

products = [   
 {name : “chocolate”, qty : 3, price : 50 }, {name: “ice cream”, qty: 5, price: 100},  
 {name : “cake”, qty : 2, price : 70 }, {name: “milk”, qty: 5, price: 20}  
];

1. Sort the above products based on the quantity
2. Display the total price of all the products as a single result i.e., qty \* price

Benefits of Node.js

* You can write server side programs using Javascript
* Developers who already have knowledge in Javascript for front end can easily switch to the back end development without learning a new language
* JSON is usually used to exchange the data between front-end and back-end & JSON is the native language in Javascript & you don’t need any complex parsers
* Node.js also follows ECMA Standard so it can understand all the new features of Javascript
* Node.js provides NPM (Node Package Manager) using which you can download any Javascript libraries from the internet

Node.js uses a file called package.json which is a very important file in any node project, it contains the complete project informations like name, version, dependencies, commands, you can use npm init -y to create package.json

Modules

These are reusable unit of code which could be a variable, function, class you can import in other javascript files and use them

There are 3 types of modules

1. Core: Available in the node itself like os, fs, http
2. Local: Created within the project
3. 3rd Party: Need to download from the internet like express, body-parser, readline-sync

Exporting & Importing

db.js  
module.exports.dbOperations = {   
 store: function() { … } ,

fetch: function() { …. }  
}

a.js

let dbop = require(“db.js”);

dbop.store();

dbop.fetch();

The above exports & imports are used before ES6, but in ES6 you have export & import keyword which are much simpler to use

db.js

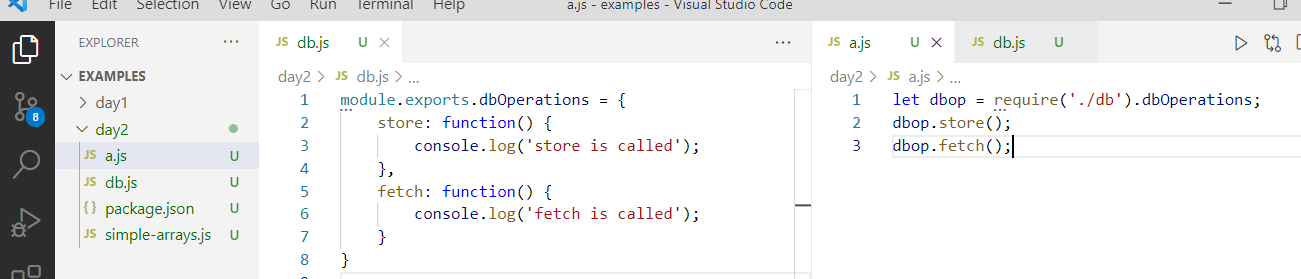
export function store() { … }  
export function fetch() { … }

a.js

import { store, fetch } from ‘./db.js’;

Note: If you want to use es6 import & export then you must specify type : module in the package.json, because by default node.js uses common module and you must use module.exports & require.

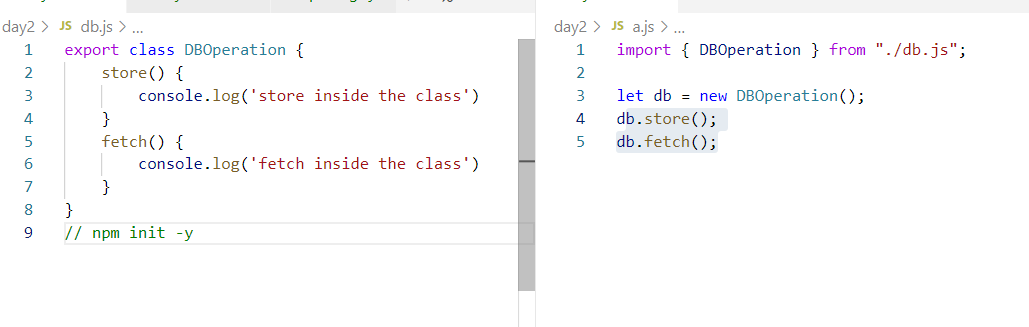
Using old way of import & export



New method of importing and exporting

package.json

{   
 type : “module”  
}



Two types of exports

1. Named exports: You need to import it by name
2. Default export: You can use any name while importing

Note: You can use default exports when the javascript file has only one module

simple.js

class Abc {   
  
}

export default Abc;

demo.js

import Abc from ‘./simple.js’

or

import anyName from ‘./simple.js’; // imports Abc class only

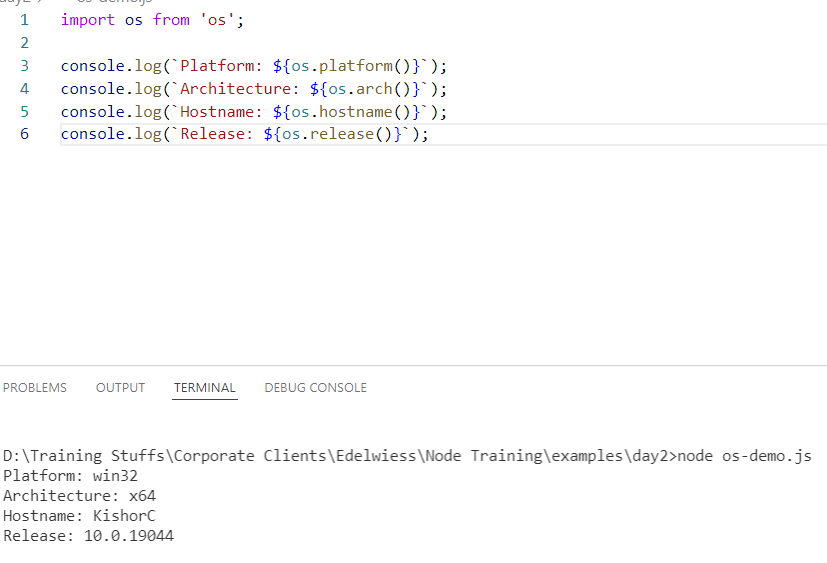
Core Module

These are inbuilt modules available in Node.js, most of them are having default exports it means you can import them using any name.

Ex: os, fs, http, url,

OS module

It helps you to get OS related information like architecture, platform, hostname and so on.



Third party module

These are the modules which are available in the internet that can be downloaded in your machine and use in the code.

npm install library-name

ex: readline-sync, express, cors, mongodb, body-parser, bootstrap

readline-sync: It is used to take input from the keyboard in the console, it gives functions that will have inbuilt feature to validate as well like questionPassword(“”) this function asks user to enter minimum 12 characters that must have digits, special characters & so on.

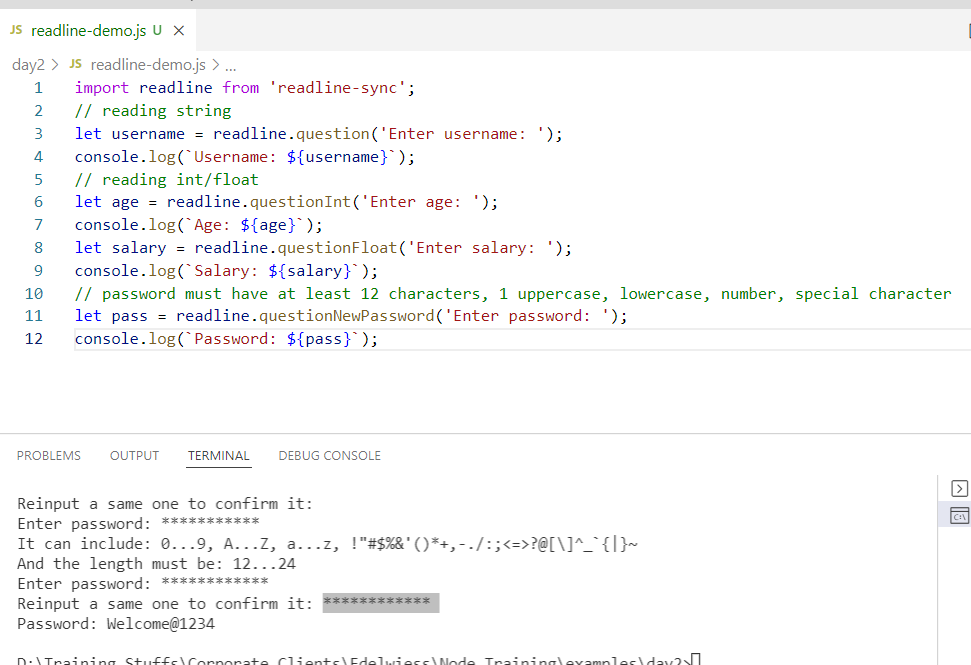
questionEmail(“”) this function asks user to enter valid email id

questionInt : to read number

questionFloat: to read floating point numbers

question: to read strings

To install: npm install readline-sync



FS Module

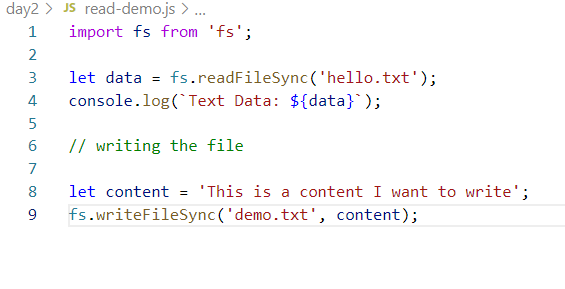
It is used to read & write files

Note: You can read the text file but you get binary data which you need to convert into string format

import fs from ‘fs’;

fs.readFileSync(filename);

fs.writeFileSync(filename, content);



Activity:

Take text from the keyboard and write the content to the file, but ensure that old content is not erased

Reading & Writing JSON data

JSON is commonly used format to exchange the data, when you want to write to JSON firstly you must a Javascript object which needs to be converted to JSON and write that JSON to the file, similarly when you read the JSON file you will get a JSON text which must be converted to Javascript object if you want to access any property of the JSON.

JSON.stringify(jsobject) converts Javascript to JSON

JSON.parse(json) converts JSON to Javascript



Take input from the keyboard repeatedly until you enter -1 to quit, ask user to enter id, name & salary and store those data in an object, then add that object to the array until you enter -1, once you enter -1 quit the loop and store that array in the JSON file so that you must see all the objects in a JSON file.

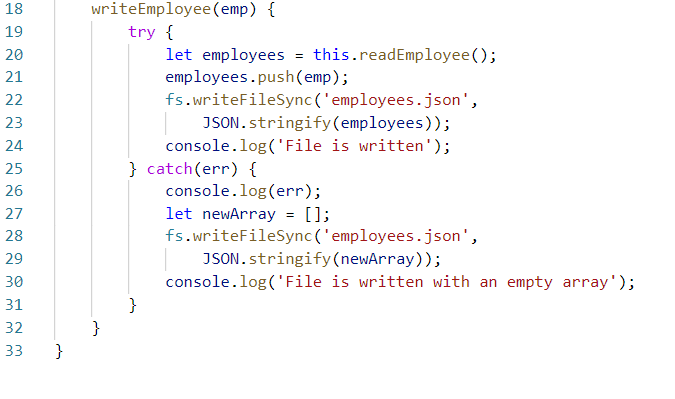
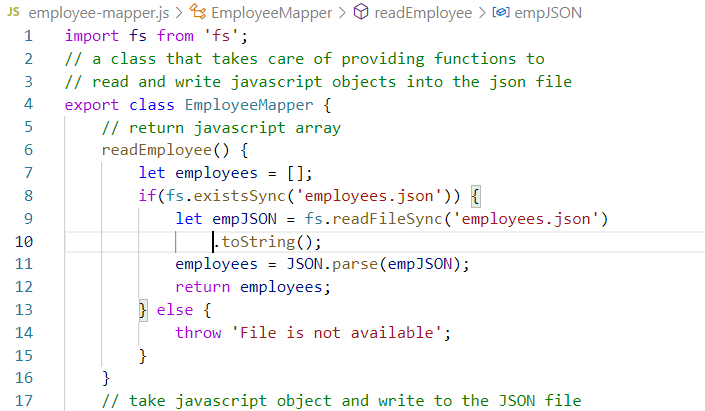
Day 3 Agenda

1. Writing & Reading JSON files
2. Web application with HTTP module
3. RESTful webservices with Express module
4. Jade & Stylus templates
5. Socket.io - custom events

How to properly read & write json files without erasing old json data.

We need to create a reusable module that can take care of reading the json data from the file and writing the json data to the file without losing the old data.

employee-mapper.js



HTTP module

It is used to create web applications using node.js, it provides features that helps in creating & starting servers

import http from ‘http’;

// creating server

let server = http.createServer( ( req, res ) => } } );

// start the server

server.listen( ( ) => { } );

[ or ]

http.createServer( (req, res) => { } ).listen (( ) => { } );



Reading query-parameters from the clients

request.url: reads the url string, but it will be in string format

urlModule.parse(request.url, true); It converts URL string to url object

urlObject.query.name: this extracts query object from the url & its parameter

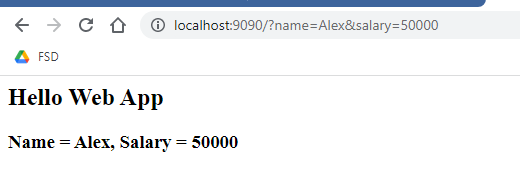
?name=Alex&age=30

urlObject.query.name

urlObject.query.age



Output:



Express Module

You can create Web Api’s (Restful webservices) in your application

Webservices are online services / APIs which helps different technologies to share the data in a common format called JSON, the end applications (front-end or backend or any type of applications) must able to convert JSON to suitable language format the end application understands

ex: Phonepay can fetch balance from any bank applications, IRCTC apps can send data to Phone pay while booking the ticket, Amazon application can send data to any banking applications

Principles of developing webservices

1. Web service must have some URL which the clients can access
2. Web service uses HTTP methods to perform CRUD operations based on the type of operations you must use different type of HTTP methods as per the standard
   1. Post: Create a new resource
   2. Put: Modify the existing resource
   3. Get: Fetch the resource
   4. Delete: Delete the existing resource

Note: Express module is a third party module, we need to install it

# npm install express

We need to create an express object that provides methods to perform operations in terms of http methods

import express from ‘express’;

let app = express();

app.listen(9090); // starts the server

app.get(url, (req, res) => { } )

app.post(url, (req, res) => { } )

app.js



Sending JSON from the client

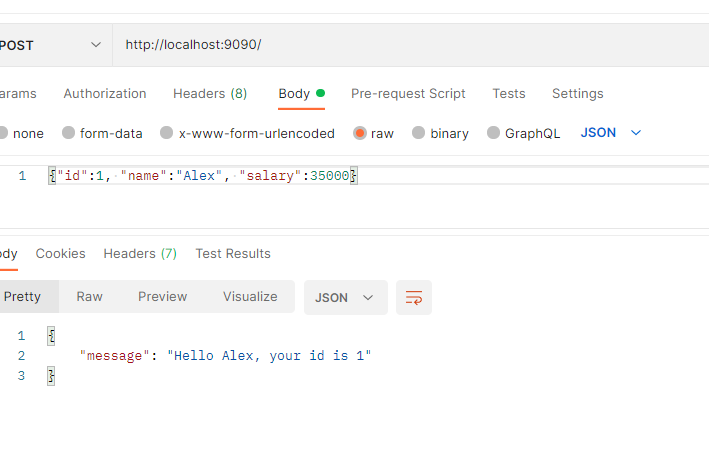
You can use postman body to enter JSON { … } in the request & send the JSON data to the webservice, your webservice must parse the JSON and convert to the appropriate types like Javascript, Java, Python and etc.

Note: You can’t send data when the method is GET.

Express uses a body parser to convert JSON to Javascript object, you need to add the middleware to parse the JSON



Output:



Path Parameters:

These are dynamic paths which will have client’s input so that the webservice with the same URL must able to extract these dynamic paths

Ex:

employee/100

employee/150

employee/200

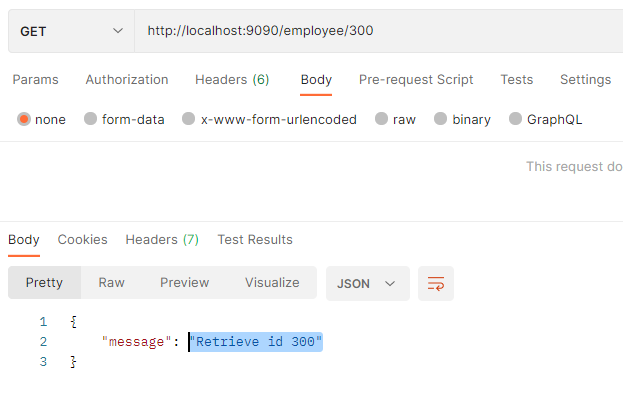
All the above URLs have same static path employee and dynamic value 100, 150 & 200, we need a webservice that will have a parameterized URL so that it can extract all the dynamic values

/employee/:someName

:someName will be a parameterized path that can accept any value



Output:



You can also pass multiple parameters if you have multiple parameter paths

ex: /employee/:empId/:empSalary

Then you can send request as /employee/100/25000

Activity:

Perform CRUD operations with HTTP methods like post, get, put & delete and manage the data in JSON file, ensure the JSON file doesn’t lose old data

1. Store: Pass the JSON data from the postman which must be stored in JSON file
2. Retrieve: When you enter /employees it must show all the JSON data in the response present in JSON file
3. Retrieve by Id: When you enter /employees/1 it must show the employee json data in the response if id is present else it must show employee id 1 not found in the JSON format
4. Delete by id: When you enter /employees/1 it must delete the JSON data from the JSON file if id is present else it must show employee id 1 not found in the JSON format
5. Update salary based on id: When you enter /employees/1/40000 it must update the salary of employee id 1, if id is not present show the error message employee id 1 not found in the JSON format

Query parameters:

These are data that will have key values in the URL which can be extracted in the express.js using request.query

path/subpath/?key=value&key=value

Template Engine:

When a node.js wants to render dynamic content to the clients in HTML format we can use template engines which will use variables in the template files that are replaced by actual values which makes the content dynamic

Note: HTML file is only for static content, they don’t support variable usage, however using template engines you can make your HTML to be dynamic using variables

There are many template engines node.js supports, but by default express.js supports Jade, some of the other template engines are:-

* Pug
* Mustache
* EJS
* Jade

Jade uses a syntax where the html elements will follow indentation spaces to represent the content

ex:

html

head

title

body

h2

h3

table

Installing Jade

npm install jade

Styling the HTML

You can use Stylus which follows the same syntax of Jade to style your template

CSS

p { color : #FF0000} // red color

Stylus

p

color : #FF0000

Note: You can also use 3rd party css in your stylus using @import

@import

url : path

How to use styles in the Jade

In HTML

<p class = ‘container’>

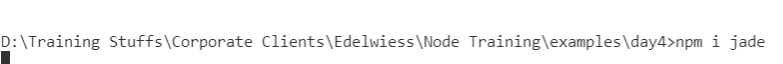
In Jade

p.container

content

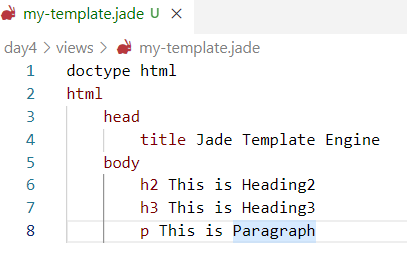
Note: We need to keep jade templates in the views folders, by default express.js locates the templates in the views folder

Installing jade



We need to set jade as the template engine hence we need jade library in our project

views/my-template.jade

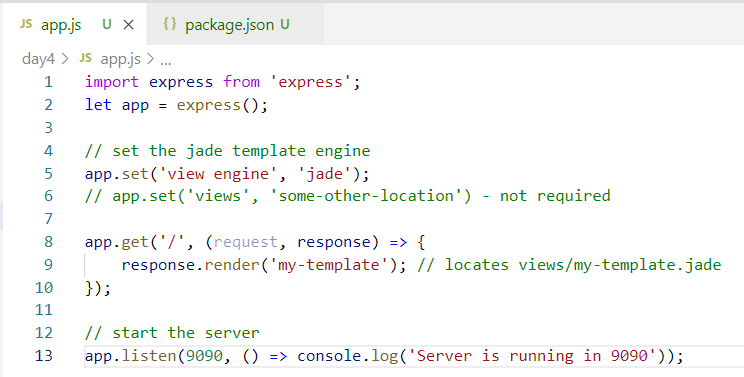


In Express.js you don’t have to set the location of template, because it by default uses views folder

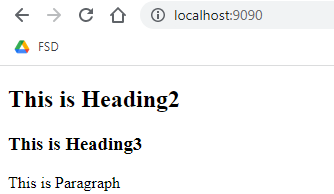
Installing express.js

npm install express

app.js



Output:



Using variables in the template

We need to share the data to the template in the form of key & value, & in the template we must use the key to show the data

Ex:

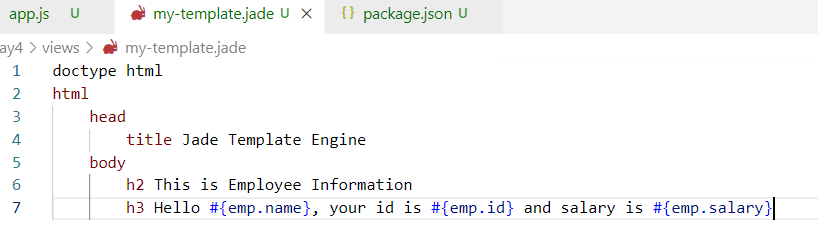
response.render(“my-template”, {emp: {id: 100, name: “Raj”, salary: 4000} } );

Template

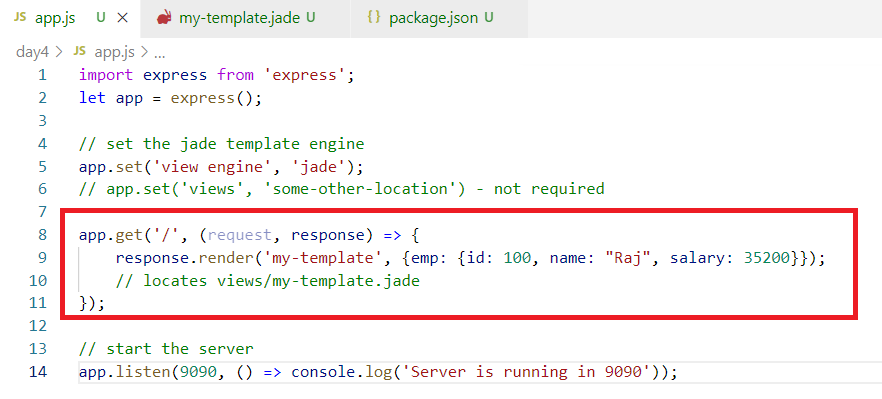
body

h2 Hello #{emp.name}, id is #{emp.id} and salary is #{emp.salary}

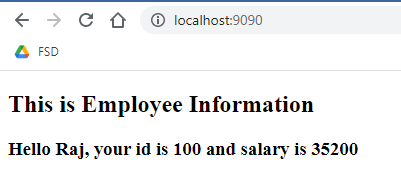
my-template.jade



app.js



Output:



How to iterate over a key that has an array

You can use each-in loop in the template

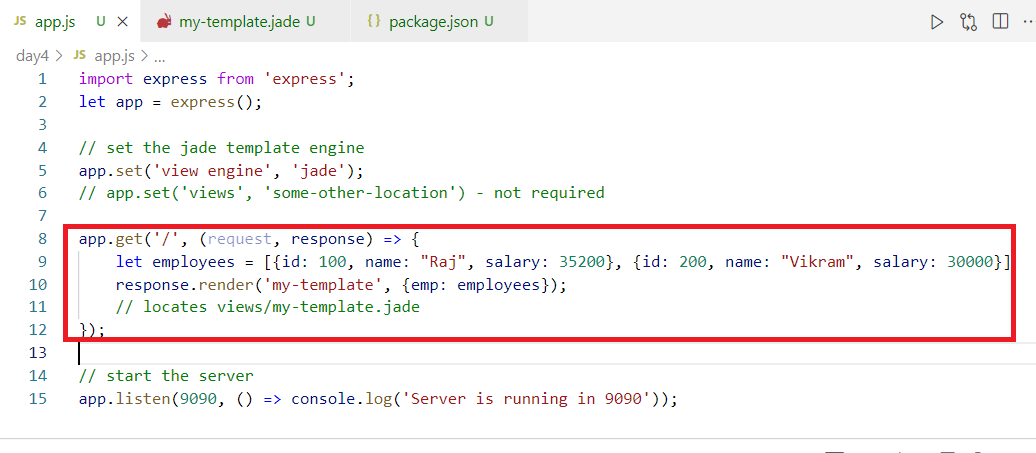
template

ul

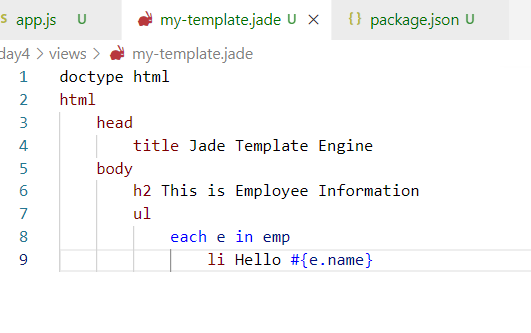
each x in employees

li Hello #{x.name}, ID is #{x.id}

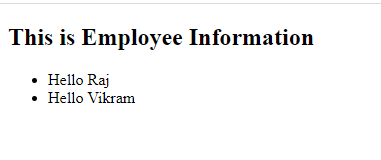
app.js



my-template.jade



Output:



MongoDB

It is a NoSQL database which maintains unstructured data in JSON format which is called as documents

Documents: These are JSON objects which are like records

These documents are maintained in a collection

Collection: It is a container that keeps the documents

MongoDB provides two commands

1. mongod: This is used to start the database server, ensure data/db is created in the root director/drive from the place you type mongod from the command prompt
2. mongo: This is a shell to directly interact with the database to perform CRUD operations

Note: MongoDB supports javascript & it is case sensitive

Create collection

db.createCollection(“employee”);

Insert the document in a collection

db.employee.insertOne( {\_id: 1, name : “Raj” } );

db.employee.insertMany ( [ { .. }, { .. }, { .. } ]); // to store multiple documents

Note: MongoDB maintains unique document using \_id, it will be auto-generated if you don’t provide value to \_id

Find all the documents or single document

db.employee.find(); // returns all the documents

db.employee.find({\_id: 100}); // returns the document having \_id: 100

Note: In MongoDB you need to create a Javascript function if you want to auto-generate value for \_id & you need to call that function while storing the document

function sequenceGen() { … } // it must return next value on each call

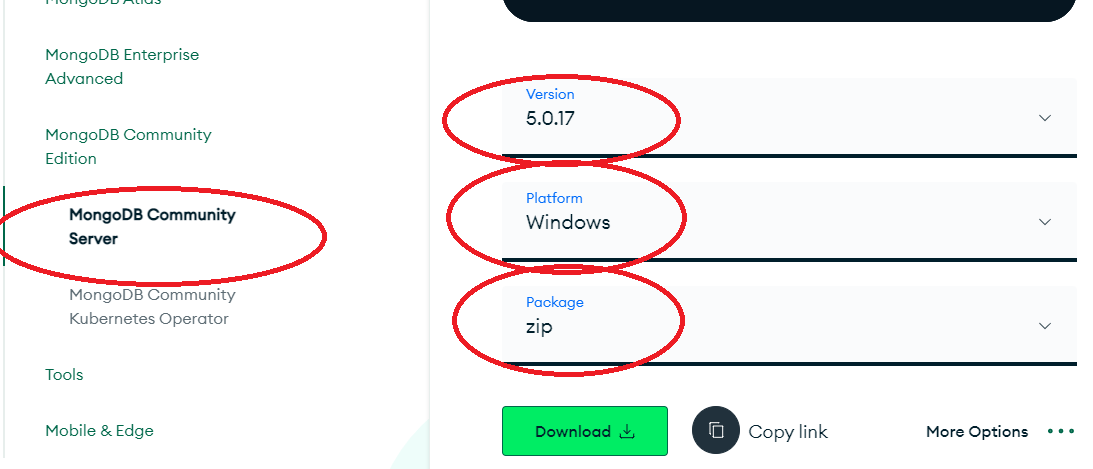
ex:

sequenceGen() : 199

sequenceGen() : 200

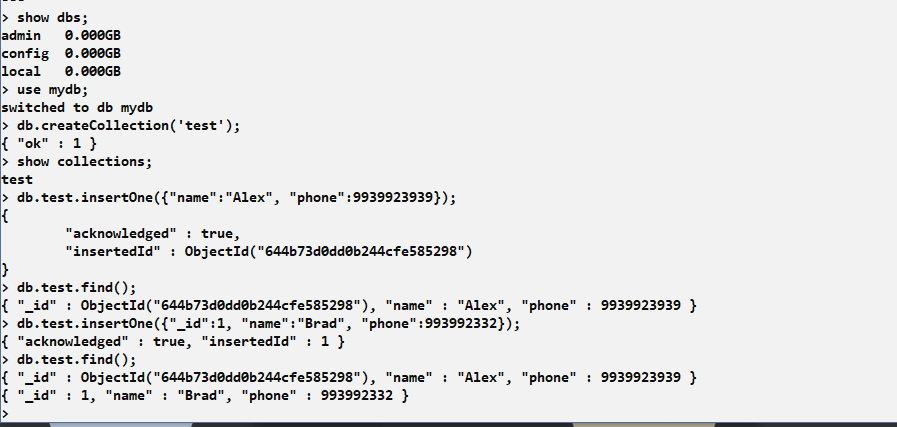
You can call this function while storing the documents so that you will have sequence for \_id

ex: db.employee.insertOne( { \_id: sequenceGen(), name: “…..” } );

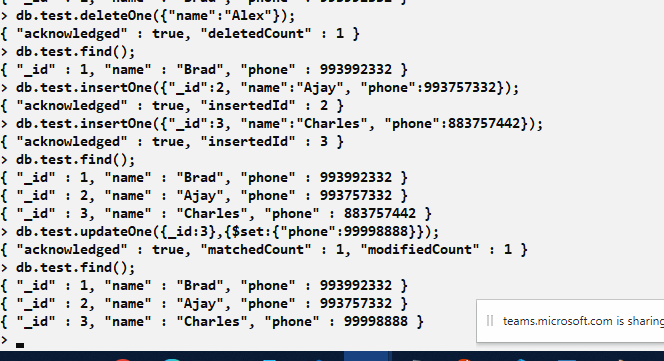


Steps

1. Extract the zip folder
2. Create data/db folder in the root drive
3. Start the mongod
4. Strat the mongo



Delete & Update



DeleteMany & UpdateMany

db.test.updateMany({condition}, {$set: { property : value }}); Updates multiple documents

db.test.deleteMany({ });

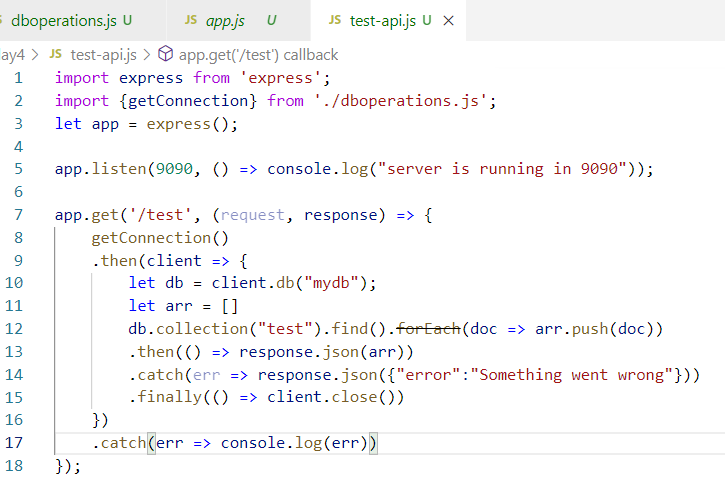
Interacting with the database using node.js & mongodb library

npm install mongodb

Performing store



Using express & mongodb



Storing the data using express



MEAN Stack

It’s the collection 4 different technologies MongoDB, Express, Angular & Node.js from Angular you can create front end applications, from other technologies you can create backend applications

CORS: Cross Origin Resource Sharing is what we need to do here as Angular runs in a different domain and backend runs in different domain, by default backend doesn’t accept request from a different domain hence we need to enable the CORS

We need to download cors library along with express, mongodb

#Installing

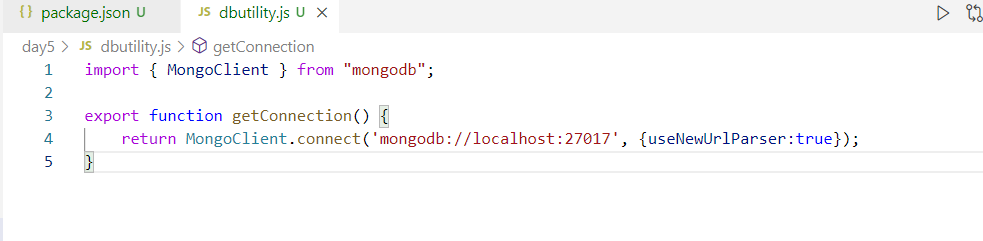
npm install express cors mongodb



Update package.json to use type: module

Create a reusable code that establishes database connection

dbutility.js



Now we will create APIs to perform CRUD operations on Users collection

app.js

import express from "express";

import { getConnection } from "./dbutility.js";

import cors from "cors";

let app = express();

// using body parser & cors in the middleware

app.use(express.json()); // parse the json from the request

app.use(cors()); // enabling the cors

// start the server in some port

app.listen(9090, () => console.log("Server running in 9090"));

// API to store

app.post('/users', (request, response) => {

    let body = request.body;

    let connection = getConnection();

    connection

    .then(client => {

        let db = client.db("mydb");

        // you can generate value for \_id before calling insertOne

        db.collection("users").insertOne(body)

        .then(value => response.status(201).json(value))

        .catch(err => response

            .status(409).json({"error":err, "message":`ID: ${err.keyValue.\_id} already exists`}))

        .finally(() => client.close())

    })

    .catch(err => response.status(500).json({"error":err, "message":"Database Connection Error"}))

});

// API to retrieve the data based on the id

app.get('/users/:id', (request, response) => {

    let id = parseInt(request.params.id);

    let connection = getConnection();

    connection.then(client => {

        let db = client.db("mydb");

        db.collection("users").findOne({\_id: id})

        .then(value => {

            if(value == null || value == undefined) {

                response.status(404)

                .json({"error":`Record not found with an id: ${id}`, "value":value})

            } else {

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

            }

        })

        .catch(err => response.status(404).json(err))

        .finally(() => client.close());

    })

    .catch(err => response.status(500).json({"error":err, "message":"Database Connection Error"}))

});

// API to retreive all the records

app.get("/users", (request, response) => {

    let connection = getConnection();

    connection.then(client => {

        let db = client.db("mydb");

        db.collection("users").find().toArray()

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

        .finally(() => client.close())

    })

    .catch(err => response.status(500>json({"error":err, "message":"Database Connection Error"})));

});

// API to delete the record based on id

app.delete("/users/:id", (request, response) => {

    let id = parseInt(request.params.id);

    let connection = getConnection();

    connection.then(client => {

        let db = client.db("mydb");

        db.collection("users").deleteOne({\_id: id})

        .then(value => {

            if(value.deletedCount == 0) {

                response.status(404).json({"error":`Record with an id: ${id} is not found`})

            } else {

                response.status(200).json({"message":`Record with an id: ${id} is deleted`})

            }

        })

        .catch(err => response.json(err))

        .finally(() => client.close());

    })

    .catch(err => response.status(500>json({"error":err, "message":"Database Connection Error"})))

});

// API to update the dob based on the id

app.put('/users/:id/:dob', (request, response) => {

    let id = parseInt(request.params.id);

    let birthday = request.params.dob;

    let connection = getConnection();

    connection

    .then(client => {

        let db = client.db("mydb");

        db.collection("users").updateOne({\_id: id}, {$set: {dob: birthday}})

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

        .catch(err => response.json(err))

        .finally(() => client.close())

    })

    .catch(err => response.status(500>json({"error":err, "message":"Database Connection Error"})))

});

Connecting the front-end & back-end

You can use front-end technologies like React.js, Angular Framework, Vue.js, Ember.js, Javascript to interact with any back-end

All these technologies helps you to create web & mobile apps

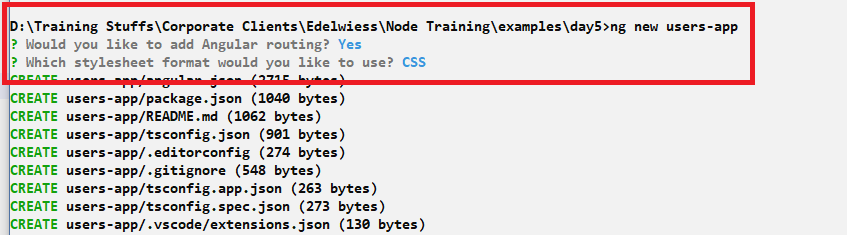
ex: React has React Native, Angular has Ionic

Installing Angular

npm i -g @angular/cli

Create angular project

ng new app-name i.e., ng new users-app



In order to access backend service you need to create angular service that utilizes HttpClient object

Note: HttpClientModule must be imported in the app.module

You must use ng g s service to create an angular service

app.module.ts

@NgModule( {

imports : [ HttpClientModule]

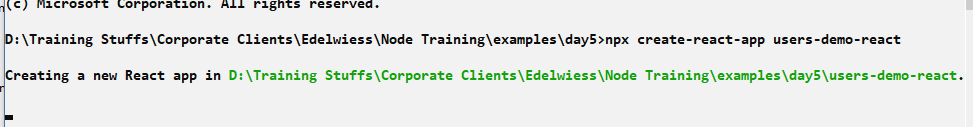
} )

ng g s service-name

Accessing the same backend from the React.js

We must use axios library to access the backend from the react.

npx create-react-app project-name



In React we need use axios to access the backend and useState to modify the components state.

let [users, setUsers] = useState([]);

The above code initializes users to [ ]

setUsers is used to modify the users variable

We need to send request to the webservice & initialize the users and through users.map() function we can iterate over the array.

How to deploy node.js application on AWS

<https://sumantmishra.medium.com/how-to-deploy-node-js-app-on-aws-with-github-db99758294f1>

Testing framework - unit testing

Mocha & Chai: Which is a testing framework to test backend & front end applications

Jest: For front-end.

Cloud DB:

It is used when the applications want to access a centralized database, ex: In AWS you have DynamoDB

npm install typescript

tsc hello.ts >> hello.js

node hello.js

Docker & Kubernetes: